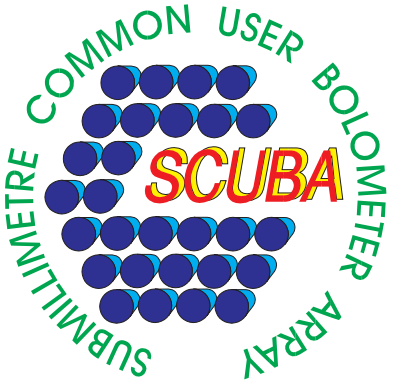
SSN/72.2

Starlink Project
Starlink System Note 72.2

Tim Jenness, John F. Lightfoot
Joint Astronomy Centre, Hilo, Hawaii

10 July 2000

SURF Programming Interface



Abstract

This document describes the SURF programming interface. Including the library API, instructions on how to build a SURF task, the SURF file format and instructions on how to write a SURF import task.

Contents

1 Introduction

The SCUBA User Reduction Facility (SURF, SUN/33) provides data reduction software for submillimetre bolometer cameras. It was specifically designed for the Submillimetre Common-User Bolometer Array (SCUBA; Holland et al 1999, MNRAS, 303, 659) but could be ported to similar bolometer cameras. This document describes the programming interface used to write SURF tasks and import foreign data formats into the system.

2 SURF File Format

SURF uses that standard Starlink NDF (SUN/33) file format. This is a hierarchical format based on HDS (SUN/92) and has native support for variances, quality flags, non-linear axes and arbritrary extensions.

2.1 Raw data format

SCUBA raw data (i.e. data in the format expected by the SURF task reduce_switch) has the following general layout:

```
STRUCT  <NDF>

   MORE           <EXT>           {structure}
      FIGARO         <FIGARO_EXT>    {structure}

      SCUCD          <SCUCD_ST>      {structure}

      SCUBA          <SCUBA_ST>      {structure}

      FITS(171)      <_CHAR*80>      'ACCEPT  = 'not used'           / ac...'
                                     ... 'WAVE_5  =                ...','END'

   DATA_ARRAY(5,130,900)  <_REAL>   0.0102353,1.9114159E-6,0.0184784,
                                    ... 1.7097691E-5,0.01525085,1.841503E-5,0
```

It is a very simple NDF format with a single DATA_ARRAY and four extensions. The extensions are described in section 2.3. The DATA_ARRAY is three dimensional with the following meanings:

(1) The size of the first dimension depends on the observing mode. For jiggle and noise observations it has size 5 corresponding to the demodulated data, variance on demodulated data, internal calibrator signal, variance on calibrator signal and data quality. For SCAN/MAP observations it has size 4 corresponding to the demodulated data, variance on demodulated data, internal calibrator signal and data quality (i.e. the calibrator variance is not stored for scan maps). The data quality is converted from a REAL number to a standard UBYTE quality (appendix 4): if the quality is odd, this indicates the flatfield was set to zero so bit 1 is turned on (this is also done using the flatfield task), if the

quality is greater than the value specified via the SPIKE_LEVEL parameter (i.e. there were more spikes in the individual sample than the specified threshold) bit 2 is turned on. For SKYDIP observations it has a size of SCUBA__N_TEMPS (see Appendix C, usually 3) corresponding to the different temperatures measured (AMBIENT, SKY and COLD).

(2) The second dimension corresponds to the number of bolometers observed. This should match the size of the BOL_CHAN and BOL_ADC arrays in the SCUBA extension.

(3) The third dimension is the time axis and is equal to the total number of samples (of length EXP_TIME). This dimension must be greater than any value stored in the DEM_PNTR array. For a JIGGLE observation the size can be calculated (assuming the observation was not aborted early)

2.2 SURF Format

The reduce_switch task converts the raw SCUBA format into a more NDF-like format with proper variance and quality arrays (extracted by reduce_switch from the SCUBA data array), axes, history reporting, a title and units.

Here is an example of the reduce_switch output for a Mars pointing observation:

```
010  <NDF>

   DATA_ARRAY(37,32)  <_REAL>      -0.003304771,0.0002364833,0.00320935,
                                   ... 0.0003040197,0.001364953,-0.007496601
   MORE           <EXT>            {structure}
      FIGARO         <FIGARO_EXT>    {structure}
         {structure is empty}

      SCUCD          <SCUCD_ST>      {structure}
         JIGL_X(16)     <_REAL>         5.3766,10.6914,10.6914,5.3766,0,
                                        ... -10.6914,-5.3766,0,5.3766,-5.3766
         JIGL_Y(16)     <_REAL>         -3.09,0,6.18,9.27,12.36,9.27,6.18,
                                        ... 6.18,0,-3.09,-6.18,-9.27,-9.27
         DEC1           <_REAL>         0
         DEC2           <_REAL>         0
         LST_STRT(2,1,2,1)  <_DOUBLE>   3.9425429572598,3.9440140603798,
                                        3.9469952223685,3.9454089275179
         RA1            <_REAL>         0
         RA2            <_REAL>         0

      SCUBA          <SCUBA_ST>      {structure}
         BOL_CALB(16,9)  <_REAL>        1.286,0.9311,0.9499,1.002,0.9585,
                                        ... 1.054,1.065,1.045,1,1,1,1,1,1,0
         BOL_DU3(16,9)  <_REAL>         57.08,45.9,36.15,24.64,14.55,3.62,
                                        ... -15.78,9.047,-54.65,0,0,0,0,0,0
         BOL_DU4(16,9)  <_REAL>         -28.14,-35.91,-41.46,-49.62,
                                        ... 63.89,77.76,71.16,0,0,0,0,0,0
         BOL_QUAL(16,9)  <_INTEGER>     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                        ... 0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1
         BOL_TYPE(16,9)  <_CHAR*20>     'SHORT','SHORT','SHORT','SHORT',
                                        ... 'P...','P1350_DC','P1100_DC','BAD'
         BOL_ADC(37)    <_INTEGER>      7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,
```

```
                                        ... 8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9
      BOL_CHAN(37)   <_INTEGER>         1,2,3,4,7,8,9,10,11,13,14,15,16,1,
                                        ... 12,13,14,15,16,1,2,3,4,5,6,7,8,9
      FLAT_ADC       <_INTEGER>         0
      FLAT_CHN       <_INTEGER>         0
      FLAT_IND       <_INTEGER>         0
      PHOT_BB        <_INTEGER>         0
      ISTART         <_INTEGER>         0
      NPIX           <_INTEGER>         0
      POINTER        <_INTEGER>         0
      DEM_PNTR(1,2,1)  <_INTEGER>       1,17

   FITS(171)      <_CHAR*80>        'ACCEPT  = 'PROMPT  '           / ac...'
                                    ... 'WAVE_5  =                ...','END'
   REDS           <SURF_EXTENSION>   {structure}
      BEAM_WT(3)     <_REAL>         0,1,0

HISTORY        <HISTORY>        {structure}
   CREATED        <_CHAR*24>       '1999-AUG-11 17:04:57.000'
   CURRENT_RECORD  <_INTEGER>      1
   RECORDS(10)    <HIST_REC>       {array of structures}

   Contents of RECORDS(1)
      DATE           <_CHAR*24>       '1999-AUG-11 17:04:57.000'
      COMMAND        <_CHAR*29>       'REDUCE_SWITCH   (SURF V1.4-3)'
      USER           <_CHAR*4>        'timj'
      HOST           <_CHAR*6>        'lapaki'
      DATASET        <_CHAR*33>       '/local/jcmt/scratch/timj/orac/o10'
      TEXT(3)        <_CHAR*72>       'Parameters: IN=@10 MSG_FILTER='N...'
                                      ... 'Software: /star/bin/surf/redu...'

QUALITY        <QUALITY>        {structure}
   QUALITY(37,32)  <_UBYTE>          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                     ... 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   BADBITS        <_UBYTE>          7

VARIANCE(37,32)  <_REAL>         1.0950339E-6,7.1230471E-7,6.6492021E-7,
                                 ... 4.2760187E-9,7.4043317E-5,4.8094225E-6
AXIS(2)        <AXIS>            {array of structures}

Contents of AXIS(1)
   DATA_ARRAY(37)  <_REAL>           1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
                                     ... 26,27,28,29,30,31,32,33,34,35,36,37
   LABEL          <_CHAR*9>         'Bolometer'

LABEL          <_CHAR*8>         'Raw data'
TITLE          <_CHAR*30>        'mars'
UNITS          <_CHAR*5>         'Volts'

End of Trace.
```

The data arrays are now 2 dimensional, corresponding to the bolometer number and time axis. The axis information is written so that the integration number is reflected on the Y axis. Additionally, a new extension, `REDS`, is created to hold the relative beam weights; for maps these

are always set to 0,1,0.

A photometry observation is slightly different from a map observation in that the output data array is now 3 dimensional rather than 2:

```
0107  <NDF>

   MORE           <EXT>           {structure}
      FIGARO         <FIGARO_EXT>    {structure}
         {structure is empty}

      SCUCD          <SCUCD_ST>      {structure}
         JIGL_X(9)      <_REAL>         0,2,2,0,-2,-2,-2,0,2
         JIGL_Y(9)      <_REAL>         0,0,2,2,2,0,-2,-2,-2
         DEC1           <_REAL>         0
         DEC2           <_REAL>         0
         LST_STRT(2,1,8,1)  <_DOUBLE>   1.8834042737017,1.8844116777459,
                                        ... 1.8981978240679,1.8971856029149
         RA1            <_REAL>         0
         RA2            <_REAL>         0

      SCUBA          <SCUBA_ST>      {structure}
         BOL_CALB(16,9)  <_REAL>        1.286,0.9311,0.9499,1.002,0.9585,
                                        ... 1.054,1.065,1.045,1,1,1,1,1,1,0
         BOL_DU3(16,9)  <_REAL>         57.08,45.9,36.15,24.64,14.55,3.62,
                                        ... -15.78,9.047,-54.65,0,0,0,0,0,0
         BOL_DU4(16,9)  <_REAL>         -28.14,-35.91,-41.46,-49.62,
                                        ... 63.89,77.76,71.16,0,0,0,0,0,0
         BOL_QUAL(16,9)  <_INTEGER>     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                        ... 0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1
         BOL_TYPE(16,9)  <_CHAR*20>     'SHORT','SHORT','SHORT','SHORT',
                                        ... 'P...','P1350_DC','P1100_DC','BAD'
         BOL_ADC(130)   <_INTEGER>      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,
                                        ... 8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9
         BOL_CHAN(130)  <_INTEGER>      1,2,3,4,5,6,7,8,9,10,11,12,13,14,
                                        ... 12,13,14,15,16,1,2,3,4,5,6,7,8,9
         FLAT_ADC       <_INTEGER>      0
         FLAT_CHN       <_INTEGER>      0
         FLAT_IND       <_INTEGER>      0
         PHOT_BB(3,2)   <_INTEGER>      0,112,0,0,46,0
         ISTART         <_INTEGER>      0
         NPIX           <_INTEGER>      0
         POINTER        <_INTEGER>      0
         DEM_PNTR(1,8,1)  <_INTEGER>    1,10,19,28,37,46,55,64

      FITS(171)      <_CHAR*80>      'ACCEPT  = 'not used'           / ac...'
                                     ... 'WAVE_5  =                ...','END'
      REDS           <SURF_EXTENSION>   {structure}
         BEAM_WT(3)     <_REAL>         0.5,1,0.5

   HISTORY        <HISTORY>       {structure}
      CREATED        <_CHAR*24>      '1999-AUG-11 17:08:24.000'
      CURRENT_RECORD  <_INTEGER>     1
      RECORDS(10)    <HIST_REC>      {array of structures}
```

```
   Contents of RECORDS(1)
      DATE           <_CHAR*24>      '1999-AUG-11 17:08:24.000'
      COMMAND        <_CHAR*29>      'REDUCE_SWITCH   (SURF V1.4-3)'
      USER           <_CHAR*4>       'timj'
      HOST           <_CHAR*6>       'lapaki'
      DATASET        <_CHAR*34>      '/local/jcmt/scratch/timj/orac/o107'
      TEXT(3)        <_CHAR*72>      'Parameters: IN=@107 MSG_FILTER='...'
                                     ... 'Software: /star/bin/surf/redu...'

DATA_ARRAY(130,72,3)  <_REAL>   -0.0002527004,-0.0006291242,
                                 ... 1.1537035E-5,0.001769435,0.001728294
QUALITY        <QUALITY>       {structure}
   QUALITY(130,72,3)  <_UBYTE>    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                  ... 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
   BADBITS        <_UBYTE>        7

VARIANCE(130,72,3)  <_REAL>    3.6919182E-6,3.6207405E-6,4.2371025E-6,
                               ... 1.3244949E-8,0.0002660675,2.1535034E-5
AXIS(3)        <AXIS>          {array of structures}

Contents of AXIS(1)
   DATA_ARRAY(130)  <_REAL>       1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
                                  ... 122,123,124,125,126,127,128,129,130
   LABEL          <_CHAR*9>       'Bolometer'

LABEL          <_CHAR*8>       'Raw data'
TITLE          <_CHAR*30>      'hltau'
UNITS          <_CHAR*5>       'Volts'

End of Trace.
```

The 3rd dimension corresponds to beam position. In most cases only beam 2 is used (the middle beam) but for two or three bolometer chopping the 1st and 3rd beams will be accessed. In general, the 1st and 3rd beams are simply scaled versions of the middle beam; the relative weighting of the beams is stored in the newly created BEAM_WT array in the REDS extension. In the above example, the middle beam is -0.5 times the left or right beams.

2.3 Extensions

The information describing the observation can be found in the NDF extensions. SURF expects to find the following extensions in data files:

2.3.1 FITS

The FITS extension is an array of 80 character strings stored in a format identical to a standard FITS header. This contains general information about the observation but no array information. Listings of full SCUBA FITS headers can be found in Appendix D. SURF only requires a subset of the full SCUBA headers. For a jiggle map these are listed in table 1.

OBJECT	Name of object
RUN	Run number of observation
MODE	Type of observation (MAP, PHOTOM, SKYDIP etc)
SAM_MODE	Sampling method (JIGGLE or RASTER)
SAM_CRDS	Coordinate frame of jiggle or scan
SAM_PA	Scan PA (not required for jiggle)
LAT	Latitude of object (eg Declination)
LONG	Longitude of object (eg Right Ascension)
CENT_CRD	Coordinate frame of LONG/LAT (RB,RJ,GA etc)
UTDATE	Date of observation in YYYY:MM:DD format
UTSTART	UT time of start of observation (HH:MM:SS)
STSTART	Local sidereal time of start of observation (HH:MM:SS)
STEND	LST of end of observation
EXP_TIME	Exposure time (seconds) of each individual sample
MAP_X	Tangent plane X offset from tracking centre
MAP_Y	Tangent plane Y offset from tracking centre
N_BOLS	Number of bolometers selected (should match data array)
STATE	State at end of observation (ABORT indicates early abort)
VERSION	Version of real-time software (see SURFLIB_PROCESS_BOLS)
JIGL_CNT	Number of offsets in jiggle pattern
J_PER_S	Number of jiggles per switch
J_REPEAT	No. of jiggle pattern repeats in switch
CNTR_DU3	Nasmyth dU3 coord of instrument centre
CNTR_DU4	Nasmyth dU4 coord of instrument centre
LAT-OBS	Latitude of observatory (degrees)
LONG-OBS	East longitude of observatory (degrees)
TELESCOP	Telescope name
INSTRUME	Instrument name
CHOP_THR	Chopper throw (arcsec)
CHOP_PA	Chopper PA 0=in lat, 90=in long
CHOP_FRQ	Chopper frequency (Hz)
CHOP_CRD	Coordinate frame of chop
CHOP_FUN	Chopper waveform (SQUARE, TRIPOS)
N_SUBS	Number of sub-instruments
SUB_1	Name of sub-instrument 1 (or 'not used')
SUB_2	Name of sub-instrument 2 (or 'not used')
SUB_3	Name of sub-instrument 3 (or 'not used')
SUB_4	Name of sub-instrument 4 (or 'not used')
SUB_5	Name of sub-instrument 5 (or 'not used')
FILT_1	Filter name for sub-instrument 1 (or 'not used')
FILT_2	Filter name for sub-instrument 2 (or 'not used')
FILT_3	Filter name for sub-instrument 3 (or 'not used')
FILT_4	Filter name for sub-instrument 4 (or 'not used')
FILT_5	Filter name for sub-instrument 5 (or 'not used')
WAVE_1	Wavelength of sub-instrument 1 (microns)
WAVE_2	Wavelength of sub-instrument 2 (microns)
WAVE_3	Wavelength of sub-instrument 3 (microns)
WAVE_4	Wavelength of sub-instrument 4 (microns)
WAVE_5	Wavelength of sub-instrument 5 (microns)

Table 1: FITS keywords required by SURF for Jiggle mapping

2.3.2 SCUBA

The SCUBA extension contains information on the instrument and the structure of the data array. It has the following format (output of hdstrace):

```
SCUBA          <SCUBA_ST>      {structure}
   BOL_CALB(16,9)  <_REAL>        1.286,0.9311,0.9499,1.002,0.9585,
                                  ... 1.054,1.065,1.045,1,1,1,1,1,1,0
   BOL_DU3(16,9)  <_REAL>         57.08,45.9,36.15,24.64,14.55,3.62,
                                  ... -15.78,9.047,-54.65,0,0,0,0,0,0
   BOL_DU4(16,9)  <_REAL>         -28.14,-35.91,-41.46,-49.62,
                                  ... 63.89,77.76,71.16,0,0,0,0,0,0
   BOL_QUAL(16,9)  <_INTEGER>     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                  ... 0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1
   BOL_TYPE(16,9)  <_CHAR*20>     'SHORT','SHORT','SHORT','SHORT',
                                  ... 'P...','P1350_DC','P1100_DC','BAD'
   BOL_ADC(128)   <_INTEGER>      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,
                                  ... 8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9
   BOL_CHAN(128)  <_INTEGER>      1,2,3,4,5,6,7,8,9,10,11,12,13,14,
                                  ... 12,13,14,15,16,1,2,3,4,5,6,7,8,9
   FLAT_ADC       <_INTEGER>      0
   FLAT_CHN       <_INTEGER>      0
   FLAT_IND       <_INTEGER>      0
   PHOT_BB(3,2)   <_INTEGER>      0,112,0,0,46,0
   DEM_PNTR(2,4,3,1)  <_INTEGER>   1,17,49,33,65,81,113,97,129,145,
                                   ... 257,273,305,289,321,337,369,353
   ISTART         <_INTEGER>      0
   NPIX           <_INTEGER>      0
   POINTER        <_INTEGER>      0
```

The first 5 entries are simply the flatfield information (see Appendix E for an example flatfield file). The shape of the arrays (in this case 16×9) corresponds to the number of channels on each A/D card and the number of A/D cards. These dimensions must match the values of the constants SCUBA__NUM_ADC and SCUBA__NUM_CHAN (Appendix C).

The full description of these entries follows:

BOL_CALB
: These are the actual flatfield values and contain the response of each pixel relative to a reference pixel (usually the centre pixel on each array, H7 and C14). This correction is applied to the data using the flatfield task (specifically the `sculib_flatfield_data` routine):

$$V_{\text{flatfielded}}(B) = V_{\text{original}}(B) \times \text{BOL\_CALB}(B) \quad (1)$$

where B is the selected bolometer.

BOL_DU3
: These are the X coordinates of every bolometer. For SCUBA, these are arcsec X (U3) offsets (i.e. d$U3$) from the left-hand Nasmyth centre.

BOL_DU4
: These are the Y coordinate of every bolometer. For SCUBA, these are arcsec Y (U4) offsets (i.e. d$U4$) from the left-hand Nasmyth centre.

BOL_QUAL
This is an array of quality flags. A 0 indicates that the bolometer is 'good', a 1 indicates that the bolometer is 'bad'. This array is used during flatfielding (`sculib_flatfield_data`)to set bit 1 in the NDF quality array.

BOL_TYPE
This array contains a textual description of the sub-instrument to which each bolometer belongs. This information is used by the extinction task to select the bolometers from the specified sub-instrument.

BOL_ADC
This array contains the A/D card numbers corresponding to each of the bolometers that were used for the observation. The order and size should match the size of the second dimension of the main data array for raw SCUBA data and the first dimension for data that has been processed by reduce_switch. The extinction task modifies this array to reflect the sub-instrument selection.

BOL_CHAN
This array contains the A/D channel numbers corresponding to each of the bolometers that were used for the observation. The order and size should match the size of the second dimension of the main data array for raw SCUBA data and the first dimension for data that has been processed by reduce_switch. The extinction task modifies this array to reflect the sub-instrument selection.

FLAT_ADC
Not used by SURF.

FLAT_CHN
Not used by SURF.

FLAT_IND
Not used by SURF.

PHOT_BB
This is only used for PHOTOM (and POLPHOT) observations. The first dimension corresponds to the number of beam positions (3 for 2 position chopping with nodding) and the second dimension corresponds to the number of sub-instruments used for the observation. The array contains the bolometer number (using the same indexing scheme as for the BOL_CHAN and BOL_ADC arrays) that is on source for each of the beam positions. For standard single bolometer photometry only the middle beam is on source and in the example shown only H7 and C14 are used for the photometry. The scuphot task uses this array to decide which bolometers to extract from the full dataset. The extinction task modifies this array to reflect the sub-instrument selection.

This information is also available in string form in the `PHOT_BBF` FITS keyword. For non-PHOTOM based observations this structure is 0.

DEM_PNTR

This array is used to convert from measurement, integration, exposure and switch number to sample number (i.e. the final dimension in the data array). The routines `SCULIB_FIND_SWITCH`

and SCULIB_FIND_INT are provided to simplify this (they also return the position of the end of the switch or integration). It has dimensions of N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, and N_MEASUREMENTS.

ISTART
Not used by SURF.

NPIX
Not used by SURF.

POINTER
Not used by SURF.

2.3.3 SCUCD

The SCUCD extension contains information on telescope movements. It is slightly different for JIGGLE (MAP and PHOTOM) and SCAN observations. For a JIGGLE observation it has the following format:

```
STRUCT.MORE.SCUCD  <SCUCD_ST>

   JIGL_X(64)      <_REAL>          -2.6883,-5.3457,-8.033999,-13.3797,
                                    ... 5.3457,2.6883,0,2.6883,0,2.6883,0,0
   JIGL_Y(64)      <_REAL>          1.545,3.09,1.545,1.545,3.09,4.635,6.18,
                                    ... -9.27,-7.725,-6.18,-4.635,-3.09,0
   DEC1            <_REAL>          0
   DEC2            <_REAL>          0
   LST_STRT(2,4,3,1)  <_DOUBLE>     5.6288711444511,5.630383297715,
                                    ... 5.6639924748021,5.6623806283313
   RA1             <_REAL>          0
   RA2             <_REAL>          0
```

and for a SCAN observation:

```
STRUCT.MORE.SCUCD  <SCUCD_ST>

   JIGL_X          <_REAL>          0
   JIGL_Y          <_REAL>          0
   DEC1(1,5,1,1)   <_REAL>          -17.06608,-17.04154,-17.10884,-17.07168,
                                    -17.11085
   DEC2(1,5,1,1)   <_REAL>          -17.03781,-17.09083,-17.04711,-17.11225,
                                    -17.09126
   LST_STRT(1,5,1,1)  <_DOUBLE>     5.521871430823,5.5231108938446,
                                    ... 5.5266851885263,5.5281428875176
   RA1(1,5,1,1)    <_REAL>          21.16225,21.16618,21.16226,21.1673,
                                    21.16577
   RA2(1,5,1,1)    <_REAL>          21.16466,21.16197,21.16755,21.1638,
                                    21.16747
   WPLATE          <_REAL>          0
```

The full description of these entries follows:

JIGL_X and JIGL_Y

These are the X and Y coordinates of the jiggle pattern in arcsecond offsets. The coordinate frame is given in the SAM_CRDS FITS keyword. They are only used for JIGGLE observations.

LST_STRT

This is the local sidereal time of the start of every switch expressed as decimal radians. It has dimensions of N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, and N_MEASUREMENTS. The current epoch of observation (Modified Julian Date) is calculated from the UTSTART and UTDATE FITS keywords in SCULIB_GET_MJD.

RA1, DEC1, RA2 and DEC2

This is the apparent RA and Dec of the start (RA1, DEC1) and end (RA2, DEC2) of the scans in radians. The dimensions are N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, and N_MEASUREMENTS. The current epoch of observation (Modified Julian Date) is calculated from the UTSTART and UTDATE FITS keywords in SCULIB_GET_MJD.

WPLATE

For polarimetry observations (POLPHOT or POLPMAP) this array contains the waveplate angle for each measurement (where a measurement is defined as a move of the waveplate). It is used by the SURFLIB_FILL_WPLATE routine.

2.3.4 REDS

The REDS[1] extension is created by SURF as a general purpose repository of information that should be passed between SURF tasks.

It can contain the following entries:

BEAM_WT
The relative weight of the beams. Only used for PHOTOM observations. This is created by reduce_switch.

SKY
An NDF containing the estimate of the sky fluctuations. This is usually created by calcsky and is used by remsky to remove the sky fluctuations. The size of the NDF should match the size of the time axis in the main data array.

BOLWT
Relative weight of each bolometer. Created by setbolwt and used by rebin.

[1]The name REDS is historical, at some time in the distant past the SCUBA data reduction software was called REDS, standing for 'REDuction Scuba'

Table 2: Quality bits used by SURF

Bit#	Value	Meaning
0	1	Infinity (eg division by zero)
1	2	Set by flatfield.
2	4	Set by reduce_switch if the transputers detected more spikes than specified by the SPIKE_LEVEL parameter.
3	8	Set by change_quality.
4	16	Set by despiking (scuclip and despike)

3 Aborted data files

When an observation completes normally (either because the specified number of measurements has been taken or a TERMINATE command has been issued) the size of the final dimension of the data array and the contents and shape of the DEM_PNTR array agree. In some cases an observation must be aborted suddenly using the ABORT command. Aborted observation files do not always have the correct shape for DEM_PNTR (e.g. the number of integrations specified by the second dimension of the array does not agree with the number that were actually observed). SURF overcomes this problem by looking in the STATE FITS keyword; if it contains the string "ABORTING" the exposure, integration and measurement values are read from the FITS keywords LAST_EXP, LAST_INT and LAST_MEAS respectively.

4 Quality flags

The SURF software conforms to the NDF standard concerning the processing of quality or bad-pixel masks. Each method of setting a pixel bad is associated with a bit in the quality masking flag (the NDF.QUALITY.BADBITS component). The bad bits and their meaning in SURF are described in table 2.

In order to remove the effect of a particular bit (i.e. to ignore a despiking), the KAPPA task setbb can be used to change the bad-bits mask in the NDF. Simply calculate the value related to the bits you are interested in keeping and use this value in setbb. Note that care must be taken in deciding which bits are to be used for masking bad data. Bits zero and one must always be set whereas the other three bits are optional. change_quality is the only task that acts on a file rather than producing a processed copy and it is probably better if change_quality is used directly if you wish to manipulate the mask associated with bit three.

Additionally, regridded images also use quality flags. Bit 0 is used to represent areas where no data were available and bit 1 is used to mask data at the edge of the regridded area via the TRIM parameter.

5 SURF Libraries

SURF consists of the top level task routines (i.e. routines that provide the structure of the task and do all disk I/O and most of the interaction with the parameter system) and lower level routines that do jobs that could be used by multiple tasks or other systems (such as the real-time observing system) or provide the data reduction algorithms. These lower level routines are stored in libraries and are discussed in the following sections.

5.1 SURFLIB

SURFLIB is a library of data reduction algorithms and routines that are directly relevant to SURF itself. All new subroutines required by SURF are added to this library unless there is an obvious reason to put it somewhere else. Contains routines for sky noise removal, instrumental polarisation removal, baseline removal from scan maps etc).

5.2 SCULIB

This library was designed to provide the fundamental building blocks for manipulating SCUBA data in the off-line and real-time system. Many of the routines in this library are not used in SURF itself. In general, new routines are added to SURFLIB rather than SCULIB although SCULIB does contain some routines that are used solely in SURF.[2]

6 Anatomy of a SURF task

SURF tasks follow the same general format internally:

(1) Request the messaging level from the parameter system (MSG_LEVEL).

(2) Read in some information concerning the object name, observation mode and run number and write this to the screen (at normal messaging level) to keep the user informed.

(3) Check the history component to make sure that the correct tasks have been run prior to running the current task. This always contains a check to see that the reduce_switch task has been run on the data.

(4) Check the dimensions of the data array, the LST_STRT array and the DEM_PNTR arrays.

(5) Retrieve information from NDF extensions (see e.g. SCULIB_GET_BOL_DESC, SCULIB_GET_JIGGLE). [as required]

(6) If required, construct an output name based on the input filename (SCULIB_CONSTRUCT_OUT). The output file is usually propogated from the input file using NDF_PROP.

(7) Do whatever has to be done.

(8) Shut down NDF, free memory, free locators.

[2] . . . for historical reasons.

7 Writing an import task

In order to import data into SURF that has not been taken with the SCUBA VAX/VMS data acquisition system, it is necessary to convert the original file format to a form that SURF can understand. The file can be converted to the same format as written by the SCUBA acquisition system (§2.1) or the format written by the reduce_switch task (§2.2).

It is expected that the conversion of the data structure to the correct format will be fairly simple. The difficulty lies in generating the required FITS information (with the correct FITS keywords) and the NDF extensions.

7.1 Case Study: DREAM

DREAM is a SCUBA observing mode where the chopping is done as part of the jiggle pattern and the resulting signal is reconstructued by tiling the bolometer patterns (le Poole and Greve, 1998, SPIE 3357). In this case it is not necessary to subtract switches, despike or flatfield the data and SURF is required simply to perform the regridding and coadding. In this case, the SURF format was chosen to simulate the output of the flatfield task (the HISTORY component of the NDF was updated to reflect this).

The required stages are:

(1) Read in the import data file.

(2) Create output NDF.

(3) Convert data array to a 2-d array of dimensions (N_BOLS $\times$ N_SAMPLES). Also create Variance and Quality arrays.

(4) Write AXIS information. This is simply bolometer number for the X axis and integration number for the Y axis.

(5) Create the SCUCD, SCUBA and REDS extensions.

(6) Construct the DEM_PNTR array.

(7) Construct the LST_STRT array.

(8) Store the jiggle patterns in JIGL_X and JIGL_Y.

(9) Construct BOL_CHAN and BOL_ADC arrays.

(10) Store the flatfield information (essentially for the X and Y offsets of each bolometer). This is done by reading in a flatfield file using SCULIB_READBOLS.

(11) Create FITS header (see table 1 for required keywords for jiggle map).

(12) Write fake history information to convince SURF that reduce_switch and flatfield have been run on the file.

(13) Shut down.

Glossary

chopping The secondary mirror is continuously moved on and off source at approximately 7 Hz in order to remove the sky to zeroth order. This is done in addition to standard jiggling.

demodulation Removal of the chop signal by the transputers. At this time, the raw data can not be accessed, only the demodulated data are stored.

exposure An exposure is the result from a complete set of switches. For example, in a JIGGLE/MAP or PHOTOM observation where the telescope is nodding the source between left and right beams, the data from each nod position is a switch and the reduced result 'left switch' - 'right switch' say, is an exposure. In a SCAN/MAP observation there is no beam switching so, in this case, an exposure is the same as a switch.

integration An integration means different things for different observations.

For one of the mapping modes it means the data from one fully-sampled coverage of the map area. In a JIGGLE/MAP, where full sampling is achieved by jiggling the secondary mirror, an integration is generally the results from one pass through the complete jiggle pattern. An integration is made up of one or more exposures.

Similarly, an integration for a SCAN/MAP observation is made up of data from the raster scans that cover the map area once.

For PHOTOM observations an integration is usually the average of a 9 point mini-jiggle.

For a SKYDIP observation, an integration is the data from a single revolution of the sector chopper in front of the cryostat window.

jiggle In order to sample an image fully the secondary mirror is moved once a second (whilst chopping) to move the position of the array on the sky; this is called 'jiggling.' There are a complete set of jiggle positions for each integration. A PHOTOM observation can also jiggle in order to correct for seeing effects.

measurement A measurement is a group of integrations. Most MAP or PHOTOM observations will consist of only one measurement.

A FOCUS or ALIGN observation consists of five measurements (one for each secondary mirror position). A SKYDIP observation consists of one measurement at each elevation.

nod In order to correct for atmospheric variation the telescope is moved off-source in each exposure so that sky can be measured.

ODF The observation definition file (ODF) is a file containing a list of instructions for an observation with SCUBA.

sub-instrument SCUBA contains bolometer arrays and photometric pixels that can operate at several wavelengths simultaneously. Each of these is called a sub-instrument. They are:

- SHORT - the short wave array containing 91 bolometers
- LONG - the long wave array containing 37 bolometers

- P1100 - the single bolometer optimised for 1100 micron.
- P1350 - the single bolometer optimised for 1350 micron.
- P2000 - the single bolometer optimised for 2000 micron.

switch The switch is the fundamental unit of data-taking in an observation. For example in a JIGGLE/MAP or PHOTOM observation each chunk of jiggle positions measured with the object in the beam of a telescope is a switch. Each scan across the source in a SCAN/MAP observation is also a switch.

tau (τ) Submillimetre extinction is measured using the zenith optical depth, tau or τ, this is a measure of the amount of water vapour present in the atmosphere. For a tau, τ at a given airmass, A, the attenuation due to the atmosphere is given as $e^{-A\tau}$. Note that tau is wavelength dependent and that the value quoted by the Caltech Submillimetre Observatory (CSO) is the τ at 225 GHz and will therefore be different at the other wavelengths used by SCUBA

A SURF coordinate frames

SURF understands a number of different coordinate frames and uses two letter codes[3] (as used by the JCMT) to describe them. The following are available:

RB	FK4 B1950
RJ	FK5 J2000
RD	Apparent RA/Dec, current epoch
GA	Galactic coordinates (J2000)
NA	Left hand Nasmyth coordinates
AZ	Az/El offsets
PLANET	JCMT input coordinate frame for moving source
PL	SURF output coordinate for offsets from tracking centre

These coordinate frames are used to specify the tracking centre (`CENT_CRD`), the coordinate frame of the chop (`CHOP_CRD`), the coordinate frame of any offsets (`LOCL_CRD`) and the coordinate frame of the sampling (`SAM_CRDS`) (eg jiggle coordinate frame or scan coordinate frame)

SURF currently assumes that the instrument is on the left hand Nasmyth platform.

[3] with the exception of PLANET frame

```
surf ---- src                          Top level surf_* and IFL files
      |
      |-- surflib                      surflib source code
      |
      |-- sculib                       sculib source code
      |
      |-- surf_kap                     Useful routines from KAPPA
      |
      |-- scripts                      Scripts (eg surf.csh, remdbm.pl)
      |
      |-- misc                         Miscellaneous (eg ipfile.dat)
      |
      |-- dream                        DREAM mode import routine
      |
      |-- starlink                     Starlink makefiles
      |
      |-- docs --------- hlp           On-line help system
                         |
                         |-- sun216    Main SURF documentation
                         |
                         |-- sc10      Photometry cookbook
                         |
                         |-- sc11      Mapping cookbook
                         |
                         |-- sc11_mini mini mapping cookbook
```

Figure 1: Directory layout of SURF development system. CVS directories are not shown.

B Building SURF

The SURF development tree is in CVS and has the layout shown in Fig. 1. The development version can be built by using the default make target in the `surf` directory or by running the individual makefiles in the `sculib`, `surflib`, `surf_kap` and `src` directories. The SURF monolith is built in the `src` directory along with compilation of the IFC files (the monolith IFL file is created automatically from the task IFL files).

The current development version number (i.e. the version number written to history components of the NDF files) is defined in `src/makefile` via the `PKG_VERS` variable. (see `surf_set_app_name.F` for more information on the setting of the version number in the monolith.

In order to generate a Starlink distribution the `export` target must be used from the root `makefile`. This generates tar files from each directory and copies them to the `starlink` directory. If required, the version number must be updated manually in the Starlink `makefile`. The platform dependencies are stored in the `mk` script that is used to run `make` with the correct environment. The environment is selected with the `SYSTEM` environment variable and can be one of `sun4_Solaris`, `ix86_Linux` or `alpha_OSF1`.

Once the `SYSTEM` environment variable has been set the `mk` script can be run to build and export SURF:

./mk export_source
Exports the source into a compressed tar file ready for distribution.

./mk build
Builds the monolith, extracting source files from tar files as required.

./mk export
Exports the built system into a compressed tar file ready for distribution. This will include the source code.

./mk export_run
Exports the run-only system into a compressed tar file ready for distribution. This will not include the source code.

./mk install
Installs the built system into the directory structure specified by the `INSTALL` environment variable. Files are installed below this directory into `bin/surf`, `docs`, `help/surf` and `dates`. Soft links are made from the installed binary directory to the current directory so that the installed files can be seen. Usually, `INSTALL` is set to `/star`.

./mk deinstall
Deinstall the package.

C SURF Constants

The SURF_PAR include file defines the global constants that are used in SURF for preallocating arrays and defining useful mathematical constants and strings. Only top level SURF routines (ie names of `surf_*` include this file; all other library routines obtain the value from argument.

C.1 Constant strings

PACKAGE = 'SURF'
Name of the software package. Used in output messages. (CHAR*10)

C.2 Convenient numeric definitions

PI = 3.14159265359D0
π [DOUBLE]

R2AS = 206264.806247D0
Used to convert radians to arcseconds:

$$\text{arcsec} = 3600\frac{180}{\pi}\theta \tag{2}$$

[DOUBLE]

C.3 Parameters controlling the observation limits

SCUBA__MAX_BEAM = 3
maximum number of bolometers in a single sub-instrument that actually observer the source in a PHOTOM observation. [INTEGER]

SCUBA__MAX_MEAS = 200
maximum number of measurements in an observation. [INTEGER]

SCUBA__MAX_INT = 1000
maximum number of integrations in a measurement. [INTEGER]

SCUBA__MAX_EXP = 128
maximum number of exposures per integration. [INTEGER]

SCUBA__MAX_SWITCH = 3
maximum number of switches per exposure. [INTEGER]

SCUBA__MAX_JIGGLE = 512
maximum number of offsets in a jiggle pattern. [INTEGER]

C.4 Parameters governing the instrument itself

SCUBA__MAX_SUB = 5
Maximum number of sub-instruments (i.e. for SCUBA there are LONG, SHORT, P2000, P1350 and P1100 sub-instruments). [INTEGER]

SCUBA__NUM_ADC = 9
Number of A/D cards (called A, B, C, D, E, F, G, H and I for SCUBA).

SCUBA__NUM_CHAN = 16
Number of channels on each A/D card.

C.5 Parameters dealing with automatic output filename generation

SUFFIX_ENV = 'SCUBA_SUFFIX'
Name of the environment variable dealing with the format of the automatically generated output filename. [CHAR*15]

SCUBA__N_SUFFIX = 3
Number of suffix options [INTEGER]

SUFFIX_OPTIONS = 'LONG','SHORT','VERBOSE'
The allowed suffix options i.e. these are the values that are valid for the environment variable specified by SUFFIX_ENV [CHAR*10 (SCUBA_N_SUFFIX)]

C.6 General limits

SCUBA__MAX_POINT = 20
Maximum number of pointing corrections.

SCUBA__MAX_FITS = 200
Maximum number of FITS items in header.

SCUBA__MAX_SECT = 10
Maximum number of SCUBA sections that can be specified.

SCUBA__N_TEMPS = 3
: Number of temperatures per reading for SKYDIP observations: AMBIENT, SKY, COLD.

D SCUBA FITS headers

This section lists the FITS headers written by SCUBA. SURF uses a subset of these keywords. Multiple headers are listed since the values are slightly different for different observing modes.

D.1 Skydip

This is a typical SKYDIP FITS header:

```
ACCEPT  = 'NO      '           / accept update; PROMPT, YES or NO
ALIGN_AX= 'not used'           / Alignment measurements in X or Y axis
ALIGN_SH=                   -1 / Distance between successive alignment offsets (
ALT-OBS =                 4092 / Height of observatory above sea level (metres)
AMEND   =             3.863703 / Airmass at end of observation
AMSTART =             1.015427 / Airmass at start of observation
APEND   =              627.851 / Air pressure at end of observation (mbar)
APSTART =              627.851 / Air pressure at start of observation (mbar)
ATEND   =             5.604397 / Air temp. at end of observation (C)
ATSTART =             5.677658 / Air temp. at start of observation (C)
BOLOMS  = 'SHORT_DC,LONG_DC'   / Names of bolometers measured
CALIBRTR=                    F / Internal calibrator is on or off
CAL_FRQ =                   -1 / Calibrator frequency (Hz)
CENT_CRD= 'AZ      '           / Centre coordinate system
CHOP_CRD= 'not used'           / Chopper coordinate system
CHOP_FRQ=                    2 / Chopper frequency (Hz)
CHOP_FUN= 'not used'           / Chopper waveform
CHOP_PA =                   -1 / Chopper P.A., 0 = in lat, 90 = in long
CHOP_THR=                   -1 / Chopper throw (arcsec)
DATA_DIR= '19990808'           / Sub-directory where datafile was stored
DATA_KPT= 'DEMOD   '           / The type of data stored to disk
END_AZD =              219.183 / Azimuth at end of observation (deg)
END_EL  =                   80 / Elevation of last SKYDIP point (deg)
END_ELD =              79.9896 / Elevation at end of observation
EQUINOX =                 2000 / Equinox of mean coordinate system
EXPOSED =                    5 / Exposure per pixel (seconds)
EXP_NO  =                    1 / Exposure number at end of observation
EXP_TIME=                  0.5 / Exposure time for each basic measurement (sec)
E_PER_I =                    1 / Number of exposures per integration
FILTER  = '450N:850N'          / Filters used
FOCUS_SH=                   -1 / Shift between focus measurements (mm)
GAIN    =                    1 / Programmable gain
HSTEND  = '20:05:49.00131'     / HST at end of observation
HSTSTART= '20:00:56.00281'     / HST at start of observation
HUMEND  =                   15 / Humidity (%) at end of observation
HUMSTART=                   15 / Humidity (%) at start of observation
INSTRUME= 'SCUBA   '           / Name of instrument used
```

```
INT_NO  =                    1 / Integration number at end of observation
JIGL_CNT=                   -1 / Number of offsets in jiggle pattern
JIGL_NAM= 'not used'            / File containing jiggle offsets
J_PER_S =                   -1 / Number of jiggles per switch position
J_REPEAT=                   -1 / No. jiggle pattern repeats in a switch
LAT     = 'not used'            / Object latitude
LAT-OBS =        19.8258323669 / Latitude of observatory (degrees)
LAT2    = 'not used'            / Object latitude at MJD2
LOCL_CRD= 'no      '            / Local offset coordinate system
LONG    = '+219:11:26.64'       / Object longitude
LONG-OBS=        204.520278931 / East longitude of observatory (degrees)
LONG2   = 'not used'            / Object Longitude at MJD2
MAP_HGHT=                   -1 / Height of rectangle to be mapped (arcsec)
MAP_PA  =                   -1 / P.A. of map vertical, +ve towards +ve long
MAP_WDTH=                   -1 / Width of rectangle to be mapped (arcsec)
MAP_X   =                   -1 / Map X offset from telescope centre (arcsec)
MAP_Y   =                   -1 / Map Y offset from telescope centre (arcsec)
MAX_EL  =                   80 / Max elevation of sky-dip (deg)
MEANDEC =            -17.93834 / -17:56:18.01437 = approx. mean Dec. (deg)
MEANRA  =             222.1444 / 222:08:39.76318 = approx. mean R.A. (deg)
MEAS_NO =                   10 / Measurement number at end of observation
MIN_EL  =                   15 / Min elevation of sky-dip (deg)
MJD1    =                   -1 / Modified Julian day planet at RA,DEC
MJD2    =                   -1 / Modified Julian day planet at RA2,DEC2
MODE    = 'SKYDIP  '            / The type of observation
N_INT   =                   10 / No. integrations in the observation
N_MEASUR=                   10 / No. measurements in the observation
OBJECT  = 'not used'            / Name of object
OBJ_TYPE= 'not used'            / Type of object
OBSDEF  = 'ss:odfsxskydip_850.obs_193300' / The observation definition file
OBSERVER= 'jmk     '            / The name of the observer
PROJ_ID = 'SCUBA   '            / The project identification
RUN     =                   26 / Run number of observation
SAM_CRDS= 'no      '            / Coordinatesystem of sampling mesh
SAM_DX  =                   -1 / Sample spacing along scan direction (arcsec)
SAM_DY  =                   -1 / Sample spacing perp. to scan (arcsec)
SAM_MODE= 'not used'            / Sampling method
SAM_PA  =                   -1 / Scan P.A. rel. to lat. line; 0=lat, 90=long
SCAN_REV=                    F / .TRUE. if alternate scans reverse direction
SPK_NSIG=                   -1 / N sigmas from fit of spike threshold
SPK_RMVL=                    F / Automatic spike removal
SPK_WDTH=                   -1 / Assumed width of spike
START_EL=                   15 / Elevation of first SKYDIP point (deg)
STATE   = 'Terminating          :   RESET_DATA_SIZE' / SCUCD state
STEND   = '16:49:09.149323'     / ST at end of observation
STRT_AZD=              219.174 / Azimuth at observation start (deg)
STRT_ELD=              14.9868 / Elevation at observation start (deg)
STSTART = '16:44:15.34744'      / ST at start of observation
SWTCH_MD= 'not     '            / Switch mode of observation
SWTCH_NO=                    1 / Switch number at end of observation
S_PER_E =                    1 / Number of switch positions per exposure
TELESCOP= 'JCMT    '            / Name of telescope
TEL_OPER= 'kkp     '            / Telescope operator
UTDATE  = '1999:8:8'            / UT date of observation
```

```
UTEND   = '6:05:48.9996'       / UT at end of observation
UTSTART = '6:00:55.99937'      / UT at start of observation
VERSION =                  1.1 / SCUCD version
ALIGN_DX=             0.027268 / SMU tables X axis alignment offset
ALIGN_DY=             0.482489 / SMU tables Y axis alignment offset
ALIGN_X =             -0.96279 / SMU tables X axis
ALIGN_Y =              3.79108 / SMU tables Y axis
AZ_ERR  =                    0 / Error in the telescope azimuth
CHOPPING=                    F / SMU Chopper chopping state
EL_ERR  =             0.154495 / Error in the telescope elevation
FOCUS_DZ=            -0.361142 / SMU tables Z axis focus offset
FOCUS_Z =             -17.2453 / SMU tables Z axis
SEEING  =                0.525 / SAO atmospheric seeing
SEE_DATE= '9908061715'         / Date and time of SAO seeing
TAU_225 =                0.066 / CSO tau
TAU_DATE= '9908080555'         / Date and time of CSO tau
TAU_RMS =              5.0E-03 / CSO tau rms
UAZ     =            -0.528088 / User azimuth pointing offset
UEL     =              4.83289 / User elevation pointing offset
UT_DATE = ' 8 Aug 1999'        / UT date at start of observation
BAD_LIM =                   32 / No. spikes before quality set bad
CALIB_LG=                   -1 / Lag of internal calibrator in samples
CALIB_PD=                   -1 / Period of internal calibrator in samples
CHOP_LG =                    0 / Chop lag in samples
CHOP_PD =             64.03689 / Chop period in samples
CNTR_DU3=                    0 / Nasmyth dU3 coord of instrument centre
CNTR_DU4=                    0 / Nasmyth dU4 coord of instrument centre
ETATEL_1=                 0.75 / Transmission of telescope
ETATEL_2=                  0.9 / Transmission of telescope
ETATEL_3=                   -1 / Transmission of telescope
ETATEL_4=                   -1 / Transmission of telescope
ETATEL_5=                   -1 / Transmission of telescope
FILT_1  = '450     '           / Filter name
FILT_2  = '850     '           / Filter name
FILT_3  = 'not used'           / Filter name
FILT_4  = 'not used'           / Filter name
FILT_5  = 'not used'           / Filter name
FLAT    = 'jcmtdata_dir:lwswphot.dat' / Name of flat-field file
JIG_DSCD=                   -1 / No. samples discarded after jiggle movement
L_GD_BOL= 'not used'           / Bol. to whose value LW guard ring is set
L_GUARD =                    F / Long wave guard ring on or off
MEAS_BOL= 'not used'           / Bolometers  actually measured in observation
N_BOLS  =                    2 / Number of bolometers selected
N_SUBS  =                    2 / Number of sub-instruments used
PHOT_BBF= 'not_used       LL,C14,NULL' / The bolometers on the source
PRE_DSCD=                    0 / No. of samples discarded before chop movement
PST_DSCD=                    0 / No. samples discarded after chop movement
REBIN   = 'LINEAR  '           / Rebinning method used by SCUIP
REF_ADC =                   -1 / A/D card of FLATFIELD reference bolometer
REF_CHAN=                   -1 / Channel of FLATFIELD reference bolometer
SAM_TIME=                  122 / A/D sample period in ticks (64musec)
SIMULATE=                    F / True if data is simulated
SKY     = 'jcmtdata_dir:skydip_startup.dat' / Name of sky opacities file
SUB_1   = 'SHORT   '           / SCUBA instrument being used
```

```
SUB_2   = 'LONG    '           / SCUBA instrument being used
SUB_3   = 'not used'           / SCUBA instrument being used
SUB_4   = 'not used'           / SCUBA instrument being used
SUB_5   = 'not used'           / SCUBA instrument being used
S_GD_BOL= 'not used'           / Bol. to whose value SW guard ring is set
S_GUARD =                    F / Short wave guard ring on or off
TAUZ_1  =                   -1 / Zenith sky optical depth
TAUZ_2  =                   -1 / Zenith sky optical depth
TAUZ_3  =                   -1 / Zenith sky optical depth
TAUZ_4  =                   -1 / Zenith sky optical depth
TAUZ_5  =                   -1 / Zenith sky optical depth
T_AMB   =             278.6532 / The ambient air temperature (K)
T_COLD_1=                  120 / Effective temperature of cold load (K)
T_COLD_2=                  100 / Effective temperature of cold load (K)
T_COLD_3=                   -1 / Effective temperature of cold load (K)
T_COLD_4=                   -1 / Effective temperature of cold load (K)
T_COLD_5=                   -1 / Effective temperature of cold load (K)
T_HOT   =                 282.6 / The temperature of the hot load (K)
T_TEL   =             282.2908 / The temperature of the telescope
USE_CAL =                    F / .TRUE. if dividing chop by cal before rebin
WAVE_1  =                  442 / Wavelength of map (microns)
WAVE_2  =                  862 / Wavelength of map (microns)
WAVE_3  =                    0 / Wavelength of map (microns)
WAVE_4  =                    0 / Wavelength of map (microns)
WAVE_5  =                    0 / Wavelength of map (microns)
END
```

D.2 Jiggle map

Jiggle map of a planet (note that for a moving source SCUBA supplies that apparent RA and Dec of the source at two different times. These are specified as `LONG` and `LAT` for `MJD1` and `LONG2` and `LAT2` for `MJD2`):

```
ACCEPT  = 'not used'           / accept update; PROMPT, YES or NO
ALIGN_AX= 'not used'           / Alignment measurements in X or Y axis
ALIGN_SH=                   -1 / Distance between successive alignment offsets (
ALT-OBS =                 4092 / Height of observatory above sea level (metres)
AMEND   =             1.296355 / Airmass at end of observation
AMSTART =             1.283144 / Airmass at start of observation
APEND   =             627.4847 / Air pressure at end of observation (mbar)
APSTART =             627.2405 / Air pressure at start of observation (mbar)
ATEND   =             7.655678 / Air temp. at end of observation (C)
ATSTART =             7.997562 / Air temp. at start of observation (C)
BOLOMS  = 'LONG,SHORT'         / Names of bolometers measured
CALIBRTR=                    F / Internal calibrator is on or off
CAL_FRQ =             2.929688 / Calibrator frequency (Hz)
CENT_CRD= 'PLANET  '           / Centre coordinate system
CHOP_CRD= 'LO      '           / Chopper coordinate system
CHOP_FRQ=               7.8125 / Chopper frequency (Hz)
CHOP_FUN= 'SQUARE  '           / Chopper waveform
CHOP_PA =                    0 / Chopper P.A., 0 = in lat, 90 = in long
CHOP_THR=                   68 / Chopper throw (arcsec)
```

```
DATA_DIR= '19990808'           / Sub-directory where datafile was stored
DATA_KPT= 'DEMOD   '           / The type of data stored to disk
END_AZD =              198.486 / Azimuth at end of observation (deg)
END_EL  =                   -1 / Elevation of last SKYDIP point (deg)
END_ELD =              50.3461 / Elevation at end of observation
EQUINOX =                 2000 / Equinox of mean coordinate system
EXPOSED =                0.128 / Exposure per pixel (seconds)
EXP_NO  =                   13 / Exposure number at end of observation
EXP_TIME=                0.128 / Exposure time for each basic measurement (sec)
E_PER_I =                   -1 / Number of exposures per integration
FILTER  = '450N:850N'          / Filters used
FOCUS_SH=                   -1 / Shift between focus measurements (mm)
GAIN    =                   10 / Programmable gain
HSTEND  = '18:52:56.00327'     / HST at end of observation
HSTSTART= '18:41:44.99954'     / HST at start of observation
HUMEND  =                   14 / Humidity (%) at end of observation
HUMSTART=                   14 / Humidity (%) at start of observation
INSTRUME= 'SCUBA   '           / Name of instrument used
INT_NO  =                    1 / Integration number at end of observation
JIGL_CNT=                   -1 / Number of offsets in jiggle pattern
JIGL_NAM= 'not used'           / File containing jiggle offsets
J_PER_S =                   -1 / Number of jiggles per switch position
J_REPEAT=                   -1 / No. jiggle pattern repeats in a switch
LAT     = '-017:55:40.89'      / Object latitude
LAT-OBS =        19.8258323669 / Latitude of observatory (degrees)
LAT2    = '-017:55:47.08'      / Object latitude at MJD2
LOCL_CRD= 'RD      '           / Local offset coordinate system
LONG    = '+014:48:26.75'      / Object longitude
LONG-OBS=        204.520278931 / East longitude of observatory (degrees)
LONG2   = '+014:48:28.04'      / Object Longitude at MJD2
MAP_HGHT=                  600 / Height of rectangle to be mapped (arcsec)
MAP_PA  =                    0 / P.A. of map vertical, +ve towards +ve long
MAP_WDTH=                  600 / Width of rectangle to be mapped (arcsec)
MAP_X   =                    0 / Map X offset from telescope centre (arcsec)
MAP_Y   =                    0 / Map Y offset from telescope centre (arcsec)
MAX_EL  =                   -1 / Max elevation of sky-dip (deg)
MEANDEC =            -17.92968 / -17:55:46.84753 = approx. mean Dec. (deg)
MEANRA  =             222.1171 / 222:07:01.490479 = approx. mean R.A. (deg)
MEAS_NO =                    1 / Measurement number at end of observation
MIN_EL  =                   -1 / Min elevation of sky-dip (deg)
MJD1    =    51398.19570601852 / Modified Julian day planet at RA,DEC
MJD2    =    51398.20612268519 / Modified Julian day planet at RA2,DEC2
MODE    = 'MAP     '           / The type of observation
N_INT   =                    1 / No. integrations in the observation
N_MEASUR=                    1 / No. measurements in the observation
OBJECT  = 'mars    '           / Name of object
OBJ_TYPE= 'UNKNOWN '           / Type of object
OBSDEF  = 'ss:scan_68dec.obs_mars_184023' / The observation definition file
OBSERVER= 'jmk     '           / The name of the observer
PROJ_ID = 'm99au34 '           / The project identification
RUN     =                   18 / Run number of observation
SAM_CRDS= 'NA      '           / Coordinatesystem of sampling mesh
SAM_DX  =                    3 / Sample spacing along scan direction (arcsec)
SAM_DY  =                   60 / Sample spacing perp. to scan (arcsec)
```

```
SAM_MODE= 'RASTER  '           / Sampling method
SAM_PA  =                15.5 / Scan P.A. rel. to lat. line; 0=lat, 90=long
SCAN_REV=                   T / .TRUE. if alternate scans reverse direction
SPK_NSIG=                  -1 / N sigmas from fit of spike threshold
SPK_RMVL=                   F / Automatic spike removal
SPK_WDTH=                  -1 / Assumed width of spike
START_EL=                  -1 / Elevation of first SKYDIP point (deg)
STATE   = 'Terminating         :   STEP' / SCUCD state
STEND   = '15:36:04.176178'    / ST at end of observation
STRT_AZD=              194.923 / Azimuth at observation start (deg)
STRT_ELD=              51.0202 / Elevation at observation start (deg)
STSTART = '15:24:51.33911'     / ST at start of observation
SWTCH_MD= 'NOSW    '           / Switch mode of observation
SWTCH_NO=                   1 / Switch number at end of observation
S_PER_E =                   1 / Number of switch positions per exposure
TELESCOP= 'JCMT    '           / Name of telescope
TEL_OPER= 'kkp     '           / Telescope operator
UTDATE  = '1999:8:8'           / UT date of observation
UTEND   = '4:52:55.99983'      / UT at end of observation
UTSTART = '4:41:44.99954'      / UT at start of observation
VERSION =                 1.1 / SCUCD version
ALIGN_DX=            0.027268 / SMU tables X axis alignment offset
ALIGN_DY=            0.482489 / SMU tables Y axis alignment offset
ALIGN_X =            -2.32166 / SMU tables X axis
ALIGN_Y =             3.78719 / SMU tables Y axis
AZ_ERR  =                   0 / Error in the telescope azimuth
CHOPPING=                   T / SMU Chopper chopping state
EL_ERR  =                   0 / Error in the telescope elevation
FOCUS_DZ=           -0.361142 / SMU tables Z axis focus offset
FOCUS_Z =            -16.5076 / SMU tables Z axis
SEEING  =               0.525 / SAO atmospheric seeing
SEE_DATE= '9908061715'         / Date and time of SAO seeing
TAU_225 =               0.097 / CSO tau
TAU_DATE= '9908080435'         / Date and time of CSO tau
TAU_RMS =            8.0E-03 / CSO tau rms
UAZ     =           -0.528088 / User azimuth pointing offset
UEL     =            4.83289 / User elevation pointing offset
UT_DATE = ' 8 Aug 1999'        / UT date at start of observation
BAD_LIM =                  32 / No. spikes before quality set bad
CALIB_LG=                   6 / Lag of internal calibrator in samples
CALIB_PD=            42.66667 / Period of internal calibrator in samples
CHOP_LG =                   4 / Chop lag in samples
CHOP_PD =                  16 / Chop period in samples
CNTR_DU3=                   0 / Nasmyth dU3 coord of instrument centre
CNTR_DU4=                   0 / Nasmyth dU4 coord of instrument centre
ETATEL_1=                  -1 / Transmission of telescope
ETATEL_2=                  -1 / Transmission of telescope
ETATEL_3=                  -1 / Transmission of telescope
ETATEL_4=                  -1 / Transmission of telescope
ETATEL_5=                  -1 / Transmission of telescope
FILT_1  = '450     '           / Filter name
FILT_2  = '850     '           / Filter name
FILT_3  = 'not used'           / Filter name
FILT_4  = 'not used'           / Filter name
```

```
FILT_5  = 'not used'           / Filter name
FLAT    = 'jcmtdata_dir:lwswphot.dat' / Name of flat-field file
JIG_DSCD=                    -1 / No. samples discarded after jiggle movement
L_GD_BOL= 'not used'           / Bol. to whose value LW guard ring is set
L_GUARD =                     F / Long wave guard ring on or off
MEAS_BOL= 'not used'           / Bolometers  actually measured in observation
N_BOLS  =                   128 / Number of bolometers selected
N_SUBS  =                     2 / Number of sub-instruments used
PHOT_BBF= 'not_used       LL,C14,NULL' / The bolometers on the source
PRE_DSCD=                     0 / No. of samples discarded before chop movement
PST_DSCD=                     0 / No. samples discarded after chop movement
REBIN   = 'LINEAR  '           / Rebinning method used by SCUIP
REF_ADC =                    -1 / A/D card of FLATFIELD reference bolometer
REF_CHAN=                    -1 / Channel of FLATFIELD reference bolometer
SAM_TIME=                   125 / A/D sample period in ticks (64musec)
SIMULATE=                     F / True if data is simulated
SKY     = 'jcmtdata_dir:skydip_startup.dat' / Name of sky opacities file
SUB_1   = 'SHORT   '           / SCUBA instrument being used
SUB_2   = 'LONG    '           / SCUBA instrument being used
SUB_3   = 'not used'           / SCUBA instrument being used
SUB_4   = 'not used'           / SCUBA instrument being used
SUB_5   = 'not used'           / SCUBA instrument being used
S_GD_BOL= 'not used'           / Bol. to whose value SW guard ring is set
S_GUARD =                     F / Short wave guard ring on or off
TAUZ_1  =                     0 / Zenith sky optical depth
TAUZ_2  =                     0 / Zenith sky optical depth
TAUZ_3  =                     0 / Zenith sky optical depth
TAUZ_4  =                     0 / Zenith sky optical depth
TAUZ_5  =                     0 / Zenith sky optical depth
T_AMB   =                    -1 / The ambient air temperature (K)
T_COLD_1=                    -1 / Effective temperature of cold load (K)
T_COLD_2=                    -1 / Effective temperature of cold load (K)
T_COLD_3=                    -1 / Effective temperature of cold load (K)
T_COLD_4=                    -1 / Effective temperature of cold load (K)
T_COLD_5=                    -1 / Effective temperature of cold load (K)
T_HOT   =                    -1 / The temperature of the hot load (K)
T_TEL   =                    -1 / The temperature of the telescope
USE_CAL =                     F / .TRUE. if dividing chop by cal before rebin
WAVE_1  =                   442 / Wavelength of map (microns)
WAVE_2  =                   862 / Wavelength of map (microns)
WAVE_3  =                     0 / Wavelength of map (microns)
WAVE_4  =                     0 / Wavelength of map (microns)
WAVE_5  =                     0 / Wavelength of map (microns)
END
```

D.3 Photometry

This is a photometry header:

```
ACCEPT  = 'not used'           / accept update; PROMPT, YES or NO
ALIGN_AX= 'not used'           / Alignment measurements in X or Y axis
ALIGN_SH=                    -1 / Distance between successive alignment offsets (
ALT-OBS =                  4092 / Height of observatory above sea level (metres)
```

```
AMEND   =            1.279808 / Airmass at end of observation
AMSTART =            1.263818 / Airmass at start of observation
APEND   =            628.7057 / Air pressure at end of observation (mbar)
APSTART =            628.7057 / Air pressure at start of observation (mbar)
ATEND   =            6.678879 / Air temp. at end of observation (C)
ATSTART =            6.776558 / Air temp. at start of observation (C)
BOLOMS  = 'H7      '          / Names of bolometers measured
CALIBRTR=                   T / Internal calibrator is on or off
CAL_FRQ =            2.929688 / Calibrator frequency (Hz)
CENT_CRD= 'RB      '          / Centre coordinate system
CHOP_CRD= 'AZ      '          / Chopper coordinate system
CHOP_FRQ=               7.8125 / Chopper frequency (Hz)
CHOP_FUN= 'SCUBAWAVE'         / Chopper waveform
CHOP_PA =                  90 / Chopper P.A., 0 = in lat, 90 = in long
CHOP_THR=                  45 / Chopper throw (arcsec)
DATA_DIR= '19990808'          / Sub-directory where datafile was stored
DATA_KPT= 'DEMOD   '          / The type of data stored to disk
END_AZD =             274.745 / Azimuth at end of observation (deg)
END_EL  =                  -1 / Elevation of last SKYDIP point (deg)
END_ELD =             51.0018 / Elevation at end of observation
EQUINOX =                2000 / Equinox of mean coordinate system
EXPOSED =                   0 / Exposure per pixel (seconds)
EXP_NO  =                   1 / Exposure number at end of observation
EXP_TIME=               1.024 / Exposure time for each basic measurement (sec)
E_PER_I =                   1 / Number of exposures per integration
FILTER  = '450N:850N'         / Filters used
FOCUS_SH=                  -1 / Shift between focus measurements (mm)
GAIN    =                  10 / Programmable gain
HSTEND  = '10:29:31.9986'     / HST at end of observation
HSTSTART= '10:25:34.0004'     / HST at start of observation
HUMEND  =                  36 / Humidity (%) at end of observation
HUMSTART=                  37 / Humidity (%) at start of observation
INSTRUME= 'SCUBA   '          / Name of instrument used
INT_NO  =                   8 / Integration number at end of observation
JIGL_CNT=                   9 / Number of offsets in jiggle pattern
JIGL_NAM= 'JCMTDATA_DIR:SQUEASY_9_2P0.JIG' / File containing jiggle offsets
J_PER_S =                   9 / Number of jiggles per switch position
J_REPEAT=                   1 / No. jiggle pattern repeats in a switch
LAT     = '+018:07:36.20'     / Object latitude
LAT-OBS =       19.8258323669 / Latitude of observatory (degrees)
LAT2    = 'not used'          / Object latitude at MJD2
LOCL_CRD= 'RB      '          / Local offset coordinate system
LONG    = '+004:28:44.42'     / Object longitude
LONG-OBS=      204.520278931 / East longitude of observatory (degrees)
LONG2   = 'not used'          / Object Longitude at MJD2
MAP_HGHT=                 180 / Height of rectangle to be mapped (arcsec)
MAP_PA  =                   0 / P.A. of map vertical, +ve towards +ve long
MAP_WDTH=                 180 / Width of rectangle to be mapped (arcsec)
MAP_X   =                   0 / Map X offset from telescope centre (arcsec)
MAP_Y   =                   0 / Map Y offset from telescope centre (arcsec)
MAX_EL  =                  -1 / Max elevation of sky-dip (deg)
MEANDEC =            18.23093 / 18:13:51.36292 = approx. mean Dec. (deg)
MEANRA  =            67.90475 / 67:54:17.08923 = approx. mean R.A. (deg)
MEAS_NO =                   1 / Measurement number at end of observation
```

```
MIN_EL  =                   -1 / Min elevation of sky-dip (deg)
MJD1    =                   -1 / Modified Julian day planet at RA,DEC
MJD2    =                   -1 / Modified Julian day planet at RA2,DEC2
MODE    = 'PHOTOM  '           / The type of observation
N_INT   =                    8 / No. integrations in the observation
N_MEASUR=                    1 / No. measurements in the observation
OBJECT  = 'hltau   '           / Name of object
OBJ_TYPE= 'UNKNOWN '           / Type of object
OBSDEF  = 'ss:odfsxphotom.t_hltau_100525' / The observation definition file
OBSERVER= 'jmk     '           / The name of the observer
PROJ_ID = 'scuba   '           / The project identification
RUN     =                  107 / Run number of observation
SAM_CRDS= 'AZ      '           / Coordinatesystem of sampling mesh
SAM_DX  =                   -1 / Sample spacing along scan direction (arcsec)
SAM_DY  =                   -1 / Sample spacing perp. to scan (arcsec)
SAM_MODE= 'JIGGLE  '           / Sampling method
SAM_PA  =                   -1 / Scan P.A. rel. to lat. line; 0=lat, 90=long
SCAN_REV=                    F / .TRUE. if alternate scans reverse direction
SPK_NSIG=                    0 / N sigmas from fit of spike threshold
SPK_RMVL=                    T / Automatic spike removal
SPK_WDTH=                    0 / Assumed width of spike
START_EL=                   -1 / Elevation of first SKYDIP point (deg)
STATE   = 'Terminating         :' / SCUCD state
STEND   = '7:15:14.03503'      / ST at end of observation
STRT_AZD=              274.523 / Azimuth at observation start (deg)
STRT_ELD=              51.8553 / Elevation at observation start (deg)
STSTART = '7:11:15.3828'       / ST at start of observation
SWTCH_MD= 'BMSW    '           / Switch mode of observation
SWTCH_NO=                    1 / Switch number at end of observation
S_PER_E =                    2 / Number of switch positions per exposure
TELESCOP= 'JCMT    '           / Name of telescope
TEL_OPER= 'kkp     '           / Telescope operator
UTDATE  = '1999:8:8'           / UT date of observation
UTEND   = '20:29:31.9986'      / UT at end of observation
UTSTART = '20:25:34.0004'      / UT at start of observation
VERSION =                  1.1 / SCUCD version
ALIGN_DX=             0.027268 / SMU tables X axis alignment offset
ALIGN_DY=             0.482489 / SMU tables Y axis alignment offset
ALIGN_X =             -2.98649 / SMU tables X axis
ALIGN_Y =              3.79108 / SMU tables Y axis
AZ_ERR  =                    0 / Error in the telescope azimuth
CHOPPING=                    T / SMU Chopper chopping state
EL_ERR  =                    0 / Error in the telescope elevation
FOCUS_DZ=            -0.504222 / SMU tables Z axis focus offset
FOCUS_Z =             -16.7523 / SMU tables Z axis
SEEING  =                0.525 / SAO atmospheric seeing
SEE_DATE= '9908061715'         / Date and time of SAO seeing
TAU_225 =                0.101 / CSO tau
TAU_DATE= '9908082019'         / Date and time of CSO tau
TAU_RMS =              3.0E-03 / CSO tau rms
UAZ     =             -5.09089 / User azimuth pointing offset
UEL     =              3.83056 / User elevation pointing offset
UT_DATE = ' 8 Aug 1999'        / UT date at start of observation
BAD_LIM =                   32 / No. spikes before quality set bad
```

```
CALIB_LG=                    6 / Lag of internal calibrator in samples
CALIB_PD=             42.66667 / Period of internal calibrator in samples
CHOP_LG =                    4 / Chop lag in samples
CHOP_PD =                   16 / Chop period in samples
CNTR_DU3=                    0 / Nasmyth dU3 coord of instrument centre
CNTR_DU4=                    0 / Nasmyth dU4 coord of instrument centre
ETATEL_1=                   -1 / Transmission of telescope
ETATEL_2=                   -1 / Transmission of telescope
ETATEL_3=                   -1 / Transmission of telescope
ETATEL_4=                   -1 / Transmission of telescope
ETATEL_5=                   -1 / Transmission of telescope
FILT_1  = '850     '           / Filter name
FILT_2  = '450     '           / Filter name
FILT_3  = 'not used'           / Filter name
FILT_4  = 'not used'           / Filter name
FILT_5  = 'not used'           / Filter name
FLAT    = 'jcmtdata_dir:lwswphot.dat' / Name of flat-field file
JIG_DSCD=                   -1 / No. samples discarded after jiggle movement
L_GD_BOL= 'not used'           / Bol. to whose value LW guard ring is set
L_GUARD =                    F / Long wave guard ring on or off
MEAS_BOL= 'LONG,SHORT,F12,F13' / Bolometers  actually measured in observation
N_BOLS  =                  130 / Number of bolometers selected
N_SUBS  =                    2 / Number of sub-instruments used
PHOT_BBF= 'NULL,H7,NULL,NULL,C14,NULL' / The bolometers on the source
PRE_DSCD=                    0 / No. of samples discarded before chop movement
PST_DSCD=                    0 / No. samples discarded after chop movement
REBIN   = 'LINEAR  '           / Rebinning method used by SCUIP
REF_ADC =                   -1 / A/D card of FLATFIELD reference bolometer
REF_CHAN=                   -1 / Channel of FLATFIELD reference bolometer
SAM_TIME=                  125 / A/D sample period in ticks (64musec)
SIMULATE=                    F / True if data is simulated
SKY     = 'jcmtdata_dir:skydip_startup.dat' / Name of sky opacities file
SUB_1   = 'LONG    '           / SCUBA instrument being used
SUB_2   = 'SHORT   '           / SCUBA instrument being used
SUB_3   = 'not used'           / SCUBA instrument being used
SUB_4   = 'not used'           / SCUBA instrument being used
SUB_5   = 'not used'           / SCUBA instrument being used
S_GD_BOL= 'not used'           / Bol. to whose value SW guard ring is set
S_GUARD =                    F / Short wave guard ring on or off
TAUZ_1  =                    0 / Zenith sky optical depth
TAUZ_2  =                    0 / Zenith sky optical depth
TAUZ_3  =                    0 / Zenith sky optical depth
TAUZ_4  =                    0 / Zenith sky optical depth
TAUZ_5  =                    0 / Zenith sky optical depth
T_AMB   =                   -1 / The ambient air temperature (K)
T_COLD_1=                   -1 / Effective temperature of cold load (K)
T_COLD_2=                   -1 / Effective temperature of cold load (K)
T_COLD_3=                   -1 / Effective temperature of cold load (K)
T_COLD_4=                   -1 / Effective temperature of cold load (K)
T_COLD_5=                   -1 / Effective temperature of cold load (K)
T_HOT   =                   -1 / The temperature of the hot load (K)
T_TEL   =                   -1 / The temperature of the telescope
USE_CAL =                    F / .TRUE. if dividing chop by cal before rebin
WAVE_1  =                  862 / Wavelength of map (microns)
```

```
WAVE_2  =                  442 / Wavelength of map (microns)
WAVE_3  =                   -1 / Wavelength of map (microns)
WAVE_4  =                   -1 / Wavelength of map (microns)
WAVE_5  =                   -1 / Wavelength of map (microns)
END
```

D.4 Scan Map

This is a scan map FITS header. Note the presence of MAP_HGHT, MAP_WDTH and MAP_PA to specify the map size (although SURF does not use these since all map parameters are encoded into the RA1, RA2, DEC1 and DEC2 arrays in the SCUCD extension.

```
ACCEPT  = 'not used'           / accept update; PROMPT, YES or NO
ALIGN_AX= 'not used'           / Alignment measurements in X or Y axis
ALIGN_SH=                   -1 / Distance between successive alignment offsets (
ALT-OBS =                 4092 / Height of observatory above sea level (metres)
AMEND   =             1.645293 / Airmass at end of observation
AMSTART =             1.602218 / Airmass at start of observation
APEND   =             628.3394 / Air pressure at end of observation (mbar)
APSTART =             628.3394 / Air pressure at start of observation (mbar)
ATEND   =             4.798538 / Air temp. at end of observation (C)
ATSTART =             4.847378 / Air temp. at start of observation (C)
BOLOMS  = 'LONG,SHORT'         / Names of bolometers measured
CALIBRTR=                    F / Internal calibrator is on or off
CAL_FRQ =             2.929688 / Calibrator frequency (Hz)
CENT_CRD= 'RB      '           / Centre coordinate system
CHOP_CRD= 'LO      '           / Chopper coordinate system
CHOP_FRQ=                7.8125 / Chopper frequency (Hz)
CHOP_FUN= 'SQUARE  '           / Chopper waveform
CHOP_PA =                   90 / Chopper P.A., 0 = in lat, 90 = in long
CHOP_THR=                   44 / Chopper throw (arcsec)
DATA_DIR= '19990808'           / Sub-directory where datafile was stored
DATA_KPT= 'DEMOD   '           / The type of data stored to disk
END_AZD =              214.199 / Azimuth at end of observation (deg)
END_EL  =                   -1 / Elevation of last SKYDIP point (deg)
END_ELD =               37.223 / Elevation at end of observation
EQUINOX =                 2000 / Equinox of mean coordinate system
EXPOSED =                0.128 / Exposure per pixel (seconds)
EXP_NO  =                   12 / Exposure number at end of observation
EXP_TIME=                0.128 / Exposure time for each basic measurement (sec)
E_PER_I =                   -1 / Number of exposures per integration
FILTER  = '450N:850N'          / Filters used
FOCUS_SH=                   -1 / Shift between focus measurements (mm)
GAIN    =                   10 / Programmable gain
HSTEND  = '21:46:43.99933'     / HST at end of observation
HSTSTART= '21:37:22.00058'     / HST at start of observation
HUMEND  =                   17 / Humidity (%) at end of observation
HUMSTART=                   16 / Humidity (%) at start of observation
INSTRUME= 'SCUBA   '           / Name of instrument used
INT_NO  =                    1 / Integration number at end of observation
JIGL_CNT=                   -1 / Number of offsets in jiggle pattern
JIGL_NAM= 'not used'           / File containing jiggle offsets
J_PER_S =                   -1 / Number of jiggles per switch position
```

```
J_REPEAT=                   -1 / No. jiggle pattern repeats in a switch
LAT     = '-024:22:00.00'      / Object latitude
LAT-OBS =        19.8258323669 / Latitude of observatory (degrees)
LAT2    = 'not used'           / Object latitude at MJD2
LOCL_CRD= 'RB      '           / Local offset coordinate system
LONG    = '+016:31:00.00'      / Object longitude
LONG-OBS=        204.520278931 / East longitude of observatory (degrees)
LONG2   = 'not used'           / Object Longitude at MJD2
MAP_HGHT=                  600 / Height of rectangle to be mapped (arcsec)
MAP_PA  =                    0 / P.A. of map vertical, +ve towards +ve long
MAP_WDTH=                  600 / Width of rectangle to be mapped (arcsec)
MAP_X   =                    0 / Map X offset from telescope centre (arcsec)
MAP_Y   =                    0 / Map Y offset from telescope centre (arcsec)
MAX_EL  =                   -1 / Max elevation of sky-dip (deg)
MEANDEC =            -24.46885 / -24:28:07.850189 = approx. mean Dec. (deg)
MEANRA  =             248.5074 / 248:30:26.53198 = approx. mean R.A. (deg)
MEAS_NO =                    1 / Measurement number at end of observation
MIN_EL  =                   -1 / Min elevation of sky-dip (deg)
MJD1    =                   -1 / Modified Julian day planet at RA,DEC
MJD2    =                   -1 / Modified Julian day planet at RA2,DEC2
MODE    = 'MAP     '           / The type of observation
N_INT   =                    1 / No. integrations in the observation
N_MEASUR=                    1 / No. measurements in the observation
OBJECT  = 'b31-22  '           / Name of object
OBJ_TYPE= 'UNKNOWN '           / Type of object
OBSDEF  = 'ss:scan_44ra.obs_b31x22_204433' / The observation definition file
OBSERVER= 'jmk     '           / The name of the observer
PROJ_ID = 'm99au34 '           / The project identification
RUN     =                   38 / Run number of observation
SAM_CRDS= 'NA      '           / Coordinatesystem of sampling mesh
SAM_DX  =                    3 / Sample spacing along scan direction (arcsec)
SAM_DY  =                   60 / Sample spacing perp. to scan (arcsec)
SAM_MODE= 'RASTER  '           / Sampling method
SAM_PA  =                 15.5 / Scan P.A. rel. to lat. line; 0=lat, 90=long
SCAN_REV=                    T / .TRUE. if alternate scans reverse direction
SPK_NSIG=                   -1 / N sigmas from fit of spike threshold
SPK_RMVL=                    F / Automatic spike removal
SPK_WDTH=                   -1 / Assumed width of spike
START_EL=                   -1 / Elevation of first SKYDIP point (deg)
STATE   = 'Terminating         :   STEP RECORD_DATA' / SCUCD state
STEND   = '18:30:20.72296'     / ST at end of observation
STRT_AZD=              212.051 / Azimuth at observation start (deg)
STRT_ELD=              38.3846 / Elevation at observation start (deg)
STSTART = '18:20:57.18613'     / ST at start of observation
SWTCH_MD= 'NOSW    '           / Switch mode of observation
SWTCH_NO=                    1 / Switch number at end of observation
S_PER_E =                    1 / Number of switch positions per exposure
TELESCOP= 'JCMT    '           / Name of telescope
TEL_OPER= 'kkp     '           / Telescope operator
UTDATE  = '1999:8:8'           / UT date of observation
UTEND   = '7:46:43.99933'      / UT at end of observation
UTSTART = '7:37:22.00058'      / UT at start of observation
VERSION =                  1.1 / SCUCD version
ALIGN_DX=             0.027268 / SMU tables X axis alignment offset
```

```
ALIGN_DY=             0.482489 / SMU tables Y axis alignment offset
ALIGN_X =             -2.02578 / SMU tables X axis
ALIGN_Y =              3.79108 / SMU tables Y axis
AZ_ERR  =                    0 / Error in the telescope azimuth
CHOPPING=                    T / SMU Chopper chopping state
EL_ERR  =                    0 / Error in the telescope elevation
FOCUS_DZ=            -0.361142 / SMU tables Z axis focus offset
FOCUS_Z =              -16.756 / SMU tables Z axis
SEEING  =                0.525 / SAO atmospheric seeing
SEE_DATE= '9908061715'         / Date and time of SAO seeing
TAU_225 =                0.063 / CSO tau
TAU_DATE= '9908080735'         / Date and time of CSO tau
TAU_RMS =              2.0E-03 / CSO tau rms
UAZ     =              1.42235 / User azimuth pointing offset
UEL     =              4.05503 / User elevation pointing offset
UT_DATE = ' 8 Aug 1999'        / UT date at start of observation
BAD_LIM =                   32 / No. spikes before quality set bad
CALIB_LG=                    6 / Lag of internal calibrator in samples
CALIB_PD=             42.66667 / Period of internal calibrator in samples
CHOP_LG =                    4 / Chop lag in samples
CHOP_PD =                   16 / Chop period in samples
CNTR_DU3=                    0 / Nasmyth dU3 coord of instrument centre
CNTR_DU4=                    0 / Nasmyth dU4 coord of instrument centre
ETATEL_1=                   -1 / Transmission of telescope
ETATEL_2=                   -1 / Transmission of telescope
ETATEL_3=                   -1 / Transmission of telescope
ETATEL_4=                   -1 / Transmission of telescope
ETATEL_5=                   -1 / Transmission of telescope
FILT_1  = '450     '           / Filter name
FILT_2  = '850     '           / Filter name
FILT_3  = 'not used'           / Filter name
FILT_4  = 'not used'           / Filter name
FILT_5  = 'not used'           / Filter name
FLAT    = 'jcmtdata_dir:lwswphot.dat' / Name of flat-field file
JIG_DSCD=                   -1 / No. samples discarded after jiggle movement
L_GD_BOL= 'not used'           / Bol. to whose value LW guard ring is set
L_GUARD =                    F / Long wave guard ring on or off
MEAS_BOL= 'not used'           / Bolometers  actually measured in observation
N_BOLS  =                  128 / Number of bolometers selected
N_SUBS  =                    2 / Number of sub-instruments used
PHOT_BBF= 'not_used       LL,C14,NULL' / The bolometers on the source
PRE_DSCD=                    0 / No. of samples discarded before chop movement
PST_DSCD=                    0 / No. samples discarded after chop movement
REBIN   = 'LINEAR  '           / Rebinning method used by SCUIP
REF_ADC =                   -1 / A/D card of FLATFIELD reference bolometer
REF_CHAN=                   -1 / Channel of FLATFIELD reference bolometer
SAM_TIME=                  125 / A/D sample period in ticks (64musec)
SIMULATE=                    F / True if data is simulated
SKY     = 'jcmtdata_dir:skydip_startup.dat' / Name of sky opacities file
SUB_1   = 'SHORT   '           / SCUBA instrument being used
SUB_2   = 'LONG    '           / SCUBA instrument being used
SUB_3   = 'not used'           / SCUBA instrument being used
SUB_4   = 'not used'           / SCUBA instrument being used
SUB_5   = 'not used'           / SCUBA instrument being used
```

```
S_GD_BOL= 'not used'           / Bol. to whose value SW guard ring is set
S_GUARD =                    F / Short wave guard ring on or off
TAUZ_1  =                    0 / Zenith sky optical depth
TAUZ_2  =                    0 / Zenith sky optical depth
TAUZ_3  =                    0 / Zenith sky optical depth
TAUZ_4  =                    0 / Zenith sky optical depth
TAUZ_5  =                    0 / Zenith sky optical depth
T_AMB   =                   -1 / The ambient air temperature (K)
T_COLD_1=                   -1 / Effective temperature of cold load (K)
T_COLD_2=                   -1 / Effective temperature of cold load (K)
T_COLD_3=                   -1 / Effective temperature of cold load (K)
T_COLD_4=                   -1 / Effective temperature of cold load (K)
T_COLD_5=                   -1 / Effective temperature of cold load (K)
T_HOT   =                   -1 / The temperature of the hot load (K)
T_TEL   =                   -1 / The temperature of the telescope
USE_CAL =                    F / .TRUE. if dividing chop by cal before rebin
WAVE_1  =                  442 / Wavelength of map (microns)
WAVE_2  =                  862 / Wavelength of map (microns)
WAVE_3  =                    0 / Wavelength of map (microns)
WAVE_4  =                    0 / Wavelength of map (microns)
WAVE_5  =                    0 / Wavelength of map (microns)
END
```

E Flatfield file format

This is the flatfield file used for SCUBA at the time of writing:

```
proc SET_BOLS
{ flat field data file written by SCUDR
{ Mon Jun 16 17:20:00 1997
{ observation run number 01
{      Name Type      dU3          dU4          Calib       Theta       A           B          Qual
 SETBOL G1  LONG      0.7830E+01 -0.7610E+02  0.1290E+01  0.4280E+01  0.7106E+01  0.7117E+01  0  66.0359490741  12 H8
 SETBOL G2  LONG      0.2679E+02 -0.6138E+02  0.1086E+01  0.2370E+00  0.7387E+01  0.6795E+01  0  66.0482523148  12 H8
 SETBOL G3  LONG      0.4785E+02 -0.4588E+02  0.1160E+01  0.1088E+00  0.7340E+01  0.6600E+01  0  66.0606250000  12 H8
 SETBOL G4  LONG      0.7119E+02 -0.3377E+02  0.1080E+01  0.2125E-01  0.7499E+01  0.6642E+01  0  66.0729050926  12 H8
 SETBOL G7  LONG     -0.1777E+02 -0.6524E+02  0.1089E+01  0.1843E+01  0.6980E+01  0.7279E+01  0  66.0852893519  12 H8
 SETBOL G8  LONG      0.2193E+01 -0.4958E+02  0.1032E+01 -0.1701E+02 -0.4981E+01 -0.9338E+00  0  66.0979166667  12 H8
 SETBOL G9  LONG      0.2291E+02 -0.3673E+02  0.1194E+01  0.9538E+01  0.7368E+01  0.6878E+01  0  66.1170254630  13 H8
 SETBOL G10 LONG      0.4500E+02 -0.2298E+02  0.1187E+01  0.1407E+02  0.6766E+01  0.7412E+01  0  66.1295138889  13 H8
 SETBOL G11 LONG      0.6850E+02 -0.1106E+02  0.1159E+01 -0.1926E+00  0.7540E+01  0.6623E+01  0  66.1417013889  13 H8
 SETBOL G13 LONG     -0.3950E+02 -0.5281E+02  0.1039E+01 -0.3004E+01  0.7097E+01  0.6988E+01  0  66.1540625000  13 H8
 SETBOL G14 LONG     -0.2157E+02 -0.3796E+02  0.1072E+01 -0.1470E+01  0.6906E+01  0.7204E+01  0  66.1663194444  13 H8
 SETBOL G15 LONG      0.0510E+00 -0.2533E+02  0.1150E+01 -0.7553E-01  0.7276E+01  0.6851E+01  0  66.1786805556  13 H8
 SETBOL G16 LONG      0.2174E+02 -0.1171E+02  0.9190E+00  0.6017E+03  0.7038E+01  0.7038E+01  0  66.1910532407  13 H8
 SETBOL H1  LONG      0.4506E+02  0.1172E+01  0.1014E+01 -0.2208E+00  0.7407E+01  0.6631E+01  0  66.2103472222  14 H8
 SETBOL H2  LONG      0.6772E+02  0.1328E+02  0.9490E+00 -0.1933E+00  0.7481E+01  0.6458E+01  0  66.2226736111  14 H8
 SETBOL I12 LONG_DC   0.0000E+00  0.0000E+00  0.1000E+01  0.0000E+00  0.0000E+00  0.0000E+00  0   0.0000000000   0 NUL
 SETBOL H4  LONG     -0.6336E+02 -0.3956E+02  0.1003E+01  0.5146E+02  0.7361E+01  0.6270E+01  0  66.2350000000  14 H8
 SETBOL H5  LONG     -0.4256E+02 -0.2631E+02  0.9240E+00 -0.2044E+02  0.7017E+01  0.7017E+01  0  66.2473611111  14 H8
 SETBOL H6  LONG     -0.2200E+02 -0.1264E+02  0.9870E+00 -0.3340E+01  0.7200E+01  0.6928E+01  0  66.2596296296  14 H8
 SETBOL H7  LONG     -0.0000E+00  0.0000E+00  0.1000E+01  0.4839E+01  0.7151E+01  0.6991E+01  0  66.2720138889  14 H8
 SETBOL H8  LONG      0.2262E+02  0.1231E+02  0.9270E+00 -0.3250E+00  0.7358E+01  0.6627E+01  0  66.5038773148  16 H8
 SETBOL H9  LONG      0.4633E+02  0.2465E+02  0.1039E+01  0.1223E+02  0.7425E+01  0.6624E+01  0  66.2913541667  15 H8
 SETBOL H10 LONG      0.7049E+02  0.3728E+02  0.1134E+01 -0.3192E+00  0.7535E+01  0.6472E+01  0  66.3037037037  15 H8
 SETBOL H11 LONG     -0.6490E+02 -0.1333E+02  0.9580E+00  0.7245E+01  0.6991E+01  0.7178E+01  0  66.3161921296  15 H8
 SETBOL H12 LONG     -0.4228E+02 -0.4700E+00  0.9160E+00  0.1142E+01  0.6943E+01  0.7179E+01  0  66.3286921296  15 H8
 SETBOL H13 LONG     -0.2029E+02  0.1298E+02  0.1068E+01 -0.4099E+00  0.7245E+01  0.6759E+01  0  66.3409490741  15 H8
 SETBOL H14 LONG      0.1250E+01  0.2529E+02  0.9650E+00  0.3552E+01  0.7087E+01  0.7087E+01  0  66.3532407407  15 H8
```

```
 SETBOL H15 LONG      0.2519E+02  0.3783E+02  0.9380E+00 -0.2608E+05  0.7130E+01  0.7130E+01  0  66.3657407407  15 H8
 SETBOL H16 LONG      0.4965E+02  0.4959E+02  0.9280E+00 -0.3867E+00  0.7513E+01  0.6436E+01  0  66.3780902778  15 H8
 SETBOL I1  LONG     -0.6385E+02  0.1301E+02  0.9940E+00 -0.2134E+01  0.6749E+01  0.7152E+01  0  66.3975578704  16 H8
 SETBOL I2  LONG     -0.4057E+02  0.2669E+02  0.1051E+01  0.4635E+03  0.7093E+01  0.7094E+01  0  66.4102546296  16 H8
 SETBOL I3  LONG     -0.1803E+02  0.3894E+02  0.1135E+01  0.8583E+02  0.6732E+01  0.7409E+01  0  66.4227893519  16 H8
 SETBOL I4  LONG      0.5930E+01  0.5154E+02  0.9620E+00 -0.5106E+01  0.6806E+01  0.7535E+01  0 192.7277662037   6 H7
 SETBOL I5  LONG      0.2938E+02  0.6318E+02  0.9410E+00 -0.4519E+00  0.7451E+01  0.6512E+01  0  66.4477083333  16 H8
 SETBOL I6  LONG     -0.6020E+02  0.4057E+02  0.9970E+00  0.3169E+01  0.7024E+01  0.7024E+01  0  66.4604976852  16 H8
 SETBOL I7  LONG     -0.3669E+02  0.5404E+02  0.1054E+01 -0.6765E+02  0.7037E+01  0.7043E+01  0  66.4728125000  16 H8
 SETBOL I8  LONG     -0.1578E+02  0.6389E+02  0.1065E+01 -0.2099E+02  0.6717E+01  0.7520E+01  0  66.4855787037  16 H8
 SETBOL I9  LONG      0.9047E+01  0.7776E+02  0.1045E+01 -0.7307E-02  0.7092E+01  0.7032E+01  0  66.4977546296  16 H8

{ short-wave array
 SETBOL A1  SHORT     0.5708E+02 -0.2814E+02  0.1286E+01 -0.7433E-01  0.4129E+01  0.3690E+01  0  64.9820833333  43 C14
 SETBOL A2  SHORT     0.4590E+02 -0.3591E+02  0.9311E+00 -0.3590E-01  0.4002E+01  0.3673E+01  0  64.9895138889  43 C14
 SETBOL A3  SHORT     0.3615E+02 -0.4146E+02  0.9499E+00  0.1720E+00  0.3987E+01  0.3677E+01  0  64.9969675926  43 C14
 SETBOL A4  SHORT     0.2464E+02 -0.4962E+02  0.1002E+01  0.2098E+00  0.3995E+01  0.3715E+01  0  65.0044212963  43 C14
 SETBOL A5  SHORT     0.1455E+02 -0.5628E+02  0.9585E+00 -0.4524E+00  0.3849E+01  0.3849E+01  0  65.0118055556  43 C14
 SETBOL A6  SHORT     0.3620E+01 -0.6315E+02  0.1115E+01 -0.1306E+01  0.3761E+01  0.3982E+01  0  65.0192824074  43 C14
 SETBOL A7  SHORT     0.5638E+02 -0.1661E+02  0.1000E+01 -0.2444E+00  0.4263E+01  0.3668E+01  0  65.0266898148  43 C14
 SETBOL A8  SHORT     0.4508E+02 -0.2315E+02  0.1126E+01 -0.1680E+00  0.3962E+01  0.3597E+01  0  65.0341319444  43 C14
 SETBOL A9  SHORT     0.3495E+02 -0.3012E+02  0.1089E+01 -0.2392E-01  0.3919E+01  0.3687E+01  0  65.5473379630   4 C14
 SETBOL A10 SHORT     0.2240E+02 -0.3762E+02  0.1107E+01 -0.8797E-02  0.3912E+01  0.3632E+01  0  65.5551041667   4 C14
 SETBOL A11 SHORT     0.1314E+02 -0.4396E+02  0.1203E+01 -0.2799E-02  0.3922E+01  0.3644E+01  0  65.5626851852   4 C14
 SETBOL A12 SHORT     0.2812E+01 -0.5070E+02  0.1067E+01 -0.6335E+00  0.3796E+01  0.3796E+01  0  65.5701967593   4 C14
 SETBOL A13 SHORT    -0.7590E+01 -0.5808E+02  0.9114E+00  0.3143E+02  0.3849E+01  0.3851E+01  0  65.5777777778   4 C14
 SETBOL A14 SHORT     0.5559E+02 -0.4683E+01  0.9639E+00 -0.1907E+00  0.4101E+01  0.3452E+01  0  65.5855555556   4 C14
 SETBOL A15 SHORT     0.4502E+02 -0.1081E+02  0.1068E+01 -0.2673E+00  0.3985E+01  0.3569E+01  0  65.5932754630   4 C14
 SETBOL A16 SHORT     0.3296E+02 -0.1807E+02  0.1170E+01 -0.1050E+00  0.3974E+01  0.3628E+01  0  65.6006944444   4 C14
 SETBOL B1  SHORT     0.2224E+02 -0.2456E+02  0.1102E+01 -0.9115E-01  0.3948E+01  0.3680E+01  0  65.6129629630   5 C14
 SETBOL B2  SHORT     0.1165E+02 -0.3120E+02  0.1157E+01 -0.6530E-02  0.3921E+01  0.3681E+01  0  65.6204513889   5 C14
 SETBOL B3  SHORT     0.2000E+01 -0.3855E+02  0.9660E+00  0.2064E-01  0.3904E+01  0.3688E+01  0  65.6279282407   5 C14
 SETBOL B4  SHORT    -0.9089E+01 -0.4490E+02  0.1076E+01  0.2914E-01  0.3910E+01  0.3702E+01  0  65.6354513889   5 C14
 SETBOL B5  SHORT    -0.1951E+02 -0.5160E+02  0.9298E+00  0.7764E+01  0.3756E+01  0.3901E+01  0  65.6433101852   5 C14
 SETBOL B6  SHORT     0.5585E+02  0.7780E+01  0.1088E+01 -0.2603E+00  0.4125E+01  0.3553E+01  0  72.3519444444  58 C14
 SETBOL B7  SHORT     0.4489E+02  0.9322E+00  0.1020E+01 -0.3361E+05  0.3717E+01  0.3717E+01  0  65.6584606481   5 C14
 SETBOL B8  SHORT     0.3283E+02 -0.5420E+01  0.1112E+01  0.5543E+00  0.3792E+01  0.3789E+01  0  65.6659953704   5 C14
 SETBOL B9  SHORT     0.2169E+02 -0.1181E+02  0.1019E+01  0.1420E+01  0.3657E+01  0.3946E+01  0  65.3514120370   1 C14
```

```
SETBOL B10 SHORT     0.1166E+02 -0.1921E+02  0.1063E+01  0.1457E+01  0.3861E+01  0.3776E+01  0  65.3589467593   1 C14
SETBOL B11 SHORT     0.7480E+00 -0.2610E+02  0.1017E+01  0.6716E+00  0.3820E+01  0.3829E+01  0  65.3664004630   1 C14
SETBOL B12 SHORT    -0.1013E+02 -0.3228E+02  0.1085E+01 -0.1378E+00  0.3904E+01  0.3757E+01  0  65.3738888889   1 C14
SETBOL B13 SHORT    -0.2043E+02 -0.3952E+02  0.1112E+01 -0.2830E+00  0.3875E+01  0.3700E+01  0  65.3814351852   1 C14
SETBOL B14 SHORT    -0.2907E+02 -0.4586E+02  0.9829E+00 -0.2085E+00  0.3860E+01  0.3679E+01  0  65.3888657407   1 C14
SETBOL B15 SHORT     0.5681E+02  0.2020E+02  0.9500E+00 -0.2601E+00  0.4107E+01  0.3470E+01  0  72.3643634259  58 C14
SETBOL B16 SHORT     0.4512E+02  0.1323E+02  0.1090E+01 -0.3063E+00  0.3952E+01  0.3450E+01  0  72.3767245370  58 C14
SETBOL C1  SHORT     0.3315E+02  0.7096E+01  0.1004E+01 -0.2430E+00  0.3927E+01  0.3510E+01  0  65.4163888889   2 C14
SETBOL C2  SHORT     0.2194E+02  0.7400E+00  0.1066E+01 -0.2254E+00  0.3982E+01  0.3568E+01  0  65.4239004630   2 C14
SETBOL C3  SHORT     0.1091E+02 -0.6110E+01  0.1044E+01  0.2300E+00  0.3858E+01  0.3727E+01  0  65.4313425926   2 C14
SETBOL C4  SHORT     0.2756E+00 -0.1215E+02  0.1254E+01 -0.1419E+01  0.3763E+01  0.3830E+01  0  65.4390509259   2 C14
SETBOL C5  SHORT    -0.1075E+02 -0.1977E+02  0.1111E+01  0.5111E+00  0.3827E+01  0.3827E+01  0  65.4468634259   2 C14
SETBOL C6  SHORT    -0.2075E+02 -0.2666E+02  0.1075E+01 -0.2159E+00  0.3901E+01  0.3764E+01  0  65.4543634259   2 C14
SETBOL C7  SHORT    -0.3146E+02 -0.3298E+02  0.1287E+01  0.7394E+01  0.3743E+01  0.3888E+01  0  65.4617824074   2 C14
SETBOL C8  SHORT    -0.4124E+02 -0.4042E+02  0.9000E+00  0.4200E+01  0.3704E+01  0.3741E+01  0  65.4693287037   2 C14
SETBOL C9  SHORT     0.5849E+02  0.3141E+02  0.1326E+01  0.3450E+00  0.3868E+01  0.3868E+01  0  72.3890740741  58 C14
SETBOL C10 SHORT     0.4631E+02  0.2581E+02  0.1066E+01  0.1101E+01  0.3526E+01  0.4008E+01  0  65.4891782407   3 C14
SETBOL C11 SHORT     0.3388E+02  0.2000E+02  0.1229E+01  0.4242E+00  0.3780E+01  0.3780E+01  0  65.4966782407   3 C14
SETBOL C12 SHORT     0.2248E+02  0.1267E+02  0.1131E+01  0.1323E+01  0.3537E+01  0.3973E+01  0  65.5042824074   3 C14
SETBOL C13 SHORT     0.1116E+02  0.6479E+01  0.1062E+01  0.5274E+00  0.3760E+01  0.3760E+01  0  65.5119328704   3 C14
SETBOL C14 SHORT     0.0000E+00 -0.0000E+00  0.1000E+01 -0.2339E+00  0.3860E+01  0.3595E+01  0  72.4200810185  58 C14
SETBOL C15 SHORT    -0.1116E+02 -0.6479E+01  0.1130E+01  0.1179E+01  0.3666E+01  0.3898E+01  0  65.5270949074   3 C14
SETBOL C16 SHORT    -0.2215E+02 -0.1332E+02  0.1101E+01 -0.3823E+00  0.3928E+01  0.3760E+01  0  65.5345717593   3 C14
SETBOL D1  SHORT    -0.3139E+02 -0.2045E+02  0.1157E+01  0.4295E+01  0.3729E+01  0.3857E+01  0  65.6783796296   6 C14
SETBOL D2  SHORT    -0.4220E+02 -0.2734E+02  0.1175E+01  0.2061E+01  0.3803E+01  0.3755E+01  0  65.6859375000   6 C14
SETBOL D3  SHORT    -0.5298E+02 -0.3271E+02  0.1142E+01 -0.2841E+00  0.3731E+01  0.3861E+01  0  65.6934606481   6 C14
SETBOL D4  SHORT     0.4815E+02  0.3794E+02  0.9828E+00 -0.2879E+01  0.3816E+01  0.3816E+01  0  65.7009027778   6 C14
SETBOL D5  SHORT     0.3558E+02  0.3207E+02  0.1027E+01 -0.2057E+01  0.3540E+01  0.3993E+01  0  65.7086342593   6 C14
SETBOL D6  SHORT     0.2340E+02  0.2565E+02  0.1080E+01  0.2692E+01  0.3989E+01  0.3572E+01  0  65.7160416667   6 C14
SETBOL D7  SHORT     0.1184E+02  0.1956E+02  0.1258E+01 -0.1224E+01  0.3779E+01  0.3779E+01  0  65.7237500000   6 C14
SETBOL D8  SHORT     0.1100E+01  0.1301E+02  0.1052E+01  0.1119E+01  0.3643E+01  0.3878E+01  0  65.7313888889   6 C14
SETBOL D9  SHORT    -0.1030E+02  0.6303E+01  0.1011E+01  0.1506E+02  0.3734E+01  0.3770E+01  0  65.7436805556   7 C14
SETBOL D10 SHORT    -0.2141E+02 -0.4385E+00  0.1088E+01 -0.2669E+00  0.3951E+01  0.3611E+01  0  65.7512615741   7 C14
SETBOL D11 SHORT    -0.3244E+02 -0.6844E+01  0.1173E+01  0.1050E+01  0.3711E+01  0.3913E+01  0  65.7587268519   7 C14
SETBOL D12 SHORT    -0.4299E+02 -0.1380E+02  0.1172E+01 -0.8284E+00  0.3793E+01  0.3728E+01  0  65.7662037037   7 C14
SETBOL D13 SHORT    -0.5295E+02 -0.2078E+02  0.9500E+00  0.5212E+01  0.3815E+01  0.3733E+01  0  65.7740393519   7 C14
SETBOL D14 SHORT     0.3682E+02  0.4498E+02  0.9517E+00 -0.6182E+00  0.4028E+01  0.3525E+01  0  65.7814583333   7 C14
SETBOL D15 SHORT     0.2543E+02  0.3744E+02  0.1036E+01 -0.6247E+01  0.3738E+01  0.3775E+01  0  65.7891203704   7 C14
```

```
 SETBOL D16 SHORT     0.1301E+02  0.3179E+02  0.1248E+01  0.2862E+01  0.3836E+01  0.3636E+01  0  65.7968518519   7 C14
 SETBOL E1  SHORT     0.1499E+01  0.2630E+02  0.1153E+01 -0.5981E+01  0.3772E+01  0.3773E+01  0  65.8090972222   8 C14
 SETBOL E2  SHORT    -0.9694E+01  0.1926E+02  0.1098E+01  0.6842E+01  0.3768E+01  0.3768E+01  0  72.4015509259  58 C14
 SETBOL E3  SHORT    -0.2096E+02  0.1302E+02  0.1142E+01 -0.3893E+00  0.3940E+01  0.3670E+01  0  65.8240972222   8 C14
 SETBOL E4  SHORT    -0.3153E+02  0.6590E+01  0.1126E+01  0.3139E+00  0.3796E+01  0.3796E+01  0  65.8315740741   8 C14
 SETBOL E5  SHORT    -0.4362E+02 -0.8783E+00  0.1026E+01  0.6693E+02  0.3609E+01  0.3792E+01  0  65.8392939815   8 C14
 SETBOL E6  SHORT    -0.5395E+02 -0.7067E+01  0.1020E+01  0.9356E+00  0.3635E+01  0.3792E+01  0  65.8469444444   8 C14
 SETBOL E7  SHORT     0.2731E+02  0.5184E+02  0.9742E+00  0.7678E+02  0.3756E+01  0.3897E+01  0  65.8546759259   8 C14
 SETBOL E8  SHORT     0.1511E+02  0.4443E+02  0.1204E+01 -0.5818E+00  0.4007E+01  0.3567E+01  0  65.8621875000   8 C14
 SETBOL E9  SHORT     0.3500E+01  0.3973E+02  0.1020E+01 -0.2097E+01  0.3605E+01  0.3976E+01  0  65.8745717593   9 C14
 SETBOL E10 SHORT    -0.8784E+01  0.3302E+02  0.1197E+01 -0.4411E+01  0.3800E+01  0.3767E+01  0  72.4138888889  58 C14
 SETBOL E11 SHORT    -0.1987E+02  0.2656E+02  0.1005E+01  0.4240E+01  0.3643E+01  0.3936E+01  0  65.8894791667   9 C14
 SETBOL E12 SHORT    -0.3045E+02  0.2024E+02  0.1067E+01  0.1210E+02  0.3940E+01  0.3664E+01  0  65.8971643519   9 C14
 SETBOL E13 SHORT    -0.4249E+02  0.1245E+02  0.9850E+00  0.8132E+00  0.3809E+01  0.3739E+01  0  65.9045254630   9 C14
 SETBOL E14 SHORT    -0.5245E+02  0.6274E+01  0.1177E+01 -0.1508E+01  0.3760E+01  0.3760E+01  0  65.9120717593   9 C14
 SETBOL E15 SHORT     0.1652E+02  0.5893E+02  0.9800E+00  0.3954E+01  0.3587E+01  0.4123E+01  0  65.9196643519   9 C14
 SETBOL E16 SHORT     0.3524E+01  0.5163E+02  0.1174E+01  0.9079E+00  0.3618E+01  0.4055E+01  0  65.9270949074   9 C14
 SETBOL F1  SHORT    -0.7665E+01  0.4592E+02  0.1168E+01  0.3404E+01  0.3775E+01  0.3842E+01  0  65.9394328704  10 C14
 SETBOL F2  SHORT    -0.1817E+02  0.3963E+02  0.1020E+01 -0.5761E+00  0.3991E+01  0.3608E+01  0  65.9469560185  10 C14
 SETBOL F3  SHORT    -0.3008E+02  0.3403E+02  0.1295E+01 -0.5724E+00  0.3977E+01  0.3659E+01  0  65.9544675926  10 C14
 SETBOL F4  SHORT    -0.4123E+02  0.2619E+02  0.1244E+01  0.8881E+00  0.3627E+01  0.3909E+01  0  65.9621064815  10 C14
 SETBOL F5  SHORT    -0.5155E+02  0.2029E+02  0.1069E+01  0.1093E+00  0.3760E+01  0.3760E+01  0  65.9695601852  10 C14
 SETBOL F6  SHORT     0.6895E+01  0.6560E+02  0.1245E+01  0.4535E-01  0.3981E+01  0.3981E+01  0  65.9770949074  10 C14
 SETBOL F7  SHORT    -0.4842E+01  0.6013E+02  0.9947E+00 -0.7757E+00  0.4069E+01  0.3691E+01  0  65.9846527778  10 C14
 SETBOL F8  SHORT    -0.1690E+02  0.5287E+02  0.1017E+01  0.8808E+00  0.3724E+01  0.4061E+01  0  65.9920138889  10 C14
 SETBOL F9  SHORT    -0.2931E+02  0.4752E+02  0.9485E+00 -0.5179E+01  0.3703E+01  0.3979E+01  0  66.0041087963  11 C14
 SETBOL F10 SHORT    -0.4009E+02  0.3977E+02  0.9474E+00  0.8256E+00  0.3667E+01  0.3968E+01  0  66.0116203704  11 C14
 SETBOL F11 SHORT    -0.5045E+02  0.3338E+02  0.1169E+01  0.7934E+00  0.3701E+01  0.3940E+01  0  66.0190625000  11 C14
 SETBOL I11 SHORT_DC  0.0000E+00  0.0000E+00  0.1000E+01  0.0000E+00  0.0000E+00  0.0000E+00  1   0.0000000000   0 NUL
{ P1100 bolometer
 SETBOL I10 P1100    -0.5465E+02  0.7116E+02  0.1000E+01 -0.1453E+03  0.1002E+02  0.9003E+01  1  69.6688541667  65 NUL
 SETBOL I15 P1100_DC  0.0000E+00  0.0000E+00  0.1000E+01  0.0000E+00  0.0000E+00  0.0000E+00  1   0.0000000000   0 NUL
{ P1350 bolometer
 SETBOL G12 P1350    -0.5827E+02 -0.7068E+02  0.1000E+01 -0.8953E+00  0.9681E+01  0.1101E+02  0  69.7100231481  66 NUL
 SETBOL I14 P1350_DC  0.0000E+00  0.0000E+00  0.1000E+01  0.0000E+00  0.0000E+00  0.0000E+00  0   0.0000000000   0 NUL
{ P2000 bolometer
 SETBOL G6  P2000     0.1000E+03  0.2375E+01  0.1000E+01 -0.1447E+00  0.1735E+02  0.1374E+02  0  69.7901157407  71 NUL
 SETBOL I13 P2000_DC  0.0000E+00  0.0000E+00  0.1000E+01  0.0000E+00  0.0000E+00  0.0000E+00  0   0.0000000000   0 NUL
```

```
end proc
```

F The DREAM2SURF import task

This section lists the complete source code for the `dream2surf` import task. This is an example of an import task for JIGGLE map type observations.

```
      SUBROUTINE DREAM2SURF ( STATUS )
*+
*  Name:
*     DREAM2SURF

*  Purpose:
*     Convert DREAM output data into SURF data format

*  Language:
*     Starlink Fortran 77

*  Type of Module:
*     ADAM A-task

*  Invocation:
*     CALL DREAM2SURF ( STATUS )

*  Arguments:
*     STATUS = INTEGER (Given and Returned)
*        The global status

*  Description:
*     This routine reads in a DREAM format file and converts it
*     into a file format that can be processed by SURF.
*     This new file will have been 'REDUCE_SWITCH'ed and
*     'FLATFIELD'ed.

*  Usage:
*     dream2surf in out

*  ADAM Parameters:
*     FLATFILE = CHAR (Read)
*        File containing the flatfield description
*     FSCYCLE = INTEGER (Read)
*        First cycle number to read from file
*     IN = FILE (Read)
*        The name of the DREAM input file to open.
*     MSG_FILTER = CHAR (Read)
*         Message filter level. Default is NORM.
*     NRCYCLE = INTEGER (Read)
*         Number of cycles to read from file
*     OUT = NDF (Write)
*        The NDF output file.

*  Authors:
*     TIMJ: T. Jenness (timj@jach.hawaii.edu)
*     GREVE: H.W. van Someren Greve (greve@nfra.nl)
```

```
*  Copyright:
*     Copyright (C) 1998 Particle Physics and Astronomy
*     Research Council. All Rights Reserved.


*  History:
*     Revision 1.4  1999/08/03 19:32:28  timj
*     Add copyright message to header.
*
*     Revision 1.3  1998/06/24 19:26:03  timj
*     Finally get jiggle pattern to work for new format sol files.
*
*     Revision 1.2  1998/06/19 19:28:26  timj
*     First released version
*
*     Revision 1.1  1998/05/14 20:41:03  timj
*     Initial revision
*

*  Bugs:
*     {note_any_bugs_here}

*-

*  Type Definitions:
      IMPLICIT NONE

*  Global constants:
      INCLUDE 'SAE_PAR'       ! Starlink status
      INCLUDE 'DAT_PAR'       ! DAT__ constants
      INCLUDE 'PRM_PAR'       ! For VAL__
      INCLUDE 'MSG_PAR'       ! For MSG_

      INCLUDE 'SCU_SOL'       ! Description of DREAM header file
      INCLUDE 'SURF_PAR'      ! Standard SCUBA include

*  COMMON data
      COMMON SOLPA            ! DREAM common block

*  Status:
      INTEGER STATUS

*  External references
      INTEGER CHR_LEN
      EXTERNAL CHR_LEN

*  Local Constants:
      INTEGER       DREAM__NBYTES
      PARAMETER     (DREAM__NBYTES = 4)  ! Number of bytes per DREAM record
      INTEGER       SRECSIZE
      PARAMETER     (SRECSIZE = 4096)          ! Small record size in bytes
      INTEGER       HEADER
      PARAMETER     (HEADER   = 5120)          ! Nr of I4 in Header records
```

```
      CHARACTER*(10) TSKNAME                    ! Task name
      PARAMETER (TSKNAME = 'DREAM2SURF')

      DOUBLE PRECISION LAT_OBS           ! Latitude of JCMT (degrees)
      PARAMETER (LAT_OBS = 19.8258323669)
      DOUBLE PRECISION LONG_OBS          ! Longitude of JCMT (degrees)
      PARAMETER (LONG_OBS = 204.520278931)



*  Local variables:
      INTEGER ADC             ! A to D number
      REAL             BOL_CALB (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! bolometer flatfield factors
      DOUBLE PRECISION BOL_DAY (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! time and day number on which
                                          ! the bolometer flatfield was
                                          ! measured
      REAL             BOL_DU3 (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! Nasmyth dU3 coords of
                                          ! bolometers
      REAL             BOL_DU4 (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! Nasmyth dU4 coords of
                                          ! bolometers
      INTEGER          BOL_QUAL (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! bolometer flatfield quality
      CHARACTER*20     BOL_REF (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! flatfield reference
                                          ! bolometers
      REAL             BOL_RTEMP (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! scratch real bolometer data
      INTEGER          BOL_RUN (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! run number on which the
                                          ! bolometer flatfield was
                                          ! measured
      CHARACTER*20     BOL_TYPE (SCUBA__NUM_CHAN, SCUBA__NUM_ADC)
                                          ! bolometer types
      INTEGER BOL_ADC ( SCUBA__NUM_ADC * SCUBA__NUM_CHAN ) ! ADC numbers
      INTEGER BOL_CHAN ( SCUBA__NUM_ADC * SCUBA__NUM_CHAN ) ! Channel numbers
      CHARACTER *10 CENT_CRD  ! Centre coordinate system
      INTEGER CHAN            ! Channel number
      CHARACTER * 80 CTEMP    ! Scratch string
      DOUBLE PRECISION DEC    ! Declination
      INTEGER DEMBNDS ( 4 )   ! Dimensions of DEM_PNTR
      INTEGER DEM_PNTR        ! Pointer to DEM_PNTR extension
      INTEGER DIM ( 4 )       ! Dimensions of an array
      DOUBLE PRECISION DLST   ! Time per cycle
      INTEGER DREAM_PTR       ! Pointer to DREAM input data
      DOUBLE PRECISION DTEMP  ! Scratch double
      CHARACTER * (80) FITS (SCUBA__MAX_FITS)  ! FITS extension
      INTEGER LBND ( 2 )      ! Lower bounds of output NDF
      INTEGER LUN             ! Logical unit number of input file
      INTEGER ERR             ! Error from DREAM system
      INTEGER FD              ! File descriptor of input
```

```
      CHARACTER*80 FLATFILE   ! Flatfield file name
      INTEGER FSCYCLE         ! First cycle to read from input file
      INTEGER I               ! Loop integer
      INTEGER IERR            ! For VEC_
      INTEGER IPOSN           ! Position in string
      INTEGER ITEMP           ! Scratch integer
      REAL    JIGL_X ( SCUBA__MAX_JIGGLE ) ! X jiggle positions
      REAL    JIGL_Y ( SCUBA__MAX_JIGGLE ) ! Y jiggle positions
      DOUBLE PRECISION LONGITUDE ! Longitude degrees west
      DOUBLE PRECISION LST    ! LST of observation
      DOUBLE PRECISION LST_STRT ! Start LST of observation (decimal hours)
      INTEGER MAXCYC          ! Maximum number of cycles that can be read
      DOUBLE PRECISION MJD    ! MJD of observation
      INTEGER NBYTES          ! Number of bytes per data record
      INTEGER NDIM            ! Number of dimensions in an array
      INTEGER NERR            ! For VEC_
      INTEGER NRCYCLE         ! Number of cycles to read
      INTEGER NREC            ! Number of records per cycle
      INTEGER NSVAL           ! Number of solved values per cycle
      INTEGER N_FITS          ! Number of FITS components
      INTEGER OFFSET          ! Y offset in output array
      INTEGER OUT_A_PTR       ! Pointer to axis
      INTEGER OUT_NDF         ! Output NDF identifier
      INTEGER OUT_PTR         ! Pointer to output data (NDF)
      INTEGER OUT_QUAL_PTR    ! Pointer to quality array
      INTEGER OUT_VAR_PTR     ! Pointer to variance array
      CHARACTER*(DAT__SZLOC) OUT_SCUCD_LOC ! Locator to SCUCD extension
      CHARACTER*(DAT__SZLOC) OUT_SCUBA_LOC ! Locator to SCUBA extension
      CHARACTER*(DAT__SZLOC) OUT_REDS_LOC ! Locator to REDS extension
      CHARACTER*(DAT__SZLOC) OUT_FIG_LOC ! Locator to FIGARO extension
      CHARACTER*(DAT__SZLOC) OUT_FITS_LOC ! Locator to FITS extension
      DOUBLE PRECISION RA     ! Right ascension of source
      INTEGER RECIN           ! Current input record
      INTEGER RECORD          ! Record number in input file
      INTEGER RECSS           ! Record size in words
      CHARACTER *20 RUNNO     ! Run number in 0 padded form (eg 0015)
      CHARACTER * 80 STEMP    ! Scratch string
      INTEGER UBND ( 2 )      ! Upper bounds of output NDF

*-

      IF (STATUS .NE. SAI__OK) RETURN

*     Set the MSG output level (for use with MSG_OUTIF)

      CALL MSG_IFGET('MSG_FILTER', STATUS)


*     Read in the DREAM file (currently assume that
*     user will supply full path or use DREAM_OUT env var.

      CALL RIO_ASSOC('IN', 'READ', 'UNFORMATTED', SRECSIZE, FD,
     :     STATUS)
```

```
*     Since we are using the DREAM I/O routines we need to get
*     the file unit number
      CALL FIO_UNIT(FD, LUN, STATUS)

*     Read header into common block

      IF (STATUS .EQ. SAI__OK) THEN
         RECORD = 1             ! Record number 1
         RECSS = SRECSIZE / DREAM__NBYTES ! Output record size in words

         CALL DISK_IO (2, LUN, RECSS, RECORD, SOLPA, HEADER, ERR)

         IF (ERR .NE. 0) THEN
            STATUS = SAI__ERROR
            CALL MSG_SETC('TSK', TSKNAME)
            CALL MSG_SETI('ERR', ERR)
            CALL MSG_SETI('REC', RECORD)
            CALL ERR_REP(' ','^TSK: Error ^ERR in reading record'//
     :           ' ^REC', STATUS)
         END IF
      END IF

*     Ask for the first cycle to select
      CALL PAR_GDR0I('FSCYCLE', 1, 1, r_Ncycle, .FALSE., FSCYCLE,
     :     STATUS)

*     Ask for the number of cycles to be selected
      MAXCYC = r_Ncycle - FSCYCLE + 1
      CALL PAR_GDR0I('NRCYCLE', MAXCYC, 1, MAXCYC, .FALSE.,
     :     NRCYCLE, STATUS)

*     Now we can start doing things.
*     Calculate total number of pixel values per cycle

      NSVAL = r_Nbol * MAX_PIX

      IF (NSVAL .LE. 0) THEN

         IF (STATUS .EQ. SAI__OK) THEN
            STATUS = SAI__ERROR
            CALL MSG_SETC('TSK', TSKNAME)
            CALL MSG_SETI('NPT', NSVAL)
            CALL ERR_REP(' ','^TSK: Number of observed points'//
     :           'is too small (^NPT)', STATUS)
         END IF
      END IF

*     Start NDF
      CALL NDF_BEGIN

*     Open an NDF file
*     Default file name constructed from the UT date

*     Year
```

```
      CALL CHR_ITOC(GR_YY, STEMP, ITEMP)
      IPOSN = CHR_LEN(STEMP)

*     Month (pad with leading zero)
      IF (GR_MN .LT. 10) CALL CHR_APPND('0', STEMP, IPOSN)
      CALL CHR_ITOC(GR_MN, CTEMP, ITEMP)
      CALL CHR_APPND(CTEMP, STEMP, IPOSN)

*     Day
      IF (GR_DD .LT. 10) CALL CHR_APPND('0', STEMP, IPOSN)
      CALL CHR_ITOC(GR_DD, CTEMP, ITEMP)
      CALL CHR_APPND(CTEMP,STEMP,IPOSN)

*     Find observation number from filename
*     Assumes string of form xxx_NNNN

      ITEMP = CHR_LEN ( SOLVE_DATA )
      RUNNO = SOLVE_DATA(ITEMP-3:)

*     Append an _
      CALL CHR_APPND('_', STEMP, IPOSN)

*     Append run number to out
      CALL CHR_APPND(RUNNO, STEMP, IPOSN)

*     Append DREAM signature
      CALL CHR_APPND('_drm', STEMP, IPOSN)


*     Set default
      CALL PAR_DEFOC('OUT', STEMP, STATUS)

*     Bounds of NDF
      LBND(1) = 1
      LBND(2) = 1
      UBND(1) = R_NBOL
      UBND(2) = NPIX * NRCYCLE

      CALL NDF_CREAT('OUT', '_REAL', 2, LBND, UBND, OUT_NDF, STATUS)

*     Map the output data array (plus the other arrays for SURF
*     compatibility)
      CALL NDF_MAP(OUT_NDF, 'QUALITY','_UBYTE', 'WRITE/ZERO',
     :     OUT_QUAL_PTR, ITEMP, STATUS)
      CALL NDF_MAP(OUT_NDF, 'DATA', '_REAL', 'WRITE', OUT_PTR,
     :     ITEMP, STATUS)
      CALL NDF_MAP(OUT_NDF, 'VARIANCE', '_REAL', 'WRITE/ZERO',
     :     OUT_VAR_PTR, ITEMP, STATUS)


*     Get some memory to store the data from each cycle
*     We are using 4 byte words but define in parameter

      NBYTES = MAX(DREAM__NBYTES * NSVAL, SRECSIZE)
```

```
      CALL PSX_MALLOC(NBYTES, DREAM_PTR, STATUS)

*     Loop over cycles

*     Calculate number of records per cycle
      NREC = (NSVAL + RECSS - 1 ) / RECSS

*     Loop
      DO I = FSCYCLE, NRCYCLE

         IF (STATUS .EQ. SAI__OK) THEN

*     Determine inpur record number
            RECIN = HEAD_S / RECSS + (I * NREC)

*     Read pixel intensities into memory
            CALL DISK_IO (2, LUN, RECSS, RECIN, %VAL(DREAM_PTR),
     :           NSVAL, ERR)

            IF (ERR .NE. 0) THEN
               print *,'in if ',ERR, STATUS
               STATUS = SAI__ERROR
               CALL MSG_SETC('TSK', TSKNAME)
               CALL MSG_SETI('ERR', ERR)
               CALL MSG_SETI('REC', RECIN)
               CALL ERR_REP(' ','^TSK: Error ^ERR in reading record'//
     :              ' ^REC', STATUS)
            END IF

*     Calculate current offset position in output array (time axis)
            OFFSET = (I - FSCYCLE) * NPIX + 1

*     Now we need to write this data to the NDF file
            CALL DREAM_DATA_TO_SURF_DATA(R_NBOL, MAX_PIX,
     :           NPIX * NRCYCLE,
     :           OFFSET, INT_POS, %VAL(DREAM_PTR), %VAL(OUT_PTR),
     :           STATUS)

         END IF

      END DO

*     Free the memory assoicated with each cycle
      CALL PSX_FREE(DREAM_PTR, STATUS)

*     Close the input file
      CALL RIO_CLOSE(FD, STATUS)

*     Unmap the data arrays
      CALL NDF_UNMAP(OUT_NDF, '*', STATUS)

*     Axis information is simply integration number.
*     Steal this from REDUCE_SWITCH
*
```

```
*     Axis 1: bolometers   2: Data

*     Deal with BOLOMETER axis

      CALL NDF_AMAP(OUT_NDF, 'CENTRE', 1, '_INTEGER', 'WRITE',
     :     OUT_A_PTR, ITEMP, STATUS)
      IF (STATUS .EQ. SAI__OK) THEN
         CALL SCULIB_NFILLI (R_NBOL, %val(OUT_A_PTR))
      END IF
      CALL NDF_ACPUT ('Bolometer', OUT_NDF, 'LABEL', 1, STATUS)
      CALL NDF_AUNMP (OUT_NDF, 'CENTRE', 1, STATUS)

*     Integrations
      CALL NDF_AMAP (OUT_NDF, 'CENTRE', 2, '_REAL', 'WRITE',
     :     OUT_A_PTR, ITEMP, STATUS)

      IF (STATUS .EQ. SAI__OK) THEN
         CALL SCULIB_NFILLR (ITEMP, %val(OUT_A_PTR))
         CALL SCULIB_MULCAR(ITEMP, %VAL(OUT_A_PTR), 1.0/REAL(NPIX),
     :        %VAL(OUT_A_PTR))
      END IF

      CALL NDF_ACPUT ('Integration', OUT_NDF, 'LABEL', 2, STATUS)
      CALL NDF_AUNMP (OUT_NDF, 'CENTRE', 2, STATUS)


*     Now we can start writing header information to the file

*     SCUCD extension
      CALL NDF_XNEW(OUT_NDF, 'SCUCD', 'SCUCD_ST', 0, 0,
     :     OUT_SCUCD_LOC, STATUS)

*     SCUBA extension
      CALL NDF_XNEW(OUT_NDF, 'SCUBA', 'SCUBA_ST', 0, 0,
     :     OUT_SCUBA_LOC, STATUS)

*     REDS extension
      CALL NDF_XNEW(OUT_NDF, 'REDS', 'SURF_EXTENSION', 0, 0,
     :     OUT_REDS_LOC, STATUS)

*     FIGARO extension (for completeness)
      CALL NDF_XNEW(OUT_NDF, 'FIGARO', 'FIGARO_EXT', 0, 0,
     :     OUT_FIG_LOC, STATUS)
      CALL DAT_ANNUL(OUT_FIG_LOC, STATUS)

*     First write DEM_PNTR array
*     Create the component
*     Note that DEM_PNTR is always 1 dimensional in this
*     since there are no switches, exposures or measurements.
*     Only has 3 dimensions since there are no switches

      DEMBNDS ( 1 ) = 1
      DEMBNDS ( 2 ) = NRCYCLE
      DEMBNDS ( 3 ) = 1
```

```
      CALL CMP_MOD(OUT_SCUBA_LOC, 'DEM_PNTR', '_INTEGER', 3,
     :     DEMBNDS, STATUS)

*     Map it (rather do this as otherwise I need to create
*     the array on the fly
      CALL CMP_MAPV(OUT_SCUBA_LOC, 'DEM_PNTR', '_INTEGER',
     :     'WRITE', DEM_PNTR, ITEMP, STATUS)

*     Loop over cycles again
      DO I = 1, NRCYCLE
         OFFSET = (I-1) * NPIX + 1
         CALL VEC_ITOI(.FALSE., 1, OFFSET,
     :        %VAL(DEM_PNTR + ((I-1) * VAL__NBI)), IERR, NERR,
     :        STATUS)
      END DO

*     Unmap DEM_PNTR
      CALL CMP_UNMAP(OUT_SCUBA_LOC, 'DEM_PNTR', STATUS)

*     Write LST information (Same dimensions as DEM_PNTR)
      DEMBNDS ( 1 ) = 1
      DEMBNDS ( 2 ) = 1
      DEMBNDS ( 3 ) = NRCYCLE
      DEMBNDS ( 4 ) = 1
      CALL CMP_MOD(OUT_SCUCD_LOC, 'LST_STRT', '_DOUBLE', 4,
     :     DEMBNDS, STATUS)

*     Map it (rather do this as otherwise I need to create
*     the array on the fly (re-use DEM_PNTR variable)
      CALL CMP_MAPV(OUT_SCUCD_LOC, 'LST_STRT', '_DOUBLE',
     :     'WRITE', DEM_PNTR, ITEMP, STATUS)

*     This is based on STIME
*     Returns LST in radians and the MJD
*     Dont know why John chose degrees east of meridian
*     convert back to degrees west.
      LONGITUDE =  LONG_OBS - 360.0D0

      CALL LST_FROM_UT(GR_YY, GR_MN, GR_DD, UT_HH, UT_MN, UT_SS,
     :     LONGITUDE, LST_STRT, MJD, STATUS)

*     increment in LST per cycle
*     I think this is stored in cycle_t and is in millisec in the header
*     Convert to radians
      DLST = (CYCLE_T / (1000.0D0 * 3600.0D0)) * 15.0D0 * PI / 180.0D0

*     Loop over cycles
      DO I = FSCYCLE, NRCYCLE
         LST = LST_STRT + (DBLE(I-1) * DLST)

         CALL VEC_DTOD(.FALSE., 1, LST,
     :        %VAL(DEM_PNTR + (I-FSCYCLE) * VAL__NBD), IERR, NERR,
     :        STATUS)
```

```
      END DO

*     Unmap LST_STRT
      CALL CMP_UNMAP(OUT_SCUCD_LOC, 'LST_STRT', STATUS)

*     Now deal with jiggle patterns
*     Create the JIGL_X and JIGL_Y components
      CALL CMP_MOD(OUT_SCUCD_LOC, 'JIGL_X', '_REAL', 1,
     :     NPIX, STATUS)
      CALL CMP_MOD(OUT_SCUCD_LOC, 'JIGL_Y', '_REAL', 1,
     :     NPIX, STATUS)

*     Now loop over jiggle positions and store the relevant
*     ones (> -1)

*     SURF requires that bolometer coordinates are derived by
*          Xpos = Bol_Xpos - Jigl_X
*          Ypos = Bol_Ypos - Jigl_Y
*
*     DREAM currently assumes
*          Xpos = -Bol_Xpos + Jigl_X
*          Ypos =  Bol_Ypos - Jigl_Y

*     I end up doing this
*          Invert x positions in flatfield (nasty)
*          Negate jiggle position for X and Y

      ITEMP = 0
      DO I = 0, MAX_PIX  - 1
         IF (INT_POS(I) .NE. - 1) THEN
            ITEMP = ITEMP + 1
            JIGL_X(ITEMP) = JIG_X(I)
            JIGL_Y(ITEMP) = JIG_Y(I)
         END IF
      END DO


*     Write the jiggle pattern to the extensions
      CALL CMP_PUT1R(OUT_SCUCD_LOC, 'JIGL_X', NPIX, JIGL_X, STATUS)
      CALL CMP_PUT1R(OUT_SCUCD_LOC, 'JIGL_Y', NPIX, JIGL_Y, STATUS)


*     Now work out the order in which the bolometers are stored.

*     Create the BOL_CHAN and BOL_ADC extensions
      CALL CMP_MOD(OUT_SCUBA_LOC, 'BOL_CHAN', '_INTEGER', 1,
     :     R_NBOL, STATUS)
      CALL CMP_MOD(OUT_SCUBA_LOC, 'BOL_ADC', '_INTEGER', 1,
     :     R_NBOL, STATUS)


*     Need to loop over input bolometers and decode the name
*     DREAM stores the storage order in BOL_ORDER and the
*     name in BOL_NAME
```

```
      DO I = 0, R_NBOL - 1

*     Determine the ADC and Channel number from the name
         CALL SCULIB_BOLDECODE ( BOL_NAME(I), ADC, CHAN, STATUS)

*     Now put these values into the correct slot of BOL_CHAN and BOL_ADC
*     (Am I supposed to use BOL_ORDER?)
         BOL_CHAN ( I + 1) = CHAN
         BOL_ADC ( I + 1) = ADC
*         BOL_CHAN ( BOL_ORDER(I) + 1) = CHAN
*         BOL_ADC ( BOL_ORDER(I) + 1) = ADC

      END DO

*     Write the bolometer order to the extensions
      CALL CMP_PUT1I(OUT_SCUBA_LOC, 'BOL_ADC', R_NBOL, BOL_ADC, STATUS)
      CALL CMP_PUT1I(OUT_SCUBA_LOC, 'BOL_CHAN', R_NBOL, BOL_CHAN,STATUS)

*     Read flatfield from a text file
      CALL PAR_GETOC ('FLATFILE', FLATFILE, STATUS)

      CALL SCULIB_READBOLS (FLATFILE, SCUBA__NUM_CHAN,
     :     SCUBA__NUM_ADC, BOL_TYPE, BOL_DU3, BOL_DU4, BOL_CALB,
     :     BOL_RTEMP, BOL_RTEMP, BOL_RTEMP, BOL_QUAL, BOL_DAY,
     :     BOL_RUN, BOL_REF, STATUS)

*     Now write the flatfield information to the extensions
      NDIM = 2
      DIM (1) = SCUBA__NUM_CHAN
      DIM (2) = SCUBA__NUM_ADC

*     Create the extensions
      CALL CMP_MODC(OUT_SCUBA_LOC, 'BOL_TYPE', 20, NDIM, DIM, STATUS)
      CALL CMP_MOD(OUT_SCUBA_LOC, 'BOL_DU3', '_REAL', NDIM, DIM,
     :     STATUS)
      CALL CMP_MOD(OUT_SCUBA_LOC, 'BOL_DU4', '_REAL', NDIM, DIM,
     :     STATUS)

*     Only really interested in the BOL_TYPE and DU3, DU4 arrays
*     Write the data
      CALL CMP_PUTNC (OUT_SCUBA_LOC, 'BOL_TYPE', NDIM, DIM, BOL_TYPE,
     :     DIM, STATUS)
      CALL CMP_PUTNR (OUT_SCUBA_LOC, 'BOL_DU3', NDIM, DIM, BOL_DU3,
     :     DIM, STATUS)
      CALL CMP_PUTNR (OUT_SCUBA_LOC, 'BOL_DU4', NDIM, DIM, BOL_DU4,
     :     DIM, STATUS)



*     FITS EXTENSION -------------------
      N_FITS = 0  ! No FITS components to start with

*     Object name (default to something if nothing in header)
```

```
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'OBJECT', 'DREAM', 'Name of object', STATUS)

*     RUN number
      CALL CHR_CTOI(RUNNO, ITEMP, STATUS)
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'RUN', ITEMP, 'Run number of observation', STATUS)

*     Observation, Sample mode (Always MAP/JIGGLE) and sample coords
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'MODE', 'MAP', 'The type of observation', STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SAM_MODE', 'JIGGLE', 'Sampling method', STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SAM_CRDS', 'NA', 'Coordinate system of sampling mesh',
     :     STATUS)
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SAM_PA', -1, 'Scan P.A. rel. to lat. line; 0=lat, 90=long',
     :     STATUS)

*     Coordinates of observation

*     Declination
*     Convert to string D:M:S
      IF (DECSN .GE. 0.0D0) THEN
         STEMP = '+'
      ELSE
         STEMP = '-'
      END IF
      IPOSN = 1

      DEC = DBLE(DECDD) + (DBLE(DECMN)/60.0D0) + (DBLE(DECSS)/3600.0D0)
      CALL CHR_RTOAN(REAL(DEC), 'DEGREES', STEMP, IPOSN)

*     Store in LAT
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'LAT', STEMP, 'Object Latitude', STATUS)

*     Store in token
      CALL MSG_SETC('DEC', STEMP)

*     Determine Right ascension
      RA = DBLE(RAHH) + (DBLE(RAMN)/60.0D0) + (DBLE(RASS)/3600.0D0)
      IPOSN = 0
      CALL CHR_RTOAN(REAL(RA), 'HOURS', STEMP, IPOSN)

*     Store in LONG
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'LONG', STEMP, 'Object longitude', STATUS)

*     Store in message token
      CALL MSG_SETC('RA', STEMP)

*     Write coordinates to display
```

```
      CALL MSG_SETC('TSK',TSKNAME)
      CALL MSG_OUTIF(MSG__NORM, ' ', '^TSK: Coordinates: '//
     :     '^RA, ^DEC', STATUS)


*     Need to ask for coordinate frame of the tracking centre
      CALL PAR_CHOIC('COORDS', 'RB', 'AZ,RD,RB,GA,RJ', .TRUE.,
     :     CENT_CRD, STATUS)

      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CENT_CRD',CENT_CRD,'Centre coordinate system', STATUS)



*     UT Date.
*     Convert Year
      CALL CHR_ITOC(GR_YY, STEMP, ITEMP)
      IPOSN = CHR_LEN(STEMP)
      CALL CHR_APPND(':',STEMP, IPOSN)

*     Convert month
      CALL CHR_ITOC(GR_MN, CTEMP, ITEMP)
      CALL CHR_APPND(CTEMP, STEMP, IPOSN)
      CALL CHR_APPND(':',STEMP, IPOSN)

*     Convert day
      CALL CHR_ITOC(GR_DD, CTEMP, ITEMP)
      CALL CHR_APPND(CTEMP, STEMP, IPOSN)

*     Write the MJD
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'MJD-OBS', MJD, 'Modified Julian Date of obsstart',
     :     STATUS)

*     Write the date
*      STEMP = '1998:1:1'
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'UTDATE', STEMP, 'UT Date of observation', STATUS)



*     Write the UT time
*     Convert hour
      CALL CHR_ITOC(UT_HH, STEMP, ITEMP)
      IPOSN = CHR_LEN(STEMP)
      CALL CHR_APPND(':',STEMP, IPOSN)

*     Convert minute
      CALL CHR_ITOC(UT_MN, CTEMP, ITEMP)
      CALL CHR_APPND(CTEMP, STEMP, IPOSN)
      CALL CHR_APPND(':',STEMP, IPOSN)

*     Convert second
      CALL CHR_ITOC(UT_SS, CTEMP, ITEMP)
```

```
      CALL CHR_APPND(CTEMP, STEMP, IPOSN)

      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'UTSTART', STEMP, 'UT at start of observation', STATUS)

*     Convert LST (Hours minutes seconds) into a time
*     First convert LST_STRT to hours (should be radians to this point)
      LST_STRT = LST_STRT * 180.0D0 / (PI * 15.0D0)

      IPOSN = 0
      CALL CHR_DTOAN(LST_STRT, 'HOURS', STEMP, IPOSN)

*     and write it out
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'STSTART', STEMP, 'ST at start of observation', STATUS)


*     Also store STEND
      IPOSN = 0

      DTEMP = LST_STRT + (DLST * DBLE(FSCYCLE + NRCYCLE - 1)
     :     * 180.0D0 / (PI * 15.0D0))
      CALL CHR_DTOAN(DTEMP,
     :     'HOURS',STEMP, IPOSN)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'STEND', STEMP, 'ST at start of observation', STATUS)

*     exposure time per sample
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'EXP_TIME', DBLE(CYCLE_T / (1000.0 * REAL(NPIX))),
     :     'Exposure time for each basic measurement (sec)',
     :     STATUS)


*     Set up MAP_X and MAP_Y offsets
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :      'MAP_X', 0.0,
     :      'Map X offset from telescope centre (arcsec)',
     :      STATUS)
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :      'MAP_Y', 0.0,
     :      'Map Y offset from telescope centre (arcsec)',
     :      STATUS)

*     Number of bolometers
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :      'N_BOLS', R_NBOL,
     :      'Number of bolometers selected', STATUS)

*     State of the observation (anything except ABORT)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'STATE','Terminating', 'SCUCD state', STATUS)

*     Version of SCUCD (set to 0 for now)
```

```
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'VERSION', 0, 'SCUCD version (DREAM data)',STATUS)

*     Jiggle info
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'JIGL_CNT', NPIX, 'Number of offsets in a jiggle pattern',
     :     STATUS)
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'J_PER_S', NPIX, 'Number of jiggles per switch',
     :     STATUS)
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'J_REPEAT', NPIX, 'No. of jiggle pattern repeats in switch',
     :     STATUS)

*     Center of array
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CNTR_DU3', 0.0D0, 'Nasmyth dU3 coord of instrument centre',
     :     STATUS)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CNTR_DU4', 0.0D0, 'Nasmyth dU4 coord of instrument centre',
     :     STATUS)

*     Position of the telescope
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'LAT-OBS', LAT_OBS, 'Latitude of observatory (degrees)',
     :     STATUS)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'LONG-OBS', LONG_OBS,
     :     'East Longitude of observatory (degrees)',
     :     STATUS)

*     Need to supply some chop information (meaningless)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CHOP_THR', 0.0D0, 'Chopper throw (arcsec)',
     :     STATUS)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CHOP_PA', 0.0D0, 'Chopper P.A., 0 = in lat, 90 = in long',
     :     STATUS)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CHOP_FRQ', SMU_F, 'Chopper frequency (Hz)',
     :     STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CHOP_CRD', 'NA', 'Chopper coordinate system',
     :     STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'CHOP_FUN', 'DREAM', 'Chopper waveform',
     :     STATUS)



*     Sub instrument information
*     Assume that we have one sub-instrument and that it is LONG
      CALL SCULIB_PUT_FITS_I(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'N_SUBS', 1, 'Number of sub-instruments used',
```

```
     :     STATUS)

      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SUB_1','LONG', 'SCUBA sub-instrument being used', STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SUB_2','not used', 'SCUBA sub-instrument being used',STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SUB_3','not used', 'SCUBA sub-instrument being used',STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SUB_4','not used', 'SCUBA sub-instrument being used',STATUS)
      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'SUB_5','not used', 'SCUBA sub-instrument being used',STATUS)

      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'TAUZ_1',0.0D0, 'Zenith sky optical depth',STATUS)

*     FILTERS and wavelength

      CALL SCULIB_PUT_FITS_C(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'FILT_1','850', 'Filter name', STATUS)
      CALL SCULIB_PUT_FITS_D(SCUBA__MAX_FITS, N_FITS, FITS,
     :     'WAVE_1',862.0D0, 'Wavelength of map (microns)', STATUS)



*     Write FITS component
      CALL NDF_XNEW (OUT_NDF, 'FITS', '_CHAR*80', 1, N_FITS,
     :     OUT_FITS_LOC, STATUS)
      CALL DAT_PUT1C (OUT_FITS_LOC, N_FITS, FITS, STATUS)

*     Annul extension locators
      CALL DAT_ANNUL(OUT_SCUCD_LOC, STATUS)
      CALL DAT_ANNUL(OUT_SCUBA_LOC, STATUS)
      CALL DAT_ANNUL(OUT_REDS_LOC, STATUS)
      CALL DAT_ANNUL(OUT_FITS_LOC, STATUS)

*     Write the HISTORY information. (DREAM info will be written
*     automatically when NDF is closed
      CALL NDF_HCRE(OUT_NDF, STATUS)

*     Close the NDF and shut down the NDF system (write DREAM history)
      CALL NDF_ANNUL(OUT_NDF, STATUS)

*     It seems that the only way to write multiple history
*     entries is to open and close the NDF multiple times!

*     REDUCE_SWITCH
*     Re-open the file
      CALL NDF_ASSOC('OUT', 'UPDATE', OUT_NDF, STATUS)

*     Need to write REDUCE_SWITCH and FLATFIELD tags to fool SURF
*     into thinking that the data have been processed by these tasks
      CALL NDF_HPUT(' ', 'REDUCE_SWITCH', .TRUE., 1,
     :     'This is a dummy history component to fool SURF',
```

```
     :     .FALSE., .TRUE., .FALSE., OUT_NDF, STATUS)

*     Close the NDF
      CALL NDF_ANNUL(OUT_NDF, STATUS)

*     FLATFIELD
*     Re-open the file
      CALL NDF_ASSOC('OUT', 'UPDATE', OUT_NDF, STATUS)


      CALL NDF_HPUT(' ', 'FLATFIELD', .TRUE., 1,
     :     'This is a dummy history component to fool SURF',
     :     .FALSE., .TRUE., .FALSE., OUT_NDF, STATUS)

*     Close the NDF
      CALL NDF_ANNUL(OUT_NDF, STATUS)


*     Shut down NDF system
      CALL NDF_END(STATUS)

      END
```

G Library APIs

This section lists the full API documentation for the SURF libraries.

G.1 SURF

These are the top-level SURF routines that are not simply task files. These are routines for disk I/O, the top level A-task interface routine and specialised code for calcsky and despike (since they are children of the general purpose rebin task interface).

SURF_GRID_CALCSKY
Calculate sky contribution from median image

Description:

This routine calculates the sky contribution by attempting to remove the source from the input data stream. The source signal can either be calculated by this routine or by reading in a model of the source from a file.

When calculating the source structure internally a similar method to that used by DESPIKE is employed. The input data are placed into bins of size one quarter beamwidth. The median of each bin is calculated and this is treated as the source model (cf. REBIN_METHOD=MEDIAN in REBIN).

Once the source model is available, it is removed from all of the input data. The source-removed data are then analysed with the sky emission derived from the mean of the signal across the array for all the sample times.

Since the sky signal is expected to vary on timesales of the order of one second, an option is included for smoothing the sky signal. This is especially useful for scan map data where samples are taken at 7.8 Hz.

Invocation:

```
CALL SURF_GRID_CALCSKY ( TSKNAME, N_FILES, N_PTS, N_POS, N_BOLS, N_BOLS, WAVELENGTH,
DIAMETER, IMNDF, N_M_FITS, MODEL_FITS, CHOP_THROW, CHOP_PA, BOX_SIZE, BOL_RA_PTR,
BOL_DEC_PTR, DATA_PTR, QUALITY_PTR, SKY_PTR, SKY_ERR, BADBIT, STATUS )
```

Arguments:

TSKNAME = CHARACTER (Given)
Name of task to be used in output messages

N_FILES = INTEGER (Given)
Number of data sets (ie files)

N_PTS (N_FILES) = INTEGER (Given)
Total number of points in each map

N_POS(N_FILES) = INTEGER (Given)
Number of positions per set (Y positions)

N_BOLS(N_FILES) = INTEGER (Given)
Number of bolometers per set (X positions)

WAVELENGTH = REAL (Given)
Wavelength of observation in microns

DIAMETER = REAL (Given)
Diameter of telescope in metres

IMNDF = INTEGER (Given)
NDF identifier of the supplied model. If this is equal to NDF__NOID then no external model is used.

N_M_FITS = INTEGER (Given)
Number of FITS keywords in the model

MODEL_FITS = CHAR∗(80) (Given)
FITS keywords from the model

CHOP_THROW = REAL (Given)
Size of chop throw for the first input image (arcsec)

CHOP_PA = REAL (Given)
Chop position angle of first input image (degrees)

BOX_SIZE (N_FILES) = INTEGER (Given)
Size of smoothing box - pixels

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (X coords) Note that each data set has positions for N_POS ∗ N_BOLS

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (Y coords)

DATA_PTR(N_FILES) = INTEGER (Given, modified data)
Pointers to actual data arrays. The arrays are modified and contain the source removed data on exit.

QUALITY_PTR(N_FILES) = INTEGER (Given)
Pointer to quality arrays

SKY_PTR (N_FILES) = INTEGER (Pointer Given, data returned)
Storage space for the sky signal

SKY_ERR (N_FILES) = INTEGER (Pointer Given, data returned)
Storage space for the error on sky signal

BADBIT (N_FILES) = BYTE (Given)
Bad bit mask for identifying bad pixels from quality

STATUS = INTEGER (Given & Returned)
Global Status

Examples:

```
calcsky test_rlb model=!  \\
```

Calculate sky for test_rlb.sdf. Only read in one file and don't use an external source model.

```
calcsky list.inp model=m82 noloop\\
```

Read in the files specified in list.inp and use m82.sdf as a model of the source.

```
calcsky file nosrc=nosrc boxsz=10.0 \\
```

Calculate sky for file.sdf. Store the source subtracted image in nosrc.sdf. Use a smoothing size of 10 seconds.

Notes:

- The model itself is only an approximation to the data (since the data points can fall anywhere within a given cell) so some source signal will remain after source subtraction.
- If a model is supplied externally (via MODEL parameter) the cell size and the map centre of the model is used for the source subtraction.
- The sky signal is stored in an NDF extension (.MORE.REDS.SKY). The file must be processed by REMSKY to actually remove the sky contribution.

Related Applications :

SURF: REMSKY

Copyright :

SURF_GRID_DESPIKE
Despike data by sorting into a given grid position

Description:

For each data point this routine places it into a bin in the output grid depending on the position of the data point on the sky. The position in the input data array is stored. This is done in two stages:

- Find the size of the output grid from the maximum extent of the input data.
- Loop through data. Find I,J coordinate of each point in the output grid.
- Find out maximum number of points for an I,J position.
- Put data onto grid in array (I,J,N) [REALS]. We also need to store positions of these data. We can either do it by storing the file number, bolometer and position (time) index OR we can just store some index in a merged data array that goes from 1..TOT_PTS. First method is easy but memory hungry. Second method is more efficient but does need some reconstruction to work out where the point was in the original data. Use the second method.

Once the data is gridded, it is first displayed and then despiked. Currently despiking is done on a simple sigma clipping basis for each bin.

Invocation:

```
CALL SURF_GRID_DESPIKE ( N_FILES, N_PTS, N_POS, N_BOLS, WAVELENGTH, DIAMETER,
BOL_RA_PTR, BOL_DEC_PTR, DATA_PTR, QUALITY_PTR, NX, NY, ICEN, JCEN, NSPIKES, BADBIT,
STATUS )
```

Arguments:

N_FILES = INTEGER (Given)
Number of data sets (ie files)

N_PTS (N_FILES) = INTEGER (Given)
Total number of points in each map

N_POS(N_FILES) = INTEGER (Given)
Number of positions per set (Y positions)

N_BOLS(N_FILES) = INTEGER (Given)
Number of bolometers per set (X positions)

BITNUM = INTEGER (Given)
Bit number to be affected by this routine

WAVELENGTH = REAL (Given)
Wavelength of data (microns)

DIAMETER = REAL (Given)
Diameter of telescope (metres)

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (X coords) Note that each data set has positions for N_POS ∗ N_BOLS

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (Y coords)

DATA_PTR(N_FILES) = INTEGER (Given)
Pointers to actual data arrays

QUALITY_PTR(N_FILES) = INTEGER (Given)
Pointer to quality arrays

NX = INTEGER (Returned)
Number of points in grid (X)

NY = INTEGER (Returned)
Number of points in grid (Y)

ICEN = INTEGER (Returned)
Reference pixel (X)

JCEN = INTEGER (Returned)
Reference pixel (Y)

NSPIKES (N_FILES) = INTEGER (Returned)
Number of spikes detected (and removed) in each file

BADBIT (N_FILES) = BYTE (Given)
Bad bit mask for identifying bad pixels from quality

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

For SMODE=NONE, DMODE is only requested if a plot is required.

Copyright :

SURF_MON
main routine for SCUBA offline data reduction package

Description:

This is the main routine for the SCUBA reduction A-task.

Invocation:

```
CALL SURF_MON( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status

Notes:

This routine is not seen by the user

Copyright :

SURF_READ_REBIN_NDF
Read an NDF into memory prior to regridding

Description:

This routines reads all necessary information from an NDF (via the given NDF identifier) for rebinning. Optionally, returns the quality array (needed for despiking).

Invocation:

```
CALL SURF_READ_REBIN_NDF( IN_NDF, MAX_FILE, NSPEC, DATA_SPEC, OUT_COORDS, N_FILE,
USE_SECTION, N_BOL, N_POS, N_INTS, N_BEAMS, MJD_STANDARD, IN_UT1, OUT_RA_CEN,
OUT_DEC_CEN, FITS, N_FITS, WAVELENGTH, SUB_INSTRUMENT, SOBJECT, SUTDATE, SUTSTART,
BOL_ADC, BOL_CHAN, BOL_RA_PTR, BOL_RA_END, BOL_DEC_PTR, BOL_DEC_END, DATA_PTR,
DATA_END, VARIANCE_PTR, VARIANCE_END, QMF, QUALITY_PTR, QUALITY_END, BADBITS,
USE_LST, LST_PTR, ANG_INT, ANG_MEAS, INT_LIST, MEAS_LIST, BOLWT, STATUS)
```

Arguments:

IN_NDF = INTEGER (Given)
NDF identifier of input NDF

MAX_FILE = INTEGER (Given)
Max number of files allowed [used for INT_LIST/MEAS only]

NSPEC = INTEGER (Given)
Number of SCUBA sections in DATA_SPEC

DATA_SPEC(NSPEC) = CHAR (Given)
SCUBA sections

OUT_COORDS = CHAR (Given)
Output coordinates system. (Passed into SURF_READ_REBIN_NDF)

N_FILE = INTEGER (Given)
Current file number (less than MAX_FILE and greater than 0).

USE_SECTION = LOGICAL (Given)
Determines whether we are using the section or the invers

N_BOL = INTEGER (Returned)
Number of bolometers associated with this observation

N_POS = INTEGER (Returned)
Number of samples in observation

N_INTS = INTEGER (Returned)
Total Number of integrations in observation (INT∗MEAS)

N_MEAS = INTEGER (Returned)
Number of measurements in observation

N_BEAMS = INTEGER (Given & Returned)
Number of beams requested in the positions arrays. N_BEAMS = 1 will return the middle-beam (standard position) N_BEAMS = 2 will return L and R beams for RASTER and JIGGLE N_BEAMS = 3 will return L, M and R beams (as M,L,R) for JIGGLE data that has been reduce_switched and for SCAN data. N_BEAM is set to 2 if a single switch of a JIGGLE map is requested.

MJD_STANDARD = DOUBLE (Given & Returned)
Modified Julian data of observation. Returned if N_FILE=1.

IN_UT1 = DOUBLE (Returned)
Modified Julian data of observation UT1 at start of an input observation, expressed as modified Julian day

IN_RA_CEN = DOUBLE (Returned)
RA of centre

IN_DEC_CEN = DOUBLE (Returned)
Dec of centre

FITS (SCUBA_MAX_FITS) = CHARACTER ∗80 (Returned)
FITS header entries

N_FITS = INTEGER (Returned)
Number of FITS entries

WAVELENGTH = REAL (Given & Returned)
Wavelength of map

SUB_INSTRUMENT = CHAR (Given & Returned)
Sub instrument of map

SOBJECT = CHAR (Returned)
Name of object

SUTDATE = CHAR (Returned)
UT date of the observation

SUTSTART = CHAR (Returned)
UT time of the observation

BOL_ADC = INTEGER (Returned)
ADC information for bolometers - only used by BOLREBIN

BOL_CHAN = INTEGER (Returned)
Channel information for bolometers - only used by BOLREBIN

BOL_RA_PTR = INTEGER (Returned)
Pointer to RA bolometer positions [Pointer is given, data is returned]

BOL_RA_END = INTEGER (Returned)
Pointer to end of RA bolometer positions

BOL_DEC_PTR = INTEGER (Returned)
Pointer to DEC bolometer positions

BOL_DEC_END = INTEGER (Returned)
Pointer to end of DEC bol positions

DATA_PTR = INTEGER (Returned)
Pointer to data values

DATA_END = INTEGER (Returned)
Pointer to end of data values

VARIANCE_PTR = INTEGER (Returned)
Pointer to variance values

VARIANCE_END = INTEGER (Returned)
Pointer to end of variance values

QMF = LOGICAL (Given)
Flag to decide whether quality is being stored (.FALSE.) or being folded into the data array (.true.). See NDF_SQMF

QUALITY_PTR = INTEGER (Returned)
Pointer to quality array

QUALITY_END = INTEGER (Returned)
Pointer to end of quality array

BADBITS = BYTE (Returned)
Bad bits mask for quality array

USE_LST = LOGICAL (Given)
Governs whether we want an LST array returned

LST_PTR(2) = INTEGER (Returned)
Array of pointers (begin and end) to array of LSTs

ANG_INT(MAX_FILE, SCUBA__MAX_INT,2) = REAL (Returned)
Array containing the polarimetry angles for each integration The 2 dimensions are for WPLATE and ANGROT

ANG_MEAS(MAX_FILE, SCUBA__MAX_MEAS,2) = REAL (Returned)
Array containing the pol angles for each measurement The 2 dimensions are for WPLATE and ANGROT

INT_LIST(MAX_FILE, SCUBA__MAX_INT+1) = INTEGER (Returned)
Position of integrations in each data file

MEAS_LIST(MAX_FILE, SCUBA__MAX_MEAS+1) = INTEGER (Returned)
Position of measurements in each data file

BOLWT (N_BOL) = REAL (Returned)
Relative weights of each bolometer

STATUS = INTEGER (Given and Returned)
The global status

Copyright :

SURF_RECURSE_READ
Allow the recursive read of REBIN text files and NDFs

Description:

This routine takes an input name and reads in all NDFs resulting from this name. SCUBA sections are parsed. Text files are expanded into NDF+parameters. Nested text files are allowed.

Invocation:

```
CALL SURF_RECURSE_READ( RLEV, NAME, MAX_FILE, OUT_COORDS, N_FILE, N_BOL, N_POS,
N_INTS, IN_UT1, IN_RA_CEN, IN_DEC_CEN, FITS, N_FITS, WAVELENGTH, SUB_INSTRUMENT,
OBJECT, UTDATE, UTSTART, FILENAME, BOL_ADC, BOL_CHAN, BOL_RA_PTR, BOL_RA_END,
BOL_DEC_PTR, BOL_DEC_END, DATA_PTR, DATA_END, VARIANCE_PTR, VARIANCE_END, QMF,
QUALITY_PTR, QUALITY_END, QBITS, ANG_INT, ANG_MEAS, INT_LIST, MEAS_LIST, BOLWT,
WEIGHT, SHIFT_DX, SHIFT_DY, NPARS, PARS, STATUS)
```

Arguments:

RLEV = INTEGER (Given)
Recursion level. This is the number of times that the routine has called itself. A value of 1 should be passed in by the external routine.

NAME = CHAR (Given)
Name of input file. This may also contain scuba section specifications. The input file can be a NDF or a text file containing a list of NDFs.

MAX_FILE = INTEGER (Given)
Maximum number of NDFs that can be read by the system.

OUT_COORDS = CHAR (Given)
Output coordinates system. (Passed into SURF_READ_REBIN_NDFS)

N_FILE = INTEGER (Given & Returned)
Current file number (less than MAX_FILE and greater than 0). This counter is incremented when an NDF has been read successfully.

N_BOL(MAX_FILE) = INTEGER (Returned)
Number of bolometers associated with each file

N_POS(MAX_FILE) = INTEGER (Returned)
Number of samples associated with each file

N_INTS(MAX_FILE) = INTEGER (Returned)
Total Number of integrations associated with each file (INT∗MEAS)

N_MEAS(MAX_FILE) = INTEGER (Returned)
Number of measurements associated with each file

IN_UT1(MAX_FILE) = DOUBLE (Returned)
Modified Julian data of observation for each file

IN_RA_CEN(MAX_FILE) = DOUBLE (Returned)
RA of centre for each file

IN_DEC_CEN(MAX_FILE) = DOUBLE (Returned)
Dec of centre for each file

FITS (N_FITS, MAX_FILE) = CHARACTER∗(80) (Returned)
FITS entries for each file

N_FITS = INTEGER (Given)
Size of FITS array for each file

WAVELENGTH = REAL (Given & Returned)
Wavelength of map

SUB_INSTRUMENT = CHAR (Given & Returned)
Sub instrument of map

OBJECT(MAX_FILE) = CHAR (Returned)
Name of object in each file

UTDATE(MAX_FILE) = CHAR (Returned)
UT date of each observation

UTSTART(MAX_FILE) = CHAR (Returned)
UT time of each observation

FILENAME(MAX_FILE) = CHAR (Returned)
Actual filename of each file read.

BOL_ADC ( ) = INTEGER (Returned)
ADC information for bolometers - only used by BOLREBIN

BOL_CHAN ( ) = INTEGER (Returned)
Channel information for bolometers - only used by BOLREBIN

BOL_RA_PTR(MAX_FILE) = INTEGER (Returned)
Array of pointers to RA bolometer positions read from each file

BOL_RA_END(MAX_FILE) = INTEGER (Returned)
Array of pointers to end of RA bolometer positions read from each file

BOL_DEC_PTR(MAX_FILE) = INTEGER (Returned)
Array of pointers to DEC bolometer positions read from each file

BOL_DEC_END(MAX_FILE) = INTEGER (Returned)
Array of pointers to end of DEC bol positions read from each file

DATA_PTR(MAX_FILE) = INTEGER (Returned)
Array of pointers to data values read from each file

DATA_END(MAX_FILE) = INTEGER (Returned)
Array of pointers to end of data values

VARIANCE_PTR(MAX_FILE) = INTEGER (Returned)
Array of pointers to variance values read from each file

VARIANCE_END(MAX_FILE) = INTEGER (Returned)
Array of pointers to end of variance values

QMF = LOGICAL (Given)
Flag to decide whether quality is being stored (.FALSE.) or being folded into the data array (.true.). See NDF_SQMF

QUALITY_PTR(MAX_FILE) = INTEGER (Returned)
Pointer to quality array

QUALITY_END(MAX_FILE) = INTEGER (Returned)
Pointer to end of quality array

QBITS(MAX_FILE) = BYTE (Returned)
Bad bits mask for each file

ANG_INT(MAX_FILE, SCUBA__MAX_INT,2) = REAL (Returned)
Array containing the polarimetry angles for each integration The 2 dimensions are for WPLATE and ANGROT

ANG_MEAS(MAX_FILE, SCUBA__MAX_MEAS,2) = REAL (Returned)
Array containing the pol angles for each measurement The 2 dimensions are for WPLATE and ANGROT

INT_LIST(MAX_FILE, SCUBA__MAX_INT+1) = INTEGER (Returned)
Position of integrations in each data file

MEAS_LIST(MAX_FILE, SCUBA__MAX_MEAS+1) = INTEGER (Returned)
Position of measurements in each data file

BOLWT (max num of bols, MAX_FILE) = REAL (Returned)
Relative Weights of each bolometer for each file

WEIGHT(MAX_FILE) = REAL (Returned)
Weight of each input file

SHIFT_DX(MAX_FILE) = REAL (Returned)
X Shift of each input file

SHIFT_DY(MAX_FILE) = REAL (Returned)
Y Shift of each input file

NPARS = INTEGER (Given)
Number of parameters in PARS array.

PARS(NPARS) = REAL (Given)
Values of input parameters. 1: WEIGHT, 2: SHIFT_DX, 3: SHIFT_DY

STATUS = INTEGER (Given and Returned)
The global status

Notes:

- This subroutine can be called recursively. A limit of 5 recursion levels is imposed. Note that recursion is not approved of in Fortran 77 (mainly because local variables remember their state on entry!).
- Text files with extension .txt are converted to NDFs by the NDF_OPEN command. This is bad - do not use .txt files as batch files.

SURF_REQUEST_OUTPUT_COORDS
Request longitude and latitude of output map from user

Description:

Prompts user for output coordinates of map in the specified coordinate frame. The default values are derived from the supplied apparent RA/Dec centre. The selected centre is returned (in apparent ra/dec).

Invocation:

```
CALL SURF_REQUEST_OUTPUT_COORDS( TASK, PARLONG, PARLAT, OUT_COORDS, LAT_OBS, DEF_RA_CEN,
DEF_DEC_CEN, MJD, HOURS, OUT_RA_CEN, OUT_DEC_CEN, OUT_ROTATION, OUT_LONG, OUT_LAT,
STATUS)
```

Arguments:

TASK = CHARACTER (Given)
Description of task to be used in error messages

PARLONG = CHARACTER (Given)
Name of the parameter used to request the LONGITUDE

PARLAT = CHARACTER (Given)
Name of the parameter used to request the LATITUDE

OUT_COORDS = CHARACTER (Given)
Coordinate frame of output coordinates.

LAT_OBS = DOUBLE (Given)
Latitude of observatory (Radians)

DEF_RA_CEN = DOUBLE (Given)
Apparent RA of the default map centre (radians)

DEF_DEC_CEN = DOUBLE (Given)
Apparent Dec of the default map centre (radians)

MJD = DOUBLE (Given)
Modified Julian date to be used as reference for apparent RA/Dec coordinates

HOURS = LOGICAL (Given)
Flag to decide whether longitude is expressed as hours or degrees

OUT_RA_CEN = DOUBLE (Returned)
Apparent RA of output map centre (radians)

OUT_DEC_CEN = DOUBLE (Returned)
Apparent Dec of output map centre (radians)

OUT_ROTATION = DOUBLE (Returned)
angle between apparent N and N of output coord system (radians)

OUT_LONG = DOUBLE (Returned)
longitude of output map centre in the selected coordinate frame (rad)

OUT_LAT = DOUBLE (Returned)
latitude of output map centre in the selected coordinate frame (rad)

STATUS = INTEGER (Given & Returned)
Global status

Notes:

- The output coordinates are always slightly different from the when defaults are accepted since there is a loss of precision converting the ra/dec to and from a string form.

Copyright :

SURF_SET_APP_NAME
Set application name for NDF history

Description:

This routine sets the application name to be of the form

TASKNAME (PACKAGE VN.n-n)

That is, the taskname (which would be the default history if we did not change it) with the package name and the version number. The version number is specified at compile time.

Invocation:

```
CALL SURF_SET_APP_NAME( TSKNAME, STATUS )
```

Arguments:

TASKNAME = _CHAR (Given)
Name of the task.

STATUS = _INTEGERS (Given & Returned)
Global status

Notes:

- This routine should be compiled using the C pre-processor with a define for the PKG_VERS cpp variable. This variable should include the required quotes (since CPP will not change quoted strings). An example could be: f77 -DPKG_VERS="'5.2-4'" surf_set_app_name.F The V is prepended automatically.

SURF_WRITE_DATA
Write bolometer positions and values to text file

Description:

This routine writes the value, variance and position of each data point to a ASCII file. It is called as part of the EXTRACT_DATA task. The interface is the same as that used in the REBIN task. The data and variance are in volts. The positions are in radians. The data are written out as columns (RA DEC DATA VAR) and subsets can be extracted by using SCUBA sections.

Invocation:

```
CALL SURF_WRITE_DATA( FD, NPTS, IN_DATA, IN_VARIANCE, BOL_RA, BOL_DEC, STATUS
)
```

Arguments:

FD = INTEGER (Given)
Output file descriptor

NPTS = INTEGER (Given)
Number of points in input data

IN_DATA(NPTS) = REAL (Given)
Data value

IN_VARIANCE(NPTS) = REAL (Given)
Variance on data

BOL_RA(NPTS) = DOUBLE (Given)
Bolometer position (radians offset from map centre)

BOL_DEC(NPTS) = DOUBLE (Given)
Bolometer position (radians offset from map centre)

STATUS = INTEGER (Given and Returned)
The global status

Notes:

For each file name that is entered, values for the parameters SELECT_INTS, WEIGHT, SHIFT_DX and SHIFT_DY are requested.

- The application can read in up to 100 separate input datasets.
- No data is returned if the DATA or positions are bad. Data is still returned if Variance is bad.

ASCII input files :

The REF and IN parameters accept ASCII text files as input. These text files may contain comments (signified by a #), NDF names, values for the parameters WEIGHT, SHIFT_DX

and SHIFT_DY, and names of other ASCII files. There is one data file per line. An example file is:

```
file1{b5}   1.0  0.5  0.0  # Read bolometer 5 from file1.sdf
file2                     # Read file 2 but you will still be
                          # prompted for WEIGHT, and shifts.
file3{i3}-  1.0  0.0  0.0  # Use everything except int 3
test.bat                  # Read in another text file
```

Note that the parameters are position dependent and are not necessary. Missing parameters are requested. This means it is not possible to specify SHIFT_DX (position 3) without specifying the WEIGHT. Also note that SCUBA sections can be specified with any input NDF.

Related Applications :

SURF: REBIN, BOLREBIN, INTREBIN, CHANGE_QUALITY

Copyright :

SURF_WRITE_MAP_INFO
Calculate FITS and WCS information for rebinned image

Description:

This routine writes Axis, FITS and WCS information to a rebinned map file. Note that this routine does not write WCS information to the FITS header - that is stored in the WCS information. Keywords CRVAL, CRPIX and CTYPE are used here to generate the WCS only.

Invocation:

```
CALL SURF_WRITE_MAP_INFO( OUT_NDF, OUT_COORDS, OUT_TITLE, MJD_STANDARD, NFILE,
FILENAME, OUT_LONG, OUT_LAT, OUT_PIXEL, I_CENTRE, J_CENTRE, NX_OUT, NY_OUT, WAVELENGTH
FILTER, WRITE_CHOP, CHOP_PA, CHOP_THROW, WPLATE, ANGROT, STATUS )
```

Arguments:

OUT_NDF = INTEGER (Given)
NDF identifier of output file

OUT_COORDS = CHAR (Given)
Output coordinate system

OUT_TITLE = CHAR (Given)
Title of map

OUT_UNITS = CHAR (Given)
Output units

MJD_STANDARD = DOUBLE (Given)
Modified Julian date of map

NFILE = INTEGER (Given)
Number of files in input map

FILENAME = CHAR(NFILE) (Given)
Names of input files

OUT_PIXEL = REAL (Given)
Size of pixel in output map (radians)

OUT_LONG = DOUBLE (Given)
Longitude of output map (radians)

OUT_LAT = DOUBLE (Given)
Latitude of output map (radians)

I_CENTRE = INTEGER (Given)
Position of X reference pixel in output map

J_CENTRE = INTEGER (Given)
Position of Y reference pixel in output map

NX_OUT = INTEGER (Given)
Size of output map in X direction

NY_OUT = INTEGER (Given)
Size of output map in Y direction

WAVELENGTH = REAL (Given)
Wavelength of the map (microns)

FILTER = CHARACTER (Given)
Name of filter

WRITE_CHOP = LOGICAL (Given)
Write the CHOP information to the header

CHOP_CRD = CHARACTER (Given)
Chop coordinate frame

CHOP_PA = REAL (Given)
Chop position angle in degrees east of north

CHOP_THROW = REAL (Given)
Chop throw in arcsec

WPLATE = REAL (Given)
Wave plate angle (ignored if bad value).

ANGROT = REAL (Given)
POLPACK rotation angle between waveplate and X pixel axis (ignored if bad value)

TELESCOPE = CHARACTER (Given)
Name of telescope to write to FITS header

INSTRUMENT = CHARACTER (Given)
Name of instrument to write to FITS header

STATUS = INTEGER (Given and Returned)
The global status

Copyright :

SURF_WRITE_PHOTOM
Routine to output ASCII results of PHOTOM reduction

Description:

This routine writes out the results for 1 sub-instrument of a PHOTOM observation. For each bolometer that measured the source:-

```
Bolometer                    :       <bolometer name>

Weight                       :       <weight to be given to its results>


 Integration     Peak     Peak_sig       Peak_x        Peak_y       Quality

<integration>  <peak>    <Error>    <x of peak>  <y of peak>  <quality>

             for all the integrations taken in the observation


Measurement results :

 Fit to coadded jiggle:       :

               <peak>  <variance>  <x of peak>  <y of peak>  <quality>

 Coadded fit results :        :

               <peak>  <variance>                           <quality>
```

Invocation:

```
CALL SURF_WRITE_PHOTOM (FD, MAX_BEAM, N_BOLS, BOL_CHAN, BOL_ADC, PHOT_BB, MAX_INT,
N_MEASUREMENTS, N_INTEGRATIONS,PEAK_D, PEAK_V, PEAK_X, PEAK_Y, PEAK_Q, BEAM_WEIGHT,
MEAS_1_D, MEAS_1_V, MEAS_1_X, MEAS_1_Y, MEAS_1_Q, MEAS_2_D, MEAS_2_V, MEAS_2_Q,
STATUS)
```

Arguments:

FD = INTEGER (Given)
ASCII file descriptor

MAX_BEAM = INTEGER (Given)
the maximum number of bolometers that can observe the source in a single observation

PARABOLA = LOGICAL (Given)
True if we fitted the coaaded jiggle with a parabola

N_BOLS = INTEGER (Given)
the number of bolometers used in the sub-instrument

BOL_CHAN (N_BOLS) = INTEGER (Given)
the channel numbers of the bolometers observing the object

BOL_ADC (N_BOLS) = INTEGER (Given)
the A/D numbers of the bolometers observing the object

PHOT_BB (MAX_BEAM) = INTEGER (Given)
the indices of the bolometers used to observe the source in each beam in the BOL_CHAN and BOL_ADC arrays

MAX_INT = INTEGER (Given)
the maximum number of integrations in an observation

N_MEASUREMENTS = INTEGER (Given)
the number of measurements in the observation

N_INTEGRATIONS = INTEGER (Given)
the number of integrations in the observation

PEAK_D (MAX_INT, MAX_BEAM) = REAL (Given)
the fitted peak value for each integration with each bolometer

PEAK_V (MAX_INT, MAX_BEAM) = REAL (Given)
the variance on PEAK

PEAK_X (MAX_INT, MAX_BEAM) = REAL (Given)
the x offset of the fitted peak for each integration with each bolometer

PEAK_Y (MAX_INT, MAX_BEAM) = REAL (Given)
the y offset of the fitted peak

PEAK_Q (MAX_INT, MAX_BEAM) = BYTE (Given)
the quality of each fitted peak (0 is good)

BEAM_WEIGHT (MAX_BEAM) = REAL (Given)
the weights assigned to the measurements with each bolometer

MEAS_1_D (MAX_BEAM) = REAL (given)
the fitted peak to the coadded integrations

MEAS_1_V (MAX_BEAM) = REAL (Given)
the variance on MEAS_1_D

MEAS_1_X (MAX_BEAM) = REAL (Given)
the x offset of the peak fitted to the coadd

MEAS_1_Y (MAX_BEAM) = REAL (Given)
the y offset

MEAS_1_Q (MAX_BEAM) = BYTE (Given)
the quality on MEAS_1_D

MEAS_2_D (MAX_BEAM) = REAL (Given)
the coadd of the peaks fitted to the individual integrations for each bolometer

MEAS_2_V (MAX_BEAM) = REAL (Given)
the variance on MEAS_2_D

MEAS_2_Q (MAX_BEAM) = BYTE (Given)
the quality on MEAS_2_D

STATUS = INTEGER (Given and returned)
global status

Copyright :

SURF_WRITE_PHOTOM_HEADER
Write photom information to text file

Description:

This routine writes out the header for 1 sub-instrument of a PHOTOM observation. If status is good on entry the routine will call FIO_ASSOC to open the ASCII file to hold the results; the filename will be read from parameter 'FILE'. FIO_WRITE will then be called to write out the results in the following format:-

```
Output from SURF reduction of a PHOTOM observation
Reduction date          :  <date>
Observation definition  :  <ODF name>
Date of observation     :  <date>
Time of observation     :  <time>
Run number              :  <number>
Object                  :  <object>
Sub-instrument          :  <name>
Filter                  :  <name>
Centre coords           :  <coord system of centre>
Latitude                :  <latitude>
Longitude               :  <longitude>
```

If the centre coordinate system is PLANET then:-

```
2nd latitude          :  <latitude on MJD2>
2nd longitude         :  <longitude on MJD2>
date of 1st position  :  <modified Julian day when source at lat,long>
date of 2nd position  :  <modified Julian day when source at lat2,long2>
```

end if

```
Offset coords          :  <coord system of source offset>
x offset               :  <x offset of source>
y offset               :  <y offset of source>
Sampe coords           :  <coord system of jiggle offsets>
Sample position angle  :  <angle that x axis of jiggle offsets is rotated
                          anticlockwise from the x axis of the sample
                          coord system>
Sky error removal      :  <TRUE if the SURF REMSKY application has
                          been run on the data>
Analysis mode          :  AVERAGE or PARABOLA
```

Invocation:

```
CALL SURF_WRITE_PHOTOM_HEADER (ODF, OBS_DATE, OBS_TIME, ANALYSIS, RUN_NUMBER,
OBJECT, SUB_INSTRUMENT, FILTER, CENTRE_COORDS, LAT, LONG, LAT2, LONG2, MJD1, MJD2,
OFFSET_COORDS, MAP_X, MAP_Y, SAMPLE_COORDS, SAMPLE_PA, SKY_SUBTRACTION, FD, STATUS)
```

Arguments:

ODF = CHARACTER∗(∗) (Given)
the name of the observation definition file

OBS_DATE = CHARACTER∗(∗) (Given)
the date of the observation

OBS_TIME = CHARACTER∗(∗) (Given)
the UT of the observation

RUN_NUMBER = INTEGER (Given)
the observation number

OBJECT = CHARACTER∗(∗) (Given)
the name of the object observed

SUB_INSTRUMENT = CHARACTER∗(∗) (Given)
the name of the sub-instrument used

FILTER = CHARACTER∗(∗) (Given)
the name of the filter used

CENTRE_COORDS = CHARACTER∗(∗) (Given)
the coordinate system of the telescope centre coords

LAT = CHARACTER∗(∗) (Given)
the latitude of the telescope centre

LONG = CHARACTER∗(∗) (Given)
the longitude of the telescope centre

LAT2 = CHARACTER∗(∗) (Given)
the second source latitude if CENTRE_COORDS is PLANET

LONG2 = CHARACTER∗(∗) (Given)
the second source longitude if CENTRE_COORDS is PLANET

MJD1 = DOUBLE PRECISION (Given)
the modified Julian date of LAT,LONG if CENTRE_COORDS is PLANET

MJD2 = DOUBLE PRECISION (Given)
the modified Julian date of LAT2,LONG2 if CENTRE_COORDS is PLANET

OFFSET_COORDS = CHARACTER∗(∗) (Given)
the coordinate system of the map centre offsets

MAP_X = REAL (Given)
the x offset of the map centre

MAP_Y = REAL (Given)
the y offset of the map centre

SAMPLE_COORDS = CHARACTER∗(∗) (Given)
the coordinate system of the jiggle offsets

SAMPLE_PA = REAL (Given)
the position angle by which the x-axis of the jiggle offsets is rotated (anti-clockwise) from the x-axis of the SAMPLE_COORDS system

SKY_SUBTRACTION = LOGICAL (Given)
.TRUE. if the sky error has been removed

MAX_BEAM = INTEGER (Given)
the maximum number of bolometers that can observe the source in a single observation

PHOT_BB (MAX_BEAM) = INTEGER (Given)
the indices of the bolometers used to observe the source in each beam in the BOL_CHAN and BOL_ADC arrays

FD = INTEGER (Returned)
File descriptor of output file

STATUS = INTEGER (Given and returned)
global status

Copyright :

G.2 SURFLIB

This section describes the subroutines that are available in the `surflib` distribution (`libsurflib.a`).

SURFLIB_2DFT_CHOP
Calculate the FT of the 2D chop throw

Description:

Given the chop parameters calculate the FT of the chop. The dual beam chop is treated as two delta functions of opposite sign separated by the chop throw and centred at the middle of the data array. The Fourier transform of this function is a sine wave in the direction of the chop (zero at the middle beam) with wavelength related to the spatial frequency of the array (pixel spacing).

Invocation:

```
CALL SURFLIB_2DFT_CHOP (CHOP_THROW, CHOP_PA, PIXSIZE, NX, NY, FT_DATA, FT_VARIANCE,
WT_DATA, WT_VARIANCE, STATUS)
```

Arguments:

CHOP_THROW = REAL (Given)
chop throw in arcsec

CHOP_PA = REAL (Given)
position angle of chop. Positive is anti-clockwise from North. This angle is in degrees.

PIXSIZE = REAL (Given)
pixel size in arcsec

NX = INTEGER (Given)
Number of pixels in X

NY = INTEGER (Given)
Number of pixels in Y

FT_DATA (IDIMS(1), IDIMS(2)) = REAL (Returned)
data array containing F.T. of chop

FT_VARIANCE (IDIMS(1), IDIMS(2)) = REAL (Returned)
variance on DATA; set to 0

WT_DATA (IDIMS(1), IDIMS(2)) = REAL (Returned)
data array containing weight of chop

WT_VARIANCE (IDIMS(1), IDIMS(2)) = REAL (Returned)
variance on DATA; set to 0

STATUS = INTEGER (Given and returned)
global status

Copyright :

SURFLIB_CALC_DUAL_BEAM
Calculates a chopped map from a single beam map

Description:

This routine can be used to added a chopped beam response to a single beam image. When NBEAMS=2 this routine calculates a middle-beam dual-beam map by calculating the difference between the left and right beams for each pixel in the map.

When NBEAMS=3 a triple beam map is constructed where the central beam is the difference between the central beam and the average of the positions a chop distance away on either side.

All pixels off the edge of the image are assumed to have flux 0 and variance 0.

Bad pixels are assumed to be equivalent to flux of 0 and variance 0, although for the triple beam image a bad pixel is retained if it coincide with the middle beam position.

Invocation:

```
CALL SURFLIB_CALC_CHOPPED_IMAGE( NBEAMS, CHOP_THR, CHOP_PA, DIM1, DIM2, IN_DATA,
OUT_DATA, USEVAR, IN_VAR, OUT_VAR, STATUS)
```

Arguments:

NBEAMS = INTEGER (Given)
Number of beams to add. (2 or 3)

CHOP_THR = REAL (Given)
Chop throw in pixels

CHOP_PA = REAL (Given)
Chop position angle (east of north) in degrees

DIM1 = INTEGER (Given)
First dimension of image

DIM2 = INTEGER (Given)
Second dimension of image [can not use DIMS as an array since the linux fortran compiler doesnt like it]

IN_DATA = REAL (Given)
Input single-beam image

OUT_DATA = REAL (Returned)
Output dual-beam image

USEVAR = LOGICAL (Given)
Are we processing a variance array?

IN_VAR = REAL (Given)
Input variance image (if USEVAR=TRUE)

OUT_VAR = REAL (Returned)
Output variance image (if USEVAR=TRUE)

STATUS = INTEGER (Given and Returned)
Global status

Copyright :

SURFLIB_CALC_GRIDIJ
Calculate the position of a pixel number in an IJ grid

Description:

Calculates the I,J grid positions related to pixel numbers 1 to NX∗NY. Must calculate all positions in one go since it is very inefficient to calculate the spiral positions on demand Three modes types are currently available.

Invocation:

```
CALL SURFLIB_CALC_GRIDIJ( TYPE, NX, NY, ICEN, JCEN, I, J, STATUS)
```

Arguments:

TYPE = CHAR (Given)

Type of unwrapping. Can be one of: XLINEAR - unfold each X strip in turn for each Y YLINEAR - unfold each Y strip in turn for each X SPIRAL - unfold in a spiral from the reference pixels (ICEN,JCEN) DIAG1 - Diagonal starting at 1,1 DIAG2 - Diagonal starting at NX,1

NX = INTEGER (Given)

Number of pixels in X

NY = INTEGER (Given)

Number of pixels in Y

ICEN = INTEGER (Given)

Location of reference pixel in X

JCEN = INTEGER (Given)

Location of reference pixel in Y

I (NX ∗ NY) = INTEGER (Returned)

X Positions in grid, indexed by pixel number

J (NX ∗ NY) = INTEGER (Returned)

Y Positions in grid, indexed by pixel number

STATUS = INTEGER (Given & Returned)

Global Status

Copyright :

SURFLIB_CALC_IJPOS
Calculate the position of data on an XY grid

Description:

Given an XY position, calculates the IJ position in the grid. The I,J's are then placed into the output array (of dimension 2, N_PTS)

Invocation:

```
CALL SURFLIB_CALC_IJPOS (N_PTS, PIXEL_SZ, I_CENTRE, J_CENTRE, X_POS, Y_POS, IJ,
STATUS )
```

Arguments:

N_PTS = INTEGER (Given)
Number of X,Y pairs supplied

PIXEL_SZ = DOUBLE (Given)
Size of each pixel (radians)

I_CENTRE = INTEGER (Given)
Location of reference pixel in X

J_CENTRE = INTEGER (Given)
Location of reference pixel in Y

X_POS = DOUBLE PRECISION (Given)
X positions

Y_POS = DOUBLE PRECISION (Given)
Y positions

IJ (2, N_PTS) = INTEGER (Given)
Positions of each point (I,J) in output grid

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SURFLIB_CALC_OUTPUT_GRID
Calculate the grid required to contain data given the X,Y positions

Description:

Finds the extent of the data (given X and Y) and then reports back the size of the grid that is necessary to contain the data. Also returns the reference pixel position.

Invocation:

```
CALL SURFLIB_CALC_OUTPUT_GRID (N_FILES, N_PTS, PIXEL_SZ, X_PTR, Y_PTR, NX, NY,
I_CENTRE, J_CENTRE, STATUS )
```

Arguments:

N_FILES = INTEGER (Given)
Number of data sets (ie files)

N_PTS = INTEGER (Given)
Number of X,Y pairs supplied

PIXEL_SZ = REAL (Given)
Size of each pixel (radians)

X_PTR(N_FILES) = INTEGER (Given)
Pointers to arrays containing DOUBLE PRECISION X positions

Y_PTR(N_FILES) = INTEGER (Given)
Pointers to arrays containing DOUBLE PRECISION Y positions

NX = INTEGER (Given)
Required number of points in X in output grid

NY = INTEGER (Given)
Required number of points in Y in output grid

I_CENTRE = INTEGER (Given)
Location of reference pixel in X

J_CENTRE = INTEGER (Given)
Location of reference pixel in Y

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

- This routine requires double precision arguments for the X and Y coordinates.

SURFLIB_CALC_POLPACK_ANGROT
Average rotations angles over integration

Description:

This routine calculates the mean rotation angle for each integration. The input data contains one angle per sample and this is averaged to calculate the rotation angle per integration. The input angle is the rotation angle (anti-clockwise) between the RA frame and the nasmyth (waveplate) frame. The output angle is the angle between the waveplate 0 and the X PIXEL axis (not the RA axis) Since the supplied angle is the angle from the Y-axis (ie all angles are from Nasmyth Y axis) we need to add 90 degrees

Invocation:

```
CALL SURFLIB_CALC_POLPACK_ANGROT(N_POS, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS,
DEM_PNTR, ANGROT_IN, ANGROT_OUT, STATUS)
```

Arguments:

N_POS = INTEGER (Given)
Size of ANGROT_IN array

N_EXPOSURES = INTEGER (Given)
Number of exposures in DEM_PNTR

N_INTEGRATIONS = INTEGER (Given)
Number of integrations in DEM_PNTR

N_MEASUREMENTS = INTEGER (Given)
Number of measurements in DEM_PNTR

DEM_PNTR(1,N_EXPOSURES,N_INTEGRATIONS,N_MEASUREMENTS) = INTEGER (Given)

Start position for each EXP, INT, MEAS

ANGROT_IN (N_POS) = REAL (Given)
Rotation for each sample (N_POS)

ANGROT_OUT (N_INTEGRATIONS, N_MEASUREMENTS) = REAL (Returned)
Rotation angle averaged over each integration

ANGROT_VAR (N_INTEGRATIONS, N_MEASUREMENTS) = REAL (Returned)
Variance on averaged angle

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SURFLIB_CLIP_GRID
Remove spikes from grid

Description:

Given the binned data calculated by SURFLIB_FILL_GRID and the limits calculated by SURFLIB_STATS_GRID, clip all points that lie outside the statistical limits provided by the stats array.

Invocation:

```
CALL SURFLIB_CLIP_GRID(N_FILES, N_PTS, NX, NY, NMAX, NSIGMA, N_BOLS, BITNUM, QUALITY_PTR,
BINS, BIN_POS, STATS, IPOS, JPOS, NSPIKES, STATUS)
```

Arguments:

N_FILES = INTEGER (Given)
Number of input data sets

N_PTS(N_FILES) = INTEGER (Given)
Number of points in each data set

NX = INTEGER (Given)
Size of X dimension

NY = INTEGER (Given)
Size of Y dimension

NMAX = INTEGER (Given)
Maximum value allowed for third dimension of BINS

N_BOLS (N_FILES) = INTEGER (Given)
Number of bolometers per file (used for constructing message for users)

BITNUM = INTEGER (Given)
Bit number that is modified when set bad

DATA_PTR(N_FILES) = INTEGER (Given & Returned)
The input data with each spike replaced with a median value

QUALITY_PTR(N_FILES) = INTEGER (Given & Returned)
Pointers to quality arrays. Note that quality arrays are modified. (The whole point of this routine) even though the pointers are not.

BINS(NX, NY, NMAX) = REAL (Given)
The data stored in relation to its position

BIN_POS (NX, NY, NMAX) = INTEGER (Given)
Position of each data point in quality arrays

STATS (NX, NY, 3) = REAL (Given)
Statistics of each bin. 1=Median, 2=High, 3=Low

PNTS(NMAX) = REAL (Given)
Scratch space for copying in the data from each I,J

IPOS(NX∗NY) = INTEGER (Given)
I position in array

JPOS(NX∗NY) = INTEGER (Given)
J position in array

NSPIKES = INTEGER (Returned)
Number of spikes removed for a given file

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SURFLIB_DECODE_REMSKY_STRING
Calculate bolometer list from remsky input

Description:

Given an array of bolometer descriptions, return an array of bolometer numbers. Input array can contain a variety of forms for identifying bolometers or lists of bolometers. This string is generated by, for example, the REMSKY task.

Invocation:

```
CALL SURFLIB_DECODE_REMSKY_STRING( SUB_INSTRUMENT, N_ELEMENTS, BOL_DESC, N_BOLS,
BOL_ADC, BOL_CHAN, BOL_LIST, N_BOLS_OUT, STATUS )
```

Arguments:

SUB_INSTRUMENT = CHARACTER (Given)
Sub instrument name. This governs the bolometers that are allowed to be returned.

N_ELEMENTS = INTEGER (Given)
Number of elements in the input array

BOL_DESC(N_ELEMENTS) = CHARACTER∗(∗) (Given)
Array of bolometer descriptions Can take the form of:

`n` bolometer number

`id` bolometer id (eg H7)

`all` all bolometers

`rn` Nth ring of bolometers

Can be prefixed with a minus sign to indicate that a bolometer should be removed from the final list. The calculation is sequential.

N_BOLS = INTEGER (Given)
Maximum allowed size for return array containing bolometer numbers An error is given if this number is exceeded. In effect this is equivalent to the actual number of bolometers on the array.

BOL_ADC(N_BOLS) = INTEGER (Given)
ADC numbers for identifying bolometers

BOL_CHAN(N_BOLS) = INTEGER (Given)
Channel number for identifying bolometers

BOL_LIST(N_BOLS) = INTEGER (Returned)
List of bolometer numbers corresponding to the supplied strings.

N_BOLS_OUT = INTEGER (Returned)
Actual number of bolometers returned in BOL_LIST

STATUS = INTEGER (Given & Returned)
Global status

Syntax :

`[all]`	Whole array
`[r0]`	Ring zero (central pixel)
`[h7,r1]`	inner ring and h7
`[r1,-h8]`	inner ring without h8
`[r1,-18]`	inner ring without bolometer 18
`[all,-r1,-h7]`	all pixels except the inner ring/h7
`[all,-r3,g1]`	all pixels except ring 3 but with g1 (g1 is in r3)
`[all,-r1,-r2,-r3,-r4,-r5]`	Selects the central pixel!!

Note that the sequence is evaluated in turn so that `[all,-all,h7]` will still end up with pixel `h7`.

Copyright :

SURFLIB_DIFF_DESPIKE
Despike data scan by detecting points more than nsigma from running 2-point mean

Description:

This routine removes spikes from SCAN/MAP observations. The scan map differential despiking algorithm uses 2 criteria to decide which points are spikes.

First, for each bolometer used a pass is made through each scan calculating for each point:-

$$diff(i) = point(i) - \frac{(point(i-1) + point(i+1))}{2.0} \tag{3}$$

Values of 'diff' for the first and last points in the scan are calculated in a similar way but subtracting the mean of points 2 and 3 and points n-1 and n-2 respectively.

The mean and standard deviation of 'diff' are calculated by coadding the 10 points at each end of the scan where, hopefully, there is no source emission. Spikes in these regions are handled by removing points from the coadd that lie further than 3 sigma from the mean, then redoing the calculation recursively until no further points need be removed.

The first criterion for a spike is that it's 'diff' value should be further from the mean of 'diff' by NSIGMA times the sigma derived from the endpoints.

The problem with this simple approach is that bright sources in the scan themselves lead to excursions in 'diff' that can be wrongly identified as spikes. To prevent this happening a second criterion is used. In this the scan values are convolved with a 3 sample wide box so that each 'box' point is the average of the point itself and the points on either side of it. 'Box' is expected to increase faster for real sources than for spikes because in them the increase will be spread over all 3 averaged points rather than just 1.

The second criterion for a spike is met, therefore, if a point's 'diff' is further from the 'diff' mean than the value of 'box' at that point.

Fixed-up values for points that have identified as spikes are calculated by interpolating between the closest healthy points on either side.

The second spike criterion also means unfortunately that the technique is less sensitive to spikes on bright sources than elsewhere. In addition, it is still possible to clip bright sources if too low a value for NSIGMA is used. It is recommended to run despike several times with different values of NSIGMA. Begin with NSIGMA=5, look at the result to see how effective despiking has been, then repeat the process with NSIGMA=4.5, 4.0 etc. until you start to clip source information.

Invocation:

```
CALL SURFLIB_DIFF_DESPIKE (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS, DEMOD_POINTER,
N_BOL, N_POS, IN_DATA, IN_VARIANCE, IN_QUALITY, BADBIT, NSIGMA, OUT_DATA, OUT_VARIANCE,
OUT_QUALITY, NUM_SPIKES, STATUS)
```

Arguments:

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)
array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the variance on IN_DATA

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

BADBIT = BYTE (Given)
bad bit mask

NSIGMA = REAL (Given)
cut-off sigma for spikes

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the deconvolved data

OUT_VARIANCE (N_BOL, N_POS) = REAL (Returned)
the variance on OUT_DATA

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

NUM_SPIKES = INTEGER (Returned)
Total number of spikes detected

STATUS = INTEGER (Given and returned)
global status

SURFLIB_FILL_GRID
Populate grid with data points corresponding to position

Description:

Given an array of I,J coordinates associated with data values, this routine places each data point onto the rectangular grid and the position in the data array into a corresponding grid. Grid is of size (NX,NY,NMAX) and each data point is placed into a different layer (up to NMAX) for each I,J. We already know the value of NMAX since the output array has been pre-allocated.

Invocation:

```
CALL SURFLIB_FILL_GRID (N_PTS, NX, NY, NMAX, OFFSET, IN_DATA, IN_QUALITY, BADBIT,
IJ, GRID, BINS, BIN_POS, STATUS )
```

Arguments:

N_PTS = INTEGER (Given)
Number of I,J pairs supplied

NX = INTEGER (Given)
Size of X dimension

NY = INTEGER (Given)
Size of Y dimension

NMAX = INTEGER (Given)
Maximum value allowed for third dimension of BINS

OFFSET = INTEGER (Given)
This is the offset in the IJ data array. Not used for IJ itself (Since this is added to the pointer of the input array if needed) in this array but is added onto each of the values placed into the BIN_POS array so that we can keep track of the input data without using a further two pieces of information.

IN_DATA(N_PTS) = REAL (Given)
Input data

IN_QUALITY(N_PTS) = BYTE (Given)
Input quality

BADBIT = BYTE (Given)
Bad bits mask for quality array

IJ (2, N_PTS) = INTEGER (Given)
Positions of each point (I,J) in output grid for each N_PTS

GRID (NX, NY) = INTEGER (Given & Returned)
Scratch space for keeping track of the number of points added into each I,J

BINS(NX, NY, NMAX) = REAL (Given & Returned)
The data stored in relation to its position

BIN_POS (NX, NY, NMAX) = INTEGER (Given & Returned)
The position in the input data arrays associated with each data point in BINS

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

- GRID is not initialised by this routine since it is incremented each time a point is placed in BINS
- BINS and BIN_POS are not initialised here but should be filled with bad before this routine is first called.
- a status of SAI__WARN is returned if the indices lie outside the model area. It is up to the calling routine to decide whether to halt.

Copyright :

SURFLIB_FILL_POLPACK_ANGLES
Average

Description:

Copy the waveplate and rotation angles from WPLATE and ANGROT (Which have dimensions (N_INT,N_MEAS) to a 2-dim array containing the angles (waveplate and rotation) for each integration and (in a separate array) the angles for each measurement. Note that since the rotation angles are stored in the file as one per integration, they need to be averaged over the integration in order to calculate the correct value for the measurement. ANG_MEAS and ANG_INT should be set to VAL__BADR before entry to this routine.

Invocation:

```
CALL SURFLIB_FILL_POLPACK_ANGLES( MAX_FILE, MAX_INT, MAX_MEAS, N_FILE, N_INT,
N_MEAS, WPLATE, ANGROT, FAST_AXIS, ANG_INT, ANG_MEAS, STATUS)
```

Arguments:

MAX_FILE = INTEGER (Given)
First Dimension of ANG_INT,ANG_MEAS

MAX_INT = INTEGER (given)
Second dimension of ANG_INT

MAX_MEAS = INTEGER (given)
Second dimension of ANG_MEAS

N_FILE = INTEGER (Given)
Current slice in ANG_INT, ANG_MEAS

N_INT = INTEGER (Given)
Number of integrations per measurement

N_MEAS = INTEGER (Given)
Number of measurements

WPLATE(N_INT, N_MEAS) = REAL (Given)
Waveplate angles for each integration

ANGROT(N_INT, N_MEAS) = REAL (Given)
Waveplate angles for each integration

FAST_AXIS = REAL (Given)
Angle of the fast axis of the waveplate and the zero position of the waveplate. Added to each waveplate angle to calculate the true angle of the waveplate.

ANG_INT(MAX_FILE, MAX_INT, 2) = REAL (Returned)
Waveplate and rotation angles for each integration 1=Waveplate, 2=Rotation. The waveplate angle is corrected for fast axis.

ANG_MEAS(MAX_FILE, MAX_MEAS, 2) = REAL (Returned)

Waveplate and rotation angles for each measurement. 1=Waveplate, 2=Rotation. The waveplate angle is corrected for fast axis. The rotation angle is average over each integration during the measurement. The waveplate angle is set to bad if it is not constant for the given measurement.

STATUS = INTEGER (Given and Returned)

Global status

Copyright :

SURFLIB_FILL_WPLATE
Populate the WavePlate array

Description:

Populate an array containing the waveplate angle for each sample. This can be calculated in two ways. If the SCUCD_WPLATE array is present (N_WPLATE>0), then this array contains the waveplate position for each measurement. If this array is not present the Waveplate angle is assumed to start at 0 degrees and increment by 22.5 degrees for each integration. If the number of measurements exceeds the number of positions in the SCUCD_WPLATE array the sequence wraps round to the first value. The final output array will contain a single number (waveplate position) per sample (up to N_POS). The output value will be in degrees.

Since integrations and measurements are stored sequentially in a data file (in DEM_PNTR) there is no need to use SCULIB_FIND_SWITCH to determine the start and end points in the data array - use SCULIB_FIND_INT.

The waveplate angles are also stored in a small array containing a single wave plate angle for each integration and each measurement. This is created to make it easier to extract the individual waveplate positions later on when storing them in output file

Invocation:

```
CALL SURFLIB_FILL_WPLATE(USE_WP, N_WPLATE, SCUCD_WPLATE, N_POS, N_EXP, N_INT,
N_MEAS, DEM_PNTR, WPLATE_OUT, WPLATE_ANG, STATUS)
```

Arguments:

N_WPLATE = INTEGER (Given)
Size of SCUCD_WPLATE array. If the value is zero then SCUCD_WPLATE will not be used.

SCUCD_WPLATE(N_WPLATE) = REAL (Given)
Waveplate positions for each measurement (degrees)

N_POS = INTEGER (Given)
Number of samples in WPLATE_OUT

N_EXP = INTEGER (Given)
Number of exposures in DEM_PNTR array

N_INT = INTEGER (Given)
Number of integrations in DEM_PNTR array

N_MEAS= INTEGER (Given)
Number of measurements in DEM_PNTR array

DEM_PNTR (1, N_EXP, N_INT, N_MEAS) = INTEGER (Given)
Start position for each EXP, INT, MEAS

WPLATE_OUT (N_POS) = REAL (Returned)
Waveplate position for each sample (degrees)

WPLATE_ANG(N_INT, N_MEAS) = REAL (Given)
Waveplate position for each integration/measurement

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SURFLIB_HISTOGRAM_GRID
Calculate a 2-D histogram of I,J coordinates on a grid

Description:

Given an array of I,J coordinates calculate the histogram of all points onto this grid of size NX $*$ NY. Just tells you how many data points there are per cell. Returns the histogram and the highest value in the histogram

Invocation:

```
CALL SURFLIB_HISTOGRAM_GRID (N_PTS, NX, NY, USEDATA, IN_DATA, IN_QUALITY, BADBIT,
IJ, GRID, IMAX, JMAX, NMAX, STATUS )
```

Arguments:

N_PTS = INTEGER (Given)
Number of I,J pairs supplied

NX = INTEGER (Given)
Size of X dimension

NY = INTEGER (Given)
Size of Y dimension

USEDATA = LOGICAL (Given)
If TRUE the histogram takes bad data points into account by not including them in the histogram. If false the straight histogram of the I,J's is constructed.

IN_DATA(N_PTS) = REAL (Given)
Input data

IN_QUALITY(N_PTS) = BYTE (Given)
Input quality

BADBIT = BYTE (Given)
Bad bits mask for quality array

IJ (2, N_PTS) = INTEGER (Given)
Positions of each point (I,J) in output grid

GRID (NX, NY) = INTEGER (Given & Returned)
Histogram. Note that array is not cleared by this routine leaving the possibility that the routine can be called multiple times to include more than one data set.

IMAX = INTEGER (Returned)
I position of maximum [first one encountered]

JMAX = INTEGER (Returned)
J position of maximum [first one encountered]

NMAX = INTEGER (Returned)
Maximum value in histogram

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

- If USEDATA is TRUE the data and quality array are used so that bad pixels can be identified and not included into the histogram of positions.
- GRID is not initialised by this routine
- A warning error is raised if the bounds of the histogram are exceeded. It is up to the caller to decide whether this is a fatal error.

Copyright :

SURFLIB_MEDIAN_REGRID
Generate image with each pixel the median of all pixels in bin

Description:

This is done in two stages:

1) Find the size of the output grid from the maximum extent of the input data. 2) Loop through data. Find I,J coordinate of each point in the output grid. 3) Find out maximum number of points for an I,J position. 4) Put data onto grid in array (I,J,N) [REALS]. We also need to store positions of these data. We can either do it by storing the file number, bolometer and position (time) index OR we can just store some index in a merged data array that goes from 1..TOT_PTS. First method is easy but memory hungry. Second method is more efficient but does need some reconstruction to work out where the point was in the original data. Use the second method.

Once the data is gridded, it is first displayed and then despiked. Currently despiking is done on a simple sigma clipping basis for each bin.

Invocation:

```
CALL SURFLIB_MEDIAN_REGRID( N_FILES, N_PTS, DIAMETER, WAVELENGTH, OUT_PIXEL, NX,
NY, ICEN, JCEN, BOL_RA_PTR, BOL_DEC_PTR, DATA_PTR, OUT_DATA, OUT_VAR, OUT_QUAL,
STATUS )
```

Arguments:

N_FILES = INTEGER (Given)
Number of data sets (ie files)

N_PTS (N_FILES) = INTEGER (Given)
Total number of points in each map

N_POS(N_FILES) = INTEGER (Given)
Number of positions per set (Y positions)

N_BOLS(N_FILES) = INTEGER (Given)
Number of bolometers per set (X positions)

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (X coords) Note that each data set has positions for N_POS $*$ N_BOLS

BOL_RA_PTR(N_FILES) = INTEGER (Given)
Array of pointers to position information (Y coords)

DATA_PTR(N_FILES) = INTEGER (Given)
Pointers to actual data arrays

QUALITY_PTR(N_FILES) = INTEGER (Given)
Pointer to quality arrays

NX = INTEGER (Returned)
Number of points in grid (X)

NY = INTEGER (Returned)
Number of points in grid (Y)

ICEN = INTEGER (Returned)
Reference pixel (X)

JCEN = INTEGER (Returned)
Reference pixel (Y)

NSPIKES (N_FILES) = INTEGER (Returned)
Number of spikes detected (and removed) in each file

BADBIT (N_FILES) = BYTE (Given)
Bad bit mask for identifying bad pixels from quality

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

For SMODE=NONE, DMODE is only requested if a plot is required.

Copyright :

SURFLIB_PLOT_GRID
Plot unwrapped grid

Description:

Given the binned data calculated by SURFLIB_FILL_GRID Plot the unwrapped data as 'data' against 'bin' Use this as a first attempt at spike detection

Invocation:

```
CALL SURFLIB_PLOT_GRID(UNIT, NX, NY, NMAX, NSIGMA, IPOS, JPOS BINS, STATS, PNTS,
POS, STATUS)
```

Arguments:

UNIT = INTEGER (Given)
Display number

NX = INTEGER (Given)
Size of X dimension

NY = INTEGER (Given)
Size of Y dimension

NMAX = INTEGER (Given)
Maximum value allowed for third dimension of BINS

NSIGMA = REAL (Given)
How many sigma away we plot the guide lines. If NSIGMA is negative we dont plot the sigma ranges on the plot.

IPOS(NX $*$ NY) = INTEGER (Given)
I coordinate for each pixel

JPOS(NX $*$ NY) = INTEGER (Given)
J coordinate for each pixe

STATS(NX, NY, 3) = REAL (Given)
Statistics for each bin. 1=Median, 2=high, 3=low

BINS(NX, NY, NMAX) = REAL (Given)
The data stored in relation to its position

PNTS(NMAX) = REAL (Given)
Scratch space for copying in the data from each I,J

POS(NMAX) = REAL (Given)
Scratch space for storing the X positions for each marker on the plot

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SURFLIB_PROCESS_BOLS
Calculate apparent RA/Dec of bolometers and some extra processing

Description:

This routine calculates the apparent RA and Dec of each bolometer for all measurements, integrations and exposures. Both JIGGLE and SCAN data are supported. In addition, if this is called from EXTINCTION (ie EXTINCTION=.TRUE.) the data is corrected for extinction using the calculated elevation of the bolometers. If EXTINCTION is .FALSE. all the bolometers positions are converted to apparent RA-Dec at the reference modified Julian date (given by MJD_STANDARD).

There is some complication for moving sources (use PL output coords):

For JIGGLE:

- [1] Calculate map RA/Dec at new MJD
- [2] Convert to tangent plane offsets for this map centre

For SCAN:

- [1] Calculate array centre for reference MJD
- [2] Calculate tangent offsets of array centre relative to map centre
- [3] Calculate map centre at the interpolated MJD
- [4] Work out array centre given new map centre

This is all because the file stores RA start and end relative to the fixed centre at the start of the observation.

Invocation:

```
CALL SURFLIB_PROCESS_BOLS(TSKNAME, N_BEAMS, N_BOL, N_POS, N_POS_BEAMS, N_SWITCHES,
N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS,START_EXP, END_EXP, START_INT, END_INT,
START_MEAS, END_MEAS,N_MAP, N_FITS, FITS, DEM_PNTR, LST_STRT, IN_ROTATION, SAMPLE_MODE,
SAMPLE_COORDS, OUT_COORDS, JIGGLE_REPEAT, JIGGLE_COUNT, JIGGLE_X, JIGGLE_Y, JIGGLE_P_SWITC
RA_CEN, DEC_CEN, RA1, RA2, DEC1, DEC2, MJD_STANDARD, IN_UT1, MJD1, LONG1, LAT1,
MJD2, LONG2, LAT2, LOCAL_COORDS, MAP_X, MAP_Y, N_POINT, POINT_LST, POINT_DAZ,
POINT_DEL, NUM_CHAN, NUM_ADC, BOL_ADC, BOL_CHAN, BOL_DU3, BOL_DU4, SCAN_REVERSAL,
FIRST_LST, SECOND_LST, FIRST_TAU, SECOND_TAU, BOL_DEC, BOL_RA, NDATA, NVARIANCE,
USE_LST, LST_DATA, WAVEPLATE_ANG, BOL_IP_DATA, STATUS)
```

Arguments:

TSKNAME = _CHAR (Given)

Taskname that is used to determine whether certain parts of the code are executed. Special values are: SCUOVER - when using SCUOVER this prevents the EXP_START loop from running properly. Only a single set of bolometer positions are returned. EXTINCTION - corrects the input data for extinction REMIP - Corrects the input data for instrumental polarisation All other values for this parameter assume that the bolometer positions are to be calculated and returned (converted to a standard MJD)

N_BEAMS = _INTEGER (Given)
Number of beams in the input data. Only used if EXTINCTION=.TRUE.

N_BOL = _INTEGER (Given)
Number of bolometers in the input data

N_POS = _INTEGER (Given)
Number of 'samples' taken

N_POS_BEAMS = INTEGER (Given & Returned)
Number of beam positions to be returned per point. Can only be reduced. If equals 3 then order is M,L,R

N_SWITCHES = _INTEGER (Given)
Number of switches

N_EXPOSURES = _INTEGER (Given)
Number of exposures

N_INTEGRATIONS = _INTEGER (Given)
Number of integrations

N_MEASUREMENTS = _INTEGER (Given)
Number of measurements

START_EXP = INTEGER (Given)
First exposure to process

END_EXP = INTEGER (Given)
Last exposure to process

START_INT = INTEGER (Given)
First integration to process

END_INT = INTEGER (Given)
End integration to process

START_MEAS = INTEGER (Given)
First measurement to process

END_MEAS = INTEGER (Given)
End measurement to process

N_MAP = _INTEGER (Given)
Map number of input data (not used for EXTINCTION or REMIP)

N_FITS = _INTEGER (Given)
Number of FITS items

FITS() = _CHAR*80 (Given)
FITS array

DEM_PNTR() = _INTEGER (Given)
DEM_PNTR array - position in file of each exposure

LST_STRT() = _DOUBLE (Given)
LST of each exposure

IN_ROTATION = _DOUBLE (Given)
Angle between apparent N and N of input coord system (radians)

SAMPLE_MODE = _CHAR (Given)
Sample mode of input file

SAMPLE_COORDS = _CHAR (Given)
Coordinate system of sample offsets

OUT_COORDS = _CHAR (Given)
Output coordinate system

JIGGLE_REPEAT = _INTEGER (Given)
Number of times jiggle pattern is repeated in a switch

JIGGLE_COUNT = _INTEGER (Given)
Number of jiggle in pattern

JIGGLE_X(JIGGLE_COUNT) = _REAL (Given)
X jiggle offsets (arcsec)

JIGGLE_Y(JIGGLE_COUNT) = _REAL (Given)
Y jiggle offsets (arcsec)

JIGGLE_P_SWITCH = _INTEGER
Number of jiggles per switch

RA_CEN = _DOUBLE (Given)
apparent RA of output map centre (radians). Used mainly for JIGGLE but also for scan/map data pre Dec 1997

DEC_CEN = _DOUBLE (Given)
apparent Dec of output map centre (radians) Used mainly for JIGGLE but also for scan/map data pre Dec 1997

RA1 = _REAL (Given)
RA (in RD, RJ or AZ) at start of scan for each exposure (SCAN only)

RA2 = _REAL (Given)
RA (in RD, RJ or AZ) at end of scan for each exposure (SCAN only)

DEC1 = _REAL (Given)
DEC (in RJ, RD or AZ) at start of scan for each exposure (SCAN only)

DEC2 = _REAL (Given)
DEC (in RD, RJ, AZ) at end of scan for each exposure (SCAN only)

MJD_STANDARD = _DOUBLE (Given)
Standard MJD to which each input map is referenced (EXTINCTION=FALSE)

IN_UT1 = _DOUBLE (Given)
MJD of input data.

MJD1 = DOUBLE (Given)
MJD of first planet position

LONG1 = DOUBLE (Given)
Longitude (apparent RA) at MJD1

LAT1 = DOUBLE (Given)
Latitude (apparent Dec) at MJD1

MJD2 = DOUBLE (Given)
MJD of second planet position

LONG2 = DOUBLE (Given)
Longitude (apparent RA) at MJD2

LAT2 = DOUBLE (Given)
Latitude (apparent Dec) at MJD2

LOCAL_COORDS = CHARACTER (Given)
Coordinate system of offsets

MAP_X = DOUBLE (Given)
X offset in LOCAL_COORDS (radians)

MAP_Y = DOUBLE (Given)
Y offset in LOCAL_COORDS (radians)

N_POINT = _INTEGER (Given)
Number of pointing corrections (should be zero if EXTINCTION)

POINT_DEL = _REAL (Given)
Elevation pointing corrections (radians) [only if EXTINCTION=FALSE]

POINT_DAZ = _REAL (Given)
Azimuth pointing corrections (radians) [only if EXTINCTION=FALSE]

POINT_LST = _DOUBLE (Given)
LST of pointing corrections (radians) [only if EXTINCTION=FALSE]

NUM_CHAN = _INTEGER (Given)
Number of channels in DAQ

NUM_ADC = _INTEGER (Given)
Number of AtoD cards.

BOL_ADC = _INTEGER (Given)
A/D numbers of bolometers measured in input file

BOL_CHAN = _INTEGER (Given)
channel numbers of bolometers measured in input file

BOL_DU3 = _REAL (Given)
dU3 Nasmyth coordinates of bolometers

BOL_DU4 = _REAL (Given)
dU4 Nasmyth coordinates of bolometers

SCAN_REVERSAL = LOGICAL (Given)
Multiply alternate exposures by -1 if SCANning

FIRST_LST = _DOUBLE (Given)
LST of first tau value (EXTINCTION only)

SECOND_LST = _DOUBLE (Given)
LST of second tau value (EXTINCTION only)

FIRST_TAU = _REAL (Given)
First tau value (EXTINCTION only)

SECOND_TAU = _REAL (Given)
Second tau value (EXTINCTION only)

BOL_DEC(N_BOL, N_POS, N_POS_BEAMS) = _DOUBLE (Returned)
Apparent DEC of bolometers for each measurement for MJD_STANDARD Depending on N_POS_BEAMS can contain M, L and R beam positions.

BOL_RA(N_BOL, N_POS, N_POS_BEAMS) = _DOUBLE (Returned)
Apparent RA of bolometers for each measurement for MJD_STANDARD Depending on N_POS_BEAMS can contain M, L and R beam positions.

NDATA(N_BOL, N_POS, N_BEAMS) = _REAL (Given & Returned)
corrected data (EXTINCTION and REMIP only)

NVARIANCE(N_BOL, N_POS, N_BEAMS) = _REAL (Given & Returned)
Extinction corrected variance (EXTINCTION only)

USE_LST = LOGICAL (Given)
Do we return the LST positions? (TRUE then we do)

LST_DATA(N_POS) = DOUBLE (Returned)
LST for each position.

WAVEPLATE_ANG = REAL (Given)
waveplate position angle (Nasmyth degrees) (Given & Returned) The supplied value is the nasmyth angle in degrees and the return value the sky rotation angle for each position. (ie parallactic angle - elevation) in degrees (REMIP only)

BOL_IP_DATA(8, NUM_CHAN, NUM_ADC) = REAL (Given)
The IP data. This array contains the ip data for each bolometer (specified by a combination of CHAN and ADC). The 4 slices represent: P0, Pslope, Theta0 and ThetaSlope (REMIP only) The next 4 slices are for the related variance (same order)

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

MAP_X and MAP_Y are only used for JIGGLE modes. The SCAN/MAP offsets are wrapped into the RA1,DEC1,RA2,DEC2 numbers by the on-line system. The offsets are added to the map centre every time round the loop. This is because it is possible to have AZ offsets for RA,Dec centres.

This routine tries to deal with the different versions of SCUCD (only affects SCAN/MAP data). For version 0:

RA/Decs of the scan were incorrectly stored as RJ. They are converted to RD before further processing. The coordinate frame of RA1, RA2, DEC1 and DEC2 fro SCAN/MAP depends on the CENTRE_COORDS of the observation. For CENTRE=RD the scan positions are in RD; for CENTRE=RB,RJ,GA the scans are in RJ and for centre=AZ the scans are in AZ. (SCULIB_SCAN_2_RD)

For version 1.0:

A bug was introduced concerning the calculation of the ends of the scans. The bug is recreated and inverted in order to compensate. (SCULIB_FIX_SCAN_V10)

Also, LO chopping for jiggle maps was broken until 19980730 such that the initial chop pa was calculated in Az but never updated as the sky rotated. This fix is only important for SCUBA2MEM where the positions of the off-beams are returned.

Bugs:

Currently the IN_UT1 is assumed to be the MJD when the data taking begins. As of 24-NOV-1997 IN_UT1 is actually the MJD of the start of the observation (ie when the telescope begins to slew). This is a problem for data using a moving centre.

SURFLIB_READ_IPFILE
Reads IP data from text file

Description:

This routine reads IP data from a text file and stores it in a multi-dimensional array. The format of the file is: Any line starting with the string 'FAST' is assumed to be specifying the angle between 0 degrees on the waveplate and the angle to the fast axis. This is wavelength dependent. The line should be of the format

```
FAST filter angle
```

eg "FAST 450 -3.0"

The list of valid filter names is passed in to the routine. An error is raised if a fast axis angle has not been specified for each filter.

For each bolometer the following information is required:

```
BolName P0 P0_Err Pslope_Err Theta0 Theta0_Err ThetaSlope ThetaSlope_Err
```

BolName	Name of bolometer (eg H7,G12..)
P0	IP percentage polarization at the horizon
P0Err	Error in P0
PSlope	Gradient in P (ie P = P0 + Pslope * Elevation)
PSlopeErr	Error in PSlope
Theta0	polarization angle of IP (degrees)
Theta0Err	Error in polarisation angle
ThetaSlope	Gradient in Theta (ie Theta = Theta0 + ThetaSlope * El)
ThSlErr	Error in ThetaSlope

For example: `H7 2.0 0.01 0.02 0.01 167 2 1.0 0.01`

{ is the comment character. Blank lines are ignored. The columns must be space separated.

All angles are converted to radians before further processing (including Pslope which is in degrees^{-1})

Invocation:

```
CALL SURFLIB_READ_IPFILE( FD, NUM_CHAN, NUM_ADC, N_FILT, FILTERS, FAST_ANG, BOL_IP_DATA,
STATUS)
```

Arguments:

FD = FILE (Given)
Identifier to the open file (opened for read access)

NUM_CHAN = INTEGER (Given)
Number of available bolometer channels

NUM_ADC = INTEGER (Given)
Number of available AtoD cards

N_FILT = INTEGER (Given)
Number of filters requested

FILTERS(N_FILT) = CHARACTER (Given)
Array of requested filter names

FAST_ANG(N_FILT) = REAL (Returned)
Fast axis angle for each filter. Will contain bad values if no angle was found

BOL_IP_DATA(8, NUM_CHAN, NUM_ADC) = REAL (Returned)
The IP data. This array contains the ip data for each bolometer (specified by a combination of CHAN and ADC). The 4 slices represent: P0, Pslope, Theta0 and ThetaSlope The last 4 slices contain the corresponding variance All angles are converted to radians (Pslope has to be converted from /degree to /radian).

Copyright :

SURFLIB_REM_GRID
Remove an image grid of data from demodulated data

Description:

Removes a gridded image from demodulated data using a lookup table containing the (I,J) coordinates (in the grid) of each input data point. Can be used to remove the source from an input data set.

Invocation:

```
CALL SURFLIB_REM_GRID( N_POS, N_BOLS, NX, NY, IJ, GRID, IN_DATA, STATUS)
```

Arguments:

N_PTS = INTEGER (Given)
Number of data points in input data

NX = INTEGER (Given)
X dimension of input grid

NY = INTEGER (Given)
Y dimension of input grid

IJ (2, N_PTS) INTEGER (Given)
Positions of each point (I,J) in input grid

GRID (NX, NY) = REAL (Given)
Image grid to be subtracted from data

IN_DATA(N_PTS) = REAL (Given & Returned)
Input data to be modified

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SURFLIB_REM_TIMESERIES
Remove a time series from a 2D array

Description:

Remove a time series (Y data) from a 2D array where the second dimension corresponds to time. It is assumed that both the time series and the 2D array have the same number of points in time (N_POS)

Invocation:

```
CALL SURFLIB_REM_TIMESERIES ( N_BOLS, N_POS, TIME_SER, TIME_VAR, IN_DATA, IN_VAR,
STATUS)
```

Arguments:

N_BOLS = INTEGER (Given)
Number of bolometers in data array (x dimension)

N_POS = INTEGER (Given)
Number of time data values (y dim)

TIME_SER (N_POS) = REAL (Given)
Time series to be removed

TIME_VAR (N_POS) = REAL (Given)
Variance associated with each time value

IN_DATA (N_BOLS, N_POS) = REAL (Given & Returned)
Input data to be modified

IN_VAR (N_BOLS, N_POS) = REAL (Given & Returned)
Input variance

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SURFLIB_REMOVE_DC_FROM_EXP
Remove linear baseline from each exposure

Description:

This routine takes a data array. It then removes a DC offset (either the mean or the median derived from the scan) from each scan.

Invocation:

```
CALL SCULIB_REMOVE_DC_FROM_EXP(DORLB, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS,
METHOD, DEM_PNTR, N_BOL, N_POS, IN_DATA, IN_QUALITY, SAMPLE_DX, CHOP_THROW, OUT_DATA,
OUT_QUALITY, BADBIT, STATUS)
```

Arguments:

DORLB = LOGICAL (Given)
control whether we are subtracting the baseline (TRUE) or storing the basline (FALSE)

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

METHOD = CHAR (Given)
Removal method. MEDIAN or MEAN supported.

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)
= INTEGER (Given) array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the measured variance

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the data with baseline removed

OUT_VARIANCE (N_BOL, N_POS) = REAL (Given)
the output variance

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SURFLIB_REMOVE_DC_VIA_SECT
Remove median baseline from each exposure

Description:

This routine takes a data array and removes a DC offset from each scan. The DC level is the median of the data specified in the SCUBA section for that particular integration. The same level is removed for each scan of an integration with a different value for each bolometer.

Invocation:

```
CALL SCULIB_REMOVE_DC_FROM_EXP(DORLB,N_SPEC,SECTION,USE_SECT,N_EXPOSURES, N_INTEGRATIONS,
N_MEASUREMENTS, METHOD, DEM_PNTR, N_BOL, N_POS, IN_DATA, IN_QUALITY, SAMPLE_DX,
CHOP_THROW, OUT_DATA, OUT_QUALITY, BADBIT, STATUS)
```

Arguments:

DORLB = LOGICAL (Given)
control whether we are subtracting the baseline (TRUE) or storing the basline (FALSE)

N_SPEC = INTEGER (Given)
Number of sections specified

SECTION (N_SPEC) = CHAR (Given)
Array of section specifications

USE_SECT = LOGICAL (Given)
Is this an inverse section

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)
= INTEGER (Given) array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the measured variance

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the data with baseline removed

OUT_VARIANCE (N_BOL, N_POS) = REAL (Given)
the output variance

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SURFLIB_REMOVE_IP
Remove instrumental polarisation

Description:

Remove the instrumental polarisation signal from the data The following formula is used:

New flux = Measured flux / (1 + %age IP)

IP = %age polarisation ∗ cos (4∗waveplate - 2∗ IP angle)

where %age polarisation and IP angle vary linearly with elevation.

This is an approximation of

Actual flux = measured flux - mean flux ∗ %age IP

where IP = P ∗ (1 + cos (4 WP - 2 THETA)) because the mean flux can not be calculated trivially since the bolometers are jiggling on and off the source

This routine corrects the data point for each bolometer by looking up the IP data in the supplied array. Only one set of bolometers can be processed at any one time (since the elevation of the array changes during the observation). The routine should be called repeatedly for each time/elevation.

Invocation:

```
CALL SURFLIB_REMOVE_IP( ELEVATION, NUM_CHAN, NUM_ADC, N_BOLS, WPLATE_ANG, BOL_CHAN,
BOL_ADC, BOL_IP_DATA, BOL_DATA, STATUS)
```

Arguments:

ELEVATION = DOUBLE (Given)
Elevation of the source (radians)

NUM_CHAN = INTEGER (Given)
Number of channels per A/D card

NUM_ADC = INTEGER (Given)
the number of A/D cards

N_BOLS = INTEGER (Given)
the actual number of bolometers

WPLATE_ANG = REAL (Given)
nasmyth angle of waveplate for these data (degrees)

BOL_CHAN (N_BOLS) = INTEGER (Given)
channel numbers of bolometers

BOL_ADC (N_BOLS) = INTEGER (Given)
ADC numbers of bolometers

BOL_IP_DATA(8, NUM_CHAN, NUM_ADC) = REAL (Given)
The IP data. This array contains the ip data for each bolometer (specified by a combination of CHAN and ADC). The 4 slices represent: P0, Pslope, Theta0 and ThetaSlope The last 4 slices represent variance of each data point All angles are in radians.

BOL_DATA (N_BOLS) = REAL (Given and returned)
bolometer data

BOL_VAR (N_BOLS) = REAL (Given and returned)
variance of bolometer data

STATUS = INTEGER (Given and returned)
global status

Notes:

- The elevation of each bolometer is assumed to be the elevation of the central bolometer.
- Output variance is calculated
- degrees are used throughout since this simplifies the calculations when the gradients and ThetaZero are specified in degrees (or inverse degrees for the slope) from the input IP file.

Copyright :

SURFLIB_STATS_GRID
Calculate statistics of binned data

Description:

Calculate the statistics of the binned data and return the median and upper and lower limits of acceptable data.

Invocation:

```
CALL SURFLIB_STATS_GRID(SMODE, NX, NY, NMAX, NSIGMA, IPOS, JPOS, BINS, PNTS, STATS,
STATUS)
```

Arguments:

SMODE = CHARACTER (Given)

Smoothing mode. The statistics can be smoothed by adjacent pixels in order to remove spikes from the clipping envelope. Options are: NONE - No smoothing (ie the statistics of each bin) HANN - Hanning smoothing. Triangular envolope across 3 points. Note that smoothing depends on the way the grid was unwrapped. since that determines which pixels are adjacent (except for SMODE=NONE.

NX = INTEGER (Given)

Size of X dimension

NY = INTEGER (Given)

Size of Y dimension

NMAX = INTEGER (Given)

Maximum value allowed for third dimension of BINS

NSIGMA = REAL (Given)

Standard deviation limit.

IPOS(NX $*$ NY) = INTEGER (Given)

I coordinate for each pixel

JPOS(NX $*$ NY) = INTEGER (Given)

J coordinate for each pixel

BINS(NX, NY, NMAX) = REAL (Given)

The data stored in relation to its position

PNTS(NMAX) = REAL (Given)

Scratch space for copying in the data from each I,J

STATS(NX, NY, 3) = REAL (Returned)

Statistics of each bin. 3 components are: Median, Upper limit lower limit.

STATUS = INTEGER (Given & Returned)

Global status.

Copyright :

SURFLIB_TRIM_IMAGE
Sets quality bit for specified distance from exisiting bad data

Description:

Will take the quality array of a 2-D image and set the quality bit of all pixels within the specified trimming distance of a pixel that has bit 0 set. The output pixel bit is only set explicitly if bit 0 is not already set. This is so that the effect of the trim can be removed simply by changing the bad bits mask.

Invocation:

```
CALL SURFLIB_TRIM_IMAGE( TRIM, NX, NY, BITNUM, IN_QUAL, OUT_QUAL, STATUS)
```

Arguments:

TRIM = REAL (Given)
The distance in pixels inside which quality bits will be set.

NX = INTEGER (Given)
Number of X pixels in image

NY = INTEGER (Given)
Number of Y pixels in image

BITNUM = INTEGER (Given)
Bit number to set to bad if within TRIM distance of a pixel that has bit 0 set. Start counting at 0.

IN_QUAL(NX,NY) = UBYTE (Given)
Input image.

OUT_QUAL(NX,NY) = UBYTE (Returned)
Output image quality.

STATUS = INTEGER (Given and Returned)
Global status

Copyright :

G.3 SCULIB

This section describes the subroutines that are available in the `sculib` distribution (`libsculib.a`).

SCULIB_1D2_JIGGLE
routine to unpack a 1-d jiggle dataset into a 2-d image

Description:

This routine unpacks the data for a specified bolometer from a dataset stored according to jiggle number into a 2-d dataset with data stored according to jiggle offset. The input dataset can contain data either from all or part of ONE run through the jiggle pattern or for several REPEATS of the entire pattern. In the first case, the routine copies the input data into the appropriate part of the 2-d image. In the second, the 2-d datum for a given pixel will be the average of the input values for it, and the variance will calculated from the spread of input points around the mean unless only one input point contributed, in which case the input variance will be used.

Invocation:

```
CALL SCULIB_1D2_JIGGLE (BOL, N_BOLS, J_START, N_JIG, J_REPEAT, J_COUNT, J_DATA,
J_VARIANCE, J_QUALITY, IDIM, JDIM, I_JIGGLE, J_JIGGLE, DATA_2D, VARIANCE_2D, QUALITY_2D,
STATUS)
```

Arguments:

BOL = INTEGER (Given)
the number of the bolometer whose data is to be unpacked

N_BOLS = INTEGER (Given)
the number of bolometers measured

J_START = INTEGER (Given)
the index within the pattern of the first jiggle position measured

N_JIG = INTEGER (Given)
the total number of jiggle positions measured

J_REPEAT = INTEGER (Given)
the number of times the jiggle pattern was repeated in the dataset

J_COUNT = INTEGER (Given)
the number of jiggles in the pattern

J_DATA (N_BOLS, N_JIG)
= REAL (Given) the measured data at each jiggle position

J_VARIANCE (N_BOLS, N_JIG)
= REAL (Given) the measured variance

J_QUALITY (N_BOLS, N_JIG)
= INTEGER (Given) the quality on the measured data

IDIM = INTEGER (Given)
the x dimension of the 2-d map

JDIM = INTEGER (Given)
the y dimension of the 2-d map

I_JIGGLE (J_COUNT) = INTEGER (Given)
the 'i' index on the 2-d map of each jiggle position

J_JIGGLE (J_COUNT) = INTEGER (Given)
the 'j' index on the 2-d map of each jiggle position

DATA_2D (IDIM, JDIM) = REAL (Returned)
the data plane of the 2-d map

VARIANCE_2D (IDIM, JDIM) = REAL (Returned)
the variance plane

QUALITY_2D (IDIM, JDIM) = INTEGER (Returned)
the quality plane

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_2POS_CONFN
Generate a convolution function to remove the 2-position chop function from raster scans

Description:

This routine will return the asymmetric convolution function required for deconvolving a dual beam map into a single beam map. It is the convolution function that is derived in Emerson Klein and Haslam (1979) Astron. Astrophys. 76 p92 paper.

For the case where UNBAL = 1, the two beams are of equal strength:-

The ideal convolution function would be the one whose FT was the inverse of the FT of the chop function. Unfortunately, the chop function FT has zeroes in it, at which the inverse FT will tend to infinity. The ideal convolution function, therefore, does not exist. This problem is avoided by generating a function whose FT is the same as the ideal function except at the problem points, where it is set to zero. This function consists of a series of delta functions separated by the chop spacing, the central 2 points being half the chop spacing on either side of the centre of the function.

The function is normalised such that convolving this function with an original chop function of 2 delta functions of unit height will give a delta function of height 1.

Don't understand what happens when UNBAL <> 1, but the code has left doing the same as NOD2.

The convolution must cover the map even when the centre of the function is at the left or right hand extremity of the map. Hence the convolution function must be twice the length of the raw map.

Since the raw data is not sampled such that the chop spacing is an integer number of samples, the actual convolution function must be rebinned onto the sample mesh by sinc interpolation.

This routine is essentially a rewritten version of CONF22 from the RESTOR program in the NOD2 package by Haslam (1974) Astron. Astrophys. Suppl. 15 p333

Invocation:

```
CALL SCULIB_2POS_CONFN (BSEP, PIXSEP, NPIX, UNBAL, NCFN, CONF, STATUS)
```

Arguments:

BSEP = REAL (Given)
The beam separation in arcseconds

PIXSEP = REAL (Given)
The pixel separation in arcseconds

NPIX = INTEGER (Given)
The number of pixels in the x direction

UNBAL = REAL (Given)
The relative amplitudes of the right and left hand beams UNBAL=amp(lhb)/abs(amp(rhb))

NCFN = INTEGER (Returned)
The length of the convolution array

CONF(*) = REAL (Returned)
The convolution function

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_2POS_DECONV
deconvolve square chop from scan

Description:

This routine deconvolves the chopped beam response from the individual scans of a SCUBA raster map. To achieve this it cycles through the scans making up the observation, calling SCULIB_FIND_SWITCH to locate the start and finish indices of each scan in the demodulated data array. If SCULIB_FIND_SWITCH indicates that there is no data for this scan, which might happen if the observation was aborted, then no further action is taken. If the scan is too long to be handled by the routine an error message will be output and the routine will return with bad status. Otherwise, SCULIB_GENSYCONFN and SCULIB_2POS_CONFN will be called to generate the convolution functions needed to deconvolve the chop. The routine then cycles through the bolometers, calling SCULIB_CONVOLVE to do the required convolutions with the scan data for each.

Invocation:

```
CALL SCULIB_2POS_DECONV (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS, DEMOD_POINTER,
N_BOL, N_POS, IN_DATA, IN_VARIANCE, IN_QUALITY, SAMPLE_DX, BEAM_SEP, OUT_DATA,
OUT_VARIANCE, OUT_QUALITY, STATUS)
```

Arguments:

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)
array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the variance on IN_DATA

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

SAMPLE_DX = REAL (Given)
the measurement spacing along the scan (arcseconds)

BEAM_SEP = REAL (Given)
the beam separation (arcseconds)

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the deconvolved data

OUT_VARIANCE (N_BOL, N_POS) = REAL (Returned)
the variance on OUT_DATA

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_3POS_CONFN
Generate a convolution function to remove the 3-position chop function from raster scans

Description:

This routine will return the convolution function required for deconvolving the 3-position chop from a scan.

This function consists of a series of delta functions at a spacing equal to that between a -ve spike and the central spike of the 3-position chop. The height of the delta dunctions peaks at the centre of the convolution function and falls off by 1 at a time for the delta functions on either side, so that the envelope of the function is an isosceles triangle.

The convolution function must cover the scan even when the centre of the function is at one end of it. Hence, the convolution function must be twice the length of the scan.

Since the raw data are not sampled such that the chop spacing is an integer number of samples, the actual convolution function must be rebinned onto the sample mesh by sinc interpolation.

Invocation:

```
CALL SCULIB_3POS_CONFN (BSEP, PIXSEP, NPIX, NCFN, CONF, STATUS)
```

Arguments:

BSEP = REAL (Given)
The beam separation in arcseconds

PIXSEP = REAL (Given)
The pixel separation in arcseconds

NPIX = INTEGER (Given)
The number of pixels in the x direction

NCFN = INTEGER (Returned)
The length of the convolution array

CONF($*$) = REAL (Returned)
The convolution function

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_3POS_DECONV
deconvolve 3-position chop from scan

Description:

This routine deconvolves the 3 position chopped beam response from the individual scans of a SCUBA raster map. To achieve this it cycles through the scans making up the observation, calling SCULIB_FIND_SWITCH to locate the start and finish indices of each scan in the demodulated data array. If SCULIB_FIND_SWITCH indicates that there is no data for this scan, which may happen if the observation was aborted, then no further action is taken. If a scan is too long to be handled by the routine an error message will be output and the routine will return with bad status. Otherwise, SCULIB_GENSYCONFN and SCULIB_3POS_CONFN will be called to generate the convolution functions needed to deconvolve the chop. The routine then cycles through the bolometers, calling SCULIB_CONVOLVE to do the required convolutions with the scan data for each.

Invocation:

```
CALL SCULIB_3POS_DECONV (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS, DEMOD_POINTER,
N_BOL, N_POS, IN_DATA, IN_VARIANCE, IN_QUALITY, SAMPLE_DX, BEAM_SEP, OUT_DATA,
OUT_VARIANCE, OUT_QUALITY, STATUS)
```

Arguments:

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)
array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the variance on IN_DATA

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

SAMPLE_DX = REAL (Given)
the measurement spacing along the scan (arcseconds)

BEAM_SEP = REAL (Given)
the beam separation (arcseconds)

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the deconvolved data

OUT_VARIANCE (N_BOL, N_POS) = REAL (Returned)
the variance on OUT_DATA

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_ADD_CHOP
Add chop throw to apparent RA/Dec

Description:

This routine takes a chop throw and adds it on the supplied apparent RA/Dec centre position returning a new apparent RA/Dec. Works with scan map and jiggle map but does not actually know the difference itself (unless using SC chop). For EKH (SC) chopping we have to convert the scan ends to tangent plane offsets from the reference centre before calculating the angle.

Invocation:

```
CALL SCULIB_ADD_CHOP(BEAM, RA_REF_CENTRE, DEC_REF_CENTRE, RA_CENTRE, DEC_CENTRE,
CHOP_CRD, CHOP_PA, CHOP_FUN, CHOP_THROW, LST, MJD, LAT_OBS, RA_START, RA_END,
DEC_START, DEC_END, OUT_RA_CEN, OUT_DEC_CEN, STATUS)
```

Arguments:

BEAM = CHARACTER * (*) (Given)
Beam name ('M','L' or 'R')

RA_REF_CENTRE = DOUBLE PRECISION (Given)
Apparent RA of the map centre. This is the centre that scan map tangent plane offsets are calculated from.

DEC_REF_CENTRE = DOUBLE PRECISION (Given)
Apparent dec of the map centre. This is the centre that scan map tangent plane offsets are calculated from.

RA_CENTRE = DOUBLE PRECISION (Given)
the apparent RA of the 'centre' of the array (radians)

DEC_CENTRE = DOUBLE PRECISION (Given)
the apparent dec of the 'centre' of the array (radians)

CHOP_CRD = CHARACTER *(*) (Given)
Coordinate system of CHOP throw (SC,AZ,RB,RJ,GA)

CHOP_PA = REAL (Given)
Position angle of chop (in radians)

CHOP_FUN = CHARACTER * (*) (Given)
Chop function (CENTER, SQUARE, TRIPOS)

CHOP_THROW = REAL (Given)
Chop throw in radians

LST = DOUBLE PRECISION (Given)
the local sidereal time (radians)

MJD = DOUBLE PRECISION (Given)
MJD of observation

LAT_OBS = DOUBLE PRECISION (Given)
the latitude of the observatory (radians)

RA_START = REAL (Given)
RA of start of scan (if SAMPLE_MODE=RASTER)

RA_END = REAL (Given)
RA of end of scan (if SAMPLE_MODE=RASTER)

DEC_START = REAL (Given)
DEC of start of scan (if SAMPLE_MODE=RASTER)

DEC_END = REAL (Given)
DEC of end of scan (if SAMPLE_MODE=RASTER)

OUT_RA_CEN = DOUBLE (Returned)
New position with CHOP

OUT_DEC_CEN = DOUBLE (Returned)
New DEC with CHOP

STATUS = INTEGER (Given and returned)
The global status

Notes:

It is assumed that the chop throw is divided by two prior to calling this routine for scan map mode since the chop is effectively half a chop either side of the middle. For jiggle modes the chop is effectively a full chop throw from the centre since the 3-beam chopping is done by a combination of chopping and nodding.

- This routine does not yet support TRIPOS chopping (I think)
- AZ, LO and SC chopping are supported
- The negative beam is always the 'left' beam, the +ve beam is the 'right'.
- The chop tracking 'droopy' beam problem should be dealt with in an earlier subroutine such that the chop is converted from LO to AZ before calling this routine

Copyright :

SCULIB_ADD_DEMOD_EXPOSURE
add demodulated data for an exposure into the integration result

Description:

This routine adds the reduced demodulated data for an exposure in a jiggle-sampled observation into the integration result. The exposure can contain data either for all or part of ONE run through the jiggle pattern or for several REPEATS of the entire jiggle pattern. In the first case, the routine copies the exposure data into the appropriate part of the integration arrays. In the second, the integration data will be the average of the input exposure values and the integration variance will be set equal to the exposure value if there was just one valid measurement, or calculated from the dispersion of the exposure data about the mean otherwise.

Invocation:

```
CALL SCULIB_ADD_DEMOD_EXPOSURE (N_BOLS, J_START, N_JIG, J_REPEAT, J_COUNT, EXP_DATA,
EXP_VARIANCE, EXP_QUALITY, INT_DATA, INT_VARIANCE, INT_QUALITY, STATUS)
```

Arguments:

N_BOLS = INTEGER (Given)
the number of bolometers measured

J_START = INTEGER (Given)
the index within the pattern of the first jiggle position measured

N_JIG = INTEGER (Given)
the number of jiggle positions measured

J_REPEAT = INTEGER (Given)
the number of times the measured pattern was repeated

J_COUNT = INTEGER (Given)
the number of jiggle positions in the entire pattern

EXP_DATA (N_BOLS, N_JIG) = REAL (Given)
the data for the exposure

EXP_VARIANCE (N_BOLS, N_JIG) = REAL (Given)
the exposure variance

EXP_QUALITY (N_BOLS, N_JIG) = INTEGER (Given)
the exposure quality

INT_DATA (N_BOLS, J_COUNT) = REAL (Returned)
the integration data

INT_VARIANCE (N_BOLS, J_COUNT) = REAL (Returned)
the integration variance

INT_QUALITY (N_BOLS, J_COUNT) = INTEGER (Returned)
the integration quality

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_ADDARE
add one real array to another into a third

Description:

Adds two real arrays. Note that any of the arrays may be the same.

Invocation:

```
CALL SCULIB_ADDARE (N, ARRAY1, ARRAY2, ARRAY3, Q1DATA, Q2DATA, Q3DATA, V1DATA,
V2DATA, V3DATA, QUALITY, FLAGGED, VARIANCE)
```

Arguments:

N = INTEGER (Given)
Number of elements in each array

ARRAY1 (N) = REAL (Given)
Input array

ARRAY2 (N) = REAL (Given)
Second input array

ARRAY3 (N) = REAL (Returned)
Result array. ARRAY3 = ARRAY1 + ARRAY2

Q1DATA (N) = INTEGER (Given)
Quality array for first input array

Q2DATA (N) = INTEGER (Given)
Quality array for second input array

Q3DATA (N) = INTEGER (Returned)
Quality array for output array

V1DATA (N) = REAL (Given)
Variance array for first input array

V2DATA (N) = REAL (Given)
Variance array for second input array

V3DATA (N) = REAL (Returned)
Variance array for output array

QUALITY = LOGICAL (Given)
True if input has quality information

FLAGGED = LOGICAL (Given)
True if input has flagged data values

VARIANCE = LOGICAL (Given)
True if both input arrays have variance arrays

Notes:

- Does not use Quality correctly. Uses INTEGER quality rather than UBYTE

Copyright :

Implementation Status:

- Propagates variance
- Checks for bad values

SCULIB_ADDCAD
add a constant to a double array

Description:
adds a real constant to a double array

Invocation:

```
CALL SCULIB_ADDCAD (N, IN, RVAL, OUT)
```

Arguments:

N = INTEGER (Given)
number of elements in arrays

IN (N) = DOUBLE PRECISION (Given)
input array

RVAL = DOUBLE PRECISION (Given)
real constant to be added to array

OUT (N) = DOUBLE PRECISION (Returned)
output array (may be same as input)

Notes:

- No range checks are performed
- No status checking
- No checks for bad values

Copyright :

SCULIB_ADDCAI
add a constant to an integer array

Description:

adds an integer constant to an integer array

Invocation:

```
CALL SCULIB_ADDCAI (N, IN, IVAL, OUT)
```

Arguments:

N = INTEGER (Given)
number of elements in arrays

IN (N) = INTEGER (Given)
input array

IVAL = INTEGER (Given)
constant to be added to array

OUT (N) = INTEGER (Returned)
output array (may be same as input)

Notes:

- No range checks are performed
- No status checking
- No checks for bad values

SCULIB_ADDCAR
add a constant to a real array

Description:

adds a real constant to a real array

Invocation:

```
CALL SCULIB_ADDCAR (N, IN, RVAL, OUT)
```

Arguments:

N = INTEGER (Given)
number of elements in arrays

IN (N) = REAL (Given)
input array

RVAL = REAL (Given)
real constant to be added to array

OUT (N) = REAL (Returned)
output array (may be same as input)

Notes:

- No range checks are performed
- No status checking
- No checks for bad values

Copyright :

SCULIB_AIRMASS
calculate the airmass for a given zenith distance

Description:

This routine calculates the airmass corresponding to the input zenith distance. The airmass is just sec(Z) until it reaches 2, after which a more complex algorithm developed by Ian Coulson is used (it is described by comments in the code). There may be a discontinuity in returned airmasses around the value 2, I haven't checked.

Invocation:

```
CALL SCULIB_AIRMASS (Z, AIRMASS, STATUS)
```

Arguments:

Z = REAL (Given)
zenith distance (radians)

AIRMASS = REAL (Returned)
airmass

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_ANALYSE_PHOTOM_JIGGLE
analyse the jiggle map made by a bolometer during a PHOTOM observation

Description:

This routine analyses the jiggle map made by a bolometer during a PHOTOM observation. If status is good on entry the routine checks that the data array holds data for the specified bolometer. If it does not an error message will be output and the bad status returned. All being well the routine will analyse the data by the method specified in PHOTOM_ANALYSIS:-

AVERAGE - the 'result' will be the mean of the jiggle points. The variance is the sum of the variances of the averaged data divided by the number of them squared. The result will be flat-fielded and reported to the user. If no valid data were available for the calculation of the mean then a warning message will be output but good status returned.

PARABOLA - the routine will call SCULIB_FIT_2D_PARABOLA to fit a 2-d parabola to the data and the 'result' will be the peak of the fitted curve. If the fit is good the results will be flat-fielded and reported to the user. Otherwise the result quality is set bad but good status will be returned.

Invocation:

```
CALL SCULIB_ANALYSE_PHOTOM_JIGGLE (PHOTOM_ANALYSIS, BOL, N_BOLS, J_COUNT, JDATA,
VARIANCE, QUALITY, RESULT_D, RESULT_V, RESULT_Q, BADBIT, STATUS)
```

Arguments:

PHOTOM_ANALYSIS = CHARACTER*(*) (Given)
the method of analysis to be applied to the data

BOL = INTEGER (Given)
the index of the bolometer whose data is to be analysed

N_BOLS = INTEGER (Given)
the number of bolometers being measured

J_COUNT = INTEGER (Given)
the number of jiggles in the pattern

JIGGLE_X = REAL (Given)
x jiggle offsets

JIGGLE_Y = REAL (Given)
y jiggle offsets

JDATA (N_BOLS, J_COUNT) = REAL (Given)
the measured data

VARIANCE (N_BOLS, J_COUNT) = REAL (Given)
the variance

QUALITY (N_BOLS, J_COUNT) = BYTE (Given)
the quality

RESULT_D = REAL (Returned)
the sum of the input data

RESULT_V = REAL (Returned)
the sum of the input variances

RESULT_Q = BYTE (Returned)
the quality on the sum

A0 = REAL (Returned)
fitted parabola parameter

A1 = REAL (Returned)
fitted parabola parameter

X0 = REAL (Returned)
x offset of peak of fitted parabola

Y0 = REAL (Returned)
y offset of peak of fitted parabola

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_APPARENT_2_MP
Calculate mean place from a given apparent RA,Dec

Description:

This routine takes the apparent RA,Dec as input and converts them to the position in a specified output coordinate system. Allowed OUTPUT_COORDS are AZ, RJ, RB, GA, EQ. This is the reverse of SCULIB_CALC_APPARENT

Invocation:

```
CALL SCULIB_APPARENT_2_MP(RA_APP, DEC_APP, OUT_COORDS, LST, MJD, LAT_OBS, LONG,
LAT, STATUS )
```

Arguments:

RA_APP = DOUBLE PRECISION (Given)
Apparent RA of point at date (radians)

DEC_APP = DOUBLE PRECISION (Given)
Apparent Dec

OUT_COORDS = CHARACTER * (*) (Given)

LST = DOUBLE PRECISION (Given)
LST for requested coordinates (for AZ and HA)

MJD = DOUBLE PRECISION (Given)
Modified Julian date of observation

LAT_OBS = DOUBLE PRECISION (Given)
Latitude of observatory in radians. For JCMT this value is 3.46026051751D-1

LONG = DOUBLE PRECISION (Returned)
longitude of centre in input coord system (radians)

LAT = DOUBLE PRECISION (Returned)
latitude of centre in input coord system (radians)

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_APPARENT_2_TP
calculate tangent plane coordinates from apparent RA, Decs

Description:

This routine converts a list of apparent RA,Decs to tangent plane offsets from a tangent point whose position is also given in apparent RA,Dec. In addition, a rotation is applied to the tangent plane so that it can be aligned with a coordinate system other than apparent RA,Dec, and a shift is added to allow the map to be moved about in that output frame.

Invocation:

```
CALL SCULIB_APPARENT_2_TP (N_POS, BOL_XPOS, BOL_YPOS, RA_CEN, DEC_CEN, ROTATION,
SHIFT_DX, SHIFT_DY, STATUS)
```

Arguments:

N_POS = INTEGER (Given)
the number of positions to be converted

BOL_XPOS (N_POS) = DOUBLE PRECISION (Given and returned)
apparent RA on input (radians), x tangent plane offset on output (radians)

BOL_YPOS (N_POS) = DOUBLE PRECISION (Given and returned)
apparent Dec on input (radians), y tangent plane offset on output (radians)

RA_CEN = DOUBLE PRECISION (Given)
apparent RA of tangent point (radians)

DEC_CEN = DOUBLE PRECISION (Given)
apparent Dec of tangent point (radians)

ROTATION = DOUBLE PRECISION (Given)
angle between output North and apparent North (radians, measured anticlockwise from output North)

SHIFT_DX = DOUBLE PRECISION (Given)
value to be added to x offsets (radians)

SHIFT_DY = DOUBLE PRECISION (Given)
value to be added to y offsets (radians)

STATUS = INTEGER (Given and returned)

SCULIB_BESSEL_WTINIT
Generate a weighting function for rebinning

Description:

This is a FORTRAN version of the C code written for the transputer rebinning. Here is the C description: Initialise the BESSEL weighting function for Bessel interpolation. The 1-D function 2.0∗J1(x)/x is initialised in the declaration, and the weighting function lookup table is calculated from this. The tabulation is at intervals of pi/RES for the range 0.0 to RADIUS∗pi, resulting in (RES∗RADIUS)∗∗2 values. The weight function is also multiplied by a cosine over the oute third to reduce edge effects.

The look up table is initialised such that one can access the weight directly given the square of the distance between the input and output pixel. We use a fixed resolution out to 10PI.

Invocation:

```
CALL SCULIB_BESSEL_WTINIT( WTFN, RADIUES, RES, STATUS)
```

Arguments:

WTFN (RADIUS ∗ RADIUS ∗ RES ∗ RES + 1) = REAL (Returned)
The weighting function generated by this routine. The index corresponds to the square of the distance from the centre in scale units.

RADIUS = INTEGER (Given)
Size of the weighting function in scale units.

RES = INTEGER (Given)
Number of points per scale lenghth.

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_BESSJ1
calculates Bessel function J1(x)

Description:

If status is good on entry this function returns the Bessel function J1(x) for any real x.

Invocation:

```
Y = SCULIB_BESSJ1 (X, STATUS)
```

Arguments:

X = REAL (Given)
the argument of the J1 Bessel function

STATUS = INTEGER (Given and returned)
global status

Returned Value:

SCULIB_BESSJ1 = REAL
Bessel function J1(X)

Notes:

Probably should be changed to use PDA_DBESJ1 (SUN/194). That routine is DOUBLE PRECISION.

Method :

Uses an algorithm from Numerical Recipes in Fortran.

Copyright :

SCULIB_BITON
turn on a bit

Description:

Turns on a bit in a byte.

Invocation:

```
NEWBYTE = SCULIB_BITON( VAL, BIT )
```

Arguments:

VAL = BYTE (Given)
The byte to be changed

BIT = INT (Given)
The bit to be turned on

Returned Value:

SCULIB_BITON = BYTE
Modified byte.

Copyright :

SCULIB_BITOFF
Turn off a bit

Description:

Turns off a bit in a byte

Invocation:

```
NEWBYTE = SCULIB_BITOFF ( VAL, BIT )
```

Arguments:

VAL = BYTE (Given)

The byte to be changed

BIT = INT (Given)

The bit to be turned off

Returned Value:

SCULIB_BITOFF = BYTE

Modified byte.

Copyright :

SCULIB_BITOR
Returns the bitwise OR of two bytes

Description:

Returns the bitwise OR

Invocation:

```
LOGOR = SCULIB_BITOR(VAL1, VAL2)
```

Arguments:

VAL1 = BYTE (Given)
First byte

VAL2 = BYTE (Given)
Second byte

Returned Value:

SCULIB_BITOR = BYTE
OR of bytes

Copyright :

SCULIB_BITAND
Calculate the bitwise AND of two bytes

Description:

Returns the bitwise AND

Invocation:

```
LOGAND = SCULIB_BITAND( VAL1, VAL2 )
```

Arguments:

VAL1 = BYTE (Given)
First byte

VAL2 = BYTE (Given)
Second byte

Returned Value:

SCULIB_BITAND = BYTE
AND of bytes.

Copyright :

SCULIB_BITTEST
Test whether a bit is on

Description:

Tests whether specified bit is turned on or not. Returns TRUE if it is on, false otherwise.

Invocation:

```
ISON = SCULIB_BITTEST( VAL, BIT )
```

Arguments:

VAL = BYTE (Given)
The byte to be tested

BIT = INT (Given)
The bit to be tested

Returned Value:

SCULIB_BITTEST = LOGICAL
True if specified bit is turned on.

Copyright :

SCULIB_BOLDECODE
decode a bolometer ID into ADC and channel number

Description:

Given a character string containing a bolometer ID, put ADC and CHANNEL number into output variables. The syntax of the bolometer ID is a3, A3 or 3. Allowed ADCs run from a-i. Channels must be in range 1-16. Bolometer IDs outside these ranges will cause an error to be returned.

Invocation:

```
CALL SCULIB_BOLDECODE (BOLCODE, ADC, CHANNEL, STATUS)
```

Arguments:

BOLCODE = CHARACTER*(*) (given)
bolometer ID

ADC = INTEGER (Returned)
ADC number

CHANNEL = INTEGER (Returned)
channel number

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_BOLNAME
generate bolometer name from ADC and channel number

Description:

Given bolometer ADC and channel, generate the bolometer name. E.g. ADC=2, CHAN=5 gives B5. If the ADC number is outside the range 1-9, or the channel number outside 1-16, and error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_BOLNAME (ADC, CHANNEL, BOLCODE, STATUS)
```

Arguments:

ADC = INTEGER (Given)
ADC number

CHANNEL = INTEGER (Given)
channel number

BOLCODE = CHARACTER*(*) (Returned)
bolometer ID

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_BOLSELECT
interpret a list of selected bolometers

Description:

Interpret a character string containing a list of bolometers, put number of bolometers to be used in N_BOL_SELECT, channel and ADC of selected bolometers in BOL_SELECT_CHAN and BOL_SELECT_ADC respectively. BOL_ENABLED entries will also be set .TRUE. for bolometers that have been selected. The number of sub-instruments involved is returned in N_SUB, and the names of the sub-instruments are given in SUB_INSTRMNT. The Nasmyth coords of the point on the focal plane to be treated as the axis of the instrument will be returned in CENTRE_DU3, CENTRE_DU4.

The given string can either be ALL, in which case all bolometer channels will be selected, or be a list of words specifying parts of the 144 channel array. Each word in such a list must be separated from the others by ',', and the words can select bolometers by type - SHORT, LONG, P1100, P1300, P2000, SHORT_DC, LONG_DC, P1100_DC, P1300_DC, P2000_DC, or by channel number, either individually or as a range, specified by ADC letter and channel number e.g. a3. Allowed ADCs are A-I and channels 1-16.

Repeated words in the list will be ignored, as will repeated selections of the same bolometer by different routes.

The instrument sections returned in SUB_INSTRMNT will be arranged in the order SHORT, LONG, P1100, P1300, P2000.

CENTRE_DU3, CENTRE_DU4 will be set equal to ARRAY_CENTRE_DU3, ARRAY_ CENTRE_DU4 if the first word in the selection is SHORT, LONG, SHORT_DC or LONG_DC. They will be set to the offsets of the bolometer concerned if the first word in the selection is P1100, P1300, P2000 or a bolometer name like A7. If either of the coordinates ends up being set to the 'bad' value, they will be reset to zero and a warning message output.

Errors will be reported and bad status returned if -

- there are too many words in BOLOMETERS
- BOLOMETERS is empty
- if when bolometers are selected by type no detectors of the desired type are found in the database
- if the type selected refers to a single bolometer (P1100, P1300, P2000, SHORT_DC, LONG_DC, P1100_DC, P1300_DC, P2000_DC) more than bolometer of the required type is found in the database
- if bolometers are selected by ID, e.g. a15, the ID or range of IDs is bad

Warnings will be reported but good status returned if -

- any selected bolometer has undefined attributes (type, calib, dU3, or dU4)

- CENTRE_DU3, CENTRE_DU4 are set to 'bad' values (they will be reset to 0 too)

Invocation:

```
CALL SCULIB_BOLSELECT (BOLOMETERS, BOL_TYPE, BOL_CALIB, BOL_DU3, BOL_DU4, BOL_QUAL,
BOL_ENABLED, NUM_CHAN, NUM_ADC, ARRAY_CENTRE_DU3, ARRAY_CENTRE_DU4, BOL_SELECT_CHAN,
BOL_SELECT_ADC, N_BOL_SELECT, MAX_SUB, SUB_INSTRMNT, N_SUB, CENTRE_DU3, CENTRE_DU4,
STATUS)
```

Arguments:

BOLOMETERS = CHARACTER∗(∗) (Given)
list of bolometer selections

BOL_TYPE (NUM_CHAN, NUM_ADC)
= CHARACTER∗(∗) (Given) type of bolometer

BOL_CALIB (NUM_CHAN, NUM_ADC)
= REAL (Given) target calibrator values for bolometers

BOL_DU3 (NUM_CHAN, NUM_ADC) = REAL (Given)
Nasmyth dU3 offset of bolometer from field centre

BOL_DU4 (NUM_CHAN, NUM_ADC) = REAL (Given)
Nasmyth dU4 offset of bolometer from field centre

BOL_QUAL (NUM_CHAN, NUM_ADC)
= INTEGER (Given) quality of bolometers

BOL_ENABLED (NUM_CHAN, NUM_ADC)
= LOGICAL (Returned) .TRUE. if bolometer was selected

NUM_CHAN = INTEGER (Given)
number of channels per A/D

NUM_ADC = INTEGER (Given)
number of A/D cards

ARRAY_CENTRE_DU3 = REAL (Given)
the DU3 offset of the centre to be used for either array

ARRAY_CENTRE_DU4 = REAL (Given)
the DU4 offset of the centre to be used for either array

BOL_SELECT_CHAN (NUM_CHAN ∗ NUM_ADC)
= INTEGER (Returned) channel numbers of selected bolometers

BOL_SELECT_ADC (NUM_CHAN ∗ NUM_ADC)
= INTEGER (Returned) A/D card numbers of selected bolometers

N_BOL_SELECT = INTEGER (Returned)
total number of bolometers selected

MAX_SUB = INTEGER (Given)
maximum number of sub-instruments

SUB_INSTRMNT (MAX_SUB) = CHARACTER∗(∗) (Returned)
names of instrument sections being used

N_SUB = INTEGER (Returned)
the number of sub-instruments being used

CENTRE_DU3 = REAL (Returned)
the DU3 of the instrument 'centre'

CENTRE_DU4 = REAL (Returned)
the DU4 of the instrument 'centre'

STATUS = INTEGER (Given and returned)
global status

Notes:

The routine will not work properly if the input BOLOMETERS string is more than 400 characters long.

Copyright :

SCULIB_CALC_APPARENT
calculate apparent RA, Dec of plate centre and angle of input coord system N relative to apparent N

Description:

This routine takes the input coordinates and coordinate system of the map centre and converts them to the apparent coords at the time of the observation. In addition, the angle between the north direction in the input coordinate frame and that in the apparent frame is calculated (measured anti-clockwise from input north, in radians). See SCU/3.0/JFL/0393.

- [AZ] coords: Calculate the apparent RA and DEC at the LST when the routine is called by:-

$$\sin(dec\_app) = \sin(lat\_obs) * \sin(el) + \cos(lat\_obs) * \cos(el) * \cos(az) \quad (4)$$

$$\sin(H\_A) = -\frac{\sin(az) * \cos(el)}{\cos(dec\_app)} \quad (5)$$

$$\cos(H\_A) = \frac{\sin(el) - \sin(dec\_app) * \sin(lat\_obs)}{\cos(dec\_app) * \cos(lat\_obs)} \quad (6)$$

$\cos(dec\_app)$ is present in the denominator of both the $\sin(H\_A)$ and $\cos(H\_A)$ terms and it could cause both to blow up – so it's left out as only the ratio is important

$$RA\_app = LST - H\_A \quad (7)$$

$$\sin(rotation) = \frac{\sin(az) * \cos(lat\_obs)}{\cos(dec\_app)} \quad (8)$$

$$\cos(rotation) = \frac{\sin(lat\_obs) - \sin(dec\_app) * \sin(el)}{\cos(dec\_app) * \cos(el)} \quad (9)$$

Again, $\cos(dec\_app)$ appears in the denominator of both sin and cos expressions and is left out of the calculation because on ly the ratio is important.

- [RB] coords: Use SLA_FK54Z to convert to RJ. Use SLA_MAP to convert to apparent, giving ra_app, dec_app. Use same method to calculate apparent position of RB N pole, giving ra_N_app, dec_N_app. Then calculate rotation from:-

$$\mathrm{d}RA = ra\_app - ra\_N\_app \quad (10)$$

$$\sin(rotation) = \frac{\sin(\mathrm{d}RA) * \cos(dec\_N\_app)}{\cos(lat)} \quad (11)$$

$$\cos(rotation) = \frac{\sin(dec\_N\_app) - \sin(dec\_app) * \sin(lat)}{\cos(dec\_app) * \cos(lat)} \quad (12)$$

Since $\cos(lat)$ is in the denominator for both sin and cos terms, is always +ve except for $\pm\pi/2$ where it goes to zero and blows up the equations, leave it out in the calculations. The ratio of sin and cos will be unaffected.
If dec_app = $\pi/2$ (i.e. at N pole of apparent system) then

$$rotation = \pi - (ra\_N\_app - ra\_app) \quad (13)$$

- [RJ] coords: Use SLA_MAP to convert to apparent. Use same method to calculate apparent position of RB N pole. Derive rotation angle in the same way as for RB.
- [GA] coords: Use SLA_GALEQ to convert to RJ. Use SLA_MAP to convert to apparent, giving ra_app, dec_app. Use same method to calculate apparent position of GA N pole, giving ra_N_app, dec_N_app. Derive rotation angle in the same way as for RB.
- [EQ] coords: Use SLA_ECLEQ to convert to RJ. Use SLA_MAP to convert to apparent, giving ra_app, dec_app. Use same method to calculate apparent position of EQ N pole, giving ra_N_app, dec_N_app. Derive rotation angle in the same way as for RB.
- [HA] coords: Apparent RA = LST - LONG
  Apparent Dec = LAT
  Rotation = 0.0D0
- [RD] coords: Apparent RA, Dec set to input values. Rotation = 0.0.
- [PLANET] coords: If MJD1 = MJD2 then apparent RA, Dec set to input LONG, LAT. Rotation = 0.0. Otherwise apparent RA, Dec interpolated (or extrapolated) between LONG, LAT, MJD1 and LONG2, LAT2, MJD2 according to MJD. Rotation = 0.

Invocation:

```
CALL SCULIB_CALC_APPARENT (LAT_OBS, LONG, LAT, LONG2, LAT2, MAP_X, MAP_Y, COORD_TYPE,
LST, MJD, MJD1, MJD2, RA_APP, DEC_APP, ROTATION, STATUS)
```

Arguments:

LAT_OBS = DOUBLE PRECISION (Given)
latitude of observatory (radians)

LONG = DOUBLE PRECISION (Given)
longitude of centre in input coord system (radians)

LAT = DOUBLE PRECISION (Given)
latitude of centre in input coord system (radians)

LONG2 = DOUBLE PRECISION (Given)
longitude of second centre in PLANET coord system (radians)

LAT2 = DOUBLE PRECISION (Given)
latitude of second centre in PLANET coord system (radians)

MAP_X = DOUBLE PRECISION (Given)
x tangent plane offset of point from centre (radians) The offset must be in the same coordinate as COORD_TYPE

MAP_Y = DOUBLE PRECISION (Given)
y tangent plane offset of point from centre (radians) The offset must be in the same coordinate as COORD_TYPE

COORD_TYPE = CHARACTER∗(∗) (Given)
Coord system of input centre, RD, RB, RJ, GA, EQ, PLANET

LST = DOUBLE PRECISION (Given)
LST for requested coordinates (for AZ and HA)

MJD = DOUBLE PRECISION (Given)
Modified Julian date of observation

MJD1 = DOUBLE PRECISION (Given)
Modified Julian date of first centre in PLANET coord system

MJD2 = DOUBLE PRECISION (Given)
Modified Julian date of second centre in PLANET coord system

RA_APP = DOUBLE PRECISION (Returned)
Apparent RA of point at date (radians)

DEC_APP = DOUBLE PRECISION (Returned)
Apparent Dec

ROTATION = DOUBLE PRECISION (Returned)
Angle between apparent north and north of input coord system (radians, measured clockwise from input north)

STATUS = INTEGER (Given and returned)
Global status

Notes:

Does not handle LOCAL_COORDS for MAP_X and MAP_Y (see SCULIB_APARRENT_2_MP for information on how to do this)

Copyright :

SCULIB_CALC_BOL_COORDS
Calculate the bolometer offsets in (apparent RA,DEC), AzEl or NA

Description:

This routine calculates the apparent RA and dec of a specified set of bolometers. It does this by:-

- calculating the offsets that must be added to the bolometer positions to cater for the fact that the origin of the bolometer coordinate system may be offset from the 'centre' specified by RA_CENTRE , DEC_CENTRE.
- Three types of offsets may be added; Nasmyth offsets which are simply subtracted from the bolometer coordinates, azimuth offsets which are subtracted from the bolometer positions in az and el, or offsets in a coordinate system fixed relative to the sky, rotated relative to apparent RA,Dec by the angle ROTATION. The latter are added to the bolometer coordinates when they are in the form of tangent plane coords in apparent RA,Dec.
- working out the elevation and parallactic angle for the sidereal time and apparent RA, dec of the 'centre'.
- calculating the Nasmyth offset of each bolometer, then rotating them into tangent plane offsets in azimuth and elevation.
- adding pointing corrections in azimuth and elevation, then rotating the coords into the apparent RA,Dec tangent plane.
- calling SLA_DTP2S to work out the apparent RA, dec of the offset positions.

Invocation:

```
CALL SCULIB_CALC_BOL_COORS (OUT_COORDS,RA_CENTRE, DEC_CENTRE, LST, LAT_OBS, OFFSET_COORDS,
OFFSET_X, OFFSET_Y, ROTATION, N_POINT, MAX_POINT, POINT_LST, POINT_DAZ, POINT_DEL,
NUM_CHAN, NUM_ADC, N_BOL, BOL_CHAN, BOL_ADC, U3, U4, U3_CENTRE, U4_CENTRE, X_BOL,
Y_BOL, ELEVATION, PAR_ANGLE, STATUS)
```

Arguments:

OUT_COORDS = CHARACTER * (*) (Given)
Output coordinate system (NA, AZ, RA)

RA_CENTRE = DOUBLE PRECISION (Given)
the apparent RA of the 'centre' (radians)

DEC_CENTRE = DOUBLE PRECISION (Given)
the apparent dec of the 'centre' (radians)

LST = DOUBLE PRECISION (Given)
the local sidereal time (radians)

LAT_OBS = DOUBLE PRECISION (Given)
the latitude of the observatory (radians)

OFFSET_COORDS = CHARACTER*(*) (Given)
the coordinate system of the offset of the array origin from the 'centre'; NA or RD or AZ

OFFSET_X = REAL (Given)
the x offset of the array origin from the 'centre' (arcseconds)

OFFSET_Y = REAL (Given)
the y offset of the array origin from the 'centre' (arcseconds)

ROTATION = DOUBLE PRECISION (Given)
for OFFSET_COORDS other than NA, this gives the angle from N in the offset coordinate system to N in apparent RA,Dec (radians, increasing clockwise)

N_POINT = INTEGER (Given)
number of elements used in pointing correction arrays

MAX_POINT = INTEGER (Given)
dimension of pointing correction arrays

POINT_LST (MAX_POINT) = DOUBLE PRECISION (Given)
LST of measured corrections (radians)

POINT_DAZ (MAX_POINT) = REAL (Given)
correction to be added in azimuth (arcsec)

POINT_DEL (MAX_POINT) = REAL (Given)
correction to be added in elevation (arcsec)

NUM_CHAN = INTEGER (Given)
the number of channels per A/D card

NUM_ADC = INTEGER (Given)
the number of A/D cards

N_BOL = INTEGER (Given)
the actual number of bolometers

BOL_CHAN (N_BOL) = INTEGER (Given)
channel numbers of bolometers

BOL_ADC (N_BOL) = INTEGER (Given)
ADC numbers of bolometers

U3 (NUM_ADC,NUM_CHAN) = REAL (Given)
the U3 offsets of the bolometers (arcsec)

U4 (NUM_ADC,NUM_CHAN) = REAL (Given)
the U4 offsets of the bolometers (arcsec)

U3_CENTRE = REAL (Given)
the U3 offset of the tracking 'centre' on the array (arcsec)

U4_CENTRE = REAL (Given)
the U4 offset of the tracking 'centre' on the array (arcsec)

X_BOL (N_BOL) = DOUBLE PRECISION (Returned)
the X offset (apparent RA or X) of the bolometer (radians)

Y_BOL (N_BOL) = DOUBLE PRECISION (Returned)
the Y offset (apparent dec or Y) of the bolometer (radians)

ELEVATION = DOUBLE (Returned)
elevation of each position

PAR_ANGLE = DOUBLE (Returned)
parallactic angle of each position

STATUS = INTEGER (Given and returned)
The global status

Copyright :

SCULIB_CALC_CLOCKERR
Calculate start up time and error in system clock

Description:

Checks the self-consistency of the FITS headers by calculating the expected LST from the Azimuth and Elevation values stored in the header and comparing this with the supplied reference LST. Returns the difference between the two values and the LST calculated from the headers.

Invocation:

```
CALL SCULIB_CALC_CLOCKERR( N_FITS, FITS, RA_CEN, LST_REF, CLOCK_ERR, LST_AZEL,
STATUS )
```

Arguments:

N_FITS = INTEGER (Given)
Number of items in the FITS array

FITS() = CHAR∗80 (Given)
FITS array

RA_CEN = DOUBLE (Given)
Apparent RA corresponding to the Az/El defined in the FITS header. In radians.

LST_REF = DOUBLE (Given)
Local sidereal time at the start of data acquisition. This usually comes from the LST_STRT array. In radians.

CLOCK_ERR = DOUBLE (Returned)
The time difference (in radians) between the times stored in the data file (header items and LST_STRT) and the actual time of the observation derived from the azimuth and elevation of the observed source. The value is the number that must be added to the supplied LST_REF in order to correct it.

LST_AZEL = DOUBLE (Returned)
Local Sidereal time calculated from the azimuth, elevation and apparent RA/Dec (radians)

STATUS = INTEGER (Given & Returned)
Global status. An error will be returned if the observation was not done in a tracking frame (eg NA or AZ) since those frames can not be used to determine the time from the telescope az and el.

Notes:

Requires the following FITS headers to be defined:

- LAT-OBS
- STRT_ELD

- STRT_AZD
- CENT_CRD

The clock error returned must be added to all times read from the data file in order to make those times correct.

The centre coordinates of the observation must corresponde to an astronomical frame. AZ can not be used (for obvious reasons).

Copyright :

SCULIB_CALC_FLATFIELD
calculate flat-field results for a bolometer

Description:

This routine calculates the image parameters from data taken as a flat-field measurement of a SCUBA bolometer. It takes as input data a map made of a point source with pixels on a square grid.

If status is good on entry, the routine will set default return values to VAL__BADR except QUALITY set to 0. As progress is made through the routine the various image quantities will be set to their derived values, and QUALITY will be set to 1 if any problem is encountered.

An attempt will then be made to estimate the 0 level of the image by averaging valid data points in the corners of the map area. If there are any such points then the 0 level will be subtracted from the map.

Next the routine will calculate the 0th and 1st order moments of the image on the map. Map pixels with bad quality are ignored.

The 0th order is calculated as

$$U_0 = \sum_{ij} f_{ij}, \tag{14}$$

where i,j are the pixel indices. The 1st order moment in X is calculated as

$$U_x = \sum_{ij} x f_{ij}, \tag{15}$$

where x is the x position of pixel i,j; and the 1st order moment in Y is calculated as

$$U_y = \sum_{ij} y f_{ij}, \tag{16}$$

where y is the y position of pixel i,j.

If the 0th order moment of the image was 0, i.e. there is no image on the map, then a warning message will be output and the routine will return with good status and QUALITY set to 1. Otherwise, the x,y centre of the image is calculated from:-

$$X_{\mathrm{cen}} = \frac{U_x}{U_0} \tag{17}$$

$$Y_{\mathrm{cen}} = \frac{U_y}{U_0} \tag{18}$$

With this information the image can be analysed further by one of 2 methods. The method used depends on the value of parameter FLAT_ANALYSIS. If there is an error reading this parameter, or its value is not either MOMENTS or FIT, then a warning message will be issued and a value of MOMENTS will be assumed.

For a MOMENTS analysis the 2nd order moments can be calculated about the centre of the image:-

$$U_{xx} = \sum_{ij}(x - X_{cen})^2 f_{ij} \tag{19}$$

$$U_{xy} = \sum_{ij}(x - X_{cen})(y - Y\_cen) f_{ij} \tag{20}$$

$$U_{yy} = \sum_{ij}(y - Y_{cen})^2 f_{ij} \tag{21}$$

The parameters of the weighted ellipse describing the data are then (following 'Analysis of Astronomical Images using Moments', R.Stobie, J.B.I.S., 33, 323):-

$$\tan(2\theta) = \frac{2U_{xy}}{U_{xx} - U_{yy}} \tag{22}$$

$$A^2 = 2(U_{xx} + U_{yy}) + 2\sqrt{(U_{xx} - U_{yy})^2 + 4U_{xy}^2} \tag{23}$$

$$B^2 = 2(U_{xx} + U_{yy}) - 2\sqrt{(U_{xx} - U_{yy})^2 + 4U_{xy}^2} \tag{24}$$

For a FIT analysis, a 2-d Gaussian is fitted by a least-squares routine to the data. SCULIB_FIT_ROUTINE is the routine that performs the fit while SCULIB_GAUSSIAN_XISQ calculates the chi-squared of the fit to the data. An error will be reported and QUALITY set to 1 if there are any problems with the fit process. The fitted function is :-

$$P \exp(\frac{-(x - X_{cen})^2 - (y - Y_{cen})^2}{\sigma^2}) \tag{25}$$

where SIGMA (σ) is an ellipse with semi-axis lengths A and B, with THETA (θ) the angle between the x axis and A (THETA in radians, measured anti-clockwise).

If all is well still, the volume under the image and the variance on it are calculated. The volume will be the sum of all map pixels under a circle of 12 arcsec radius for short-wave array bolometers, or 25 arcsec for other types. If this area laps over the edges of the map, or any pixels inside it have bad quality then a warning message will be output but QUALITY will stay good.

Invocation:

```
CALL SCULIB_CALC_FLATFIELD (BOLNAME, BOL_TYPE, IDIM, JDIM, MAP_DATA, MAP_VARIANCE,
MAP_QUALITY, X, Y, VOLUME, VOLUME_VAR, X_CENTRE, Y_CENTRE, THETA, A, B, QUALITY,
STATUS)
```

Arguments:

BOLNAME = CHARACTER*(*) (Given)
the name of the bolometer being measured

BOL_TYPE = CHARACTER*(*) (Given)
the type of the bolometer being measured

IDIM = INTEGER (Given)
'I' dimension of map

JDIM = INTEGER (Given)
'J' dimension of map

MAP_DATA (IDIM, JDIM) = REAL (Given)
map data values

MAP_VARIANCE (IDIM, JDIM) = REAL (Given)
map variance

MAP_QUALITY (IDIM, JDIM) = REAL (Given)
map quality

X (IDIM) = REAL (Given)
x axis of map

Y (JDIM) = REAL (Given)
y axis of map

VOLUME = REAL (Returned)
volume under image within MULT times the fitted ellipse

VOLUME_VAR = REAL (Returned)
variance on VOLUME_DATA

X_CENTRE = REAL (Returned)
x of image centre

Y_CENTRE = REAL (Returned)
y of image centre

THETA = REAL (Returned)
angle of tilt of ellipse (radians)

A = REAL (Returned)
semi-major axis of ellipse

B = REAL (Returned)
semi-minor axis of ellipse

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_CALC_GRID
calculate the minimum rectangular grid that would contain the input jiggle pattern

Description:

This routine takes the x,y coords of the mapped points and attempts to fit them onto a rectangular grid. The grid contains the minimum number of points required to hold the measured positions.

Invocation:

```
CALL SCULIB_CALC_GRID (N, X, Y, XMIN, XMAX, XSPACE, NX, YMIN, YMAX, YSPACE, NY,
IPOS, JPOS, STATUS)
```

Arguments:

N = INTEGER (Given)
the number of positions in the jiggle pattern

X (N) = REAL (Given)
the x offsets of the jiggle

Y (N) = REAL (Given)
the y offsets

XMIN = REAL (Returned)
the minimum of the output map's x-axis

XMAX = REAL (Returned)
the maximum of the x-axis

XSPACE = REAL (Returned)
the pixel spacing in x

NX = INTEGER (Returned)
the map dimension in x

YMIN = REAL (Returned)
the minimum of the output map's y-axis

YMAX = REAL (Returned)
the maximum of the y-axis

YSPACE = REAL (Returned)
the pixel spacing in y

NY = INTEGER (Returned)
the map dimension in y

IPOS (N) = INTEGER (Returned)
the I index of each jiggle position in the map

JPOS (N) = INTEGER (Returned)
the J index of each jiggle position in the map

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_CALC_OUTPUT_COORDS
calculate output coords of map centre and angle of output coord system N relative to apparent N

Description:

This routine takes the apparent centre coords at the time of the observation and converts them to the output coordinate system. In addition, the angle between the north direction in the output coordinate frame and that in the apparent frame is calculated (measured anticlockwise from output north, in radians).

Invocation:

```
CALL SCULIB_CALC_OUTPUT_COORDS (RA_APP, DEC_APP, MJD, OUTPUT_COORDS, LONG, LAT,
STATUS)
```

Arguments:

RA_APP = DOUBLE PRECISION (Given)
apparent RA of map centre on MJD (radians)

DEC_APP = DOUBLE PRECISION (Given)
apparent declination of map centre on MJD (radians)

MJD = DOUBLE PRECISION (Given)
U1 of observation expressed as modified Julian day

OUTPUT_COORDS = CHARACTER*(*) (Given)
output coordinated system; RB, RJ, RD, GA or EQ

LONG = DOUBLE PRECISION (Returned)
longitude of map centre in output coord system (radians)

LAT = DOUBLE PRECISION (Returned)
latitude of map centre in output coord system (radians)

STATUS = INTEGER (Given and returned)
global status

Notes:

SLA routines are used to perform the coordinate conversions.

Copyright :

SCULIB_CALC_SKYDIP_TEMPS
Calculate all the skydip temps from raw data

Description:

This routine accepts the SKYDIP (3 dimensional) raw data and works out the temperature associated with each integration. The 2 dimensional array (bolometers, integrations) is then returned.

Invocation:

```
CALL SCULIB_CALC_SKYDIP_TEMPS( N_TEMPS, N_FITS, FITS, PARAM, N_INTEGRATIONS, N_MEASUREMENT
N_POS, N_BOLS, DEM_PNTR, IN_DATA, OUT_DATA, OUT_VAR, OUT_QUAL, OUT_DEM_PNTR, OUT_DATA_AV,
OUT_VAR_AV, OUT_QUAL_AV, SCRATCH, STATUS)
```

Arguments:

N_TEMPS = INTEGER (Given)
Number of SKYDIP temperatures.

SUB_REQUIRED = INTEGER (Given)
The number of the sub-instrument that is requested. If this number is less than 1 or greater than N_BOLS then all subinstrumentes are returned. This should be 0 when called from REDUCE_SWITCH and the required number if called from SKYDIP. Exists purely to prevent multiple requests for T_COLD if we are only interested in one wavelength

N_FITS = INTEGER (Given)
Number of FITS items

FITS = CHARACTER∗80 (Given)
The FITS items

PARAM = CHARACTER∗(PAR__SZNAM) (Given)
Name of parameter from which T_COLD can be read

N_INTEGRATIONS = INTEGER (Given)
Number of integrations

N_MEASUREMENTS = INTEGER (Given)
Number of measurements

N_POS = INTEGER (Given)
Number of data points

N_BOLS = INTEGER (Given)
Number of bolometers (sub instruments)

NUM_CHAN = INTEGER (Given)
number of A/D channels per A/D card

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_TYPE (NUM_CHAN, NUM_ADC)
= CHARACTER∗(∗) (Given) the types of the bolometers

BOL_CHAN (N_BOLS) = INTEGER (Given)
channel numbers of selected bolometers

BOL_ADC (N_BOLS) = INTEGER (Given)
ADC numbers of selected bolometers

DEM_PNTR(1, 1, N_INTEGRATIONS, N_MEASUREMENTS) = INTEGER (Given)
Pointer to positions in the input data

IN_DATA(N_TEMPS, N_BOLS, N_POS) = REAL (Given)
Input data [NTEMPS, NBOLS, NPOS]

OUT_DATA(N_BOLS, N_POS) = REAL (Returned)
Processed sky brightness temperatures.

OUT_VAR(N_BOLS, N_POS) = REAL (Returned)
Variance on each point in OUT_DATA

OUT_QUAL(N_BOLS, N_POS) = BYTE (Returned)
Quality of OUT_DATA

OUT_DEM_PNTR(1, 1, N_INTEGRATIONS, N_MEASUREMENTS) = INTEGER (Returned)

Modified DEM_PNTR to reflect changes in data array

OUT_DATA_AV(N_MEASUREMENTS, N_BOLS) = REAL (Returned)
Average sky temp for each measurement

OUT_VAR_AV(N_MEASUREMENTS, N_BOLS) = REAL (Returned)
Variance of the Average sky temp for each measurement

OUT_QUAL_AV(N_MEASUREMENTS, N_BOLS) = BYTE (Returned)
Quality of the Average sky temp for each measurement

SCRATCH(N_TEMPS, N_BOLS, N_INTEGRATIONS) = REAL (Given)
Work space to store the data for each measurement (since I cant simply pass a pointer in as the data is non-contiguous)

STATUS = INTEGER (Given)
Global Status

Copyright :

SCULIB_CALC_SUB_BOLS
find the positions of bolometers belonging to a specified sub-instrument in a demodulated data array

Description:

This subroutine finds the location of data from bolometers belonging to a specified sub-instrument in a demodulated data array that may contain data for several sub-instruments. The number of bolometers found, their indices in the data array, and their ADC,channel numbers are returned by the routine.

Invocation:

```
CALL SCULIB_CALC_SUB_BOLS (N_BOL_IN, IN_BOL_ADC, IN_BOL_CHAN, NUM_CHAN, NUM_ADC,
BOL_TYPE, SUB_INSTRUMENT, N_BOL_OUT, OUT_BOL_ADC, OUT_BOL_CHAN, IN_POINTER, STATUS)
```

Arguments:

N_BOL_IN = INTEGER (Given)
number of bolometers in data array

IN_BOL_ADC (N_BOL_IN) = INTEGER (Given)
ADC numbers of bolometers in data array

IN_BOL_CHAN (N_BOL_IN) = INTEGER (Given)
channel numbers of bolometers in data array

NUM_CHAN = INTEGER (Given)
number of channels per A/D card

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_TYPE (NUM_CHAN,NUM_ADC) = CHARACTER∗(∗) (Given)
types of bolometers

SUB_INSTRUMENT = CHARACTER∗(∗) (Given)
the name of the sub-instrument for which data are to be extracted

N_BOL_OUT = INTEGER (Returned)
the number of bolometers in the data array that belong to the specified sub-instrument

OUT_BOL_ADC (N_BOL_IN) = INTEGER (Returned)
ADC numbers of bolometers in specified sub-instrument

OUT_BOL_CHAN (N_BOL_IN) = INTEGER (Returned)
channel numbers of bolometers in specified sub-instrument

IN_POINTER (N_BOL_IN) = INTEGER (Returned)
array pointing to indices in data array that contain data for bolometers in the specified sub-instrument

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_CFILLB
fill a byte array with a constant

Description:
fills an byte array with a constant

Invocation:

```
CALL SCULIB_CFILLB (N, IVAL, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

IVAL = BYTE (Given)
constant to which array is to be set

ARRAY (N) = BYTE (Returned)
array to be set

Copyright :

Implementation Status:

- No status checking

SCULIB_CFILLD
fill a double precision array with a constant

Description:
fills a double precision array with a constant

Invocation:

```
CALL SCULIB_CFILLD (N, DVAL, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

DVAL = DOUBLE PRECISION (Given)
constant to which array is to be set

ARRAY (N) = DOUBLE PRECISION (Returned)
array to be set

Copyright :

Implementation Status:

- No status checking

SCULIB_CFILLI
fill an integer array with a constant

Description:
fills an integer array with a constant

Invocation:

```
CALL SCULIB_CFILLI (N, IVAL, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

IVAL = INTEGER (Given)
constant to which array is to be set

ARRAY (N) = INTEGER (Returned)
array to be set

Copyright :

Implementation Status:

- No status checking

SCULIB_CFILLR
fill a real array with a constant

Description:
fills a real array with a constant

Invocation:

```
CALL SCULIB_CFILLR (N, RVAL, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

RVAL = REAL (Given)
constant to which array is to be set

ARRAY (N) = REAL (Returned)
array to be set

Copyright :

Implementation Status:

- No status checking

SCULIB_CLIP_BOL
To clip a bolometer at +/- nsigma

Description:

This routine despikes a single bolometer at the +/- n sigma level. A 1-dimensional data set is assumed.

Invocation:

```
CALL SCULIB_CLIP_BOL(N_POS, SCUDATA, SCUQUAL, N_SIGMA, BADBIT, NSPIKES, STATUS)
```

Arguments:

N_POS = INTEGER (Given)
Number of jiggle positions in data set

SCUDATA(N_POS) = REAL (Given)
The data

SCUQUAL(N_POS) = BYTE (Given & Returned)
The data quality

N_SIGMA = DOUBLE (Given)
Number of sigma to clip at. If this number is positive then a clipped mean and standard deviation are calculated. If it is negative the statistics are calculated without iterative clipping.

BADBIT = BYTE (Given & Returned)
Bad bit mask

NSPIKES = INTEGER (Returned)
Number of spikes that were detected.

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SCULIB_COADD
coadd exposure into coadded result

Description:

This routine coadds the current exposure to the input coadd arrays and puts the result in the output coadd arrays. The input and output arrays can be the same. The coadd result is the average of exposures that have been input. The coadd variance is calculated from the spread of the input exposures about the mean if more than one exposure has been added in. If only one exposure is available then the coadd variance is set equal to variance on the input data if present, otherwise it is set equal to zero. Input pixels with bad quality are ignored.

Invocation:

```
CALL SCULIB_COADD (N, IN_DATA, IN_VARIANCE, IN_QUALITY, INCOADD_DATA, INCOADD_VAR,
INCOADD_QUAL, INCOADD_NUMBER, OUTCOADD_DATA, OUTCOADD_VAR, OUTCOADD_QUAL, OUTCOADD_NUMBER,
BADBIT, VARIANCE, STATUS)
```

Arguments:

N = INTEGER (Given)
Number of elements in arrays.

IN_DATA (N) = REAL (Given)
data to be added to coadd

IN_VARIANCE (N) = REAL (Given)
variance on input data

IN_QUALITY (N) = BYTE (Given)
quality on input data

INCOADD_DATA (N) = REAL (Given)
Input coadd.

INCOADD_VAR (N) = REAL (Given)
Input coadd variance.

INCOADD_QUAL (N) = BYTE (returned)
Input coadd quality.

INCOADD_NUMBER (N) = INTEGER (Given)
Number of exposures coadded in input coadd.

OUTCOADD_DATA (N) = REAL (Returned)
Output coadd.

OUTCOADD_VAR (N) = REAL (Returned)
Output coadd variance.

OUTCOADD_QUAL (N) = BYTE (returned)
Output coadd quality.

OUTCOADD_NUMBER (N) = INTEGER (Returned)
Number of exposures coadded in output coadd.

BADBIT = BYTE (Given)
Bad bit mask

VARIANCE = LOGICAL (Given)
T if input data has variance associated with it

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_COADD_MAPS
Average together integration and output map

Description:

This routine coadds the current integration image to the output image. The coadd variance is calculated from the spread of the integrations about the mean if more than one integration has been added in. The averaging is done over integrations, so that each point in an image is separate. An output point is good if either of the inputs is good. If only one dataset is coadded th variance is set zero. Input pixels with bad quality are ignored.

Invocation:

```
CALL SCULIB_COADD_MAPS(NPTS, DATA_IN, QUALITY_IN, WEIGHT, NCOADD, DATA_OUT, VARIANCE_OUT,
QUALITY_OUT, WEIGHT_OUT, STATUS)
```

Arguments:

NPTS = INTEGER (Given)
Number of input data points

DATA_IN(NPTS) = REAL (Given)
Input data

QUALITY_IN(NPTS) = BYTE (Given)
Input data quality

WEIGHT = REAL (Given)
Relative weight of input data

NCOADD(NPTS) = INTEGER (Given & Returned)
Number of points coadded into each data point

DATA_OUT(NPTS) = REAL (Given & Returned)
Coadded output data

VARIANCE_OUT(NPTS) = REAL (Given & Returned)
Variance of coadded output data.

QUALITY_OUT(NPTS) = BYTE (Given & Returned)
Quality of coadded output data

WEIGHT_OUT(NPTS) = REAL (Given & Returned)
Total weight of output data

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_COADD_REMOVE
remove exposure from coadded result

Description:

This routine removes an exposure from the input coadd and puts the result in the output coadd arrays. The input and output arrays can be the same. The coadd result is the average of exposures remaining in the coadd. The coadd variance is calculated from the spread of the exposures about the mean if more than one exposure remains, 0 otherwise. Exposure pixels with bad quality are ignored.

Invocation:

```
CALL SCULIB_COADD_REMOVE (N, IN_DATA, IN_VARIANCE, IN_QUALITY, INCOADD_DATA, INCOADD_VAR,
INCOADD_QUAL, INCOADD_NUMBER, OUTCOADD_DATA, OUTCOADD_VAR, OUTCOADD_QUAL, OUTCOADD_NUMBER,
VARIANCE)
```

Arguments:

N = INTEGER (Given)
Number of elements in arrays.

IN_DATA (N) = REAL (Given)
data to be added to coadd

IN_VARIANCE (N) = REAL (Given)
variance on input data

IN_QUALITY (N) = INTEGER (Given)
quality on input data

INCOADD_DATA (N) = REAL (Given)
Input coadd.

INCOADD_VAR (N) = REAL (Given)
Input coadd variance.

INCOADD_QUAL (N) = INTEGER (Given)
Input coadd quality

INCOADD_NUMBER (N) = INTEGER (Given)
Number of exposures coadded in input coadd.

OUTCOADD_DATA (N) = REAL (Returned)
Output coadd.

OUTCOADD_VAR (N) = REAL (Returned)
Output coadd variance.

OUTCOADD_QUAL (N) = INTEGER (returned)
Output coadd quality.

OUTCOADD_NUMBER (N) = INTEGER (Returned)
Number of exposures coadded in output coadd.

VARIANCE = LOGICAL (Given)
T if input data has variance associated with it

Copyright :

SCULIB_COMPRESS_DEMOD
get demodulated data for a bolometer and coadd jiggles in each integration

Description:

This routine selects data for a specified bolometer from a demodulated data array, and coadds the results for each jiggle in each integration to give an average for that integration. Data with bad quality are ignored. The variance on the average will also be derived; set equal to the variance on the input data if only one jiggle contributes to the average, otherwise calculated from the spread of the input points about the mean.

Invocation:

```
CALL SCULIB_COMPRESS_DEMOD (N_BOLS, N_JIGS, N_INTS, IN_DATA, ADC_INDEX, CHAN_INDEX,
INT_INDEX, ADC, CHAN, OUT_DATA, OUT_VARIANCE, OUT_QUALITY, STATUS)
```

Arguments:

N_BOLS = INTEGER (Given)
number of bolometers measured

N_JIGS = INTEGER (Given)
number of jiggles in pattern

N_INTS = INTEGER (Given)
number of integrations taken

IN_DATA (4, N_BOLS, N_JIGS $*$ N_INTS)
= REAL (Given) the demodulated data; 1=data, 2=variance, 3=calibrator, 4=quality

ADC_INDEX (N_BOLS) = INTEGER (Given)
the ADC numbers of the measured bolometers

CHAN_INDEX (N_BOLS) = INTEGER (Given)
the channel numbers of the measured bolometers

INT_INDEX (N_INTS) = INTEGER (Given)
the k index in IN_DATA of the first jiggle of each integration

ADC = INTEGER (Given)
the ADC number of the requested bolometer

CHAN = INTEGER (Given)
the channel number of the requested bolometer

OUT_DATA (N_INTS) = REAL (Returned)
the values for each integration averaged over the jiggles

OUT_VARIANCE (N_INTS) = REAL (Returned)
the variance on OUT_DATA

OUT_QUALITY (N_INTS) = INTEGER (Returned)
the quality on OUT_DATA

STATUS = INTEGER (Given and returned)
global status

Notes:

Only works for data where SWITCH_PER_EXP=1, EXP_PER_INT=1 and N_MEASUREMENTS=1.

Copyright :

SCULIB_CONSTRUCT_OUT
Append a identification string to a string

Description:

Appends an ID string to an input string given the contents of the environment variable specified by ENV_NAME Allowed values of ENV_NAME are stored in OPTIONS and the corresponding string stored in OPTION_STRINGS. Default is to go for the first entry in the IDSTRING array. if the value of ENV_NAME is either not set or unrecognized.

Invocation:

```
CALL SCULIB_CONSTRUCT_OUT(IN_STRING, ENV_NAME, N_OPTIONS, OPTIONS, OPTION_STRINGS,
OUT_STRING, STATUS)
```

Arguments:

IN_STRING = CHAR (Given)
Input string

ENV_NAME = CHAR (Given)
Name of environment variable to examine for suffix preference

N_OPTIONS = INTEGER (Given)
Number of suffix options

OPTIONS(N_OPTIONS) = CHAR (Given)
Names of different options

OPTION_STRINGS(N_OPTIONS) = CHAR (Given)
Suffixes for each option. If the string starts with a '!' the input string is chopped from the last '_' before adding the new string.

OUT_STRING = CHAR (Returned)
Input string with suffix

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_CONVOLVE
convolve array A with array B to give result in R

Description:

This routine convolves array A with array B using multiplication only (i.e. no FFTs are used). The routine assumes that the A arrays are N_A long and that the convolution array B is N_B long, centred at N_MIDDLE. The final result is normalised by dividing by NORM. Elements of the A array that have bad quality are ignored. The input variances are propagated as:-

$$R_{Variance} = \frac{\sum B^2 A_{Variance}}{norm^2} \tag{26}$$

After this process the errors can no longer be considered independent.

Invocation:

```
CALL SCULIB_CONVOLVE (A_DATA, A_VARIANCE, A_QUALITY, B, N_A, N_B, N_MIDDLE, NORM,
R_DATA, R_VARIANCE, R_QUALITY, STATUS)
```

Arguments:

A_DATA (N_A) = REAL (Given)
input data array

A_VARIANCE (N_A) = REAL (Given)
variance on A_DATA

A_QUALITY (N_A) = BYTE (Given)
quality on A_DATA

B (N_B) = REAL (Given)
convolution function

N_A = INTEGER (Given)
the size of A_DATA

N_B = INTEGER (Given)
length of convolution function

N_MIDDLE = INTEGER (Given)
index of zero point of convolution function

NORM = REAL (Given)
normalisation factor for convolution function

R_DATA (N_A) = REAL (Returned)
result of the convolution

R_VARIANCE (N_A) = REAL (Returned)
variance on R_DATA

R_QUALITY (N_A) = QUALITY (Returned)
quality on R_DATA

BADBIT = BYTE (Given)
badbit mask

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_COPY_DEMOD_SWITCH
copy a switch of demodulated data into arrays holding the switch chop data, chop variance, calibrator, cal variance and quality separately

Description:

This routine copies the demodulated data for the jiggle positions measured in the last switch into separate arrays for the switch chop data, chop variance, calibrator, cal variance and quality. Quality is marked bad if BIT 1 is set and LEVEL is greater than requested

Invocation:

```
CALL SCULIB_COPY_DEMOD_SWITCH (NBOLS, NDATA, NJIG, SWITCH, SWITCH_DATA, SWITCH_VARIANCE,
SWITCH_CALIBRATOR, SWITCH_CAL_VARIANCE, SWITCH_QUALITY, STATUS)
```

Arguments:

NBOLS = INTEGER (Given)
number of bolometers being measured

NDATA = INTEGER (Given)
number of data per measured position; 5 for jiggle data, 4 for raster

NJIG = INTEGER (Given)
number of jiggle positions measured

LEVEL = INTEGER (Given)
level at which spikes are marked as bad quality

SWITCH (5, NBOLS, NJIG) = REAL (Given)
switch datablock

SWITCH_DATA (NBOLS, NJIG) = REAL (Returned)
chop data for this switch of the exposure

SWITCH_VARIANCE (NBOLS, NJIG) = REAL (Returned)
variance on the chop signal

SWITCH_CALIBRATOR (NBOLS, NJIG) = REAL (Returned)
calibrator for this switch of the exposure

SWITCH_CAL_VARIANCE (NBOLS, NJIG)= REAL (Returned)
variance on the calibrator signal

SWITCH_QUALITY (NBOLS, NJIG) = BYTE (Returned)
quality for this switch of the exposure

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_COPY_GOOD
Go through data set and throw away bad values

Description:

This routine copies good data and variance from an input array to to an output array, leaving behind BAD values.

Invocation:

```
CALL SCULIB_COPY_GOOD(NPTS, INDATA, INVAR, INARR1, INARR2, NGOOD, OUTDATA, OUTVAR,
OUTARR1, OUTARR2, STATUS)
```

Arguments:

NPTS = INTEGER (Given)
The number of points in the input array

INDATA (NPTS) = REAL (Given)
Input data

INVAR (NPTS) = REAL (Given)
Input variance

INARR1 (NPTS) = DOUBLE (Given)
Auxilliary double precision array

INARR2 (NPTS) = DOUBLE (Given)
Auxilliary double precision array

NGOOD = INTEGER (Returned)
Number of good points in input data

OUTDATA (NPTS) = REAL (Returned)
Output data

OUTVAR (NPTS) = REAL (Returned)
Output variance

OUTARR1 (NPTS) = REAL (Returned)
Good auxiliary data converted to real

OUTARR2 (NPTS) = REAL (Returned)
Good auxiliary data converted to real

STATUS = INTEGER (Given & Returned)
Global status value

Notes:

- Data are still good even if VARIANCE or AUX are bad

Copyright :

Implementation Status:

- Only a routine for REAL numbers exists

SCULIB_COPYB
copy one byte array to another

Description:

copies one integer array into another

Invocation:

```
CALL SCULIB_COPYB (N, FROM, TO)
```

Arguments:

N = INTEGER (Given)
number of integers in arrays

FROM (N) = BYTE (Given)
array copied from

TO (N) = BYTE (Returned)
array copied to

Notes:

This routine is deprecated. Use VEC_BTOB (SUN/39) instead.

SCULIB_COPYD
copy one double precision array to another

Description:

copies one double precision array into another

Invocation:

```
CALL SCULIB_COPYD (N, FROM, TO)
```

Arguments:

N = INTEGER (Given)
number of elements in arrays

FROM (N) = DOUBLE PRECISION (Given)
array copied from

TO (N) = DOUBLE PRECISION (Returned)
array copied to

Notes:

This routine is deprecated. Use VEC_DTOD (SUN/39) instead.

Copyright :

SCULIB_COPYI
copy one integer array to another

Description:

copies one integer array into another

Invocation:

```
CALL SCULIB_COPYI (N, FROM, TO)
```

Arguments:

N = INTEGER (Given)
number of integers in arrays

FROM (N) = INTEGER (Given)
array copied from

TO (N) = INTEGER (Returned)
array copied to

Notes:

This routine is deprecated. Use VEC_ITOI (SUN/39) instead.

Copyright :

SCULIB_COPYR
copy one real array to another

Description:

copies one real array into another

Invocation:

```
CALL SCULIB_COPYR (N, FROM, TO)
```

Arguments:

N = INTEGER (Given)
number of reals in arrays

FROM (N) = REAL (Given)
array copied from

TO (N) = REAL (Returned)
array copied to

Notes:

This routine is deprecated. Use VEC_RTOR (SUN/39) instead.

Copyright :

SCULIB_CORRECT_EXTINCTION
correct bolometers for sky opacity

Description:

This routine corrects bolometer data for the effect of sky opacity. It does this by calculating the airmass of the point that each bolometer was looking at, then multiplying the data by exp (airmass $*$ TAUZ). Bolometers with bad data quality will be ignored.

Invocation:

```
CALL SCULIB_CORRECT_EXTINCTION (SIZE_BOL, N_BOL, BOL_DATA, BOL_VARIANCE, BOL_RA,
BOL_DEC, LST, LAT_OBS, TAUZ, STATUS)
```

Arguments:

SIZE_BOL = INTEGER (Given)
dimension of arrays

N_BOL = INTEGER (Given)
used size of arrays

BOL_DATA (SIZE_BOL) = REAL (Given and returned)
bolometer data

BOL_VARIANCE (SIZE_BOL) = REAL (Given and returned)
variance on BOL_DATA

BOL_RA (SIZE_BOL) = DOUBLE PRECISION (Given)
apparent RA of bolometer (radians)

BOL_DEC (SIZE_BOL) = DOUBLE PRECISION (Given)
apparent dec of bolometer (radians)

LST = DOUBLE PRECISION (Given)
sidereal time (radians)

LAT_OBS = DOUBLE PRECISION (Given)
latitude of observatory (radians)

TAUZ = REAL (Given)
the zenith sky opacity

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_COVSRT
Numerical Recipes in Fortran routine called by SCULIB_MRQMIN

Description:

Expand in storage the covariance matrix COVAR, so as to take into account parameters that are being held fixed. (For the latter return zero covariances). Derived from COVSRT in Numerical Recipes in Fortran, p.669, with STATUS added.

Invocation:

```
CALL SCULIB_COVSRT (COVAR, NPC, MA, IA, MFIT, STATUS)
```

Arguments:

COVAR (NPC, NPC) = REAL (Given & Returned)
Covariance matrix

INTEGER NPC = INTEGER (Given)
Size of covariance matrix

INTEGER MA = INTEGER (Given)
Size of IA array. Total number of parameters.

INTEGER IA (MA) = INTEGER (Given)
Ordering of parameters. Not a free parameter if containes 0.

INTEGER MFIT = INTEGER (Given)
Number of required parameters

STATUS = INTEGER (Given & Returned)
Global status

Notes:

Derived from the 'Numerical Recipes in Fortran' COVSRT routine.

Copyright :

SCULIB_CROSSTALK
crosstalk measurements

Description:

Experimental routine for calculating crosstalk between A-to-D cards. Averages signal for each bolometer and writes to a file. The file name comes from the FILE ADAM parameter.

Invocation:

```
CALL SCULIB_CROSSTALK (SWITCH_PER_EXP, EXP_PER_INT, N_INTEGRATIONS, N_MEASUREMENTS,
DEMOD_POINTER, N_BOLS, NUMPOS, DATA, BOL_CHAN, BOL_ADC, STATUS)
```

Arguments:

SWITCH_PER_EXP = INTEGER (Given)
number of switches per exposure

EXP_PER_INT = INTEGER (Given)
number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations

N_MEASUREMENTS = INTEGER (Given)
number of measurments

DEMOD_POINTER (SWITCH_PER_EXP, EXP_PER_INT, N_INTEGRATIONS,
N_MEASUREMENTS) = INTEGER (Given) pointer to start of demodulated data for each switch in DATA array

N_BOLS = INTEGER (Given)
number of bolometers being measured

NUMPOS = INTEGER (Given)
total number of positions measured

DATA (4, N_BOLS, NUMPOS)
= REAL (Given) the demodulated data

BOL_CHAN (N_BOLS) = INTEGER (Given)
the channel number of the measured bolometers

BOL_ADC (N_BOLS) = INTEGER (Given)
the ADC numbers of the measured bolometers

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_DAY
returns date and time as day number since 1st Jan

Description:

Calculates the time since 1st January. The time is returned as a day number and fractional day.

Invocation:

```
DAY = SCULIB_DAY ()
```

Arguments:

None

Returned Value:

SCULIB_DAY = DOUBLE PRECISION
date and time as a day number since 1st Jan

Copyright :

SCULIB_DECODE_ANGLE
convert angle string to double precision angle

Description:

This routine converts an angle in dd:mm:ss.dd format into a double precision number in radians. It assumes that the input string specifies the angle in degrees. The process involves removing the ':' delimiters, then calling SLA_DAFIN to perform the conversion. An error will be returned if there are less than 2 ':' delimiters in the string, or if the SLA routine errors.

Invocation:

```
CALL SCULIB_DECODE_ANGLE (STRING, ANGLE, STATUS)
```

Arguments:

STRING = CHARACTER*(*) (Given)
angle in xx:mm:ss.ddd format

ANGLE = DOUBLE PRECISION (Returned)
angle in radians

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_DECODE_COMPONENT
This routine decodes the value of a single component in a SCUBA data-spec

Description:

This routine decodes the value of a single component in a SCUBA data-spec. It is called by SCULIB_DECODE_SPEC. The component can be:-

- [1] "∗" select all data
- [2] "<index>" select data at position <index>
- [3] "<index1>,<index2>" select data at positions <index1> and <index2>
- [4] "<index1>:<index2>" select all data in the range <index1> to <index2>

or any combination of 2, 3 and 4 separated by commas. Example component values are:-

- "∗" select all data
- "1,5" select data at indices 1 and 5
- "5:10,16" select data at indices 5 through 10 and 16

Errors will occur if:-

- You mix the ∗ format with selection by index.
- Any index selected lies outside the range 1 to N.
- In a selection range <index2> is less than <index1>.
- The component does not conform to the design syntax.

Output from the routine is in the form of a mask array SELECT. Data indices selected will be marked as 1 in SELECT, while indices not selected will be zeroes.

Invocation:

```
CALL SCULIB_DECODE_COMPONENT (COMPONENT, N, SELECT, STATUS)
```

Arguments:

COMPONENT = CHARACTER∗(∗) (Given)
the component to be decoded

N = INTEGER (Given)
the number of items that can be selected

SELECT (N) = INTEGER (Returned)
the items selected; 1 for selected, 0 for not selected

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_DECODE_FILTER
decode filter name into names and wavelengths of filters in front of each instrument section in use

Description:

This routine decodes the filter name to give the name and central wavelength of the filter in front of each sub-instrument being used.

The filter name should be in format <short>:<long> where these are the names of the filters in front of the short and long-wave focal planes.

```
<short> can be:-
'350'      short wavelength  =   350.0   name =  '350'
'450'                            450.0           '450'
'600'                            600.0           '600'
'750'                            750.0           '750'
'850'                            850.0           '850'
'PHOT'                           -1.0            'BLANK'


<long> can be:-
'350'           long array   =   350.0   name =  '350'
       P1100, P1300, P2000       350.0           '350'
'450'           long array   =   450.0           '450'
       P1100, P1300, P2000       450.0           '450'
'600'           long array   =   600.0           '600'
       P1100, P1300, P2000       600.0           '600'
'750'           long array   =   750.0           '750'
       P1100, P1300, P2000       750.0           '750'
'850'           long array   =   850.0           '850'
       P1100, P1300, P2000       850.0           '850'
'PHOT'          long array   =    -1.0           'BLANK'
                     P1100   =  1100.0           '1100'
                     P1300   =  1300.0           '1300'
                     P2000   =  2000.0           '2000'
```

If the filter name does not fit the 'pattern' described an error message will be output and the routine return with bad status. The routine is insensitive to the case of 'PHOT'.

Invocation:

```
CALL SCULIB_DECODE_FILTER (FILTER, N_SUB, SUB_INSTRUMENT, SUB_FILTER, WAVELENGTH,
STATUS)
```

Arguments:

FILTER = CHARACTER*(*) (Given)
name of filter

N_SUB = INTEGER (Given)
number of sub instruments being used

SUB_INSTRUMENT (N_SUB) = CHARACTER*(*) (Given)
the names of the sub instruments being used

SUB_FILTER (N_SUB) = CHARACTER*(*) (Returned)
the name of the filter in front of each sub-instrument

WAVELENGTH (N_SUB) = REAL (Returned)
the wavelength in microns of the filters in front of the sub instruments being used. An inappropriate filter will have wavelength set to VAL__BADR

STATUS = INTEGER (Given and returned)
global status

Notes:

The wavelength values are probably out of step with those currently in use by the real-time system.

Copyright :

SCULIB_SPLIT_DECODE_REBIN_LINE
Splits a string into a filename and parameters for REBIN

Description:

This routine takes a string and splits it into bits. The format is assumed to be:

FILESPEC WEIGHT SHIFT_DX SHIFT_DY

The string is broken up from left to right. There must always be a weight if there is a shift There must always be a shift if there is a section. etc... Comment character is #. A line contains a comment once a # appears on the line. Only chars before the # are valid.

Invocation:

```
CALL SCULIB_DECODE_REBIN_LINE(LINE, N_FOUND, NAME, WEIGHT, SHIFT_DX, SHIFT_DY,
STATUS)
```

Arguments:

LINE = CHAR (Given)
Line read from file

N_FOUND = INTEGER (Returned)
Number of parameters found in LINE

NAME = CHARACTER (Returned)
File name + (optional) SCUBA section

WEIGHT = REAL (Returned)
Relative weight given to filename

SHIFT_DX = REAL (Returned)
Shift in X

SHIFT_DY = REAL (Returned)
Shift in Y

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SCULIB_DECODE_SPEC
Decode SCUBA-style data specifications

Description:

This routine decodes a SCUBA-style data specification. The data-spec will be of the form {<component>;<component>;...}, where components are one of the following:-

- B<index_spec> - specifying bolometer indices
- P<index_spec> - position indices
- S<index_spec> - switch indices
- E<index_spec> - exposure indices
- I<index_spec> - integration indices
- M<index_spec> - measurement indices

and the <index_spec> is a list like, for example, 2,5:7,17 to select indices 2, 5 through 7 and 17. Alternatively, <index_spec> can be $*$ which will select all data in that component coordinate. The data spec can be passed in as an array of data specs. The number is specified by N_SPEC. By default all components in a dataset are selected. Thus the empty data-spec {} will return all components selected. Example data-specs are:-

- "{} - select all data
- "{B7,12;P57}" - select data for bolometers 7 and 12 at measurement position 57
- "{S2;E1;I3:M2}" - select data for all bolometers in switch 2 of exposure 1 of integration 3 in measurement 2 of the observation
- "{B29}" - select all data for bolometer 29
- "{B29;E1}" - select data for bolometer 29 in the first exposure of each integration

The data-spec is case-insensitive and blanks are ignored.

Errors will occur:-

- If you attempt to select indices outside the dimensions input to the routine.
- If you attempt to select data both by position Pxxx and by switch Sxxx, exposure Exxx, integration Ixxx or measurement Mxxx.
- If you attempt to select by switch Sxxx when the SWITCH_EXPECTED flag is input .FALSE.

Output consists of a flag POS_SELECTED, to say whether or not the data were selected by position, and mask arrays that are set to 1 at the coordinate of selected data and 0 otherwise. Even if the data were not selected by position the POS_S array, which is the

mask for the position coordinate, will be set correctly. Conversely, however, the MEAS_S, INT_S, EXP_S and SWITCH_S arrays, which are masks for the measurement, integration, exposure and switch will not be set to sensible values if the data are position selected.

Invocation:

```
CALL SCULIB_DECODE_SPEC (SPEC, DEMOD_POINTER, N_SWITCHES,N_EXPOSURES, N_INTEGRATIONS,
N_MEASUREMENTS, N_POS, N_BOLS, SWITCH_EXPECTED, POS_SELECTED, POS_S, SWITCH_S,
EXP_S, INT_S, MEAS_S, BOL_S, STATUS)
```

Arguments:

N_SPEC = INTEGER (Given)
Number of specifications supplied in SPEC

SPEC(N_SPEC) = CHARACTER∗(∗) (Given)
the specification to be decoded

DEMOD_POINTER (N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)

= INTEGER (Given)
the pointer to the location in the main data array of the data for each switch of the observation

N_SWITCHES = INTEGER (Given)
the number of switches per exposure

N_EXPOSURES = INTEGER (Given)
the number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
the number of integrations per measurement

N_MEASUREMENTS = INTEGER (Given)
the number of measurements in the observation

N_POS = INTEGER (Given)
the number of positions measured in the observation

N_BOLS = INTEGER (Given)
the number of bolometers measured in the observation

SWITCH_EXPECTED = LOGICAL (Given)
.TRUE. if a switch component is allowed in the data-spec

POS_SELECTED = LOGICAL (Returned)
.TRUE. if the P component is used in the data-spec

POS_S (N_POS) = INTEGER (Returned)
the position mask array; 1 for selected positions, 0 otherwise

SWITCH_S (N_SWITCHES) = INTEGER (Returned)
the switch mask array; 1 for selected switches, 0 otherwise

EXP_S (N_EXPOSURES) = INTEGER (Returned)
the exposure mask array; 1 for selected exposures, 0 otherwise

INT_S (N_INTEGRATIONS) = INTEGER (Returned)
the integration mask array; 1 for selected integrations, 0 otherwise

MEAS_S (N_MEASUREMENTS) = INTEGER (Returned)
the measurement mask array; 1 for selected measurements, 0 otherwise

BOL_S (N_BOLS) = INTEGER (Returned)
the bolometer mask array; 1 for selected bolometers, 0 otherwise

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_DIV_CALIBRATOR
divides the calibrator signal into the chop signal

Description:

This routine divides the calibrator signal into the chop signal and variance over an array of measurements. No division will occur if the measurement quality is bad or the square of the calibrator signal is zero.

Invocation:

```
CALL SCULIB_DIV_CALIBRATOR (NELM, DATA, VARIANCE, CALIBRATOR, QUALITY)
```

Arguments:

NELM = INTEGER (Given)
Number of elements in each array

DATA (NELM) = REAL (Given and returned)
Demodulated data array

VARIANCE (NELM) = REAL (Given and returned)
Variance on DATA

CALIBRATOR (NELM) = REAL (Given)
Calibration array

QUALITY (NELM) = BYTE (Given)
Quality on DATA

STATUS = INTEGER (Given & Returned)
inherited status

Copyright :

SCULIB_DIV_CALIBRATOR_2
divides the mean of the calibrator signal into the chop signal

Description:

This routine divides the calibrator signal into the chop signal and variance over an array of measurements. No division will occur if the measurement quality is bad or the square of the calibrator signal is zero.

Invocation:

```
CALL SCULIB_DIV_CALIBRATOR_2 (N_BOLS, N_POS, DATA, VARIANCE, CALIBRATOR, QUALITY)
```

Arguments:

N_BOLS = INTEGER (Given)
Number of bolometers measured

N_POS = INTEGER (Given)
Number of positions measured

DATA (N_BOLS,N_POS) = REAL (Given and returned)
Demodulated data array

VARIANCE (N_BOLS,N_POS)
= REAL (Given and returned) Variance on DATA

CALIBRATOR (N_BOLS,N_POS)
= REAL (Given) Calibration array

QUALITY (N_BOLS,N_POS)
= BYTE (Given and returned) Quality on DATA

STATUS = INTEGER (Given & Returned)
inherited status

Copyright :

SCULIB_EXTRACT_2DIM_B
To extract the second dimension (at a given X) from a 2d array

Description:

This routine extracts a second dimension from a 2D byte array.

Invocation:

```
CALL SCULIB_EXTRACT_2DIM_B(X, NX, NY, TWOD, ONED, STATUS)
```

Arguments:

X = INTEGER (Given)
The X index in the 2D array to be extracted

NX = INTEGER (Given)
Size of the array in x

NY = INTEGER (Given)
Size of the array in y

TWOD = BYTE (Given)
2D array input dataset

ONED = BYTE (Returned)
1D strip extracted from 2D array

STATUS = INTEGER (Given & Returned)
Global status

Notes:

This is the BYTE routine.

Copyright :

SCULIB_EXTRACT_2DIM_D
To extract the second dimension (at a given X) from a 2d array

Description:

This routine extracts a second dimension from a 2D DOUBLE array.

Invocation:

```
CALL SCULIB_EXTRACT_2DIM_D(X, NX, NY, TWOD, ONED, STATUS)
```

Arguments:

X = INTEGER (Given)
The X index in the 2D array to be extracted

NX = INTEGER (Given)
Size of the array in x

NY = INTEGER (Given)
Size of the array in y

TWOD = DOUBLE (Given)
2D array input dataset

ONED = DOUBLE (Returned)
1D strip extracted from 2D array

STATUS = INTEGER (Given & Returned)
Global status

Notes:

This is the DOUBLE PRECISION routine.

Copyright :

SCULIB_EXTRACT_2DIM_R
To extract the second dimension (at a given X) from a 2d array

Description:

This routine extracts a second dimension from a 2D REAL array.

Invocation:

```
CALLSCULIB_EXTRACT_2DIM_R(X, NX, NY, TWOD, ONED, STATUS)
```

Arguments:

X = INTEGER (Given)
The X index in the 2D array to be extracted

NX = INTEGER (Given)
Size of the array in x

NY = INTEGER (Given)
Size of the array in y

TWOD = REAL (Given)
2D array input dataset

ONED = REAL (Returned)
1D strip extracted from 2D array

STATUS = INTEGER (Given & Returned)
Global status

Notes:

This is the REAL routine.

Copyright :

SCULIB_EXTRACT_BOL
To extract a single bolometer from a data array

Description:
This routine extracts a single bolometer from a SCUBA data array.

Invocation:

```
CALL SCULIB_EXTRACT_BOL(NBOL, N_BOLS, N_POS, SCUDATA, SCUQUAL, BOLDATA, BOLQUAL,
STATUS)
```

Arguments:

NBOL = INTEGER (Given)
The bolometer number

N_BOLS = _INTEGER (Given)
Total number of bolometers in data array

N_POS = INTEGER (Given)
Number of jiggle positions in data set

SCUDATA(N_BOLS, N_POS) = REAL (Given)
The data

SCUQUAL(N_BOLS, N_POS) = BYTE (Given)
The data quality

BOLDATA(N_POS) = REAL (Returned)
The specified bolometer data

BOLQUAL(N_POS) = _BYTE (Returned)
The bolometer quality

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

Implementation Status:
Propogates quality

SCULIB_FIND_INT
Return the indices of the start and end of the specified integration

Description:

This routine find that start and end indices of the specified integration. In general these can be found directly from the DEMOD_POINTER array but with the warning that the last integration will not necessarily point to the end of the data array if multiple measurements are being taken. For this reason, put this knowledge in one place.

Invocation:

```
CALL SCULIB_FIND_INT(DEMOD_POINTER, N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENT
N_POS, INTEG, MEAS, I_START, I_END, STATUS)
```

Arguments:

DEMOD_POINTER (N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS) = INTEGER (Given)
an array of pointers to the start of each switch

N_SWITCHES = INTEGER (Given)
the number of switches per exposure

N_EXPOSURES = INTEGER (Given)
the number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
the number of integrations per measurement

N_MEASUREMENTS = INTEGER (Given)
the number of measurements in the observation

N_POS = INTEGER (Given)
the total number of positions measured in the observation

INTEG = INTEGER (Given)
the integration number of the switch

MEAS = INTEGER (Given)
the measurement number of the switch

I_START = INTEGER (Returned)
pointer to the start of the integration

I_END = INTEGER (Returned)
pointer to the end of the integration

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIND_SWITCH
find the start and end indices of a switch in the demodulated data array

Description:

This routine finds the start and end indices of the specified switch in a SCUBA observation. The array DEMOD_POINTER contains the start indices, so this part is easy. However, the end index will be 1 less than the start index of the next, or the index of the last position measured if the specified switch is the last in the observation. This makes things a little complicated because consecutive switches within an exposure need not have been taken and stored consecutively, though exposures, integrations and measurements are ordered as one would expect.

Invocation:

```
CALL SCULIB_FIND_SWITCH (DEMOD_POINTER, N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS,
N_MEASUREMENTS, N_POS, SWITCH, EXP, INT, MEAS, S_START, S_END, STATUS)
```

Arguments:

DEMOD_POINTER (N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS) = INTEGER (Given)
an array of pointers to the start of each switch

N_SWITCHES = INTEGER (Given)
the number of switches per exposure

N_EXPOSURES = INTEGER (Given)
the number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
the number of integrations per measurement

N_MEASUREMENTS = INTEGER (Given)
the number of measurements in the observation

N_POS = INTEGER (Given)
the total number of positions measured in the observation

SWITCH = INTEGER (Given)
the index of the switch

EXP = INTEGER (Given)
the exposure number of the switch

INT = INTEGER (Given)
the integration number of the switch

MEAS = INTEGER (Given)
the measurement number of the switch

S_START = INTEGER (Returned)
pointer to the start of the switch

S_END = INTEGER (Returned)
pointer to the end of the switch

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_2D_GAUSSIAN
fit a 2D Gaussian to an image

Description:

This routine fits a 2D Gaussian to an image of a source. It takes as input data a map made of the source with pixels on a square grid.

If status is good on entry, the routine will calculate the 0th (U_0) and 1st order (U_x, U_y) moments of the image on the map. Map pixels with bad quality are ignored:

The 0th order is calculated as

$$U_0 = \sum_{ij} f_{ij}, \tag{27}$$

where i,j are the pixel indices. The 1st order moment in X is calculated as

$$U_x = \sum_{ij} x f_{ij}, \tag{28}$$

where x is the x position of pixel i,j; and the 1st order moment in Y is calculated as

$$U_y = \sum_{ij} y f_{ij}, \tag{29}$$

where y is the y position of pixel i,j.

If the 0th order moment of the image was 0, i.e. there is no image on the map, then a warning message will be output and all image parameters set to 0 or bad quality. Otherwise, the x,y centre of the image is calculated from:-

$$X_{\rm cen} = \frac{U_x}{U_0} \tag{30}$$

$$Y_{\rm cen} = \frac{U_y}{U_0} \tag{31}$$

With this information the 2nd order moments can be calculated about the centre of the image:-

$$U_{xx} = \sum_{ij} (x - X_{cen})^2 f_{ij} \tag{32}$$

$$U_{yy} = \sum_{ij} (y - Y_{cen})^2 f_{ij} \tag{33}$$

If the image had a 2D Gaussian profile,

$$f = A \exp(-a_x^2 x^2) \exp(-a_y^2 y^2), \tag{34}$$

then it can be shown that for x:-

$$a_x^2 = \frac{U_0}{2U_{xx}} \tag{35}$$

and the half-width at half-maximum is calculated from:-

$$X_{\mathrm{HWHM}} = \sqrt{\ln(2)}a_x \tag{36}$$

Lastly, the volume under the image and the variance on it are calculated. These will be the sum of all map pixels and their variances that lie within a radius of 2 $*$ max(HWHM_x, HWHM_y) from X_CENTRE, Y_CENTRE. If the HWHM in either axis is zero then a warning message will be output and PEAK_QUAL set bad. Otherwise, PEAK will be set to volume / (π $*$ HWHM_x $*$ HWHM_y).

Invocation:

```
CALL SCULIB_FIT_2D_GAUSSIAN (IDIM, JDIM, MAP_DATA, MAP_VARIANCE, MAP_QUALITY,
X, Y, PEAK, PEAK_VAR, PEAK_QUAL, X_CENTRE, Y_CENTRE, X_HWHM, Y_HWHM, STATUS)
```

Arguments:

IDIM = INTEGER (Given)
'I' dimension of map

JDIM = INTEGER (Given)
'J' dimension of map

MAP_DATA (IDIM, JDIM) = REAL (Given)
map data values

MAP_VARIANCE (IDIM, JDIM) = REAL (Given)
map variance

MAP_QUALITY (IDIM, JDIM) = REAL (Given)
map quality

X (IDIM) = REAL (Given)
x axis of map

Y (JDIM) = REAL (Given)
y axis of map

PEAK = REAL (Returned)
peak height of Gaussian

PEAK_VAR = REAL (Returned)
variance on PEAK

PEAK_QUAL = INTEGER (Returned)
quality of PEAK

X_CENTRE = REAL (Returned)
x of image centre

Y_CENTRE = REAL (Returned)
y of image centre

X_HWHM = REAL (Returned)
half-width half-max of image in x-axis

Y_HWHM = REAL (Returned)
half-width half-max of image in y-axis

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_2D_PARABOLA
fit 2d parabola to data

Description:

This routine performs a least-squares fit of a 2d parabola to a set of data. The form of the parabola is:-

$$z = A0 + A1 * (x - X0)^2 + A1 * (y - Y0)^2 \tag{37}$$

The apex of the parabola is at X0,Y0 and its value there is A0.

If status is good on entry the routine will work through the data calculating the various sums required for the least-squares method. Data with bad quality or zero variance are ignored. If no 'good' data are found then an error will be reported and the routine will return with bad status.

A matrix equation of the form $M * B = Z$ is constructed, where:-

$$M = \begin{pmatrix} \sum \frac{1}{var} & \sum \frac{x}{var} & \sum \frac{y}{var} & \sum \frac{x^2+y^2}{var} \\ \sum \frac{x}{var} & \sum \frac{x^2}{var} & \sum \frac{xy}{var} & \sum \frac{x^3+xy^2}{var} \\ \sum \frac{y}{var} & \sum \frac{xy}{var} & \sum \frac{y^2}{var} & \sum \frac{x^2y+y^3}{var} \\ \sum \frac{x^2+y^2}{var} & \sum \frac{x^3+xy^2}{var} & \sum \frac{x^2y+y^3}{var} & \sum \frac{(x^2+y^2)^2}{var} \end{pmatrix} \tag{38}$$

$$B = \begin{pmatrix} A0 + A1 * X0^2 + A1 * Y0^2 \\ -2 * A1 * X0 \\ -2 * A1 * Y0 \\ A1 \end{pmatrix} \tag{39}$$

$$Z = \begin{pmatrix} \sum \frac{z}{var} \\ \sum \frac{xz}{var} \\ \sum \frac{yz}{var} \\ \sum \frac{(x^2+y^2)z}{var} \end{pmatrix} \tag{40}$$

where x, y, z and var are the x,y coords, value and variance of the measured points and the sums are over all valid points.

SCULIB_INVERT_MATRIX is called to invert the matrix and SCULIB_FIT_MULT to multiply Z by the inverse. This yields the fitted value for B from which the other fit parameters are derived.

Invocation:

```
CALL SCULIB_FIT_2D_PARABOLA (N, DATA, VARIANCE, QUALITY, X, Y, A0, A1, X0, Y0,
Z_PEAK, Z_PEAK_VAR, BADBIT, STATUS)
```

Arguments:

N = INTEGER (Given)
number of data points

DATA (N) = REAL (Given)
the data

VARIANCE (N) = REAL (Given)
variance on the data

QUALITY (N) = BYTE (Given)
the quality on the data

X (N) = REAL (Given)
the x offsets of the data

Y (N) = REAL (Given)
the y offsets of the data

A0 = REAL (Returned)
fit result

A1 = REAL (Returned)
fit result

X0 = REAL (Returned)
x coord of parabola apex

Y0 = REAL (Returned)
y coord of parabola apex

Z_PEAK = REAL (Returned)
the value of the parabola apex

Z_PEAK_VAR = REAL (Returned)
the variance on Z_PEAK

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and returned)
global status

Notes:

If the variances are not a true reflection of the errors on the data then very strange numbers can result.

Copyright :

SCULIB_FIT_D2XISQ_DAJ2
calculate second differential of chi-squared with respect to a fit parameter

Description:

If status is good on entry this routine will calculate the 2nd order differential of chi-squared with respect to A(J). It does this numerically by calling the supplied routine XISQ_ROUTINE to calculate chi-squared at the current parameter values, then again at A(J)+DELTA_AJ and A(J)-DELTA_AJ, then using the formula:-

$$\frac{\partial^2\chi^2}{\partial AJ^2} = \frac{\chi^2(AJ+\delta AJ) - 2\chi^2(AJ) + \chi^2(AJ-\delta AJ)}{\delta AJ^2} \tag{41}$$

DELTA_AJ is equal to the absolute value of $0.001 * \mathrm{A(J)}$ or, if this is zero, 0.001.

Invocation:

```
CALL SCULIB_FIT_D2XISQ_DAJ2 (XISQ_ROUTINE, N, A, J, D2XISQ_DAJ2, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
name of routine that will calculate chi-squared

N = INTEGER (Given)
the number of parameters in the fit

A(N) = DOUBLE PRECISION (Given and returned)
the fit parameters

J = INTEGER (Given)
the index of the parameter to be varied

D2XISQ_DAJ2 = DOUBLE PRECISION (Returned)
the second differential of chi-squared with respect to A(J)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_D2XISQ_DAJK
calculate differential of chi-squared with respect to fit parameters J and K

Description:

If status is good on entry this routine will calculate the differential of chi-squared with respect to A(J) and A(K). It does this numerically by calling the supplied routine XISQ_ROUTINE to calculate chi-squared at [A(J)+DELTA_AJ,A(K)+DELTA_AK], [A(J)-DELTA_AJ,A(K)+DELTA_AK], [A(J)+DELTA_AJ,A(K)-DELTA_AK] and [A(J)-DELTA_AJ,A(K)-DELTA_AK]. The result is then calculated using the formula:-

$$\begin{aligned}\frac{\partial^2\chi^2}{\partial AJ\partial AK} =& \chi^2(AJ+\delta AJ, AK+\delta AK) \\ & - \chi^2(AJ-\delta AJ, AK+\delta AK) \\ & - \frac{\chi^2(AJ-\delta AJ, AK+\delta AK)+\chi^2(AJ-\delta AJ, AK-\delta AK)}{4\delta AJ\delta AK}\end{aligned} \tag{42}$$

DELTA_AJ is equal to the absolute value of $0.001 * \mathrm{A(J)}$ or, if this is zero, 0.001. DELTA_AK is calculated in a similar way.

Invocation:

```
CALL SCULIB_FIT_D2XISQ_DAJK (XISQ_ROUTINE, N, A, J, K, D2XISQ_DAJK, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
name of routine that will calculate xi-squared

N = INTEGER (Given)
the number of parameters in the fit

A(N) = DOUBLE PRECISION (Given and returned)
the fit parameters

J = INTEGER (Given)
the index of the first parameter to be varied

K = INTEGER (Given)
the index of the second parameter to be varied

D2XISQ_DAJK = DOUBLE PRECISION (Returned)
the differential of chi-squared with respect to A(J) and A(K)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_DXISQ_DAJ
calculate gradient in chi-squared with 'a'

Description:

If status is good on entry this routine will calculate the gradient in the chi-squared of the fit with variation in fit parameter J. It does this numerically by calling the supplied routine XISQ_ROUTINE to calculate chi-squared for (A(J)+DELTA_AJ) and (A(J)-DELTA_AJ), then using the equation:-

$$\frac{\partial \chi^2}{\partial AJ} = \frac{\chi^2(A(J) + \delta AJ) - \chi^2(A(J) - \delta AJ)}{2.0 \delta AJ} \tag{43}$$

DELTA_AJ is equal to the absolute value of $0.001 * \mathrm{A(J)}$ or, if this is zero, 0.001.

Invocation:

```
CALL SCULIB_FIT_DXISQ_DAJ (XISQ_ROUTINE, N, A, J, DXISQ_DAJ, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
name of routine that will calculate xi-squared

N = INTEGER (Given)
the number of parameters in the fit

A (N) = DOUBLE PRECISION (Given and returned)
the fit parameters

J = INTEGER (Given)
the index of the parameter to be varied

DXISQ_DAJ = DOUBLE PRECISION (Returned)
the gradient of chi-squared with A(J)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_FUNCTION
routine to fit a general function to data

Description:

This routine fits a general function to a data-set by searching the chi-squared plane. Chi-squared at each test fit position will be evaluated by the external routine XISQ_ROUTINE. The data to be fit are passed to the external routine through common; see SCULIB_GAUSSIAN_XISQ for an example. Each call to the routine will generate one iteration of the search. On exit the fit should have an improved chi-squared that should be checked for convergence (i.e. has chi-squared changed by less than CHICUT between 2 iterations) before the routine is called again to perform the next iteration. It is recommended that the routine be entered for the first iteration with LAMBDA set to 0.001. Thereafter, the routine will return the value of LAMBDA to be used in the next iteration.

The search method used is that due to Marquardt as described in chapter 8 of 'Data Reduction and Error Analysis' by Bevington and Robinson. The code is an adaptation of the Pascal routine Marquardt listed in that book.

If the routine is entered with LAMBDA = 0 then it will return the 'error' matrix for the given fit parameters.

Invocation:

```
CALL SCULIB_FIT_FUNCTION (XISQ_ROUTINE, CHICUT, N, A, LAMBDA, ALPHA, BETA, IK,
JK, DA, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
routine to calculate chi-squared of fit

CHICUT = DOUBLE PRECISION (Given)
increase in chi-squared that will make routine search with larger LAMBDA

N = INTEGER (Given)
number of fitted parameters

A (N) = DOUBLE PRECISION (Given and returned)
the fitted parameters

LAMBDA = DOUBLE PRECISION (Given and returned)
curvature factor for alpha matrix

ALPHA (N,N) = DOUBLE PRECISION (Scratch)
storage for alpha array

BETA (N) = DOUBLE PRECISION (Scratch)
storage for beta array

IK (N) = INTEGER (Scratch)
used for inverting matrix

JK (N) = INTEGER (Scratch)
used for inverting matrix

DA (N) = DOUBLE PRECISION (Scratch)
increment in A

STATUS = INTEGER (Given and returned)
global status

Copyright :

Method :

If status is good on entry the routine will enter a loop

- SCULIB_FIT_MAKEBETA is called to calculate the 'beta' matrix
- SCULIB_FIT_MAKEALPHA is called to calculate the 'alpha' matrix
- The diagonal elements of the 'alpha' matrix are multiplied by 1 + LAMBDA following the method due to Marquardt
- SCULIB_INVERT_MATRIX is called to invert the modified 'alpha' matrix
- If LAMBDA = 0 this is all that's required because this input is given if the routine is required to calculate the 'error' matrix
- Otherwise, SCULIB_FIT_MULT is called to multiply the inverted 'alpha' matrix by the 'beta' to give the next iteration of the fit for this LAMBDA
- Chi-squared is calculated for the new fit and compared with that for the previous fit. If the chi-squared is greater than the previous value plus CHICUT then things have got worse. In this case the routine will increase LAMBDA by a factor of 10, return the fit to the previous value and return to the start of the loop. If LAMBDA exceeds 1000 status is set bad, an error is reported and the loop is exited
- However, if the chi-squared is an improvement then LAMBDA is decreased by a factor of 10 and the loop is exited

End of loop

SCULIB_FIT_MAKEALPHA
calculates alpha matrix for SCULIB_FIT_FUNCTION

Description:

This routine calculates the elements of the alpha matrix for non-linear fitting as described in 'Data Reduction and Error Analysis for the Physical Sciences' by Bevington and Robinson, section 8.6 'the Marquardt Method'.

The routine does this by calling routines SCULIB_FIT_D2XISQ_DAJ2 and SCULIB_FIT_D2XISQ_DAJK for each element required. The routine will only execute if entered with good status.

Invocation:

```
CALL SCULIB_FIT_MAKEALPHA (XISQ_ROUTINE, N, A, ALPHA, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
routine to calculate chi-squared of current fit

N = INTEGER (Given)
the number of fit parameters

A (N) = DOUBLE PRECISION (Given)
the values of the fit parameters

ALPHA (N,N) = DOUBLE PRECISION (Returned)
the alpha matrix

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_MAKEBETA
calculates beta matrix for SCULIB_FIT_FUNCTION

Description:

This routine calculates the elements of the beta matrix for non-linear fitting as described in 'Data Reduction and Error Analysis for the Physical Sciences' by Bevington and Robinson, section 8.6 'the Marquardt Method'.

The routine does this by calling the routine SCULIB_FIT_DXISQ_DAJ for each element required. The routine will only execute if entered with good status.

Invocation:

```
CALL SCULIB_FIT_MAKEBETA (XISQ_ROUTINE, N, A, BETA, STATUS)
```

Arguments:

XISQ_ROUTINE (XISQ, N, A, STATUS) = EXTERNAL ROUTINE (Given)
routine to calculate chi-squared of current fit

N = INTEGER (Given)
the number of fit parameters

A (N) = DOUBLE PRECISION (Given)
the values of the fit parameters

BETA (N) = DOUBLE PRECISION (Returned)
the beta vector

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_MULT
matrix vector multiplication routine

Description:

If status is good on entry, this routine will multiply the N-vector BETA by the NxN matrix ALPHA to give the N-vector GAMMA. GAMMA should not be the same as BETA as this would mean BETA being overwritten during the multiplication process giving an incorrect answer.

Invocation:

```
CALL SCULIB_FIT_MULT (N, ALPHA, BETA, GAMMA, STATUS)
```

Arguments:

N = INTEGER (Given)
dimensions of square matrix

ALPHA (N,N) = DOUBLE PRECISION (Given)
the multiplying matrix

BETA (N) = DOUBLE PRECISION (Given)
the multiplied vector

GAMMA (N) = DOUBLE PRECISION (Returned)
the result vector (GAMMA and BETA should not be the same array)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FIT_PLANE
fits a plane to some x, y, z data

Description:

This routine does a least squares fit of data to a plane. The plane is described by an equation of form:-

$$z = M_X * x + M_Y * y + C \tag{44}$$

Invocation:

```
CALL SCULIB_FIT_PLANE (N, X, Y, Z, QUALITY, MX, MY, C, STATUS)
```

Arguments:

N = INTEGER (Given)
the number of data points

X (N) = REAL (Given)
the x coords of the measured points

Y (N) = REAL (Given)
the y coords of the measured points

Z (N) = REAL (Given)
the measured values

QUALITY (N) = INTEGER (Given)
the quality on the data

MX = REAL (Returned)
the slope of the fitted plane in the x-axis

MY = REAL (Returned)
the slope of the fitted plane in the y-axis

C = REAL (Returned)
the constant of the fitted plane

STATUS = INTEGER (Given and returned)
global status

Method :

If status is good on entry the routine will loop through the input data calculating the sums required and the number of valid data points (i.e. those with good quality flags).

If there are less than 3 valid data then an error message will be output and bad status returned else the plane coefficients will be calculated from the following formulae:-

$$M_X = \frac{\sum y^2(n_v \sum xz - \sum x \sum z) - (\sum y)^2 \sum xz - n_v \sum xy \sum yz + \sum y(\sum xy \sum z + \sum x \sum yz)}{2\sum xy \sum x \sum y - n_v(\sum xy)^2 - (\sum x)^2 \sum y^2 + \sum x^2(n_v \sum y^2 + (\sum y)^2)} \tag{45}$$

$$M_Y = \frac{n_v \sum yz - \sum y \sum z + M_X(\sum x \sum y - n_v \sum xy)}{n_v \sum y^2 - (\sum y)^2} \tag{46}$$

$$C = \frac{\sum z - M_X \sum x - M_Y \sum y}{n_v} \tag{47}$$

where n_v is the number of valid data points. If the denominator of the expression for MX is 0 then an error message will be output and bad status returned.

end if

Copyright :

SCULIB_FIT_SKYDIP
Fit the skydip data

Description:

This routine fits a sub-instrument's measurements of the sky brightness at a range of airmasses to obtain the sky opacity, ETAtel and 'b' parameters. The fit is to the function:-

$$J_{\rm meas} = (1-\eta_{\rm tel})J_{\rm tel} + \eta_{\rm tel}J_{\rm atm} - b\eta_{\rm tel}J_{\rm atm}e^{-A\tau}, \quad (48)$$

Jtel is known and Jatm is related to Jamb by:-

$$J_{\rm atm} = J_{\rm amb}\int_0^{40} A\left[k\exp\left(-\frac{h}{h_2}\right)\exp\left[Akh_2\left(\exp\left(-\frac{h}{h_2}\right)-1\right)\right]\left(1-\frac{h}{h_1}\right)\right]{\rm d}h, \quad (49)$$

where h2 = scale height of absorption (= 2km), h1 = coefficient to give 6.5K/km temperature drop in absorber, A = airmass, k = extinction.

which relation has been fudged by Bill Duncan to:-

$$J_{\rm atm} = J_{\rm amb}X_{\rm g}\left[1-\exp\left(-Akh_2\right)\right], \quad (50)$$

with $X_{\rm g}$ having the form

$$X_{\rm g} = 1 + \frac{h_2 T_{\rm lapse}}{T_{\rm amb}}\exp\left(-\frac{A\tau}{X_{\rm gconst}}\right) \quad (51)$$

with

h1 = 2, h2 = -6.5 (note this h2 is defined differently to that in the previous equation), $X_{\rm gconst}$ = 3.669383

See 'Calibration of mm and sub-mm Photometers by Skydipping', W.D.Duncan preprint and 'Inversion of Sky Dips', SCU/WDD/31.1/1093 for further details.

The fit can be made with ETAtel and/or b either fixed or varying. To allow one of these parameters to vary it should be input to the routine with a value below zero. If the input value is greater than zero then the routine will fix it at that for the fit.

Invocation:

```
CALL SCULIB_FIT_SKYDIP (CVAR, N_MEASUREMENTS, AIRMASS, J_MEASURED, J_VARIANCE,
SUB_WAVELENGTH, SUB_INSTRUMENT, SUB_FILTER, T_TEL, T_AMB, ETA_TEL_IN, B_IN, ETA_TEL_FIT,
B_FIT, TAUZ_FIT, REXISQ, TAU_ERROR, ETA_ERROR, B_ERROR, RESIDUAL, SIGMA, STATUS)
```

Arguments:

CVAR = LOGICAL (Given)

flag to govern whether to use a fixed variance (true) or the actual variance. The fixed variance is the mean of the actual variances.

N_MEASUREMENTS = INTEGER (Given)
the number of SKYDIP measurements

AIRMASS (N_MEASUREMENTS) = REAL (Given)
the airmasses at which the measurements were made

DATA (N_MEASUREMENTS) = REAL (Given)
the measured sky brightness temperatures

VARIANCE (N_MEASUREMENTS) = REAL (Given)
the variance on DATA

SUB_WAVELENGTH = REAL (Given)
the wavelength of the measurements

SUB_INSTRUMENT = CHARACTER*(*) (Given)
the name of the sub-instrument used

SUB_FILTER = CHARACTER*(*) (Given)
the name of the filter used

T_TEL = REAL (Given)
the telescope temperature (K)

T_AMB = REAL (Given)
the ambient temperature (K)

ETA_TEL_IN = REAL (Given)
if >= 0 then this will the ETAtel assumed in the fit. if < 0 then ETAtel will be allowed to vary in the fit.

B_IN = REAL (Given)
if >=0 then this value of b will be assumed in the fit. if < 0 then b will be allowed to vary in the fit.

ETA_TEL_FIT = REAL (Returned)
the result for ETAtel

B_FIT = REAL (Returned)
the result for b

TAUZ_FIT = REAL (Returned)
the fitted result for tauz

REXISQ = REAL (Returned)
the reduced chi square of the fit

TAU_ERROR = REAL (Returned)
error in the tau

ETA_ERROR = REAL (Returned)
error in eta_tel

B_ERROR = REAL (Returned)
error in B

RESIDUAL = DOUBLE (Returned)
Absolute difference between the model and the fit.

SIGMA = DOUBLE (Returned)
standard deviation of the difference between the model and the fit.

STATUS = INTEGER (Given and returned)
Global status

SCULIB_FIX_SCAN_V10
Correct scan positions for version 1.0 data from SCUCD

Description:

For data taken with version 1.0 of SCUCD the positions of the scan ends were incorrect. This was because of a bug in the software that calculated the scan ends concerning the use of arcseconds instead of radians. This routine attempts to recreate the bug so that the correct scan offsets can be calculated.

Invocation:

```
CALL SCULIB_FIX_SCAN_V10(CENTRE_COORDS, LAT_OBS, LONG, LAT, MJD, RA_OFF_START,
DEC_OFF_START, RA_OFF_END, DEC_OFF_END, RA_NEW_START, DEC_NEW_START, RA_NEW_END,
DEC_NEW_END, STATUS)
```

Arguments:

CENTRE_COORDS = CHAR (Given)
The coordinate system of the map centre

LAT_OBS = DOUBLE PRECISION (Given)
Latitude of observatory (radians)

LONG = DOUBLE (Given)
The longitude (ra) of the map centre (radians)

LAT = DOUBLE (Given)
The latitude (dec) of the map centre (radians)

MJD = DOUBLE (Given)
The modified Julian date at which the data was taken

RA_OFF_START = DOUBLE (Given)
The RA start position of the scan as stored in the header (RD)

DEC_OFF_START = DOUBLE (Given)
The Dec start position of the scan as stored in the header (RD)

RA_OFF_END = DOUBLE (Given)
The RA end position of the scan as stored in the header (RD)

DEC_OFF_END = DOUBLE (Given)
The DEC end position of the scan as stored in the header (RD)

RA_NEW_START = DOUBLE (Returned)
The corrected RA start position of the scan (RD)

DEC_NEW_START = DOUBLE (Returned)
The corrected Dec start position of the scan (RD)

RA_NEW_END = DOUBLE (Returned)
The corrected RA end position of the scan (RD)

RA_NEW_END = DOUBLE (Returned)
The corrected Dec end position of the scan (RD)

STATUS = INTEGER (Given & Returned)

Notes:

The scan ends are all stored as apparent RA/Decs.

Copyright :

SCULIB_FLATFIELD_DATA
flatfield the data in an array

Description:

This routine multiplies the data in the input array by the flatfield values of the bolometers that took it. Output quality will be set bad if input quality was bad or the quality of the flatfield for that bolometer was bad.

Invocation:

```
CALL SCULIB_FLATFIELD_DATA (N_BOL, N_POS, N_BEAM, BDATA, VARIANCE, QUALITY, BOL_CHAN,
BOL_ADC, NUM_CHAN, NUM_ADC, BOL_FLAT, BOL_QUALITY, STATUS)
```

Arguments:

N_BOL = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions at which the bolometers were measured

N_BEAM = INTEGER (Given)
number of beams data have been reduced into

BDATA (N_BOL, N_POS, N_BEAM)
= REAL (Given and returned) measured data

VARIANCE (N_BOL, N_POS, N_BEAM)
= REAL (Given and returned) variance on BDATA

QUALITY (N_BOL, N_POS, N_BEAM)
= BYTE (Given and returned) quality on BDATA

BOL_CHAN (N_BOL) = INTEGER (Given)
the channel numbers of the measured bolometers

BOL_ADC (N_BOL) = INTEGER (Given)
the A/D numbers of the measured bolometers

NUM_CHAN = INTEGER (Given)
number of channels per A/D

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_FLAT (NUM_CHAN, NUM_ADC)
= REAL (Given) the flatfield values

BOL_QUALITY (NUM_CHAN, NUM_ADC)
= INTEGER (Given) quality on BOL_FLAT

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FLATFIELD_SEQUENCE
get bolometer measurement sequence for FLATFIELD

Description:

This routine works out the bolometer measurement sequence for a FLATFIELD observation. Only one of SCUBA's sub-instruments can be flat-fielded at a time. If the sub-instrument is one of the arrays then the measurement sequence will be ref-bol-ref-bol-.....-bol-ref, where the 'bol's are the bolometers to be measured and 'ref' the reference bolometer on the array. If the sub-instrument is one of the photometry pixels then only that bolometer will be measured.

Whatever bolometer is the target of the measurement, data will be taken from all A/D channels.

If status is good on entry SCULIB_BOLSELECT is called to decode the bolometers to be measured and the sub-instruments involved. If the bolometers belong to more than one sub-instrument an error will be reported and bad status returned.

If the sub-instrument is one of the arrays then the name of the reference bolometer will be read from parameters LONGREF_BOL or SHORTREF_BOL as appropriate. The number of measurements and their sequence is set as described above in N_MEASUREMENTS, FLAT_CHAN and FLAT_ADC. If the sub-instrument is not one of the arrays then the reference bolometer will not be used and the bolometers(s) will be measured in sequence.

For each measurement of a target bolometer data will be taken from all data channels; BOLS_MEASURED is set to 'ALL' and SCULIB_BOLSELECT called to decode this to channel and ADC numbers. A check is made that the reference bolometer, if used, is among those being measured. If not, an error message will be output and bad status returned.

Lastly, the FLAT_INDEX array is set so that it points to the position in the datablock of data from the target bolometer at each measurement.

Invocation:

```
CALL SCULIB_FLATFIELD_SEQUENCE (BOLOMETERS, NUM_CHAN, NUM_ADC, NUM_SUB, BOL_TYPE,
BOL_CALB, BOL_DU3, BOL_DU4, BOL_QUAL, BOL_ENABLED, BOLS_MEASURED, N_BOLS, BOL_SELECT_CHAN,
BOL_SELECT_ADC, N_SUBS, SUB_INSTRUMENT, FLATREF_CHAN, FLATREF_ADC, N_MEASUREMENTS,
FLAT_CHAN, FLAT_ADC, FLAT_INDEX, STATUS)
```

Arguments:

BOLOMETERS = CHARACTER*(*)
the bolometers to be measured in this FLATFIELD

NUM_CHAN = INTEGER (Given)
the number of channels per A/D

NUM_ADC = INTEGER (Given)
the number of A/Ds

NUM_SUB = INTEGER (Given)
the number of sub-instruments in SCUBA

BOL_TYPE (NUM_CHAN,NUM_ADC) = CHARACTER∗(∗) (Given)
the type of each bolometer

BOL_CALB (NUM_CHAN,NUM_ADC) = REAL (Given)
the flat-field factor for each bolometer

BOL_DU3 (NUM_CHAN,NUM_ADC) = REAL (Given)
the dU3 coord of each bolometer

BOL_DU4 (NUM_CHAN,NUM_ADC) = REAL (Given)
the dU4 coord of each bolometer

BOL_QUAL (NUM_CHAN,NUM_ADC) = INTEGER (Given)
the quality of each bolometer

BOL_ENABLED (NUM_CHAN,NUM_ADC) = LOGICAL (Returned)
.TRUE. if a bolometer is to be measured

BOLS_MEASURED = CHARACTER∗(∗) (Returned)
the bolometers to be measured for each FLATFIELD measurement

N_BOLS = INTEGER (Returned)
the number of bolometers to be measured at each measurement

BOL_SELECT_CHAN (NUM_CHAN ∗ NUM_ADC) = INTEGER (Returned)
the channel numbers of the bolometers to be measured at each measurement

BOL_SELECT_ADC (NUM_CHAN ∗ NUM_ADC) = INTEGER (Returned)
the A/D numbers of the bolometers to be measured at each measurement

N_SUBS = INTEGER (Returned)
the number of sub-instruments being used

SUB_INSTRUMENT (NUM_SUB) = CHARACTER∗(∗) (Returned)
name of sub-instrument being used

FLATREF_CHAN = INTEGER (Returned)
channel number of reference bolometer

FLATREF_ADC = INTEGER (Returned)
A/D number of reference bolometer

N_MEASUREMENTS = INTEGER (Returned)
the number of measurements in this FLATFIELD observation

FLAT_CHAN (2 ∗ NUM_CHAN ∗ NUM_ADC) = INTEGER (Returned)
the channel number of the target bolometer in each measurement

FLAT_ADC (2 ∗ NUM_CHAN ∗ NUM_ADC) = INTEGER (Returned)
the A/D number of the target bolometer in each measurement

FLAT_INDEX (2 ∗ NUM_CHAN ∗ NUM_ADC) = INTEGER (Returned)
the index in the datablock of the target bolometer in each measurement

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_FREE
release virtual memory

Description:

This routine frees virtual memory obtained by SCULIB_MALLOC.

If status is bad on entry the routine will return immediately.

If START_PTR is not equal to 0 then:

- The sentinel integers above and below the used piece of memory will be checked against the values they were set to by SCULIB_MALLOC.
- PSX_FREE will be called to free the virtual memory. If that's successful START_PTR and END_PTR will be set to the bad value.
- If either of the sentinel integers was corrupted then an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_FREE (NAME, START_PTR, END_PTR, STATUS)
```

Arguments:

NAME = CHARACTER*(*) (Given)
name associated with VM

START_PTR = INTEGER (Given and returned)
pointer to beginning of virtual memory

END_PTR = INTEGER (Given and returned)
pointer to end of virtual memory

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GAUSS_WTINIT
Generate a weighting function for rebinning

Description:

A Gaussian weighting function The look up table is initialised such that one can access the weight directly given the square of the distance between the input and output pixel. The scale length is taken to be radius at half-width-half-max The gaussian should go out to a radius of at least 3 scale lengths (<0.2% of peak). Governed by the RADIUS argument The function is simply taken as F(x) = exp(-X**2) where the scale size (FWHM) is at X = SQRT(loge(2))

Invocation:

```
CALL SCULIB_GAUSS_WTINIT (WTFN, RADIUS, RES, STATUS)
```

Arguments:

WTFN (RADIUS * RADIUS * RES * RES + 1) = REAL (Returned)
The weighting function generated by this routine. The index corresponds to the square of the distance from the centre in scale units.

RADIUS = INTEGER (Given)
Size of the weighting function in scale units.

RES = INTEGER (Given)
Number of points per scale lenghth.

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_GAUSSIAN_XISQ
calculate chi-squared of Gaussian fit

Description:

If entered with good status this routine calculates the chi-squared between a dataset and a Gaussian function. Data points with bad quality will be ignored, as will points with zero variance. A warning will be issued if any data points with zero variance are encountered. If no valid data points are found then an error will be reported and bad status returned. The data are passed in via common.

Invocation:

```
CALL SCULIB_GAUSSIAN_XISQ (XISQ, N, FIT, STATUS)
```

Arguments:

XISQ = DOUBLE PRECISION (Returned)
the chi-squared of the current fit

N = INTEGER (Given)
the number of parameters being fit, should be 6

FIT(N) = DOUBLE PRECISION (Given)
the fit parameters:-

- FIT(1) = peak height
- FIT(2) = length of the 'a' axis of the sigma ellipse
- FIT(3) = length of the 'b' axis of the sigma ellipse
- FIT(4) = the angle between the 'a' axis and the x axis (+ve anticlockwise, radians)
- FIT(5) = the x coord of the centre
- FIT(6) = the y coord of the centre

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GAUSSJ
Numerical Recipes in Fortran routine for solution of linear equations by Gauss-Jordan elimination

Description:

Linear equation solution by Gauss-Jordan elimination. A(1:N,1:N) is an input matrix stored in an array of physical dimensions NP by NP. B(1:N,1:M) is an input matrix containing the M right-hand side vectors, stored in an array of physical dimensions NP by MP. On output, A(1:N,1:N) is replaced by its matrix inverse and B(1:N,1:M) is replaced by the corresponding set of solution vectors. Parameter NMAX is the largest anticipated value of N. Copied from GAUSSJ on p.30 of Numerical Recipes in Fortran, with STATUS added.

Invocation:

```
CALL SCULIB_GAUSSJ (A, N, NP, B, M, MP, STATUS)
```

Arguments:

A (NP, NP) = REAL (Given & Returned)
Input matrix. On exit, contains its inverse.

N = INTEGER (Given)
Required size of A ($<$ NP)

NP = INTEGER (Given)
Dimensions of input matrix.

B (NP, MP) = REAL (Given & Returned)
Input matrix containing M right-hand side vectors. On exit, contains the solution vectors.

M = INTEGER (Given)
Required size of B. ($<$ MP)

MP = INTEGER (Given)
Size of second dimension of B.

STATUS = INTEGER (Given & Returned)
Global status.

Copyright :

SCULIB_GENSYCONFN
Generate a convolution function which will set to zero the spatial frequencies with no signal in Dual Beam maps

Description:

This routine will return the symmetric convolution function required for setting to zero the spatial frequencies that have zero sensitivity in the raw chop-scan data. This function is not described in Emerson et al., 1979. A&A 96, 92, which (I think) describes an early version of the NOD2 algorithm before all the wrinkles had been properly understood.

This convolution function has to set to zero those points in the FT of the raw data at zero frequency and harmonics of 1/chop spacing. In the Fourier domain this means multiplying by a function which is 1 at all points apart from those specified, where it is zero. This is the sum of 2 functions, the first being 1 everywhere, the second being -1 at the points to be zeroed. The convolution function required is the sum of the inverse FTs of these two functions. The inverse FT of the first function is simply a delta function at the origin, that of the second is a series of negative spikes with the first at the origin and the others separated by the chop spacing.

The convolution function must cover the map even when the centre of the function is at the left or right hand end of the map. Hence the convolution function must be twice the length of the raw map.

Since the raw data is not sampled such that the chop spacing is an integer number of samples, the actual convolution function must be rebinned onto the sample mesh by sinc interpolation.

Otherwise the only tricky part of the algorithm is the need to normalise the two component functions such that in the Fourier domain the addition of the 2 functions does result in zeroes at the desired points. This is most easily achieved by concentrating on the zero spatial frequency which is just the sum of all the points in each of the 2 functions. The first function is a delta function set to 1, so the sum of that function is 1. The sum of the second function is that of all the points in the convolution function from its centre to one end, a length that corresponds to the map size. This is the length to be used because, in the Fourier domain, the width of the sample to be set to zero is 1/map_size.

Invocation:

```
CALL SCULIB_GENSYCONFN( BSEP, PIXSEP, NPIX, NCFN, CONF, STATUS )
```

Arguments:

BSEP = REAL (Given)
The beam separation in arcseconds

PIXSEP = REAL (Given)
The pixel separation in arcseconds

NPIX = INTEGER (Given)
The number of pixels in the x direction

UNBAL = REAL (Given)
The relative amplitudes of the right and left hand beams UNBAL=amp(lhb)/abs(amp(rhb))

NCFN = INTEGER (Returned)
The length of the convolution array

CONF(*) = REAL (Returned)
The convolution function

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_GET_BOL_DESC
Get the bolometer description arrays and check their dimensions

Description:

This routine obtains the bolometer description arrays. ie: BOL_TYPE, BOL_ADC, BOL_CHAN, BOL_DU3 and BOL_DU4 arrays. The dimensions of these arrays are checked and an error returned if necessary.

Invocation:

```
CALL SCULIB_GET_BOL_DESC(LOC, NUM_CHAN, NUM_ADC, BOL_TYPE, BOL_DU3, BOL_DU4, BOL_ADC,
BOL_CHAN, STATUS)
```

Arguments:

LOC = _CHARACTER (Given)
Locator to the SCUBA extension

NUM_CHAN = _INTEGER (Given)
Number of SCUBA channels per ADC

NUM_ADC = _INTEGER (Given)
Number of ADC cards

N_BOL = _INTEGER (Given)
Number of bolometers

BOL_TYPE = _CHARACTER*20() (Returned)
Bolometer types

BOL_DU3 = _REAL() (Returned)
du3 Nasmyth coordinated of bolometers

BOL_DU4 = _REAL() (Returned)
du4 Nasmyth coordinated of bolometers

BOL_ADC = _INTEGER() (Returned)
A/D numbers of bolometers measured

BOL_CHAN = _INTEGER() (Returned)
Channel numbers of bolometers measured

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_GET_DEM_PNTR
Get the DEM_PNTR array and its dimensions

Description:

This routine obtains the dem_pntr array and returns the dimensions in terms of switches, exposures, inegrations and measurements.

Invocation:

```
CALL SCULIB_GET_DEM_PNTR(ACTDIM, LOC, DEM_PNTR_PTR, N_SWITCHES, N_EXPOSURES, N_INTEGRATION
N_MEASUREMENTS, STATUS)
```

Arguments:

ACTDIM = _INTEGER (Given)
Number of expected dimensions

LOC = _CHARACTER (Given)
Locator to the SCUBA extension

DEM_PNTR_PTR = _INTEGER (Returned)
Pointer to location of DEM_PTR array in memory

N_SWITCHES = _INTEGER (Returned)
Number of switches indicated by DEM_PNTR

N_EXPOSURES = _INTEGER (Returned)
Number of exposures indicated by DEM_PNTR

N_INTEGRATIONS = _INTEGER (Returned)
Number of integrations indicated by DEM_PNTR

N_MEASUREMENTS = _INTEGER (Returned)
Number of measurements indicated by DEM_PNTR

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_GET_FILENAME
Find the filename associated with a parameter

Description:

This routine finds the name of the filename associated with a NDF parameter.

Invocation:

```
CALL SCULIB_GET_FILENAME(PARAM, FILENAME, STATUS)
```

Arguments:

PARAM = CHAR (Given)
Name of the parameter

FILENAME = CHAR (Returned)
Name of the file associated with PARAM

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

Implementation Status:

Uses SUBPAR

SCULIB_GET_FITS_C
get the value of specified FITS character keyword

Description:

This routine will get the value of a specified FITS character keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD= 'VALUE' / this is a comment"

The string is extracted from between the quotes and the '/' is not required.

It will search the input array for a line containing the required keyword and return VALUE. If the keyword is not found and error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_GET_FITS_C (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is required

VALUE = CHARACTER*(*) (Returned)
the value of the FITS keyword

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_FITS_D
get the value of specified FITS double keyword

Description:

This routine will get the value of a specified FITS double precision keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=VALUE / this is a comment"

It will search the input array for a line containing the required keyword and return VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_GET_FITS_D (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is required

VALUE = DOUBLE PRECISION (Returned)
the value of the FITS keyword

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_FITS_I
get the value of specified FITS integer keyword

Description:

This routine will get the value of a specified FITS integer keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=INTEGER_VALUE / this is a comment"

It will search the input array for a line containing the required keyword and return VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_GET_FITS_I (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is required

VALUE = INTEGER (Returned)
the value of the FITS keyword

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_FITS_L
get the value of specified FITS logical keyword

Description:

This routine will get the value of a specified FITS logical keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=LOGICAL / this is a comment"

It will search the input array for a line containing the required keyword and return VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_GET_FITS_L (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is required

VALUE = LOGICAL (Returned)
the value of the FITS keyword

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_FITS_R
get the value of specified FITS real keyword

Description:

This routine will get the value of a specified FITS real keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=VALUE / this is a comment"

It will search the input array for a line containing the required keyword and return VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_GET_FITS_R (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is required

VALUE = REAL (Returned)
the value of the FITS keyword

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_JIGGLE
Get the jiggle parameters

Description:

This routine obtains the parameters of the JIGGLE pattern: JIGL_X, JIGL_Y and some FITS parameters. The dimensions of these arrays are checked and an error returned if necessary.

Invocation:

```
CALL SCULIB_GET_JIGGLE(LOC, MAX_JIGGLE, N_FITS, FITS, JIGGLE_COUNT, JIGGLE_REPEAT,
JIGGLE_P_SWITCH, SAMPLE_PA, SAMPLE_COORDS, JIGGLE_X, JIGGLE_Y, STATUS)
```

Arguments:

LOC = _CHARACTER (Given)
Locator to the SCUBA extension

MAX_JIGGLE = _INTEGER (Given)
Maximum number of jiggle positions

N_FITS = _INTEGER (Given)
Size of FITS array

FITS = _CHARACTER (Given)
FITS values

JIGGLE_COUNT = _INTEGER (Returned)
Actual size of jiggle pattern

JIGGLE_REPEAT = _INTEGER (Returned)
Number of times jiggle pattern is repeated in a switch

JIGGLE_P_SWITCH = _INTEGER (Returned)
Number of jiggles per switch

SAMPLE_PA = _REAL
position angle of sample x axis relative to x axis of SAMPLE_COORDS

SAMPLE_COORDS = _CHARACTER
coordinate system of sample offsets

JIGGLE_X = _REAL (Returned)
X jiggle positions

JIGGLE_Y = _REAL (Returned)
Y jiggle positions

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_GET_LST_STRT
Get the LST_STRT array and check its dimensions

Description:

This routine obtains the lst_strt array and compares the dimensions with the number of exposures, inegrations and measurements. The number of switches is returned.

Invocation:

```
CALL SCULIB_GET_LST_STRT(LOC, LST_STRT_PTR, N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS,
N_MEASUREMENTS, STATUS)
```

Arguments:

LOC = _CHARACTER (Given)
Locator to the SCUBA extension

LST_STRT_PTR = _INTEGER (Returned)
Pointer to location of DEM_PTR array in memory

N_SWITCHES = _INTEGER (Returned)
Number of switches indicated by LST_STRT

N_EXPOSURES = _INTEGER (Given)
Actual number of exposures

N_INTEGRATIONS = _INTEGER (Given)
Actual number of integrations

N_MEASUREMENTS = _INTEGER (Given)
Actual number of measurements

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator. The array must be unmapped before finishing the program.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_GET_MJD
Obtain the Modified Julian date from the UT stored in FITS

Description:

This routine retrieves the UTSTART and UTDATE values from the FITS array and calculates the MJD from these. It can corrects for the time taken before beginning the observation by comparing the LST stored in the header (written when the observation begins and before the telescope gets on source) and the reference LST supplied to the routine (which is assumed to be the LST when the data acquisition starts).

Invocation:

```
CALL SCULIB_GET_MJD(N_FITS, FITS, MJD, STATUS)
```

Arguments:

N_FITS = _INTEGER (Given)
Size of FITS array

FITS = _CHARACTER (Given)
FITS values

LST_REF = DOUBLE PRECISION (Given)
Reference LST. This is the LST at which data acquisition begins. The MJD should be calculated for the start of data taking rather than the time the telescope began its slew. This number is normally read from the LST_STRT array and should be in radians. If a negative value or a zero value is given it is assumed that no correction should be applied. This does mean that LST_REF can never be exactly zero but that is thought be a remarkably unlikely event.

MJD = DOUBLE PRECISION (Returned)
Modified Julian date of observation

EPOCH = _REAL (Returned)
Epoch of observation

STARTUP_TIME = REAL (Returned)
The time taken, in seconds, for the observation to begin. If the time stored in the STSTART is larger than LST_REF the assumption is that a day boundary has been crossed since the LST_REF should always be more recent than STSTART (the data can not be taken before the observation starts). A startup time longer than 5 minutes or so should be treated with suspicion. If the startup time is longer than 10 minutes it is not used for the MJD calculation since this should never be the case. Set to zero if LST_REF was negative.

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

Uses the UTDATE and UTSTART and STSTART FITS headers. These must be present.

- The date must be stored in UTDATE in format YYYY:M:D
- The time must be stored in UTSTART in format HH:MM:SS.SS
- The MJD is calculated for the start of data taking rather than simply using the values stored in UTSTART. This is because there is always a delay between writing the FITS headers and taking data.

Copyright :

SCULIB_GET_RASTER
Get the raster parameters

Description:

This routine obtains the parameters of the SCAN/MAP raster: RA1, RA2, DEC1 and DEC2. The dimensions of these arrays are checked and an error returned if necessary.

Invocation:

```
CALL SCULIB_GET_RASTER(LOC, N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS,
RA1_PTR, RA2_PTR, DEC1_PTR, DEC2_PTR, STATUS)
```

Arguments:

LOC = _CHARACTER (Given)
Locator to the SCUBA extension

N_SWITCHES = _INTEGER (Given)
Actual number of switches

N_EXPOSURES = _INTEGER (Given)
Actual number of exposures

N_INTEGRATIONS = _INTEGER (Given)
Actual number of integrations

N_MEASUREMENTS = _INTEGER (Given)
Actual number of measurements

RA1_PTR = _INTEGER (Given)
Pointer to mapped RA1 array

RA2_PTR = _INTEGER (Given)
Pointer to mapped RA2 array

DEC1_PTR = _INTEGER (Given)
Pointer to mapped DEC1 array

DEC2_PTR = _INTEGER (Given)
Pointer to mapped DEC2 array

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_GET_SUB_BOLS
copy bolometers belonging to a particular sub- instrument from the input data array to the output

Description:

This routine extracts data for bolometers belonging to a particular sub-instrument from an input data array which may contain data for several sub-instruments. The input array IN_POINTER points to the indices in the first dimension of the input data array that contain the data of interest. IN_POINTER should have been calculated by an earlier call to SCULIB_CALC_SUB_BOLS.

Invocation:

```
CALL SCULIB_GET_SUB_BOLS (N_BOL_IN, N_POS, N_BEAM, IN_DATA, IN_VARIANCE, IN_QUALITY,
N_BOL_OUT, IN_POINTER, OUT_DATA, OUT_VARIANCE, OUT_QUALITY, STATUS)
```

Arguments:

N_BOL_IN = INTEGER (Given)
number of bolometers in input array

N_POS = INTEGER (Given)
number of positions measured in input array

N_BEAM = INTEGER (Given)
the number of beams in the input array

IN_DATA (N_BOL_IN,N_POS,N_BEAM)
= REAL (Given) input data array

IN_VARIANCE (N_BOL_IN,N_POS,N_BEAM)
= REAL (Given) variance on IN_DATA

IN_QUALITY (N_BOL_IN,N_POS,N_BEAM)
= BYTE (Given) quality on IN_DATA

N_BOL_OUT = INTEGER (Given)
number of bolometers in output array

IN_POINTER (N_BOL_OUT) = INTEGER (Given)
pointers from bolometers in output array to their indices in input

OUT_DATA (N_BOL_OUT,N_POS,N_BEAM)
= REAL (Returned) output data array

OUT_VARIANCE (N_BOL_OUT,N_POS,N_BEAM)
= REAL (Returned) variance on OUT_DATA

OUT_QUALITY (N_BOL_OUT,N_POS,N_BEAM)
= BYTE (Returned) quality on OUT_DATA

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_GET_SUB_INST
Ask for the specific sub-instrument

Description:

This routine finds which sub-instruments are present in the file. If there is only one this is returned, otherwise the user is given a choice.

Invocation:

```
CALL SCULIB_GET_SUB_INST(PACKAGE, N_FITS, FITS, PARAM, SUB_POINTER, WAVE, INST,
FILT, STATUS)
```

Arguments:

PACKAGE = _CHAR (Given)
Name of software package for informational message

N_FITS = _INTEGER (Given)
Number of FITS items

FITS(N_FITS) = _CHAR∗80 (Given & Returned)
The FITS array

PARAM = _CHAR (Given)
Name of the ADAM parameter used to ask for sub instrument

N_SUB = _INTEGER (Returned)
Number of sub instruments in file

SUB_POINTER = _INTEGER (Returned)
Index of selected sub instrument

WAVE = _REAL (Returned)
Wavelength (microns) of selected sub instrument

INST = _CHAR (Returned)
Name of selected sub instrument

FILT = _CHAR (Returned)
Name of selected filter

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

I actually need to include REDS_SYS (for SCUBA__MAX_SUB) but I am doing it by hand at the moment.

Copyright :

SCULIB_INSERT_BOL
To insert a single bolometer into a data array

Description:
This routine inserts a single bolometer into a SCUBA data array.

Invocation:

```
CALL SCULIB_INSERT_BOL(NBOL, N_BOLS, N_POS, BOLDATA, BOLQUAL, SCUDATA, SCUQUAL,
STATUS)
```

Arguments:

NBOL = INTEGER (Given)
The bolometer number

N_BOLS = _INTEGER (Given)
Total number of bolometers in data array

N_POS = INTEGER (Given)
Number of jiggle positions in data set

BOLDATA(N_POS) = REAL (Given)
The specified bolometer data

BOLQUAL(N_POS) = _BYTE (Given)
The bolometer quality

SCUDATA(N_BOLS, N_POS) = REAL (Given & Returned)
The data

SCUQUAL(N_BOLS, N_POS) = BYTE (Given & Returned)
The data quality

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

Implementation Status:
Propogates quality

SCULIB_INTEGRATE_PHOTOM_JIGGLE
integrate the jiggle map made by a bolometer during a PHOTOM observation

Description:

This routine just sums the data for the specified bolometer over the jiggle pattern.

After checking status on entry the routine checks that the bolometer to be integrated is among those that were measured. All being well it then loops through the jiggle pattern summing the valid data and variance measured at each position for that bolometer. If no valid data were obtained then the result quality will be set bad.

Invocation:

```
CALL SCULIB_INTEGRATE_PHOTOM_JIGGLE (BOL, N_BOLS, J_COUNT, DATA, VARIANCE, QUALITY,
RESULT_D, RESULT_V, RESULT_Q, STATUS)
```

Arguments:

BOL = INTEGER (Given)
the index of the bolometer whose data is to be integrated

N_BOLS = INTEGER (Given)
the number of bolometers being measured

J_COUNT = INTEGER (Given)
the number of jiggles in the pattern

DATA (N_BOLS, J_COUNT) = REAL (Given)
the measured data

VARIANCE (N_BOLS, J_COUNT) = REAL (Given)
the variance

QUALITY (N_BOLS, J_COUNT) = INTEGER (Given)
the quality

RESULT_D = REAL (Returned)
the sum of the input data

RESULT_V = REAL (Returned)
the sum of the input variances

RESULT_Q = INTEGER (Returned)
the quality on the sum

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_INVERT_MATRIX
invert a square matrix

Description:

This routine inverts a matrix by a method opaque to casual inspection but which has the advantage that it doesn't require any more space than the input matrix itself provides. The method is described in 'Data Reduction and Error Analysis for the Physical Sciences' by Bevington and Robinson, and the code is adapted from the Pascal version listed there.

Invocation:

```
CALL SCULIB_INVERT_MATRIX (M, ARRAY, DET, IK, JK, STATUS)
```

Arguments:

M = INTEGER (Given)
the dimensions of the matrix

ARRAY (M,M) = DOUBLE PRECISION (Given and returned)
the matrix to be inverted

DET = DOUBLE PRECISION (Returned)
the determinant of the matrix

IK (M) = INTEGER (Scratch)
i indices of largest array elements

JK (M) = INTEGER (Scratch)
j indices of largest array elements

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_J_THEORETICAL
calculate the theoretical sky brightness temperature

Description:

This routine calculates the theoretical sky brightness temperature according to Bill Duncan's model:-

$$J_{\rm meas} = (1-\eta_{\rm tel})J_{\rm tel} + \eta_{\rm tel}J_{\rm atm} - b\eta_{\rm tel}J_{\rm atm}e^{-A\tau}, \quad (52)$$

Jtel is known and Jatm is related to Jamb by:-

$$J_{\rm atm} = J_{\rm amb}\int_0^{40} A\left[k\exp\left(-\frac{h}{h_2}\right)\exp\left[Akh_2\left(\exp\left(-\frac{h}{h_2}\right)-1\right)\right]\left(1-\frac{h}{h_1}\right)\right]{\rm d}h, \quad (53)$$

where h2 = scale height of absorption (= 2km), h1 = coefficient to give 6.5K/km temperature drop in absorber, A = airmass, k = extinction.

which relation has been fudged by Bill Duncan to:-

$$J_{\rm atm} = J_{\rm amb}X_{\rm g}\left[1-\exp\left(-Akh_2\right)\right], \quad (54)$$

with $X_{\rm g}$ having the form

$$X_{\rm g} = 1 + \frac{h_2 T_{\rm lapse}}{T_{\rm amb}}\exp\left(-\frac{A\tau}{X_{\rm gconst}}\right) \quad (55)$$

with

h1 = 2, h2 = -6.5 (note this h2 is defined differently to that in the previous equation), $X_{\rm gconst}$ = 3.669383

where Jatm is the effective brightness temperature of the sky.

The routine will return immediately if entered with bad status. Errors will be reported and bad status returned if:-

- TAUZ is less than 0
- AIRMASS is less than 0
- WAVELENGTH is less than or equal to 0
- T_AMB is less than or equal to 0
- ETA_TEL is outside the range 0 to 1
- B is outside the range 0 to 1

Invocation:

```
CALL SCULIB_J_THEORETICAL (TAUZ, AIRMASS, T_TEL, T_AMB, WAVELENGTH, ETA_TEL, B,
J_THEORETICAL, STATUS)
```

Arguments:

TAUZ = REAL (Given)
zenith sky optical depth

AIRMASS = REAL (Given)
airmass

T_TEL = REAL (Given)
temperature of the telescope (K)

T_AMB = REAL (Given)
temperature of ambient air (K)

WAVELENGTH = REAL (Given)
wavelength of interest (microns)

ETA_TEL = REAL (Given)
telescope transmission

B = REAL (Given)
bandwidth factor

J_THEORETICAL = REAL (Returned)
the theoretical sky brightness temperature

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_JNU
Calculate the Rayleigh-Jeans corrected brightness temperature

Description:

This function calculates the Rayleigh-Jeans corrected brightness temperature of radiation at frequency NU (ν) and temperature T.

$$J_\nu = \frac{XT}{\exp(X) - 1} \tag{56}$$

where

$$X = \frac{h\nu}{kT} \tag{57}$$

If the absolute value of X is less than 1e-4 JNU = T, or if the absolute value of X is greater than 20 JNU = 0.

Invocation:

```
JNU = SCULIB_JNU (NU, T, STATUS)
```

Arguments:

NU = REAL (Given)
frequency (Hz)

T = REAL (Given)
temperature (K)

STATUS = INTEGER (Given and returned)
global status

Returned Value:

SCULIB_JNU = REAL
Rayleigh-Jeans corrected brightness temperature

Copyright :

SCULIB_LINEAR_WTINIT
Set up weighting function for linear rebinning

Description:

This is a FORTRAN version of the C transputer code. Here is the C description: Initialise the array containing the weighting function for linear interpolation. The weighting function is a cone, going from 1.0 at the centre to 0.0 at the edge. The look-up table represents a radial slice through the cone, tabulated to enable a look-up in terms of the square of the radius.

Invocation:

```
CALL SCULIB_LINEAR_WTINIT(WTFN, RES, STATUS)
```

Arguments:

WTFN (RES * RES) = REAL (Returned)

The weighting function generated by this routine. The index corresponds to the square of the distance from the centre in scale units.

RES = INTEGER (Given)

Number of points per scale lenghth.

STATUS = INTEGER (Given & Returned)

Global status

Copyright :

SCULIB_LST
returns LST in radians

Description:

Calculates the current LST in radians.

Invocation:

```
LST = SCULIB_LST
```

Arguments:

None

Returned Value:

LST = DOUBLE PRECISION
LST in radians

Copyright :

SCULIB_MALLOC
get virtual memory

Description:

This routine gets SIZE bytes of virtual memory.

If status is bad on entry the routine will return immediately.

If START_PTR is not equal to 0 then

- It's possible that VM may have already been allocated to START_PTR so issue an error message and set status bad.

else

- Call PSX_MALLOC to allocate the required memory plus space for 2 integers at either end.
- If status is good then Set START_PTR and END_PTR to point to the first and last bytes in the section of memory to be used and call SCULIB_CFILLI to set the sentinel integers at either end of the block to 37. SCULIB_FREE will check these for corruption when the time comes to free the VM.

end if

Invocation:

```
CALL SCULIB_MALLOC (SIZE, START_PTR, END_PTR, STATUS)
```

Arguments:

SIZE = INTEGER (Given)
number of bytes required

START_PTR = INTEGER (Given and returned)
pointer to beginning of virtual memory

END_PTR = INTEGER (Returned)
pointer to end of virtual memory

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_MAP_ALLAN_VARIANCE
incorporate latest set of MAP demodulated data into Allan variance

Description:

This routine incorporates a data slice into a run of data and updates the Allan variance of the run. The Allan variance is calculated for a range of simulated integration times for which artificial samples are calculated from the input data. The Allan variance for a particular integration time is calculated from:-

$$variance = \sum_{i=1 \to n-1} \frac{(sample(i) - sample(i+1))^2}{2n} \tag{58}$$

where n is the number of artificial samples available.

Invocation:

```
CALL SCULIB_MAP_ALLAN_VARIANCE (DEMOD, N_BOLS, N_SAMPLES, N_ALLAN_BOL, ALLAN_BOL,
KMAX, SUM, N_SUM, ARTIFICIAL, N_ARTIFICIAL, ALLAN_VARIANCE, ALLAN_QUALITY, STATUS)
```

Arguments:

DEMOD (4, N_BOLS, N_SAMPLES) = REAL (Given)
the demodulated data (data,-,-,quality)

N_BOLS = INTEGER (Given)
the number of bolometers taking data

N_SAMPLES = INTEGER (Given)
the number of samples taken for each bolometer

N_ALLAN_BOL = INTEGER (Given)
the number of bolometers whose data is to be averaged together at each sample position

ALLAN_BOL (N_ALLAN_BOL) = INTEGER (Given)
the positions in the demodulated data array of the bolometers whose measurements are to be used for the calculation of Allan variance

KMAX = INTEGER (Given)
the size of the Allan variance array

SUM (KMAX) = REAL (Given and returned)
the accumulator for current artificial sample of length K real samples

N_SUM (KMAX) = INTEGER (Given and returned)
the number of samples currently in the artificial sample accumulator

ARTIFICIAL (KMAX) = REAL (Given and returned)
the last artificial sample of length K samples that was calculated

N_ARTIFICIAL (KMAX) = INTEGER (Given and returned)
the number of artificial samples of length K samples that have been calculated

ALLAN_VARIANCE (KMAX) = REAL (Given and returned)
the Allan variance

ALLAN_QUALITY (KMAX) = INTEGER (Returned)
quality on the Allan variance

STATUS = INTEGER (Given and returned)
global status

Method :

If status on entry is good the routine will:-

loop through the data samples to be incorporated -

loop through the bolometers whose data is to be averaged into this sample -

if the bolometer was measured - if the measurement had good quality -

add the measurement to the average for this sample

end if end if

end of loop

if any bolometers had good data for this sample -

calculate the sample average

loop through the Allan variances -

add the sample to the buffer used to calculate the latest artificial sample at the simulated integration time of this variance

if all the data has been obtained for the latest artificial sample -

calculate the artificial sample

if there is only one artificial sample for this simulated integration time we can't calculate the Allan variance, so set its quality to bad

if there are 2 artifical samples then the Allan variance can be calculated from -

$$variance = \frac{(artificial\_2 - artificial\_1)^2}{2 \times 2} \tag{59}$$

if there are more than 2 artificial samples

the sum of the squares of the differences between the previous samples is recovered from the current value of the Allan variance

the square of the difference between the current artificial sample and the previous one is added to the sum

the Allan variance is re-calculated

end if

the current artifical sample is stored to be used as the 'previous' sample next time round

the variables used to calculate the artificial samples at this simulated integration time are reset

end if

end of loop through Allan variances

end if

end of loop through samples in this dataslice

Copyright :

SCULIB_MASK_DATA
Set a data array from a SCUBA section

Description:

From an array of SCUBA sections, this routine creates a mask which is then used to modify the given data array

Invocation:

```
CALL SCULIB_MASK_DATA( STATUS )
```

Arguments:

USE_SECT = LOGICAL (Given)
Are we using the section (TRUE) or inverse (FALSE)

TYPE = CHAR (Given)
Type of data. Allowed values are: REAL - data is real array BYTE - data is byte array BIT - affect single bit of byte array

N_SPEC = INTEGER (Given)
Number of specifications supplied in SPEC

SPEC(N_SPEC) = CHARACTER∗(∗) (Given)
the specification to be decoded

DEMOD_POINTER (N_SWITCHES, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS)

= INTEGER (Given)
the pointer to the location in the main data array of the data for each switch of the observation

N_SWITCHES = INTEGER (Given)
the number of switches per exposure

N_EXPOSURES = INTEGER (Given)
the number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
the number of integrations per measurement

N_MEASUREMENTS = INTEGER (Given)
the number of measurements in the observation

N_POS = INTEGER (Given)
the number of positions measured in the observation

N_BOLS = INTEGER (Given)
the number of bolometers measured in the observation

N_BEAM = INTEGER (Given)
Number of beams in DATA_PTR

SWITCH_EXPECTED = LOGICAL (Given)
.TRUE. if a switch component is allowed in the data-spec

VALUE = REAL (Given)
If TYPE is 'REAL' then use this value for masked data

BVALUE = BYTE (Given)
If TYPE is 'BYTE' then use this value for masked data

BITNUM = INTEGER (Given)
If TYPE is 'BIT' then affect this bit position

BIT_STATE = LOGICAL (Given)
If TYPE is 'BIT' then clear the bit (FALSE) or set it (TRUE)

DATA_PTR = INTEGER (Given & Returned)
Pointer to data array that is to be masked

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_MJD_TO_DATEOBS
Convert a modified julian date to the correct FITS DATE-OBS format

Description:

Given a modified Julian date generate a correctly formatted FITS DATE-OBS string.

Invocation:

```
CALL SCULIB_MJD_TO_DATEOBS ( MJD, DATE_OBS, STATUS )
```

Arguments:

MJD = DOUBLE PRECISION (Given)

Modified Julian date (Defined as JD - 2400000.5)

DATE_OBS = CHARACTER * (*) (Returned)

FITS DATE-OBS string. Should have at least 24 characters for YYYY-MM-DDThh:mm:ss.sssZ format

STATUS = INTEGER (Given & Returned)

Global Status

Notes:

The format used for the DATE-OBS keyword depends on the value of the keyword. For DATE-OBS $<$ 1999.0, use the old "dd/mm/yy" format. Otherwise, use the new "ccyy-mm-ddThh:mm:ss[.ssss]Z" format.

Copyright :

SCULIB_MRQCOF
called by SCULIB_MRQMIN

Description:

Utility routine called by SCULIB_MRQMIN, copied from Numerical Recipes in Fortran, p. 681. Used by SCULIB_MRQMIN to evaluate the linearised fitting matrix ALPHA, and vector BETA, and calculate chi-squared. Copied from MRQCOF on p.681 of Numerical Recipes in Fortran, with STATUS added.

Invocation:

```
CALL SCULIB_MRQCOF (X, Y, SIG, NDATA, A, IA, MA, ALPHA, BETA, NALP, CHISQ, FUNCS,
STATUS)
```

Arguments:

X(NDATA) = REAL (Given)
X Data points

Y(NDATA) = REAL (Given)
Y data points

SIG (NDATA) = REAL (Given)

NDATA = INTEGER (Given)
Number of data points in arrays

A(MA) = REAL (Given & Returned)
The input parameter values. Contains the best-fit values on exit.

IA(MA) = INTEGER (Given)
Array specifying which components should be fitted. A zero indicates the parameter should not be fitted – the parameters are kept at their input values. Non-zero indicates the parameter is free to be modified.

MA = INTEGER (Given)
Number of coefficients/parameters for function.

ALPHA(NALP, NALP) = REAL (Returned)

BETA(MA) = REAL (Returned)

NALP = INTEGER (Given)
Size of ALPHA.

CHISQ = REAL (Returned)
ChiSq of fit.

FUNCS = REAL FUNCTION (Given)

The function to be fitted. Must take arguments of X, A, YFIT, DYDA and MA, where YFIT is the fitting function and DYDA the the derivatives with respect to parameters A at X.

STATUS = INTEGER (Given & Returned)

Global Status

Copyright :

SCULIB_MRQMIN
Levenberg-Marquardt method non-linear least-squares fit from Numerical Recipes in FORTRAN

Description:

Levenberg-Marquardt method, attempting to reduce the value of chi-squared of a fit between a set of data points X(1:NDATA), Y(1:NDATA), and a non-linear function dependent on MA coefficients A(1:MA). The input array IA(1:MA) indicates by non-zero components those components of A that should be fitted for, and by 0 entries those components that should be held fixed at their input values. The program returns current best-fit values for the parameters A(1:MA), and chi-squared = CHISQ. The arrays COVAR (1:NCA,1:NCA), ALPHA (1:NCA,1:NCA) with physical dimension NCA (>= the number of fitted parameters) are used as working space during most iterations. Supply a subroutine FUNCS (X, A, YFIT, DYDA, MA) that evaluates the fitting function YFIT, and its derivatives DYDA with respect to the the fitting paramters A at X. On the first call provide an initial guess for the parameters A, and set ALAMDA < 0 for initialisation (which then sets ALAMDA = 0.001). If a step succeeds CHISQ becomes smaller and ALAMDA decreases by a factor of 10. If a step fails ALAMDA grows by a factor of 10. You must call this routine repeatedly until convergence is achieved. Then, make one final call with ALAMDA = 0, so that COVAR (1:MA,1:MA) returns the covariance matrix, and ALPHA the curvature matrix. Parameters held fixed will return zero covariance. Copied from MRQMIN on p.680 of Numerical Recipes in Fortran, with STATUS added.

Invocation:

```
CALL SCULIB_MRQMIN (X, Y, SIG, NDATA, A, IA, MA, COVAR, ALPHA, NCA, CHISQ, FUNCS,
ALAMDA, STATUS)
```

Arguments:

X(NDATA) = REAL (Given)
X Data points

Y(NDATA) = REAL (Given)
Y data points

SIG (NDATA) = REAL (Given)

NDATA = INTEGER (Given)
Number of data points in arrays

A(MA) = REAL (Given & Returned)
The input parameter values. Contains the best-fit values on exit.

IA(MA) = INTEGER (Given)
Array specifying which components should be fitted. A zero indicates the parameter should not be fitted – the parameters are kept at their input values. Non-zero indicates the parameter is free to be modified.

MA = INTEGER (Given)
Number of coefficients/parameters for function.

COVAR (NCA, NCA) = REAL (Returned)
The covariance matrix. Fixed parameters will return zero covariance.

ALPHA (NCA, NCA) = REAL (Returned)
The curvature matrix.

NCA = INTEGER (Given)
Size of output arrays. Must be greater than the number of fitted parameters

CHISQ = REAL (Returned)
ChiSq of fit.

FUNCS = REAL FUNCTION (Given)
The function to be fitted. Must take arguments of X, A, YFIT, DYDA and MA, where YFIT is the fitting function and DYDA the the derivatives with respect to parameters A at X.

ALAMDA = REAL (Given & Returned)
The stepping factor for the iteration. Should be set to <0 initially.

STATUS = INTEGER (Given & Returned)
Global Status

Copyright :

SCULIB_MULCAD
multiply double precision array by a constant double

Description:

multiplies a double precision array by a double constant

Invocation:

```
CALL SCULIB_MULCAD (N, IN, DVAL, OUT, STATUS)
```

Arguments:

N = INTEGER (Given)
number of array elements

IN (N) = DOUBLE (Given)
array to be multiplied

DVAL = DOUBLE (Given)
multiplication factor

OUT (N) = DOUBLE (Returned)
output array (can be same as input)

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_MULCAR
multiply real array by a constant

Description:

multiplies a real array by a real constant

Invocation:

```
CALL SCULIB_MULCAR (N, IN, RVAL, OUT)
```

Arguments:

N = INTEGER (Given)
number of array elements

IN (N) = REAL (Given)
array to be multiplied

RVAL = REAL (Given)
multiplication factor

OUT (N) = REAL (Returned)
output array (can be same as input)

Copyright :

SCULIB_MULTARE
multiplies 2 real arrays with optional error and quality handling

Description:

Multiplies two floating point arrays. The arrays may have any dimensions; they are treated here as linear in order to generate more efficient code.

Invocation:

```
CALL SCULIB_MULTARE (NELM, ARRAY1, ARRAY2, ARRAY3, Q1DATA, Q2DATA, Q3DATA, E1DATA,
E2DATA, E3DATA, QUAL, FLAGS, FBAD, VARIANCE)
```

Arguments:

NELM = INTEGER (Given)
Number of elements in each array

ARRAY1 (NELM) = REAL (Given)
Input array

ARRAY2 (NELM) = REAL (Given)
Second input array.

ARRAY3 (NELM) = REAL (Returned)
Result array. ARRAY3=ARRAY1∗ARRAY2

Q1DATA (NELM) = INTEGER (Given)
Quality array for first input array

Q2DATA (NELM) = INTEGER (Given)
Quality array for second input array

Q3DATA (NELM) = INTEGER (Returned)
Quality array for output array

E1DATA (NELM) = REAL (Given)
Variance array for input array

E2DATA (NELM) = REAL (Given)
Variance array for second input array

E3DATA (NELM) = REAL (Returned)
Variance array for output array

QUAL = LOGICAL (Given)
True if input has quality information

FLAGS = LOGICAL (Given)
True if input has flagged data values

FBAD = REAL (Given)
Flag value

VARIANCE = LOGICAL (Given)
True if both input arrays have variance arrays

Notes:

- Any of the arrays may be the same.
- Consider using VEC_MULR (SUN/39) instead.

SCULIB_NFILLI
fill an integer array with its indices

Description:
fills an integer array with its indices

Invocation:

```
CALL SCULIB_NFILLI (N, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

ARRAY (N) = INTEGER (Returned)
array to be filled

Copyright :

Implementation Status:
No status checking

SCULIB_NFILLR
fill a real array with its indices

Description:
fills a real array with its indices

Invocation:

```
CALL SCULIB_NFILLR (N, ARRAY)
```

Arguments:

N = INTEGER (Given)
number of array elements

ARRAY (N) = REAL (Returned)
array to be filled

Copyright :

Implementation Status:
No status checking

SCULIB_NOISE_MEAN
Calculate statistics of NOISE data from raw demodulated data

Description:

This routine calculates the mean and variance of a set of integrations in a NOISE measurement. If, only one integration was taken then the demodulated chop signal, chop variance, calibrator signal, calibrator variance and quality will be returned in CHOP_DATA, CHOP_VARIANCE, CAL_DATA, CAL_VARIANCE and QUALITY as they were obtained from the transputers. If more than 1 integration was obtained then:-

If, for a given bolometer, no valid data was obtained for any of the integrations then the QUALITY for that bolometer will be returned as 1.

If, for that bolometer, one valid integration was obtained then the demodulated chop signal and variance will be returned in CHOP_DATA and CHOP_VARIANCE, the calibrator signal and variance will be returned in CAL_DATA and CAL_VARIANCE, and QUALITY will be set to 0.

If more than one valid integration was obtained then the means of the chop and calibrator signals will be returned in CHOP_DATA and CAL_DATA, and the variances of the individual samples in CHOP_VARIANCE and CAL_VARIANCE, and QUALITY will be set to 0.

Invocation:

```
SCULIB_NOISE_MEAN ( N_INTEGRATIONS, N_BOLS, DEMOD, CHOP_DATA, CHOP_VARIANCE, CAL_DATA,
CAL_VARIANCE, QUALITY, WORKSPACE, STATUS)
```

Arguments:

N_INTEGRATIONS = INTEGER (Given)
number of integrations taken in measurement

N_BOLS = INTEGER (Given)
number of bolometers measured

DEMOD (5, N_BOLS, N_INTEGRATIONS) = REAL (Given)
the demodulated data; chop, chop variance, calibrator, cal variance, quality

CHOP_DATA (N_BOLS) = REAL (Returned)
the mean of the demodulated chop signal over the integrations

CHOP_VARIANCE (N_BOLS) = REAL (Returned)
the variance on CHOP_DATA, calculated from the dispersion about the mean, or the demodulated variance if there was only one integration

CAL_DATA (N_BOLS) = REAL (Returned)
the mean of the demodulated calibrator signal over the integrations

CAL_VARIANCE (N_BOLS) = REAL (Returned)
the variance on CAL_DATA, calculated from the dispersion about the mean, or the demodulated variance if there was only one integration

QUALITY (N_BOLS) = BYTE (Returned)
the quality on the returned data

WORKSPACE (N_BOLS) = INTEGER (Scratch)
some scratch space for keeping track of the number of valid observations to be included in the averaging for each bolometer

STATUS = INTEGER (Given & Returned)
inherited status

Copyright :

SCULIB_PAR_GET0?
Wrapper for the standard PAR_GET0 routines

Description:

This routine provides a wrapper for PAR_GET0? so that STATUS can be checked and an informative error message added without adding large numbers of lines to other routines.

Invocation:

```
CALL SCULIB_PAR_GET0?(PARAM, VALUE, STATUS)
```

Arguments:

PARAM = CHARACTER (Given)
Name of requested parameter

VALUE = TYPE (Returned)
Parameter value (bad if bad status from PAR_GET)

STATUS = INTEGER (Given & Returned)
Global status value

Copyright :

Implementation Status:

- All return types are present as different subroutines

SCULIB_PHOTOM_BOLSELECT
select photometers for a PHOTOM observation

Description:

This routine selects the bolometers to be used in a PHOTOM observation. If status is good on entry the routine will start by setting to zero the arrays describing which bolometers have been selected. Then it will call SCULIB_BOLSELECT to get the chan/ADC numbers of the bolometers directly specified in BOLOMETERS.

The observer can specify between 1 and 3 bolometers by name. Otherwise an error will be reported and the routine will return with bad status. Likewise, if any of these directly selected bolometers have bad quality or have identical Nasmyth offsets.

Now the routine branches on the number of directly selected bolometers:-

- 3 bolometers specified. In this case the observer must want to chop between 3 bolometers on an array. All 3 bolometers should belong to a single array, if not the routine will error and return with bad status. The routine will arbitrarily assign the 3 bolometers to the 'left', 'middle' and 'right' projected positions in the order they were selected. The routine now checks that the 3 bolometers do lie in roughly a straight line on the sky (allowing for distortion) and that the left and right bolometers lie roughly the same distance on either side of the middle. Errors will occur and the routine return with bad status if these conditions are not met. If all is OK the routine will now store the channel/ADC numbers of the directly selected bolometers, set CHOP_COORDS to Nasmyth, the bolometer spacing to half the distance between the left and right, the chopper position angle, and the instrument 'centre' to the coords of the middle bolometer. Next, the routine will search the bolometers belonging to the other array. If 3 can be found that match the positions of the 3 direct bolometers and have good quality then the routine will store the channel/ADC numbers of those bolometers as well. Lastly, the routine will set the bolometers to actually be measured to all those belonging to the array(s) containing the bolometers already selected.
- 2 bolometers specified. In this case the observer must want to observe a source by chopping between the 2 named bolometers. The routine will arbitrarily call the first of the 2 bolometers the 'middle' projected bolometer, the second 'right'. Then it will store the channel/ADC numbers of these directly selected bolometers and set the types of sub-instruments that they imply. If any of the sub-instruments are arrays the routine will search for bolometer(s) at the same Nasmyth offset in the other array and select them too. Lastly, the routine will set the bolometers to actually be measured to all those in the selected sub-instruments. CHOP_COORDS is set to Nasmyth, the bolometer spacing to the distance between the directly selected bolometers, the chopper position angle is set, and the instrument 'centre' to the Nasmyth coords of the middle bolometer.
- 1 bolometer specified. In this case the observer must want to observe a source without chopping between different bolometers. The routine will arbitrarily call the primary selected bolometer the 'middle' projected bolometer. 'Left' and 'right' bolometers

will not be assigned. The routine will now store the channel/ADC numbers of the directly selected bolometer and set the type of sub-instrument that is implied. If the sub-instrument is one of the arrays the routine will search for a bolometer at the same Nasmyth offset in the other array and select it too. Lastly, the routine will set the bolometers to actually be measured to all those in the selected sub-instruments. The instrument 'centre' is set to the Nasmyth offset of the primary selected bolometer. CHOP_COORDS, the bolometer spacing and chopper position angle are all set to bad values so that the calling routine knows these have to be read explicitly from the observation definition file.

Finally, SCULIB_BOLSELECT is called to select for measurement all the bolometers belonging to the sub-instruments involved in this PHOTOM observation. An array is then calculated holding the index in the array of ALL the photometers to be measured of each projected bolometer in each sub-instrument that was directly selected.

Invocation:

```
CALL SCULIB_PHOTOM_BOLSELECT (BOLOMETERS, BOL_TYPE, BOL_CALIB, BOL_DU3, BOL_DU4,
BOL_QUAL, BOL_ENABLED, NUM_CHAN, NUM_ADC, BOL_SELECT_CHAN, BOL_SELECT_ADC, N_BOLS,
MAX_SUB, SUB_INSTRMNT, N_SUB, CENTRE_DU3, CENTRE_DU4, CHOP_COORDS, BOL_SPACING,
CHOP_PA, N_BOL_SUB, BOLS_MEASURED, PHOT_BEAM_CHAN, PHOT_BEAM_ADC, PHOT_BEAM_BOL,
STATUS)
```

Arguments:

BOLOMETERS = CHARACTER∗(∗) (Given)
list of bolometer selections

BOL_TYPE (NUM_CHAN, NUM_ADC)
= CHARACTER∗(∗) (Given) type of bolometer

BOL_CALIB (NUM_CHAN, NUM_ADC)
= REAL (Given) target calibrator values for bolometers

BOL_DU3 (NUM_CHAN, NUM_ADC) = REAL (Given)
Nasmyth DU3 offset of bolometer from field centre

BOL_DU4 (NUM_CHAN, NUM_ADC) = REAL (Given)
Nasmyth DU4 offset of bolometer from field centre

BOL_QUAL (NUM_CHAN, NUM_ADC)
= INTEGER (Given) quality of bolometers

BOL_ENABLED (NUM_CHAN, NUM_ADC)
= LOGICAL (Returned) .TRUE. if bolometer was selected

NUM_CHAN = INTEGER (Given)
number of channels per A/D

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_SELECT_CHAN (NUM_CHAN ∗ NUM_ADC)
= INTEGER (Returned) channel numbers of selected bolometers

BOL_SELECT_ADC (NUM_CHAN ∗ NUM_ADC)
= INTEGER (Returned) A/D card numbers of selected bolometers

N_BOLS = INTEGER (Returned)
total number of bolometers selected

MAX_SUB = INTEGER (Given)
maximum number of sub instruments

SUB_INSTRMNT (MAX_SUB) = CHARACTER∗(∗) (Returned)
names of sub instrument sections to be used

N_SUB = INTEGER (Returned)
the number of sub instruments to be used

CENTRE_DU3 = REAL (Returned)
Nasmyth DU3 offset from instrument centre to which telescope is to pointed

CENTRE_DU4 = REAL (Returned)
Nasmyth DU4 offset from instrument centre to which telescope is to pointed

CHOP_COORDS = CHARACTER∗(∗) (Returned)
Chopper coordinate system required

BOL_SPACING = REAL (Returned)
Spacing between bolometers (arcsec)

CHOP_PA = REAL (Returned)
Chop position angle required (degrees)

N_BOL_SUB (MAX_SUB) = INTEGER (Returned)
Number of bolometers selected in each sub instrument

BOLS_MEASURED = CHARACTER∗(∗) (Returned)
Bolometers to be measured by transputer system

PHOT_BEAM_CHAN (3, MAX_SUB) = INTEGER (Returned)
Channel numbers of selected bolometers projected left, middle, right on sky for each sub-instrument (I index = 1 for left, 2 for middle and 3 for right)

PHOT_BEAM_ADC (3, MAX_SUB) = INTEGER (Returned)
ADC numbers of selected bolometers projected left, middle, right on sky for each sub-instrument

PHOT_BEAM_BOL (3, MAX_SUB) = INTEGER (Returned)
The index in the array of selected bolometers of those projected left, middle, right on sky for each sub-instrument.

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_POWER2
Calculate next highest power of 2

Description:

Given a number, returns that number or the next highest number that is a power of 2, and returns the power itself. If N<1, GEN_POWER2 is returned as -1, and NP2 is returned as 0.

Invocation:

```
I = SCULIB_POWER2 (N, NP2)
```

Arguments:

N = INTEGER (Given)

The number in question.

NP2 = INTEGER (Returned)

The next number that is a power of 2. Returns 0 if N is less than 1.

Returned Value:

SCULIB_POWER2 = INTEGER (Returned)

The power of 2. ie NP2=2**SCULIB_POWER2. Returns -1 if N is less than 1.

Copyright :

SCULIB_PUT_FITS_C
write a string to a FITS item specified

Description:

This routine writes a FITS character item into a character array ready to be written out to the .MORE.FITS structure in an NDF file. An error will be reported and bad status returned if the name of the FITS item is blank or more than 8 characters long. An error will also occur if the FITS character array is full. The FITS array must contain a final entry of simply 'END' unless N_FITS is 0. The END entry is automatically moved to the last field as entries are added to the FITS array.

Invocation:

```
CALL SCULIB_PUT_FITS_C (MAX_FITS, N_FITS, FITS, NAME, VALUE, COMMENT, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the size of the FITS array

N_FITS = INTEGER (Given and returned)
the number of FITS items currently in the FITS array. This should include an 'END' entry. If N_FITS is 1 it will be assumed to be an array with just an 'END'. If N_FITS=0 on entry the array will be empty and N_FITS will be changed to 2 on exit (the entry and the END card).

FITS (MAX_FITS) = CHARACTER*80 (Given and returned)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS item to be written

VALUE = CHARACTER*(*) (Given)
the value to be assigned to the item

COMMENT = CHARACTER*(*) (Given)
a comment applying to the FITS item

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_PUT_FITS_D
write a DOUBLE to a FITS item specified

Description:

This routine writes a FITS double precision item into a character array ready to be written out to the .MORE.FITS structure in an NDF file. An error will be reported and bad status returned if the name of the FITS item is blank or more than 8 characters long. An error will also occur if the FITS character array is full. The FITS array must contain a final entry of simply 'END' unless N_FITS is 0. The END entry is automatically moved to the last field as entries are added to the FITS array.

Invocation:

```
CALL SCULIB_PUT_FITS_D (MAX_FITS, N_FITS, FITS, NAME, VALUE, COMMENT, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the size of the FITS array

N_FITS = INTEGER (Given and returned)
The number of FITS items currently in the FITS array. This should include an 'END' entry. If N_FITS is 1 it will be assumed to be an array with just an 'END'. If N_FITS=0 on entry the array will be empty and N_FITS will be changed to 2 on exit (the entry and the END card).

FITS (MAX_FITS) = CHARACTER∗80 (Given and returned)
array containing the FITS items

NAME = CHARACTER∗(∗) (Given)
the name of the FITS item to be written

VALUE = DOUBLE PRECISION (Given)
the value to be assigned to the item

COMMENT = CHARACTER∗(∗) (Given)
a comment applying to the FITS item

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_PUT_FITS_I
write an integer to a FITS item specified

Description:

This routine writes a FITS integer item into a character array ready to be written out to the .MORE.FITS structure in an NDF file. An error will be reported and bad status returned if the name of the FITS item is blank or more than 8 characters long. An error will also occur if the FITS character array is full. The FITS array must contain a final entry of simply 'END' unless N_FITS is 0. The END entry is automatically moved to the last field as entries are added to the FITS array.

Invocation:

```
CALL SCULIB_PUT_FITS_I (MAX_FITS, N_FITS, FITS, NAME, VALUE, COMMENT, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the size of the FITS array

N_FITS = INTEGER (Given and returned)
The number of FITS items currently in the FITS array. This should include an 'END' entry. If N_FITS is 1 it will be assumed to be an array with just an 'END'. If N_FITS=0 on entry the array will be empty and N_FITS will be changed to 2 on exit (the entry and the END card).

FITS (MAX_FITS) = CHARACTER∗80 (Given and returned)
array containing the FITS items

NAME = CHARACTER∗(∗) (Given)
the name of the FITS item to be written

VALUE = INTEGER (Given)
the value to be assigned to the item

COMMENT = CHARACTER∗(∗) (Given)
a comment applying to the FITS item

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_RAD2STRING
Translate an angle or time in radians to a nicely formatted string

Description:

Converts an angle or time in radians to a string that can be used for display. The sexagesimal format is colon separated. For time output, the output is format SHH:MM:SS.NDP. For angle output, the format used is SDD:MM:SS.NDP. Note that the sign is used for the first character but this can be either blank or a '-'.

Invocation:

```
CALL SCULIB_RAD2STRING( ANGLE, NDP, ISTIME, RESLT, STATUS )
```

Arguments:

ANGLE = DOUBLE PRECISION (Given)
Angle to be translated (radians)

NDP = INTEGER (Given)
Number of decimal places to use for the seconds Can not be greater than 6.

ISTIME = LOGICAL (Given)
If true the angle will be assumed to be a time and will be converted to HH:MM:SS.NDP format. If false it will be assumed to be an angle and converted to SDDD:MM:SS.NDP.

RESLT = CHARACTER (Returned)
String containing the translated angle. Colon separated. Should be at least 11+NDP characters long.

STATUS = INTEGER (Given & Returned)
Global Status

Notes:

Essentially a wrapper around SLA_DR2AF and SLA_DR2TF

Copyright :

SCULIB_RANGED
Finds the maximum and minimum values in a double precision array

Description:

Find the maximum and minimum values in a double precision array.

Invocation:

```
CALL SCULIB_RANGED( ARRAY, IST, IEN, VMAX, VMIN, STATUS)
```

Arguments:

ARRAY(IEN) = DOUBLE PRECISION (Given)
Array containing the values to be checked.

IST = INTEGER (Given)
The first element of ARRAY to be examined.

IEN = INTEGER (Given)
The last element of ARRAY to be examined.

VMAX = DOUBLE PRECISION (Returned)
The maximum value of those examined

VMIN = DOUBLE PRECISION (Returned)
The minumum value of those examined

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_READ_JIGGLE
read a jiggle pattern

Description:

This routine reads in a jiggle pattern and sets some variables associated with it; the number of jiggles in the pattern, the number of jiggles to be measured in each exposure, the number of times the pattern will be repeated in each exposure, the maximum offsets in the pattern.

After checking status on entry, SCULIB_READ_NUMBERS is called to read in the jiggle offsets from the file named in JIGGLE_NAME, an error will be reported and bad status returned if no offsets are read. The maximum and minimum x and y offsets in the pattern are calculated.

Next, the variables governing the way the jiggle pattern will be divided among the exposures making up each integration is worked out. An error will be reported and bad status returned if JIGGLE_P_SWITCH is less than or equal to zero. Otherwise, the number of exposures required to execute the whole jiggle pattern, each exposure containing JIGGLE_P_SWITCH jiggles (except perhaps the last which will hold less), is calculated. If JIGGLE_P_SWITCH is larger than the number of jiggles in the pattern then the jiggle pattern will be repeated an integer number of times during each switch of the exposure and JIGGLE_P_SWITCH will be reset accordingly.

Invocation:

```
CALL SCULIB_READ_JIGGLE (JIGGLE_NAME, MAX_JIGGLE, JIGGLE_P_SWITCH, JIGGLE_REPEAT,
EXP_PER_INT, JIGGLE_X, JIGGLE_Y, JIGGLE_COUNT, JIGGLE_X_MAX, JIGGLE_X_MIN, JIGGLE_Y_MAX,
JIGGLE_Y_MIN, STATUS)
```

Arguments:

JIGGLE_NAME = CHARACTER*(*) (Given)
the name of the file containing the jiggle pattern

MAX_JIGGLE = INTEGER (Given)
the maximum number of jiggles that can be read

JIGGLE_P_SWITCH = INTEGER (Given and returned)
the requested (given) and actual (returned) number of jiggles that will be performed in each exposure of the integration

JIGGLE_REPEAT = INTEGER (Returned)
the number of times the jiggle pattern wil be repeated in each exposure

EXP_PER_INT = INTEGER (Returned)
the number of exposures required per integration

JIGGLE_X (MAX_JIGGLE) = REAL (Returned)
the x jiggle offsets

JIGGLE_Y (MAX_JIGGLE) = REAL (Returned)
the y jiggle offsets

JIGGLE_COUNT = INTEGER (Returned)
the number of jiggles in the pattern

JIGGLE_X_MAX = REAL (Returned)
the maximum jiggle offset in the x axis

JIGGLE_X_MIN
the minimum jiggle offset in the x axis

JIGGLE_Y_MAX = REAL (Returned)
the maximum jiggle offset in the y axis

JIGGLE_Y_MIN
the minimum jiggle offset in the y axis

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_READ_NUMBERS
routine to read numbers from an ASCII file

Description:

This routine reads a file of ASCII numbers. The name of the file is specified by FILENAME, and the routine assumes that the numbers are in NARRAY columns each containing SIZE numbers. The maximum value for NARRAY is 3. The read operation itself is free format. An error will be returned if there is trouble reading from the file. A warning will be given if there are more numbers in the file than can be read into the arrays.

Invocation:

```
CALL SCULIB_READ_NUMBERS (FILENAME, NARRAY, SIZE, ARRAY1, ARRAY2, ARRAY3, LENGTH,
STATUS)
```

Arguments:

FILENAME = CHARACTER*(*) (Given)
Name of file containing numbers

NARRAY = INTEGER (Given)
Number of columns in file (<=3)

SIZE = INTEGER (Given)
Dimension of arrays

ARRAY1 (SIZE) = REAL (Returned)
Output array to contain first column of numbers

ARRAY2 (SIZE) = REAL (Returned)
Output array to contain second column of numbers (if present)

ARRAY3 (SIZE) = REAL (Returned)
Output array to contain third column of numbers (if present)

LENGTH = INTEGER (Returned)
Number of items read into output arrays

STATUS = INTEGER (Given and returned)
Global status

Copyright :

SCULIB_READ_SKY
read sky parameters from a named file

Description:

This routine reads sky and telescope parameters for each SCUBA sub-instrument/filter combination from an ASCII file named in FILE. The format of each line in the file that contains sky information is assumed to be:-

FIT1 <qual> <sub> <filter> <eta_tel> <b> <tauz> <date> <day> <run>

The values returned for a given filter/sub-instrument will be from the good quality entry (<qual> = 0) with the highest associated <day> number in the file.

Errors will be reported and bad status returned if;

- there is an error opening or reading from the file
- <qual> does not convert to an integer
- <sub> is not one of SHORT, LONG, P1100, P1300, P2000
- <eta_tel> does not convert to a real
- <b> does not convert to a real
- <tauz> does not convert to a real
- <run> does not convert to an integer
- More than MAX_SKY combinations are read.

Blank lines are ignored as are those parts of lines following a ! character (comments) and lines not beginning with the word FIT1.

This routine reads a file written by SCUDR_END_SKYDIP, so these 2 routines should be changed together when necessary.

If the file name is 'NULL' (case insensitive) then the sky parameters will be reset to null values.

Invocation:

```
CALL SCULIB_READ_SKY (FILE, MAX_SKY, N_SKY, SUB, FILTER, ETA_TEL, B, TAUZ, DATEM,
DAY, RUN, STATUS)
```

Arguments:

FILE = CHARACTER*(*) (Given)
the name of the file

MAX_SKY = INTEGER (Given)
the maximum number of sub-instrument/filter combinations

N_SKY = INTEGER (Returned)
the number of sub-instrument/filter combinations read

SUB (MAX_SKY) = CHARACTER∗(∗) (Returned)
the name of the sub-instrument

FILTER (MAX_SKY) = CHARACTER∗(∗) (Returned)
the name of the filter

ETA_TEL (MAX_SKY) = REAL (Returned)
telescope transmission

B (MAX_SKY) = REAL (Returned)
atmospheric bandwidth b factor

TAUZ (MAX_SKY) = REAL (Returned)
zenith atmospheric optical depth

DATEM (MAX_SKY) = CHARACTER∗(∗) (Returned)
the date on which a measurement was made

DAY (MAX_SKY) = DOUBLE PRECISION (Returned)
the date and time at which the measurement was made, measured as a day number from 1st Jan

RUN (MAX_SKY) = INTEGER (Returned)
the run number of a measurement

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_READ_TAUZ
read sky zenith optical depths from a named file

Description:

This routine reads filter names and associated sky zenith opacities from an ASCII file named in FILE. The format of each line in the file that contains sky information is assumed to be:-

FIT1 <qual> <sub_inst> <filter> <eta_tel> <b> <tauz> <date> <day> <run>

Errors will be reported and bad status returned if;

- there is an error opening or reading the file
- <qual> is not an integer
- <sub-inst> is not one of SHORT, LONG, P1100, P1300, P2000
- <eta_tel> does not convert to a real
- <b> does not convert to a real
- <tauz> does not convert to a real
- <day> does not convert to a double
- <run> does not convert to an integer

The value of tauz returned for a given filter will be the good quality entry (<qual> = 0) with the highest associated <day> number in the file.

Blank lines are ignored as are those parts of lines following a ! character (comments) and any line whose first line is not FIT1.

This routine reads a file written by SCUDR_END_SKYDIP, so these 2 routines should be changed together when necessary.

Invocation:

```
CALL SCULIB_READ_TAUZ (FILE, MAX_FILT, N_FILT, FILTER, TAUZ, DATEM, DAY, RUN,
STATUS)
```

Arguments:

FILE = CHARACTER*(*) (Given)
the name of the file

MAX_FILT = INTEGER (Given)
the maximum number of filters

N_FILT = INTEGER (Returned)
the number of filters read

FILTER (MAX_FILT) = CHARACTER*(*) (Returned)
the names of the filters

TAUZ (MAX_FILT) = REAL (Returned)
zenith atmospheric optical depth for that filter

DATEM (MAX_FILT) = CHARACTER∗(∗) (Returned)
the date on which the measurement was made

DAY (MAX_FILT) = DOUBLE PRECISION (Returned)
the date and time at which the measurement was made, measured as a day number from 1st Jan

RUN (MAX_FILT) = INTEGER (Returned)
the run number of the measurement

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_READBOLS
read bolometer data from a name file

Description:

This routine reads bolometer data from an ASCII file named in FILE. The format of the file is assumed to be:-

```
 proc SET_BOLS
{ flat field data file written by SCUDR
{ Tues Apr 20 18:09:20 1993
{ observation run number 6
{ long-wave array
SETBOL name type du3 du4 calib theta a b qual day run ref
" ditto for other long-wave array bolometers "
{ short-wave array SETBOL name type du3 du4 calib theta a b qual day run ref
" ditto for other short-wave array bolometers "
{ P1100
SETBOL name type du3 du4 calib theta a b qual day run ref
{ P1300
SETBOL name type du3 du4 calib theta a b qual day run ref
{ P2000
SETBOL name type du3 du4 calib theta a b qual day run ref
end proc
```

where:-

<name> is the name of a bolometer (e.g. a16), case insensitive
<type> is the type of the bolometer, e.g. SHORT, LONG, P1100, P1300, P2000
<du3> is the Nasmyth DU3 coordinate of the bolometer
<du4> is the Nasmyth DU4 coordinate of the bolometer
<calib> is the flat-field value of the bolometer
<theta> is the angle between x axis an 'a' axis of fitted ellipse
<a> is the semi-length of the 'a' axis
<b> is the semi-length of the 'b' axis
<qual> is the quality of the bolometer; 0=good, 1=bad
<day> is the date when the information was measured, in days from 1 Jan
<run> is the number of the run when the information was measured
<ref> is the name of the reference bolometer

Values for bolometers that are not specified in the file will be set to default values; 'BAD' for strings, 0 for all other values except BOL_QUAL which will be 1.

Errors will be reported and bad status returned if:-

- <name> is not a valid bolometer name

- <du3>, <du4>, <calib>, <theta>, <a>, <b> do not convert to reals
- <qual>, <run> do not convert to integers
- <day> does not convert to double
- An attempt is made to set values for a bolometer twice

Each line in the file will be converted to upper case. Characters to the right of a { character in a line will be treated as comments and ignored.

Invocation:

```
CALL SCULIB_READBOLS (FILE, NUM_CHAN, NUM_ADC, BOL_TYPE, BOL_DU3, BOL_DU4, BOL_CALIB,
BOL_THETA, BOL_A, BOL_B, BOL_QUAL, BOL_DAY, BOL_RUN, BOL_REF, STATUS)
```

Arguments:

FILE = CHARACTER∗(∗) (Given)
the name of the file

NUM_CHAN = INTEGER (Given)
number of channels per ADC

NUM_ADC = INTEGER (Given)
number of ADC cards

BOL_TYPE (NUM_CHAN, NUM_ADC) = CHARACTER∗(∗) (Returned)
the type of bolometer; SHORT, LONG, P1100, etc.

BOL_DU3 (NUM_CHAN, NUM_ADC) = REAL (Returned)
the DU3 Nasmyth coordinate of the bolometer

BOL_DU4 (NUM_CHAN, NUM_ADC) = REAL (Returned)
the DU4 Nasmyth coordinate of the bolometer

BOL_CALIB (NUM_CHAN, NUM_ADC) = REAL (Returned)
the target calibration of the bolometer

BOL_THETA (NUM_CHAN, NUM_ADC) = REAL (Returned)
angle between x axis and 'a' axis of fitted ellipse (radians)

BOL_A (NUM_CHAN, NUM_ADC) = REAL (Returned)
half length of 'a' axis of fitted ellipse

BOL_B (NUM_CHAN, NUM_ADC) = REAL (Returned)
half length of 'b' axis of fitted ellipse

BOL_QUAL (NUM_CHAN, NUM_ADC) = INTEGER (Returned)
the bolometer quality

BOL_DAY (NUM_CHAN, NUM_ADC) = DOUBLE PRECISION (Returned)
the date when the bolometer was measured

BOL_RUN (NUM_CHAN, NUM_ADC) = INTEGER (Returned)
the run when the bolometer was measured

BOL_REF (NUM_CHAN, NUM_ADC) = CHARACTER∗(∗) (Returned)
the name of the reference bolometer used

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_REDUCE_SWITCH
reduce the demodulated data from the switches of an exposure into the exposure result

Description:

This routine reduces the switches in an exposure to give the exposure result, and returns the weight to be given to the specified projected bolometer position when combining different projected bolometers into the result for a sub-instrument. Exposure results are calculated assuming that:-

For a SQUARE chop-function -
switch 1 has the object in the R beam of the 'middle' projected bolometer and in the L beam of the 'right' bolometer.
switch 2 has the object in the L beam of the 'middle' bolometer and in R beam of the 'left' bolometer.

For a TRIPOS (3-position) chop-function -
switch 1 has the object in the M beam of the 'middle' bolometer, in the L beam of the right and the R beam of the 'left'.
switch 2 has the object in the R beam of the 'middle' bolometer and in the M beam of the 'right'.
switch 3 has the object in the L beam of the 'middle' bolometer and in the M beam of the 'left'.

The reduction method depends on the number of switches taken per exposure, the chop function used and the projected beam that the bolometers are assumed to be in.

SWITCH_PER_EXP = 1
CHOP_FUN = SQUARE (SCUBAWAVE or RAMPWAVE)
left bolometer = BAD (was never looking at source), weight = 0.0
middle bolometer = switch 1, weight = 1.0
right bolometer = - switch 1, weight = 1.0
CHOP_FUN = TRIPOS
left bolometer = - 2 ∗ switch 1, weight = 0.5
middle bolometer = switch 1, weight = 1.0
right bolometer = - 2 ∗ switch 1, weight = 0.5

SWITCH_PER_EXP = 2
CHOP_FUN = SQUARE
left bolometer = switch 2 - switch 1, weight = 0.5
middle bolometer = (switch 1 - switch 2) / 2, weight = 1.0
right bolometer = switch 2 - switch 1, weight = 0.5
CHOP_FUN = TRIPOS
left bolometer = - 2 ∗ (switch 1 - switch 2), weight = 0.5
middle bolometer = 2/3 ∗ (switch 1 - switch 2), weight = 1.5
right bolometer = -2/3 ∗ (switch 1 - switch 2), weight = 1.5

SWITCH_PER_EXP = 3
CHOP_FUN = SQUARE
error
CHOP_FUN = TRIPOS
left bolometer = -1/2 $*$ (2 $*$ switch 1 - (switch 2 + switch 3)), weight = 2/3
middle bolometer = 1/3 $*$ (2 $*$ switch 1 - (switch 2 + switch 3)), weight = 1.0
right bolometer = -1/2 $*$ (2 $*$ switch 1 - (switch 2 + switch 3)), weight = 2/3

Any other combinations of parameters will give rise to an error report and the routine will return with bad status.

Invocation:

```
CALL SCULIB_REDUCE_SWITCH (CHOP_FUN, SWITCH_PER_EXP, N_DATA, SWITCH_1_DATA, SWITCH_1_VARIA
SWITCH_1_QUALITY, SWITCH_2_DATA, SWITCH_2_VARIANCE, SWITCH_2_QUALITY, SWITCH_3_DATA,
SWITCH_3_VARIANCE, SWITCH_3_QUALITY, BEAM, EXP_DATA, EXP_VARIANCE, EXP_QUALITY,
WEIGHT, STATUS)
```

Arguments:

CHOP_FUN = CHARACTER$*$($*$) (Given)
the chop function used

SWITCH_PER_EXP = INTEGER (Given)
the number of switches per exposure

N_DATA = INTEGER (Given)
the number of measurements

SWITCH_1_DATA (N_DATA) = REAL (Given)
data for switch 1

SWITCH_1_VARIANCE (N_DATA) = REAL (Given)
variance for switch 1

SWITCH_1_QUALITY (N_DATA) = BYTE (Given)
quality for switch 1

SWITCH_2_DATA (N_DATA) = REAL (Given)
likewise for switch 2

SWITCH_2_VARIANCE (N_DATA) = REAL (Given)

SWITCH_2_QUALITY (N_DATA) = BYTE (Given)

SWITCH_3_DATA (N_DATA) = REAL (Given)
and switch 3

SWITCH_3_VARIANCE (N_DATA) = REAL (Given)

SWITCH_3_QUALITY (N_DATA) = BYTE (Given)

BEAM = INTEGER (Given)
the projected beam assumed for the bolometers; 1 = LEFT 2 = MIDDLE, 3 = RIGHT

EXP_DATA (N_DATA) = REAL (Returned)
the exposure result

EXP_VARIANCE (N_DATA) = REAL (Returned)
the variance on the result

EXP_QUALITY (N_DATA) = BYTE (Returned)
the quality on the result

WEIGHT = REAL (Returned)
the relative weight to assign to this projected-beam compared to the others when adding them together to give a final result

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_REMOVE_DEMOD_INT
remove demodulated data for one sub-instrument in an integration from a coadded result

Description:

This routine removes the demodulated data for a specified sub-instrument and integration from the coadded measurement.

After checking status on entry the routine cycles through the bolometers for which data has been taken. If the bolometer belongs to the sub-instrument whose data is to be removed, the routine will then cycle through the jiggle pattern removing the integration data from the coadd. If this reduces the number of coadded integrations to 0 then all the coadd numbers are set to 0 apart from quality, which is set to 1. If the number of coadded integrations is reduced to 1 then the coadd variance can no longer be estimated from the spread of data about the mean and will be set to 0.

Invocation:

```
CALL SCULIB_REMOVE_DEMOD_INT (REMOVE_TYPE, NUM_CHAN, NUM_ADC, BOL_TYPE, N_BOLS,
BOL_CHAN, BOL_ADC, J_COUNT, INT_DATA, INT_QUALITY, COADD_DATA, COADD_VARIANCE,
COADD_QUALITY, COADD_NUMBER, STATUS)
```

Arguments:

REMOVE_TYPE = CHARACTER*(*) (Given)
the type of the sub-instrument whose data is to be removed

NUM_CHAN = INTEGER (Given)
the number of channels per ADC in SCUBA

NUM_ADC = INTEGER (Given)
the number of ADCs in SCUBA

BOL_TYPE (NUM_CHAN, NUM_ADC) = CHARACTER*(*) (Given)
the types of the bolometers in SCUBA

N_BOLS = INTEGER (Given)
the number of bolometers being measured

BOL_CHAN (N_BOLS) = INTEGER (Given)
the channel numbers of the bolometers being measured

BOL_ADC (N_BOLS) = INTEGER (Given)
the ADC numbers of the bolometers being measured

J_COUNT = INTEGER (Given)
the number of jiggle positions in the pattern

INT_DATA (N_BOLS, J_COUNT) = REAL (Given)
the integration data

INT_QUALITY (N_BOLS, J_COUNT) = INTEGER (Given)
quality on the integration data

COADD_DATA (N_BOLS, J_COUNT) = REAL (Given and returned)
the mean of the coadded data

COADD_VARIANCE (N_BOLS, J_COUNT)
= REAL (Given and returned) the variance on the mean of the coadded data

COADD_QUALITY (N_BOLS, J_COUNT)
= INTEGER (Given and returned) the quality on the mean of the coadded data

COADD_NUMBER (N_BOLS, J_COUNT)= INTEGER (Given and returned)
the number of data coadded

STATUS = INTEGER (Given and returned)
the global status

Copyright :

SCULIB_REMOVE_LINEAR_BASELINE
Remove linear baseline from each exposure

Description:

This routine takes a data array. It then removes a linear baseline from each scan. The ends of the scan are the size of the chop throw (should be no source at the ends of a scan)

Invocation:

```
CALL SCULIB_REMOVE_LINEAR_BASELINE(DORLB, N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS,
DEM_PNTR, N_BOL, N_POS, IN_DATA, IN_QUALITY, SAMPLE_DX, CHOP_THROW, OUT_DATA,
OUT_QUALITY, BADBIT, STATUS)
```

Arguments:

DORLB = LOGICAL (Given)
control whether we are subtracting the baseline (TRUE) or storing the basline (FALSE)

N_EXPOSURES = INTEGER (Given)
maximum number of exposures per integration

N_INTEGRATIONS = INTEGER (Given)
number of integrations in the observation

N_MEASUREMENTS = INTEGER (Given)
number of measurements in the observation

DEMOD_POINTER (N_EXPOSURES, N_INTEGRATIONS, N_MEASUREMENTS) = INTEGER (Given)
array pointing to start and finish of scans in IN_DATA

N_BOL = INTEGER (Given)
the number of bolometers for which data was taken

N_POS = INTEGER (Given)
the number of positions measured in the scan

IN_DATA (N_BOL, N_POS) = REAL (Given)
the measured data

IN_VARIANCE (N_BOL, N_POS) = REAL (Given)
the measured variance

IN_QUALITY (N_BOL, N_POS) = BYTE (Given)
the quality on IN_DATA

NSTART = INTEGER (Given)
Number of points used for fit at start of scan

NEND = INTEGER (Given)
Number of points from end of scan

OUT_DATA (N_BOL, N_POS) = REAL (Returned)
the data with baseline removed

OUT_VARIANCE (N_BOL, N_POS) = REAL (Given)
the output variance

OUT_QUALITY (N_BOL, N_POS) = BYTE (Returned)
the quality on OUT_DATA

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SCULIB_REMOVE_OPACITY
remove sky opacity from demodulated data

Description:

This routine corrects demodulated data for the effect of sky opacity. All data is corrected assuming the same airmass of observation, but the zenith sky opacity used will depend which SCUBA sub-instrument each bolometer belongs to.

After checking status on entry, the routine will loop through the bolometers measured. For each bolometer it will then ascertain the appropriate zenith sky opacity from the parent sub-instrument and wavelength of observation. The total sky optical depth will then be calculated and, if this is in the range 0 to 20, the flux correction factor derived. The routine will then loop through the positions measured by this bolometer, correcting the fluxes and variances. If the bolometer data quality was bad, or if sky optical depth lay outside the above range, the data quality will be set bad and no correction applied.

Invocation:

```
CALL SCULIB_REMOVE_OPACITY (N_BOLS, N_POS, BOL_SELECT_CHAN, BOL_SELECT_ADC, NUM_CHAN,
NUM_ADC, BOL_TYPE, N_SUB, SUB_INSTRUMENT, TAUZ, AIRMASS, EXP_DATA, EXP_VARIANCE,
EXP_QUALITY, STATUS)
```

Arguments:

N_BOLS = INTEGER (Given)
the number of bolometers measured

N_POS = INTEGER (Given)
the number of positions they were measured at

BOL_SELECT_CHAN (N_BOLS) = INTEGER (Given)
the channel numbers of the measured bolometers

BOL_SELECT_ADC (N_BOLS) = INTEGER (Given)
the ADC numbers of the measured bolometers

NUM_CHAN = INTEGER (Given)
the number of channels per ADC

NUM_ADC = INTEGER (Given)
the number of ADCs

BOL_TYPE (NUM_CHAN, NUM_ADC) = CHARACTER∗(∗) (Given)
the type of the bolometer on each channel

N_SUB = INTEGER (Given)
the number of SCUBA sub-instruments being measured

SUB_INSTRUMENT (N_SUB) = CHARACTER∗(∗)
the names of the sub-instruments being measured

TAUZ (NSUB) = REAL (Given)
the zenith sky opacity for each sub-instrument

AIRMASS = REAL (Given)
the airmass at which the observations were made

EXP_DATA (N_BOLS, N_POS) = REAL (Given and returned)
the bolometer data

EXP_VARIANCE (N_BOLS, N_POS) = REAL (Given and returned)
the variance on the data

EXP_QUALITY (N_BOLS, N_POS) = REAL (Given and returned)
the quality on the data

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_REM_SKY
To remove sky background from SCUBA data

Description:

This routine goes through each jiggle position in turn, finding the sky background level by using the selected SKY bolometers. This level is removed from each bolometer. The background level is determined by the median of the SKY data for each jiggle. If requested, via ADD_BACK, the mean of the level that was removed is added back onto the data so that flux is conserved in the image.

Invocation:

```
CALL SCULIB_REM_SKY(MODE, ADD_BACK, N_BOLS, N_POS, SCUDATA, SCUVAR, SCUQUAL, CLIP,
N_SKYBOLS, SKYBOLS, BADBIT, STATUS)
```

Arguments:

MODE = CHARACTER * (*) (Given)
Clip mode

ADD_BACK = LOGICAL (Given)
Add the mean removed level back onto the final data set. (if TRUE)

N_BOLS = INTEGER (Given)
Number of bolometers in data set

N_POS = INTEGER (Given)
Number of jiggle positions in data set

SCUDATA(N_BOLS, N_POS) = REAL (Given & Returned)
The data

SCUVAR(N_BOLS, N_POS) = REAL (Given & Returned)
The variance on the data

SCUQUAL(N_BOLS, N_POS) = BYTE (Given & Returned)
The data quality

CLIP = _REAL (Given)
Iterative sigma clipping level

N_SKYBOLS = INTEGER (Given)
Number of sky bolometers

SKYBOLS (N_SKYBOLS) = INTEGER (Given)
List of sky bolometers

BADBIT = BYTE (Given)
Bad bit mask

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SCULIB_REWRITE_FITS_C
rewrite the value of specified FITS character keyword

Description:

This routine will overwrite the value of a specified FITS character keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called and be written out again afterwards for the change to take effect.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD='VALUE' / this is a comment"

It will search the input array for a line containing the required keyword and replace VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_REWRITE_FITS_C (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given and returned)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is to be changed

VALUE = CHARACTER*(*) (Given)
the value that the FITS keyword is to have

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_REWRITE_FITS_I
rewrite the value of specified FITS integer keyword

Description:

This routine will overwrite the value of a specified FITS integer keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called and be written out again afterwards for the change to take effect.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=VALUE / this is a comment"

It will search the input array for a line containing the required keyword and replace VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_REWRITE_FITS_I (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER∗(∗) (Given and returned)
array containing the FITS items

NAME = CHARACTER∗(∗) (Given)
the name of the FITS keyword whose value is to be rewritten

VALUE = INTEGER (Given)
the value that the FITS keyword is to have

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_REWRITE_FITS_R
rewrite the value of specified FITS real keyword

Description:

This routine will overwrite the value of a specified FITS real keyword held in the FITS extension of an NDF file. The FITS extension must have been read into the input array FITS before this routine is called and be written out again afterwards for the change to take effect.

The routine assumes that each line in the FITS array will contain a string with format:-

"KEYWORD=VALUE / this is a comment"

It will search the input array for a line containing the required keyword and replace VALUE. If the keyword is not found an error will be reported and bad status returned. If the keyword is found but the line does not conform to the above format an error will be reported and bad status returned.

Invocation:

```
CALL SCULIB_REWRITE_FITS_R (MAX_FITS, N_FITS, FITS, NAME, VALUE, STATUS)
```

Arguments:

MAX_FITS = INTEGER (Given)
the maximum number of items in the FITS array

N_FITS = INTEGER (Given)
the actual number of items in the FITS array

FITS (MAX_FITS) = CHARACTER*(*) (Given and returned)
array containing the FITS items

NAME = CHARACTER*(*) (Given)
the name of the FITS keyword whose value is to be rewritten

VALUE = REAL (Given)
the value that the FITS keyword is to have

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SCAN_2_RD
Calculate the apparent RA/Dec of a scan

Description:

The start and end of each scan is calculated in different coordinate systems dependent on the coordinate system of the tracking centre. This routine calculates the apparent RA/DEC centre for the supplied long and Lat using knowledge of the transputer system.

Invocation:

```
CALL SCULIB_SCAN_2_RD(VERSION, CENTRE_COORDS, RA_CEN, DEC_CEN, LONG, LAT, LST,
MJD, LAT_OBS, RA_APP, DEC_APP, STATUS )
```

Arguments:

VERSION = REAL (Given)
Version number of the file. This governs whether we need to even run this subroutine.

CENTRE_COORDS = CHAR (Given)
Centre coordinates of tracking centre

RA_CEN = DOUBLE (Given)
Apparent RA of map centre

DEC_CEN = DOUBLE (Given)
Apparent dec of map centre

LONG = DOUBLE (Given)
Longitude of array at LST

LAT = DOUBLE (Given)
Latitude of array at LST

LST = DOUBLE (Given)
Local sidereal time (radians)

MJD = DOUBLE (Given)
Modified Julian data of observation (should be the MJD of the time for which lst = LST).

LAT_OBS = DOUBLE PRECISION (Given)
Latitude of observatory in radians.

RA_APP = DOUBLE PRECISION (Returned)
Apparent RA of point at date (radians)

DEC_APP = DOUBLE PRECISION (Returned)
Apparent Dec

STATUS = INTEGER (Given and returned)
Global status

Notes:

Before November 1997 the situation is a bit tricky:

- [1] Scan ends were assumed to be RD by the transputers
- [2] The telescope assumed RJ offsets

Therefore I need to do the following to recreate what actually happened:

- [1] Calculate tangent plane offsets from the RD centre
- [2] Add these offsets onto the actual RJ tracking centre
- [3] Convert back into RD

Post November 1997 the offsets really are RD even when the telescope goes to RJ so this routine should not be called. Version 1.0 data will not be modified.

Copyright :

SCULIB_SCAN_APPARENT_TP_2_AZNA
Calculate NAsmyth and AZel coordinates for SCAN/MAP

Description:

This routine calculates the NA/AZ offsets from tangent plane offsets for SCAN map data.

Invocation:

```
CALL SCULIB_SCAN_APPARENT_TP_2_AZNA(OUT_COORDS, N_POS, N_BOL, ELEVATION, PAR_ANGLE,
BOL_DEC, BOL_RA, STATUS)
```

Arguments:

OUT_COORDS = _CHAR (Given)
Output coordinate system

N_POS = _INTEGER (Given)
Number of 'samples' taken

N_BOL = _INTEGER (Given)
Number of bolometers in the input data

ELEVATION (N_POS) = _DOUBLE (Given)
Elevation of each exposure

PAR_ANGLE (N_POS) = _DOUBLE (Given)
Parallactic angle of each exposure

RA_CEN = _DOUBLE (Given)
apparent RA of output map centre (radians) Only used for JIGGLE data.

DEC_CEN = _DOUBLE (Given)
apparent Dec of output map centre (radians) Only used for JIGGLE data.

BOL_DEC(N_BOL, N_POS) = _DOUBLE (Returned)
Apparent DEC of bolometers for each measurement for MJD_STANDARD

BOL_RA(N_BOL, N_POS) = _DOUBLE (Returned)
Apparent RA of bolometers for each measurement for MJD_STANDARD

STATUS = _INTEGER (Given & Returned)
Global status

Notes:

This routine does not annul the locator.

Prior Requirements :

The locator to the structure must already be available.

Copyright :

SCULIB_SEARCH_DATADIR
Open an NDF using the parameter system whilst searching DATADIR

Description:

This routine reads a parameter and attempt to open an NDF. If it fails it searches the directory DATADIR and then opens an NDF there. If that fails it asks again. This routine also recognises the SCUBA_PREFIX environment variable. When a number is given it is expanded to 4 digites (leading zeroes) and prepended with SCUBA_PREFIX and '_dem_' before opening the file. The priority is:

- File specified in current directory
- Check if filename was a number and look in current directory then either
- Look for full filename in DATADIR or
- Look for numbered name in DATADIR Only one of the last two will happen since I will know from the status return of CHR_CTOI whether a number was given.

Invocation:

```
CALL SCULIB_SEARCH_DATADIR(PACKAGE, PARAM, INDF, STATUS)
```

Arguments:

PACKAGE = CHAR (Given)
String to identify the software system when the information messages appear.

PARAM = CHAR (Given)
Name of parameter associated with NDF

INDF = INTEGER (Returned)
NDF identifier of successfully opened file

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

Implementation Status:

- Uses some non PSX calls for accessing Current working directory These are GETCWD and CHDIR. This may be a portability issue.
- Checks for PAR__ABORT or PAR__NULL status from parameter.

SCULIB_SET_DATA
set data to a real value given a byte mask

Description:

This routine uses a byte mask (N_BOLS ∗ N_POS) to set a data value in the output. A mask value of 1 indicates that a value should be changed. This routine does not distinguish 'beams'

Invocation:

```
CALL SCULIB_SET_DATA (USE_THIS, N_BOLS, N_POS, N_BEAM, MASK, VALUE, IN_DATA, STATUS)
```

Arguments:

USE_THIS = LOGICAL (Given)
Describes whether the mask should be set to the value (TRUE) or whether the rest of the data should be set (FALSE)

N_BOLS = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions measured

N_BEAM = INTEGER (Given)
number of beams used

MASK(N_BOLS, N_POS) = BYTE (Given)
input mask

VALUE = REAL (Given)
The value of the masked/unmasked data

IN_DATA (N_BOLS, N_POS, N_BEAM) = REAL (Given and returned)
the data to be masked

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SET_DATA_BIT
set a bit in data given a byte mask

Description:

This routine uses a byte mask (N_BOLS * N_POS) to set a data bit in the output. A mask value of 1 indicates that a value should be changed. This routine does not distinguish 'beams' A bit can be set or unset.

Invocation:

```
CALL SCULIB_SET_DATA_BIT(USE_THIS, N_BOLS, N_POS, N_BEAM, MASK, BITNUM, STATE,
IN_DATA, STATUS)
```

Arguments:

USE_THIS = LOGICAL (Given)
Describes whether the mask should be set to the value (TRUE) or whether the rest of the data should be set (FALSE)

N_BOLS = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions measured

N_BEAM = INTEGER (Given)
number of beams used

MASK(N_BOLS, N_POS) = BYTE (Given)
input mask

BITNUM = INTEGER (Given)
Bit number to affect

STATE = LOGICAL (Given)
Set the bits if true. Otherwise clear them

IN_DATA (N_BOLS, N_POS, N_BEAM) = BYTE (Given and returned)
the data to be masked

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SET_DATA_UB
set data to a byte value given a byte mask

Description:

This routine uses a byte mask (N_BOLS * N_POS) to set a data value in the output. A mask value of 1 indicates that a value should be changed. This routine does not distinguish 'beams'

Invocation:

```
CALL SCULIB_SET_DATA_UB(USE_THIS, N_BOLS, N_POS, N_BEAM, MASK, BVALUE, IN_DATA,
STATUS)
```

Arguments:

USE_THIS = LOGICAL (Given)
Describes whether the mask should be set to the value (TRUE) or whether the rest of the data should be set (FALSE)

N_BOLS = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions measured

N_BEAM = INTEGER (Given)
number of beams used

MASK(N_BOLS, N_POS) = BYTE (Given)
input mask

BVALUE = BYTE (Given)
The value to be given to the masked/unmasked data

IN_DATA (N_BOLS, N_POS, N_BEAM) = BYTE (Given and returned)
the data to be masked

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SET_QUAL
set quality bits in a subset of a quality array

Description:

Set the bits in the quality array as specified by a mask. Can be used to set or unset bits as well as using the mask or the inverse of the mask. This is used to mask scuba sections.

Invocation:

```
CALL SCULIB_SET_QUAL (USE_SECT, QUALITY, N_BOLS, N_POS, N_BEAM, BOL_S, POS_S,
BIT_POS, BIT_SWITCH, STATUS)
```

Arguments:

USE_SECT = LOGICAL (Given)
am I changing SECTION or .NOT. SECTION

QUALITY (N_BOLS, N_POS, N_BEAM) = BYTE (Given and returned)
the quality array

N_BOLS = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions measured

N_BEAM = INTEGER (Given)
number of beams used

BOL_S (N_BOLS) = INTEGER (Given)
array containing 1s for bolometers whose quality is to be changed

POS_S (N_POS) = INTEGER (Given)
array containing 1s for positions whose quality is to be changed

BIT_POS = INTEGER (Given)
position of bit ot be set (0 - 7)

BIT_SWITCH = LOGICAL (Given)
If .TRUE. we set the bit using SCULIB_BITON. If .FALSE. we unset the bit with SCULIB_BITOFF

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SET_QUALITY
set quality bits in a subset of a quality array

Description:
set quality bits in a subset of a quality array

Invocation:

```
CALL SCULIB_SET_QUALITY (N_BOLS, N_POS, N_BEAM, QUALITY, START_BOL, END_BOL, START_POS,
END_POS, START_BEAM, END_BEAM, BIT_POS, BIT_VALUE, STATUS)
```

Arguments:

N_BOLS = INTEGER (Given)
number of bolometers measured

N_POS = INTEGER (Given)
number of positions measured

N_BEAM = INTEGER (Given)
number of beams used

QUALITY (N_BOLS, N_POS, N_BEAM) = BYTE (Given and returned)
the quality array

START_BOL = INTEGER (Given)
index of first bolometer to be set bad

END_BOL = INTEGER (Given)
index of last bolometer to be set bad

START_POS = INTEGER (Given)
index of first position to be set bad

END_POS = INTEGER (Given)
index of last position to be set bad

START_BEAM = INTEGER (Given)
index of first beam to be set bad

END_BEAM = INTEGER (Given)
index of last beam to be set bad

BIT_POS = INTEGER (Given)
position of bit ot be set (0 - 7)

BIT_VALUE = INTEGER (Given)
value to which bit is to be set (0 if zero, 1 otherwise)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SET_USER
set the USER array used by SCULIB_SKYFUNC_1

Description:

E04UPF is a NAG routine that is used to fit a theoretical sky-dip curve to the measured data by varying ETA_TEL, B and TAU. SCULIB_SKYFUNC_1 is called by E04UPF to calculate the M sub-functions fi(x) at the current solution vector X, and/or their Jacobian matrix (see NAG manual for further info). That routine obtains the data needed for its calculations via the USER array passed into it, and this routine fills USER with the necessary numbers:-

- USER (1) = J_TEL
- USER (2) = J_ATM
- USER (3) = not used
- USER (4:M+3) = the measured airmasses
- USER (M+4:2M+3) = the measured sky temperatures
- USER (2M+4:3M+3) = the errors on the measured sky temperatures

Invocation:

```
CALL SCULIB_SET_USER (J_TEL, J_ATM, N_MEASUREMENTS, AIRMASS, J_MEAS_D, J_MEAS_V,
USER)
```

Arguments:

J_TEL = REAL (Given)
the temperature of the telescope

J_ATM = REAL (Given)
the temperature of the atmosphere

N_MEASUREMENTS = INTEGER (Given)
the number of measurements taken in the sky-dip

AIRMASS (N_MEASUREMENTS) = REAL (Given)
the airmass at each measurement

J_MEAS_D (N_MEASUREMENTS) = REAL (Given)
the brightness temperature of the atmosphere at each airmass

J_MEAS_V (N_MEASUREMENTS) = REAL (Given)
the variance on J_MEAS_D

USER (3 $*$ N_MEASUREMENTS $+$ 3) = DOUBLE PRECISION (Returned)
the USER array required

Copyright :

SCULIB_SINC
Calculate SIN(PI∗X)/(PI∗X)

Description:

Calculate sinc function.

Invocation:

```
RESULT = SCULIB_SINC( X )
```

Arguments:

X = REAL (Given)

The argument to the sinc function - In turns

Returned Value:

SCULIB_SINC = REAL

SINC(PI∗X)

Copyright :

Implementation Status:

- No status checking
- No bad value checking

SCULIB_SKYCON_1
routine to calculate non-linear constraints for NAG non-linear fitting routine E04UPF when fitting ETA_TEL, B and TAU in sky-dip analysis

Description:

E04UPF is a NAG routine used to fit a theoretical SKYDIP curve to the measured data by varying ETA_TEL, B and TAU. This routine is called by E04UPF to calculate the 1 non-linear constraint in the problem and/or it's Jacobian at the current solution vector X. See the NAG manual if you need to know more about E04UPF.

The solution vector is composed as follows:-

- x(1) = ETA_TEL
- x(2) = B
- x(3) = TAU

The non-linear constraint limits ETA_TEL $*$ B. So, evaluation of the constraint gives:-

C(1) = ETA_TEL $*$ B

and its Jacobian has components :-

$$\frac{\partial C}{\partial x(1)} = B \tag{60}$$

$$\frac{\partial C}{\partial x(2)} = \eta_{\mathrm{tel}} \tag{61}$$

$$\frac{\partial C}{\partial x(2)} = 0.0 \tag{62}$$

Invocation:

```
CALL SCULIB_SKYCON_1 (MODE, NCNLN, N, LDCJ, NEEDC, X, C, CJAC, NSTATE, IUSER,
USER)
```

Arguments:

See NAG manual description of CONFUN parameter in E04UPF.

SCULIB_SKYDIP_ALLAN_VARIANCE
incorporate latest set of SKYDIP data samples into Allan variance

Description:

This routine incorporates a data slice into a run of data and updates the Allan variance of the run.

The Allan variance is calculated for a range of simulated integration times for which artificial samples are calculated from the input data. The Allan variance for a particular integration time is calculated from:-

$$variance = \sum_{i=1 \rightarrow n-1} \frac{(sample(i) - sample(i+1))^2}{2n} \tag{63}$$

where n is the number of artificial samples available.

Invocation:

```
CALL SCULIB_SKYDIP_ALLAN_VARIANCE (SUB_INSTRUMENT, NUM_CHAN, NUM_ADC, BOL_TYPE,
N_BOLS, BOL_CHAN, BOL_ADC, N_SAMPLES, SAMPLE, SAMPLE_QUALITY, KMAX, SUM, N_SUM,
ARTIFICIAL, N_ARTIFICIAL, ALLAN_VARIANCE, ALLAN_QUALITY, STATUS)
```

Arguments:

SUB_INSTRUMENT = CHARACTER∗(∗) (Given)
the name of the sub-instrument for which the Allan variance is to be calculated

NUM_CHAN = INTEGER (Given)
number of channels per A/D card

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_TYPE (NUM_CHAN, NUM_ADC) = CHARACTER∗(∗) (Given)
the types of the bolometers

N_BOLS = INTEGER (Given)
the number of bolometers making measurements

BOL_CHAN (N_BOLS) = INTEGER (Given)
channel numbers of selected bolometers

BOL_ADC (N_BOLS) = INTEGER (Given)
ADC numbers of selected bolometers

N_SAMPLES = INTEGER (Given)
the number of samples

SAMPLE (N_BOLS, N_SAMPLES) = REAL (Given)
the samples

SAMPLE_QUALITY (N_BOLS, N_SAMPLES) = INTEGER (Given)
quality on the samples

KMAX = INTEGER (Given)
the size of the Allan variance array

SUM (KMAX) = REAL (Given and returned)
the accumulator for current artificial sample of length K real samples

N_SUM (KMAX) = INTEGER (Given and returned)
the number of samples currently in the artificial sample accumulator

ARTIFICIAL (KMAX) = REAL (Given and returned)
the last artificial sample of length K samples that was calculated

N_ARTIFICIAL (KMAX) = INTEGER (Given and returned)
the number of artificial samples of length K samples that have been calculated

ALLAN_VARIANCE (KMAX) = REAL (Given and returned)
the Allan variance

ALLAN_QUALITY (KMAX) = INTEGER (Returned)
quality on the Allan variance

STATUS = INTEGER (Given and returned)
global status

Copyright :

Method :

If status on entry is good the routine will:-

loop through the data samples to be incorporated -

loop through the bolometers for which data is available and calculate an average value for the sample from those bolometers belonging to the specified sub-instrument

if the quality of the average sample is good -

loop through the Allan variances -

add the sample to the buffer used to calculate the latest artificial sample at the simulated integration time of this variance

if all the data has been obtained for the latest artificial sample -

calculate the artificial sample

if there is only one artificial sample for this simulated integration time we can't calculate the Allan variance, so set its quality to bad

if there are 2 artifical samples then the Allan variance can be calculated from -

$$variance = \frac{(artificial\_2 - artificial\_1)^2}{2 \times 2} \tag{64}$$

if there are more than 2 artificial samples

- the sum of the squares of the differences between the previous samples is recovered from the current value of the Allan variance
- the square of the difference between the current artificial sample and the previous one is added to the sum
- the Allan variance is re-calculated

end if

the current artifical sample is stored to be used as the 'previous' sample next time round

the variables used to calculate the artificial samples at this simulated integration time are reset

end if

end of loop through Allan variances

end if

end of loop through samples in this dataslice

SCULIB_SKYDIP_BOLS
returns bolometers to be measured in a SKYDIP

Description:

This routine returns a string containing the names of the bolometers to be measured in a SKYDIP observation.

Only those sub-instruments that are looking out through a suitable filter will be measured. Thus, if the filter in front of the SHORT array is '350' or '450' the returned string will contain the word SHORT_DC. Other filter/sub/bolometer combinations are as follows:-

- 600 or 750 or 850 in front of the LONG array gives LONG_DC
- 1100 in front of the P1100 photometer gives P1100_DC
- 1300 in front of the P1300 photometer gives P1300_DC
- 2000 in front of the P2000 photometer gives P2000_DC

The returned BOLOMETERS string will contain the words for each sub-instrument to be measured separated by commas.

Invocation:

```
CALL SCULIB_SKYDIP_BOLS (FILTER, BOLOMETERS, STATUS)
```

Arguments:

FILTER = CHARACTER*(*) (Given)
the name of the filter combination in use

BOLOMETERS = CHARACTER*(*) (Returned)
the names of the bolometers to be measured

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SKYDIP_TEMPERATURES
derive sky temperatures from chopper-wheel data

Description:

This routine derives sky brightness temperatures from data containing the measured signals from cold and ambient loads and sky. The sky temperature is calculated for each sample by linear interpolation between the signals from the cold and ambient loads.

In addition, mean values of the sky temperature for each bolometer are derived from the samples taken. If more than one sample contributes to the mean then the variance is calculated from the spread of samples about the mean.

Invocation:

```
CALL SCULIB_SKYDIP_TEMPERATURES (T_COLD, T_HOT, N_SUB, :  SUB_INSTRUMENT, WAVELENGTH,
NUM_CHAN, NUM_ADC, BOL_TYPE, :  N_BOLS, BOL_CHAN, BOL_ADC, N_SAMPLES_IN, RAW_DATA,
:  N_SAMPLES_OUT, J_SKY, J_SKY_VARIANCE, J_SKY_QUALITY, J_SKY_AV, :  J_SKY_AV_VARIANCE,
J_SKY_AV_QUALITY, STATUS)
```

Arguments:

T_COLD (N_SUB) = REAL (Given)
the temperature of the cold load for each sub-instrument

T_HOT (N_SUB) = REAL (Given)
the temperature of the hot load for each sub-instrument

N_SUB = INTEGER (Given)
the number of sub-instruments being used

SUB_INSTRUMENT (N_SUB) = CHARACTER∗(∗) (Given)
the names of the sub-instruments being used

WAVELENGTH (N_SUB) = REAL (Given)
the wavelengths of the filters in front of the sub-instruments (microns)

NUM_CHAN = INTEGER (Given)
number of A/D channels per A/D card

NUM_ADC = INTEGER (Given)
number of A/D cards

BOL_TYPE (NUM_CHAN, NUM_ADC) = CHARACTER∗(∗) (Given)
the types of the bolometers

N_BOLS = INTEGER (Given)
the number of bolometers making measurements

BOL_CHAN (N_BOLS) = INTEGER (Given)
channel numbers of selected bolometers

BOL_ADC (N_BOLS) = INTEGER (Given)
ADC numbers of selected bolometers

N_SAMPLES_IN = INTEGER (Given)
the number of samples taken by each bolometer

RAW_DATA (3, N_BOLS, N_SAMPLES_IN) = REAL (Given)
the measured signals from the cold and ambient loads and sky

- RAW_DATA (1,i,j) = ambient load signal
- RAW_DATA (2,i,j) = sky signal
- RAW_DATA (3,i,j) = cold load signal

N_SAMPLES_OUT = INTEGER (Given)
the number of samples requested for each bolometer

J_SKY (N_BOLS, N_SAMPLES_OUT) = REAL (Returned)
the brightness temperature of the sky

J_SKY_VARIANCE (N_BOLS, N_SAMPLES_OUT) = REAL (Returned)
the variance on J_SKY

J_SKY_QUALITY (N_BOLS, N_SAMPLES_OUT) = BYTE (Returned)
the quality on J_SKY

J_SKY_AV (N_BOLS) = REAL (Returned)
the average of J_SKY for each bolometer

J_SKY_AV_VARIANCE (N_BOLS) = REAL (Returned)
the variance on the average of J_SKY

J_SKY_AV_QUALITY (N_BOLS) = BYTE (Returned)
the quality of the average of J_SKY

STATUS = INTEGER (Given and returned)
Global status

Copyright :

Implementation Status:

- Should be converted to use double precision internally since sometimes can return a negative temperature.

SCULIB_SKYDIP_VAR
calculate variance of skydip data points

Description:

If entered with good status this routine calculates the variance between a SKYDIP dataset and the theoretical curve generated by the the sky and telescope parameters in FIT. Additionally, the residual of the fit is also returned (absolute value of measured data mins fitted values). Data points with bad quality will be ignored. If no valid data points are found then an error will be reported and bad status returned. The data are passed in via common.

The FIT vector is composed as follows:-

- x(1) = ETA_TEL
- x(2) = B
- x(3) = TAU

For a given airmass:-

$$J_{\text{theory}} = (1 - \eta_{\text{tel}})J_{\text{tel}} + \eta_{\text{tel}}J_{\text{atm}}(1 - B\exp(-\tau Airmass)) \tag{65}$$

$$J_{\text{atm}} = J_{\text{amb}}X_G \tag{66}$$

$$X_G = 1 + \frac{h1 \times h2}{J_{\text{amb}}}\exp(-\tau \times Airmass(i)/X\_Gconst) \tag{67}$$

Invocation:

```
CALL SCULIB_SKYDIP_VAR (RESIDUAL, VAR, N, FIT, NDEG, STATUS)
```

Arguments:

RESIDUAL = DOUBLE PRECISION (Returned)
residual of the fit (absolute difference between the model and the data)

VAR = DOUBLE PRECISION (Returned)
the chi-squared value of the current fit

N = INTEGER (Given)
the number of parameters being fit, should be 3

FIT (N) = DOUBLE PRECISION (Given)
the fit parameters:-

- FIT (1) = ETA_TEL
- FIT (2) = B

- FIT (3) = TAUZ

NDEG = INTEGER (Given)
the number of degrees of freedom

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SKYDIP_XISQ
calculate chi-square of sky fit

Description:

If entered with good status this routine calculates the chi-squared between a SKYDIP dataset and the theoretical curve generated by the the sky and telescope parameters in FIT. Data points with bad quality will be ignored. If no valid data points are found then an error will be reported and bad status returned. The data are passed in via common.

The FIT vector is composed as follows:-

- x(1) = ETA_TEL
- x(2) = B
- x(3) = TAU

For a given airmass:-

$$J_{\text{theory}} = (1 - \eta_{\text{tel}})J_{\text{tel}} + \eta_{\text{tel}}J_{\text{atm}}(1 - B\exp(-\tau Airmass)) \tag{68}$$

$$J_{\text{atm}} = J_{\text{amb}}X_G \tag{69}$$

$$X_G = 1 + \frac{h1 \times h2}{J_{\text{amb}}}\exp(-\tau \times Airmass(i)/X\_Gconst) \tag{70}$$

Invocation:

```
CALL SCULIB_SKYDIP_XISQ (XISQ, N, FIT, STATUS)
```

Arguments:

XISQ = DOUBLE PRECISION (Returned)
the chi-squared value of the current fit

N = INTEGER (Given)
the number of parameters being fit, should be 3

FIT (N) = DOUBLE PRECISION (Given)
the fit parameters:-

- FIT (1) = ETA_TEL
- FIT (2) = B
- FIT (3) = TAUZ

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SKYFUNC
Return the value of the skydip function

Description:

This routine returns the value of the skydip function. The parameters are passed as follows:

- P(1) = ETA_TEL
- P(2) = B
- P(3) = TAU
- P(4) = J_AMB
- P(5) = J_TEL

This subroutine returns the theoretical value of the skydip for the given input parameters.

$$J_{\text{theory}} = (1 - \eta_{\text{tel}})J_{\text{tel}} + \eta_{\text{tel}}J_{\text{atm}}(1 - B\exp(-\tau Airmass)) \tag{71}$$

$$J_{\text{atm}} = J_{\text{amb}}X_G \tag{72}$$

$$X_G = 1 + \frac{h1 \times h2}{J_{\text{amb}}}\exp(-\tau \times Airmass(i)/X\_Gconst) \tag{73}$$

Invocation:

```
CALL SCULIB_SKYFUNC(F, X, P, M)
```

Arguments:

F = REAL (Returned)
The value of the skydip function

X = REAL (Given)
The airmass used to calculate the function

P = REAL (Given)
The parameter values

M = INTEGER (Given)
The number of parameters

Copyright :

SCULIB_SKYFUNC_1
calculate f(x) and its Jacobian for NAG non-linear fitting routine E04UPF when fitting ETA_TEL, B and TAU in SKYDIP analysis

Description:

E04UPF is a NAG routine that is used to fit a theoretical sky-dip curve to the measured data by varying ETA_TEL, B and TAU. This routine is called by E04UPF to calculate the M sub-functions fi(x) at the current solution vector X, and/or their Jacobian matrix. See the NAG manual if you need to know more about E04UPF.

The solution vector is composed as follows:-

- x(1) = ETA_TEL
- x(2) = B
- x(3) = TAU

For each of the M measured airmasses the subfunction f(x) is:-

$$f(x) = \frac{J_{\text{theory}}(Airmass(i)) - J_{\text{meas}}(Airmass(i))}{J_{\text{noise}}(Airmass(i))} \tag{74}$$

where:-

$$J_{\text{theory}} = (1-\eta_{\text{tel}})J_{\text{tel}} + \eta_{\text{tel}}J_{\text{atm}}(1 - B\exp(-\tau Airmass)) \tag{75}$$

$$J_{\text{atm}} = J_{\text{amb}}X_G \tag{76}$$

$$X_G = 1 + \frac{h1 \times h2}{J_{\text{amb}}}\exp(-\tau \times Airmass(i)/X\_Gconst) \tag{77}$$

The corresponding Jacobian has component (i,j):-

$$\frac{\partial f(x)}{\partial x(j)} = \frac{\partial}{\partial x(j)}J_{\text{theory}}(Airmass(i)) \times \frac{1}{J_{\text{noise}}(i)} \tag{78}$$

leading to:

$$\frac{\partial f(i)}{\partial x(1)} = -\frac{J_{\text{tel}} + J_{\text{atm}} - BJ_{\text{tel}}\exp(-\tau A(i))}{J_{\text{noise}}(i)} \tag{79}$$

$$\frac{\partial f(i)}{\partial x(2)} = -\frac{\eta_{\text{tel}}J_{\text{atm}}\exp(-\tau A(i))}{J_{\text{noise}}(i)} \tag{80}$$

$$\frac{\partial f(i)}{\partial x(3)} = \frac{\eta_{\rm tel}\frac{\partial J_{\rm atm}}{\partial \tau} - \eta_{\rm tel} B \exp(-\tau A(i)) \left(\frac{\partial J_{\rm atm}}{\partial \tau} - J_{\rm atm} A(i)\right)}{J_{\rm noise}(i)} \tag{81}$$

where

$$\frac{\partial J_{\rm atm}}{\partial \tau} = J_{\rm amb}\frac{\partial X_G}{\partial \tau} \tag{82}$$

and

$$\frac{\partial X_G}{\partial \tau} = -\frac{H_1 H_2 A(i)}{J_{\rm amb} X_{\rm Gconst}} \exp(-\tau A(i)/X_{\rm Gconst}) \tag{83}$$

The array USER is used to get the necessary ancillary information into the routine:-

- USER (1) = J_TEL
- USER (2) = J_AMB
- USER (3) = not used
- USER (4:M+3) = the measured airmasses
- USER (M+4:2M+3) = the measured sky temperatures
- USER (2M+4:3M+3) = the errors on the measured sky temperatures

Invocation:

```
CALL SCULIB_SKYFUNC_1 (MODE, M, N, LDFJ, X, F, FJAC, NSTATE, IUSER, USER)
```

Arguments:

See NAG manual description of OBJFUN parameter in E04UPF.

Copyright :

SCULIB_SKYFUNC_2
calculate f(x) and its Jacobian for NAG non-linear fitting routine E04UPF when fitting B and TAU in sky-dip analysis

Description:

E04UPF is a NAG routine that is used to fit a theoretical sky-dip curve to the measured data by varying B and TAU. This routine is called by E04UPF to calculate the M sub-functions f(x) at the current solution vector X, and/or the Jacobian matrix. Please see the NAG manual if you need to know more about E04UPF.

The solution vector is composed as follows:-

- x(1) = B
- x(2) = TAU

For each of the M measured airmasses the subfunction f(x) is:-

$$f(x) = \frac{J_{\rm theory}(Airmass(i)) - J_{\rm meas}(Airmass(i))}{J_{\rm noise}(Airmass(i))} \tag{84}$$

where:-

$$J_{\rm theory} = (1-\eta_{\rm tel})J_{\rm tel} + \eta_{\rm tel}J_{\rm atm}(1 - B\exp(-\tau Airmass)) \tag{85}$$

The corresponding Jacobian has component (i,j):-

$$\frac{\partial f(x)}{\partial x(j)} = \frac{\partial}{\partial x(j)} J_{\rm theory}(Airmass(i)) \times \frac{1}{J_{\rm noise}(i)} \tag{86}$$

leading to:

$$\frac{\partial f(i)}{\partial x(1)} = -\eta_{\rm tel}J_{\rm atm}\exp(-\tau A(i)) \tag{87}$$

$$\frac{\partial f(i)}{\partial x(2)} = A(i)B\eta_{\rm tel}J_{\rm atm}\exp(-\tau A(i)) \tag{88}$$

The array USER is used to get the necessary ancillary information into the routine:-

- USER (1) = J_TEL
- USER (2) = J_ATM
- USER (3) = ETA_TEL
- USER (4:M+3) = the measured airmasses
- USER (M+4:2M+3) = the measured sky temperatures
- USER (2M+4:3M+3) = the errors on the measured sky temperatures

Invocation:

```
CALL SCULIB_SKYFUNC_2 (MODE, M, N, LDFJ, X, F, FJAC, NSTATE, IUSER, USER)
```

Arguments:

See NAG manual description of OBJFUN parameter in E04UPF.

SCULIB_SKYFUNCD
Return the value of the partial derivatives of the skydip function

Description:

This routine returns the value of the skydip function. The parameters are passed as follows:

- P(1) = ETA_TEL
- P(2) = B
- P(3) = TAU
- P(4) = J_AMB
- P(5) = J_TEL

The theoretical value of the skydip for the given input parameters is given by:

$$f(x) = \frac{J_{\text{theory}}(Airmass) - J_{\text{meas}}(Airmass)}{J_{\text{noise}}(Airmass)} \tag{89}$$

where:-

$$J_{\text{theory}} = (1 - \eta_{\text{tel}})J_{\text{tel}} + \eta_{\text{tel}}J_{\text{atm}}(1 - B\exp(-\tau Airmass)) \tag{90}$$

$$J_{\text{atm}} = J_{\text{amb}}X_G \tag{91}$$

$$X_G = 1 + \frac{h1 \times h2}{J_{\text{amb}}}\exp(-\tau \times Airmass/X\_Gconst) \tag{92}$$

The partial derivatives are:

$$\frac{\partial f(i)}{\partial \eta_{\text{tel}}} = DR(1) = -J_{\text{tel}} + J_{\text{atm}} - BJ_{\text{tel}}\exp(-\tau A) \tag{93}$$

$$\frac{\partial f(i)}{\partial B} = DR(2) = -\eta_{\text{tel}}J_{\text{atm}}\exp(-\tau A) \tag{94}$$

$$\frac{\partial X_G}{\partial \tau} = -\frac{H_1 H_2 A}{J_{\text{amb}}X_{\text{Gconst}}}\exp(-\tau A/X_{\text{Gconst}}) \tag{95}$$

$$\frac{\partial J_{\text{atm}}}{\partial \tau} = J_{\text{amb}}\frac{\partial X_G}{\partial \tau} \tag{96}$$

$$\frac{\partial f}{\partial \tau} = DR(3) = \eta_{\text{tel}}\frac{\partial J_{\text{atm}}}{\partial \tau} - \eta_{\text{tel}}B\exp(-\tau A)\left(\frac{\partial J_{\text{atm}}}{\partial \tau} - J_{\text{atm}}A\right) \tag{97}$$

$$\frac{\partial f}{\partial J_{\mathrm{amb}}} = \frac{\partial f}{\partial J_{\mathrm{tel}}} = 0.0 \quad (\text{fixed parameters}) \tag{98}$$

Invocation:

```
CALL SCULIB_SKYFUNCD(X, P, DR, M)
```

Arguments:

X = REAL (Given)
The airmass used to calculate the function

P = REAL (Given)
The parameter values

M = INTEGER (Given)
The number of parameters

DR = REAL (Returned)
The values of the partial derivatives

Copyright :

SCULIB_SPLINE_PDA_IDBVIP
Fit a surface to an irregular grid using the PDA function PDA_IDBVIP

Description:

This routine provides a wrapper for the PDA_IDBVIP spline interpolation routine. The quirks of the algorithm are dealt with here so that SCULIB_SPLINE_REGRID does not have to know anything about the interpolation routine.

Invocation:

```
CALL SCULIB_SPLINE_PDA_IDBVIP (NDP, X_IN, Y_IN, DATA_IN, N_PTS, X_OUT, Y_OUT,
DATA_OUT, STATUS)
```

Arguments:

NDP = INTEGER (Given)
Number of data points on irregular grid

X_IN (NDP) = REAL (Given)
X coordinates of input data

Y_IN (NDP) = REAL (Given)
Y coordinates of input data

DATA_IN (NDP) = REAL (Given)
Data values for each X,Y

N_PTS = INTEGER (Given)
Number of points in output grid

X_OUT (N_PTS) = REAL (Given)
X coordinates of output points

Y_OUT (N_PTS) = REAL (Given)
Y coordinates of output points

DATA_OUT (N_PTS) = REAL (Returned)
Output data values

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_SPLINE_PDA_IDSFFT
Fit a surface to an irregular grid using the PDA function PDA_IDSFFT

Description:

This routine provides a wrapper for the PDA_IDSFFT spline interpolation routine. The quirks of the algorithm are dealt with here so that SCULIB_SPLINE_REGRID does not have to know anything about the interpolation routine.

Invocation:

```
CALL SCULIB_SPLINE_PDA_IDSFFT (NDP, X_IN, Y_IN, DATA_IN, NX_OUT, NY_OUT, X_OUT,
Y_OUT, DATA_OUT, STATUS)
```

Arguments:

NDP = INTEGER (Given)
Number of data points on irregular grid

X_IN (NDP) = REAL (Given)
X coordinates of input data

Y_IN (NDP) = REAL (Given)
Y coordinates of input data

DATA_IN (NDP) = REAL (Given)
Data values for each X,Y

NX_OUT = INTEGER (Given)
Number of pixels in X direction

NY_OUT = INTEGER (Given)
Number of pixels in Y direction

X_OUT (NX_OUT) = REAL (Given)
X coordinates of output points

Y_OUT (NY_OUT) = REAL (Given)
Y coordinates of output points

DATA_OUT (NX_OUT, NY_OUT) = REAL (Returned)
Output data values

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_SPLINE_PDA_SURFIT
Fit a surface to an irregular grid using the PDA function PDA_SURFIT

Description:

This routine provides a wrapper for the PDA_SURFIT spline interpolation routine. The quirks of the algorithm are dealt with here so that SCULIB_SPLINE_REGRID does not have to know anything about the interpolation routine.

Invocation:

```
CALL SCULIB_SPLINE_PDA_SURFIT (NDP, X_IN, Y_IN, DATA_IN, VARIANCE_IN, NX_OUT,
NY_OUT, X_OUT, Y_OUT, DATA_OUT, STATUS)
```

Arguments:

NDP = INTEGER (Given)
Number of data points on irregular grid

S = REAL (Given)
Smoothness factor

X_IN (NDP) = REAL (Given)
X coordinates of input data

Y_IN (NDP) = REAL (Given)
Y coordinates of input data

DATA_IN (NDP) = REAL (Given)
Data values for each X,Y

VARIANCE_IN (NDP) = REAL (Given)
Variance for each data value

NX_OUT = INTEGER (Given)
Number of pixels in X direction

NY_OUT = INTEGER (Given)
Number of pixels in Y direction

X_OUT (NX_OUT) = REAL (Given)
X coordinates of output points

Y_OUT (NY_OUT) = REAL (Given)
Y coordinates of output points

DATA_OUT (NX_OUT, NY_OUT) = REAL (Returned)
Output data values

STATUS = INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_SPLINE_REGRID
Regrid supplied data onto a rectangular grid using a spline

Description:

This routine takes data with a variance array and x y positions and regrids it onto a rectangular grid using a spline interpolation algorithm (from PDA). Each integration is regridded separately and added into a mean output image.

Invocation:

```
CALL SCULIB_SPLINE_REGRID( METHOD, SFACTOR, MAX_MAPS, N_MAPS, N_BOLS, N_INTS,
DIAMETER, WAVELENGTH, PXSIZE, NX_OUT, NY_OUT, I_CENTRE, J_CENTRE, WEIGHT, INT_PTR,
DATA_PTR, VAR_PTR, XPOS_PTR,YPOS_PTR, OUT_DATA, OUT_VARIANCE, OUT_QUALITY, OUT_WEIGHT,
STATUS )
```

Arguments:

METHOD = CHARACTER (Given)
Spline method to use.

SFACTOR = REAL (Given)
Smoothing factor to be used for PDA_SURFIT

MAX_MAPS = INTEGER (Given)
Max allowed number of input maps (eg FILES in rebin)

N_MAPS = INTEGER (Given)
Number of input maps

N_BOLS(N_MAPS) = INTEGER (Given)
Number of bolometers in each map

MAX_INTS = INTEGER (Given)
Maximum number of separate integrations

N_INTS(N_MAPS) = INTEGER (Given)
Number of integrations in each input map

DIAMETER = REAL (Given)
Diameter of dish in metres

WAVELENGTH = REAL (Given)
Wavelength of observation

PXSIZE = REAL (Given)
Pixel size of output map (radians)

NX_OUT = INTEGER (Given)
Number of X pixels in output map

NY_OUT = INTEGER (Given)
Number of Y pixels in output map

I_CENTRE = INTEGER (Given)
Reference pixel in X

J_CENTRE = INTEGER (Given)
Reference pixel in Y

WEIGHT(N_MAPS) = REAL (Given)
Relative weight of each input map

INT_LIST(MAX_MAPS, MAX_INTS + 1) = INTEGER (Given)
Position of each integration in data (cf DEM_PNTR for integrations)

DATA_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to input data

VAR_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to input data variance

XPOS_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to input data X positions

YPOS_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to input data Y positions

OUT_DATA(NX_OUT, NY_OUT) = REAL (Returned)
Regular gridded data

OUT_VARIANCE(NX_OUT, NY_OUT) = REAL (Returned)
Variance of output data

OUT_DATA(NX_OUT, NY_OUT) = BYTE (Returned)
Quality of output map

OUT_WEIGHT(NX_OUT, NY_OUT) = REAL (Returned)
Total weight of each pixel

STATUS = _INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_SPLINE_REGRID_1
Calculate the areas of the output map that contain good data points

Description:

Calculate the areas of the output map that will be affected by points in the input data array. This is used to generate a mask so that the rebinned output map only contains data points close to input data. Similar to SCULIB_WTFN_REGRID_1.

Invocation:

```
CALL SCULIB_SPLINE_REGRID_1 (DIAMETER, WAVELENGTH, IN_DATA, X, Y, NPIX, PIXSPACE,
NI, NJ, ICEN, JCEN, QUALITY, STATUS)
```

Arguments:

DIAMETER = REAL (Given)
Diameter of telescope

WAVELENGTH = REAL (Given)
Wavelength of the observation

IN_DATA (NPIX) = REAL (Given)
The input data values. (Used to define quality)

X(NPIX) = REAL (Given)
The x coordinates of the input pixels

Y(NPIX) = REAL (Given)
The y coordinates of the input pixels

NPIX = INTEGER (Given)
the number of input pixels

PIXSPACE = DOUBLE PRECISION (Given)
the pixel spacing of the output pixels

NI = INTEGER (Given)
The number of output pixels in the x direction

NJ = INTEGER (Given)
The number of output pixels in the y direction

ICEN = INTEGER (Given)
the x index of the centre of the output array

JCEN = INTEGER (Given)
the y index of the centre of the output array

SCRATCH (NI, NJ) = UBYTE (Returned)
Quality mask

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_SPLIT_FILE_SPEC
Splits a string into a filename and SCUBA section

Description:

This routine takes a string and returns the filename (ie text before a {) and the data specifications. Each specification is indicated by a {}. More than one spec can be returned. If the string contains a '!' then we will use the invers of the specified section.

Invocation:

```
CALL SCULIB_SPLIT_FILE_SPEC( NAME, MAX_SPEC, FILENAME, NSPEC, DATA_SPEC, STATUS)
```

Arguments:

NAME = CHAR (Given)
Input string

MAX_SPEC = INTEGER (Given)
Max number of specifications allowed

FILENAME = CHAR (Returned)
The name of the file (text before first spec)

NSPEC = INTEGER (Returned)
Number of specs found/returned (can be 0)

DATA_SPEC(NSPEC) = CHAR (Returned)
The actual specification for the SCUBA section

USE_SECTION = LOGICAL (Returned)
Set to true by default and false if a '-' is found at the end of the section (ie after last closing brace) – a '-' indicates that we are using the inverse of the section.

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

SCULIB_SQROOTR
take the square root of a real array

Description:

Puts square root of real input array into output array. If FLAGGED and QUALITY are both false then negative input numbers give a zero output. If FLAGGED is true then input bad values, or negative input numbers, give a bad output. If QUALITY is true then input values with bad quality, or negative input numbers, lead to zero output data and bad output quality.

Invocation:

```
CALL SCULIB_SQROOTR (N, IN_DATA, IN_QUAL, ROOT, ROOT_QUAL, QUALITY, FLAGGED)
```

Arguments:

N = INTEGER (Given)
the number of array elements

IN_DATA (N) = REAL (Given)
the array whose root is to be taken

IN_QUAL (N) = INTEGER (Given)
the quality on the input data

ROOT (N) = REAL (Returned)
the array to hold the square roots (may be the same as IN_DATA)

ROOT_QUAL (N) = INTEGER (Returned)
the quality on the output root

QUALITY = LOGICAL (Given)
.TRUE. if input quality array exists

FLAGGED = LOGICAL (Given)
.TRUE. if input data has flagged values

Notes:

Consider using VEC_SQRTR (SUN/39) instead.

Copyright :

Implementation Status:
Uses INTEGER rather than BYTE Quality.

SCULIB_STANDARD_APPARENT
convert apparent RA,Decs from one date to another

Description:

This routine converts a list of apparent RA,Decs for one date into apparent coordinates at another. To do this it calls SLA_MAPPA and SLA_AMPQK to convert the input coordinates into mean coordinates for a J2000.0 equinox, then SLA_MAPPA and SLA_MAPQKZ to convert the mean coordinates into apparent on the output date.

Invocation:

```
CALL SCULIB_STANDARD_APPARENT (N, RA_APP, DEC_APP, IN_MJD, OUT_MJD, STATUS)
```

Arguments:

N = INTEGER (Given)
number of positions to be converted

RA_APP (N) = DOUBLE PRECISION (Given and returned)
apparent RA (radians)

DEC_APP (N) = DOUBLE PRECISION (Given and returned)
apparent Dec (radians)

IN_MJD = DOUBLE PRECISION (Given)
date of input coordinates (modified Julian day)

OUT_MJD = DOUBLE PRECISION (Given)
date of output coordinates (modified Julian day)

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_STATD
To calculate mean and standard deviation of a DOUBLE array

Description:

This routine calculates the mean and standard deviation of a double array. If a positive value of CLIP is given, the data set will be clipped at CLIP sigma repeatedly until all points are within CLIP sigma. Mean and standard deviation are those values derived after iterative clipping.

Invocation:

```
CALL SCULIB_STATD(N_POS, SCUDATA, SCUQUAL, BADBIT, NGOOD, MEAN, MEDIAN, SUM, SUMSQ,
STDEV, QSORT, STATUS)
```

Arguments:

N_POS = _INTEGER (Given)
Number of jiggle positions in data set

CLIP = _REAL (Given)
Number of sigma to clip iteratively

SCUDATA(N_POS) = _DOUBLE (Given)
The data

SCUQUAL(N_POS) = _BYTE (Given)
The data quality

BADBIT = _BYTE (Given)
Bad bit mask

NGOOD = _INTEGER (Returned)
Number of good pixels in mean

MEAN = _DOUBLE (Returned)
Mean value

MEDIAN = _DOUBLE (Returned)
Median of good data

SUM = _DOUBLE (Returned)
Data sum

SUMSQ = _DOUBLE (Returned)
Sum of squares

STDEV = _DOUBLE (Returned)
Standard deviation

QSORT = _DOUBLE (Returned)
Sorted data set

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

Implementation Status:

- Deals with bad pixels
- Uses PDA sort routine

SCULIB_STATR
To calculate mean and standard deviation of a REAL array

Description:

This routine calculates the mean and standard deviation of a real array. If a positive value of CLIP is given, the data set will be clipped at CLIP sigma repeatedly until all points are within CLIP sigma. Mean and standard deviation are those values derived after iterative clipping.

Invocation:

```
CALL SCULIB_STATR(N_POS, SCUDATA, SCUQUAL, BADBIT, NGOOD, MEAN, MEDIAN, SUM, SUMSQ,
STDEV, QSORT, STATUS)
```

Arguments:

N_POS = _INTEGER (Given)
Number of jiggle positions in data set

CLIP = _REAL (Given)
Number of sigma to clip iteratively

SCUDATA(N_POS) = _REAL (Given)
The data

SCUQUAL(N_POS) = _BYTE (Given)
The data quality

BADBIT = _BYTE (Given)
Bad bit mask

NGOOD = _INTEGER (Returned)
Number of good pixels in mean

MEAN = _DOUBLE (Returned)
Mean value

MEDIAN = _DOUBLE (Returned)
Median of good data

SUM = _DOUBLE (Returned)
Data sum

SUMSQ = _DOUBLE (Returned)
Sum of squares

STDEV = _DOUBLE (Returned)
Standard deviation

QSORT = _REAL (Returned)
Sorted data set

STATUS = INTEGER (Given and Returned)
Global Status value

Copyright :

Implementation Status:

- Deals with bad pixels
- Uses PDA sort routine

SCULIB_SUB_TAUZ
get tauz appropriate for each sub-instrument

Description:

This routine finds the zenith sky optical depth appropriate for the filter in front of each sub-instrument being used. If it cannot find an optical depth for the required filter the routine will output a warning message and return an optical depth of 0.0.

The times and dates at which the optical depths were measured are also returned. If the optical depth for the required filter cannot be found then the date will be that at which the routine is run.

Invocation:

```
CALL SCULIB_GET_TAUZ (N_SUBS, SUB_FILTER, N_SKY, SKY_FILTER, SKY_TAUZ, SKY_DAY,
SUB_TAUZ, SUB_DAY, STATUS)
```

Arguments:

N_SUBS = INTEGER (Given)
number of sub-instruments

SUB_FILTER (N_SUBS) = CHARACTER∗(∗) (Given)
names of filters in front of sub-instruments

N_SKY = INTEGER (Given)
number of filters for which data is available

SKY_FILTER (N_SKY) = CHARACTER∗(∗) (Given)
filters for which data is available

SKY_TAUZ (N_SKY) = REAL (Given)
tauz for each possible filter

SKY_DAY (N_SKY) = DOUBLE PRECISION (Given)
the date and time of the tauz measurement specified as day number since 1st Jan.

SUB_TAUZ (N_SUBS) = REAL (Returned)
tauz appropriate to sub-instruments

SUB_DAY (N_SUBS) = DOUBLE PRECISION (Returned)
the date and time of the tauz measurement

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_SUBARE
subtract one real array from another into a third

Description:

Subtracts two real arrays. Note that any of the arrays may be the same.

Invocation:

```
CALL SCULIB_SUBARE (N, ARRAY1, ARRAY2, ARRAY3, Q1DATA, Q2DATA, Q3DATA, V1DATA,
V2DATA, V3DATA, QUALITY, FLAGGED, VARIANCE)
```

Arguments:

N = INTEGER (Given)
Number of elements in each array

ARRAY1 (N) = REAL (Given)
Input array

ARRAY2 (N) = REAL (Given)
Second input array

ARRAY3 (N) = REAL (Returned)
Result array. ARRAY3 = ARRAY1 - ARRAY2

Q1DATA (N) = INTEGER (Given)
Quality array for first input array

Q2DATA (N) = INTEGER (Given)
Quality array for second input array

Q3DATA (N) = INTEGER (Returned)
Quality array for output array

V1DATA (N) = REAL (Given)
Variance array for first input array

V2DATA (N) = REAL (Given)
Variance array for second input array

V3DATA (N) = REAL (Returned)
Variance array for output array

QUALITY = LOGICAL (Given)
True if input has quality information

FLAGGED = LOGICAL (Given)
True if input has flagged data values

VARIANCE = LOGICAL (Given)
True if both input arrays have variance arrays

Notes:

Consider using VEC_SUBR (SUN/39) instead

SCULIB_SUMAD
sum the elements of a double precision array

Description:

This routine adds up the NELM elements of the double precision ARRAY and puts the result in SUM.

Invocation:

```
CALL SCULIB_SUMAD (NELM, ARRAY, SUM)
```

Arguments:

NELM = INTEGER (Given)
the number of elements in the array

ARRAY (NELM) = DOUBLE PRECISION (Given)
the array whose elements are to be summed

SUM = DOUBLE PRECISION (Returned)
the sum

Copyright :

Implementation Status:

No status checking

SCULIB_TIDY_LINE
remove tabs and comments from a line

Description:

This routine tidies up a character string and returns its length ignoring trailing blanks. If the returned length is 0 the returned string will be ' ', otherwise it will be equal to the input string truncated at the used length.

The tidying involves:-

- calling CHR_CLEAN to remove non-printable characters.
- replacing all HT (tab) characters by spaces.
- truncating the string at the character before a COMCHAR character, if present (the characters after COMCHAR are assumed to be comments).

Invocation:

```
CALL SCULIB_TIDY_LINE (COMCHAR, LINE, LENGTH)
```

Arguments:

COMCHAR = CHARACTER∗1 (Given)
the character at the beginning of a comment

LINE = CHARACTER∗(∗) (Given and returned)
the line to be tidied

LENGTH = INTEGER (Returned)
the length of the string, ignoring trailing blanks

Copyright :

SCULIB_UNPACK
routine to unpack compressed resampled map

Description:

This routine unpacks a compressed SCUBA resampled data array into a 2-d image. It does this by cycling up the j slices of the image and -

if the packed data has pixels for that slice

- fill the image slice data (data sum, variance sum and weight sum) with zeroes and bad quality before the packed slice starts.
- copy the data and quality of the packed slice into the appropriate part of the image slice
- fill the image slice with zeroes and bad quality after the packed slice ends.

else

- fill the image slice with zeroes and bad quality

endif

Invocation:

```
CALL SCULIB_UNPACK (NBS_DIM1, NBS_DIM2, NBS_DIM3, RESNBS, ISTART, NPIX, POINTER,
SUB, RES_DIM1, RES_DIM2, RES_DS, RES_VS, RES_WS, RES_QS)
```

Arguments:

NBS_DIM1 = INTEGER (Given)
first dimension of the RESNBS array

NBS_DIM2 = INTEGER (Given)
second dimension of the RESNBS array

NBS_DIM3 = INTEGER (Given)
third dimension of the RESNBS array

RESNBS (NBS_DIM1, NBS_DIM2, NBS_DIM3)) = REAL (Given)
the noticeboard 'resampled data'

ISTART (RES_DIM2) = INTEGER (Given)
the I index of the start of each packed slice

NPIX (RES_DIM2) = INTEGER (Given)
the number of pixels in each packed slice

POINTER (RES_DIM2) = INTEGER (Given)
the pointer to the start of each packed slice

SUB = INTEGER (Given)
the number of the sub-instrument map to be unpacked

RES_DIM1 = INTEGER (Given)
first dimension of resampled image array

RES_DIM2 = INTEGER (Given)
second dimension of resampled image array

RES_DS (RES_DIM1, RES_DIM2) = REAL (Returned)
the unpacked data sum

RES_VS (RES_DIM1, RES_DIM2) = REAL (Returned)
the unpacked variance sum

RES_WS (RES_DIM1, RES_DIM2) = REAL (Returned)
the unpacked weight sum

RES_QS (RES_DIM1, RES_DIM2) = INTEGER (Returned)
the unpacked quality

Copyright :

SCULIB_UNPACK_CHOPSCAN
routine to unpack compressed chop-scan data

Description:

Used by the on-line system to unpack notice-board entries and retrieve demodulated data.

Invocation:

```
CALL SCULIB_UNPACK_CHOPSCAN (RESNBS, N_POINTS, ISTART, NPIX, POINTER, RESDATA_450,
RESWT_450, RESDATA_850, RESWT_850, RESDIM1, RESDIM2, BAD)
```

Arguments:

RESNBS (4, N_POINTS) = REAL (Given)
the 'resampled data' noticeboard

N_POINTS = INTEGER (Given)
the 2nd dimension of the RESNBS array

ISTART (RESDIM2) = INTEGER (Given)
the I index of the start of the packed slice at index J

NPIX (RESDIM2) = INTEGER (Given)
the number of pixels in the packed slice for row J

POINTER (RESDIM2) = INTEGER (Given)
the pointer to the start of the slice in the packed array

RESDATA_450 (RESDIM1, RESDIM2) = REAL (Returned)
the unpacked 450 data

RESWT_450 (RESDIM1, RESDIM2) = REAL (Returned)
450 weights

RESDATA_850 (RESDIM1, RESDIM2) = REAL (Returned)
the unpacked 850 image

RESWT_850 (RESDIM1, RESDIM2) = REAL (Returned)
850 weights

RESDIM1 = INTEGER (Given)
first dimension of resampled image array

RESDIM2 = INTEGER (Given)
second dimension

BAD = REAL (Given)
value signalling bad pixel

Copyright :

SCULIB_UNPACK_JIGGLE
unpack demodulated data onto 2-d map

Description:

This routine unpacks the demodulated data from a switch onto a rectangular 2-d map.

If status is good on entry the data and variance of the output map will be initialised to 'bad' values, the quality to 1. If there are any data to unpack the routine will then attempt to do so. The method used depends on whether the switch covered part/all of the entire jiggle pattern or contains data for several repeats of the jiggle pattern.

In the first case, the routine will check that the indices of the section of jiggle pattern covered by this switch lie within the bounds of the full jiggle pattern. If not, an error will be reported and bad status returned. Otherwise, the datablock will be unpacked into the map as specified by the I_JIGGLE, J_JIGGLE arrays. Each map point just has its data, variance and quality copied from the demodulated data.

When the data covers several repeats of the jiggle pattern, the routine will check that the jiggle index of the first point in the datablock is 1 and that the number of data points is an integer multiple of the size of the jiggle. If not, an error will be reported and bad status returned. Otherwise, the datablock will be unpacked into the map as specified by I_JIGGLE, J_JIGGLE. The unpacking differs from the first case in that demodulated data with bad quality will be ignored. Points in the unpacked map will be set to the average of the values contributed by the separate jiggles, variances will be calculated from the dispersion of the values about the mean if there was more than 1, or set to the demodulated variance otherwise.

Invocation:

```
CALL SCULIB_UNPACK_JIGGLE (N_JIGGLES, N_BOLS, DEMOD, J_START, JIGGLE_COUNT, I_JIGGLE,
J_JIGGLE, IDIM, JDIM, MAP_DATA, MAP_VARIANCE, MAP_QUALITY, J_END, STATUS)
```

Arguments:

N_JIGGLES = INTEGER (Given)
number of jiggles in datablock

N_BOLS = INTEGER (Given)
number of bolometers measured

DEMOD (4, N_BOLS, N_JIGGLES) = REAL (Given)
demodulated data

J_START = INTEGER (Given)
index of first jiggle in datablock in the overall jiggle pattern

JIGGLE_COUNT = INTEGER (Given)
number of jiggles in jiggle pattern

I_JIGGLE (JIGGLE_COUNT) = INTEGER (Given)
i index in output map of each jiggle position in pattern

J_JIGGLE (JIGGLE_COUNT) = INTEGER (Given)
j index in output map of each jiggle position in pattern

IDIM = INTEGER (Given)
i dimension of output map

JDIM = INTEGER (Given)
j dimension of output map

MAP_DATA (IDIM, JDIM, N_BOLS) = REAL (Returned)
output map data

MAP_VARIANCE (IDIM, JDIM, N_BOLS) = REAL (Returned)
output map variance

MAP_QUALITY (IDIM, JDIM, N_BOLS) = INTEGER (Returned)
output map quality

J_END = INTEGER (Returned)
index of last jiggle in datablock in the overall jiggle pattern

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_UNPACK_JIGGLE_SEPARATES
unpack jiggle data arrays onto 2-d map

Description:

This routine unpacks data from a jiggle observation into a rectangular 2-d map.

If status is good on entry the data and variance of the output map will be initialised to 'bad' values, the quality to 1. If there are any data to unpack the routine will then attempt to do so. The method used depends on whether the switch covered part/all of the entire jiggle pattern or contains data for several repeats of the jiggle pattern.

In the first case, the routine will check that the indices of the section of jiggle pattern covered by this switch lie within the bounds of the full jiggle pattern. If not, an error will be reported and bad status returned. Otherwise, the data will be unpacked into the map as specified by the I_JIGGLE, J_JIGGLE arrays.

When the data cover several repeats of the jiggle pattern, the routine will check that the jiggle index of the first point in the datablock is 1 and that the number of data points is an integer multiple of the size of the jiggle. If not, an error will be reported and bad status returned. Otherwise, the data will be unpacked into the map as specified by I_JIGGLE, J_JIGGLE. The unpacking differs from the first case in that points in the unpacked map will be set to the average of the good quality values contributed by the separate jiggles, variances will be calculated from the dispersion of the values about the mean if there was more than 1, or set to the input variance otherwise.

Invocation:

```
CALL SCULIB_UNPACK_JIGGLE_SEPARATES (N_JIGGLES, N_BOLS, J_DATA, J_VARIANCE, J_QUALITY,
J_START, JIGGLE_COUNT, I_JIGGLE, J_JIGGLE, IDIM, JDIM, MAP_DATA, MAP_VARIANCE,
MAP_QUALITY, MAP_NUMPTS, J_END, BADBIT, STATUS)
```

Arguments:

N_JIGGLES = INTEGER (Given)
number of jiggles in datablock

N_BOLS = INTEGER (Given)
number of bolometers measured

J_DATA (N_BOLS, N_JIGGLES) = REAL (Given)
data for each jiggle measured

J_VARIANCE (N_BOLS, N_JIGGLES) = REAL (Given)
variance on J_DATA

J_QUALITY (N_BOLS, N_JIGGLES) = BYTE (Given)
quality on J_DATA

J_START = INTEGER (Given)
index of first jiggle in datablock in the overall jiggle pattern

JIGGLE_COUNT = INTEGER (Given)
number of jiggles in jiggle pattern

I_JIGGLE (JIGGLE_COUNT) = INTEGER (Given)
i index in output map of each jiggle position in pattern

J_JIGGLE (JIGGLE_COUNT) = INTEGER (Given)
j index in output map of each jiggle position in pattern

IDIM = INTEGER (Given)
i dimension of output map

JDIM = INTEGER (Given)
j dimension of output map

MAP_DATA (IDIM, JDIM, N_BOLS) = REAL (Returned)
output map data

MAP_VARIANCE (IDIM, JDIM, N_BOLS) = REAL (Returned)
output map variance

MAP_NUMPTS (IDIM, JDIM, N_BOLS) = INTEGER (Returned)
output map quality

MAP_QUALITY (IDIM, JDIM, N_BOLS) = BYTE (Returned)
output map quality

J_END = INTEGER (Returned)
index of last jiggle in datablock in the overall jiggle pattern

BADBIT = BYTE (Given)
bad bit mask

STATUS = INTEGER (Given and returned)
global status

Copyright :

SCULIB_UT1
returns UT1 as a modified Julian day

Description:

Return the current modified Julian day calculated for Hawaiian local time.

Invocation:

```
UT1 = SCULIB_UT1
```

Arguments:

None

Returned Value:

SCULIB_UT1 = DOUBLE PRECISION

UT1 expressed as a modified Julian day (JD - 2400000.5)

Notes:

- Depends on system clock for local time and date
- Uses the IDATE function to retrieve the current date. This only returns a 2-digit year (not Y2K compliant) but is not a problem since the output is immediately passed to the SLALIB routine SLA_CALDJ which uses windowing to select the correct year.

Copyright :

Bugs:

The time zone is hard-wired for Hawaii (10).

SCULIB_WTFN_REGRID
Regrid supplied data onto a rectangular grid

Description:

This routine takes data with a variance array and x y positions and regrids it onto a rectangular grid using a weight function interpolation.

This is an image space implementation of fourier techniques. In Fourier terms the technique could be described as follows:-

- [1] Do a 2-d discrete Fourier transform of the data points. The result is a repeating pattern made up of copies of the transform of the map as a continuous function, each copied displaced from its neighbours by 1/dx, where dx is the sample spacing of the input points (assumed equal in x and y). Different copies of the 'continuous map' transforms will overlap ('alias') if there is any power in the map at frequencies greater than 1/(2dx). It is not possible to unravel aliased spectra and this constraint leads to the Nyquist sampling criterion.
- [2] We want to derive the 'continuous map' so that map values on the new grid mesh can be derived. Do this by multiplying the transform of the data by a function that has zero value beyond a radius of 0.5/dx from the origin. This will get rid of all the repeats in the pattern and leave just the transform of the 'continuous map'.
- [3] Do an inverse FT on the remaining transform for the points where you wish the resampled points to be (note: FFTs implicitly assume that the data being transformed DO repeat ad infinitum so we'd have to be careful when using them to do this).

The analogue of these process steps in image space is as follows:

- [1] Nothing.
- [2] Convolve the input data with the FT of the function used to isolate the 'continuous map' transform.
- [3] Nothing.

If the method is done properly, the rebinned map is in fact the map on the new sample mesh that has the same FT as the continuous function going through the original sample points.

Convolution Functions:-

- [Bessel:] For good data and with no time constraint on reduction the best convolution function would be one whose FT is a flat-topped cylinder in frequency space, centred on the origin and of a radius such that frequencies to which the telescope is not sensitive are set to zero. Unfortunately, this function is a Bessel function, which extends to infinity along both axes and has significant power out to a large radius. To work correctly this would require an infinite map and infinite computing time. However, a truncated Bessel function should work well on a large map, except near the edges. Edge effects can be removed by pretending that the map extends further - of course, this only works if you know what data the pretend map area should

contain, i.e. zeros. Another problem with a Bessel function arises from the fact that it does truncate the FT of the map sharply. If the data are good then there should be nothing but noise power at the truncation radius and the truncation of the FT should have no serious effect. However, if the data has spikes (power at all frequencies in the FT) or suffers from seeing effects such that the data as measured DO have power beyond the truncation radius, then this will cause ringing in the rebinned map.

- [Gaussian:] In fact, any function that is finite in frequency space will have infinite extent in image space (I think). As such they will all drag in some power from the aliased versions of the map transform and all suffer from edge effects and large compute times. Some are worse than others, however. For example, a Gaussian can have most of its power concentrated over a much smaller footprint than a Bessel function, so the convolution calculation will be much faster. It is also more robust in the presence of spikes and seeing problems because it does not truncate the map FT as sharply as the Bessel function - such effects give rise to smoother defects in the rebinned map though the defects will still BE there.

Invocation:

```
CALL SCULIB_WTFN_REGRID(USEGRD, N_MAPS, N_PTS, WTFNRAD, WTFNRES, WEIGHTSIZE, SCALE,
DIAMETER, WAVELENGTH, PXSIZE, NX_OUT, NY_OUT, I_CENTRE, J_CENTRE, WTFN, WEIGHT,
BOLWT, N_BOL, MAX_BOLS, DATA_PTR, VAR_PTR, XPOS_PTR,YPOS_PTR, OUT_DATA, OUT_VARIANCE,
OUT_QUALITY, CONV_WEIGHT, STATUS)
```

Arguments:

USEGRD = LOGICAL (Given)
Use guard ring for final stage of rebin

N_MAPS = INTEGER (Given)
Number of data files read in

N_PTS (N_MAPS) = INTEGER (Given)
Number of points in each input dataset

WTFNRAD = INTEGER (Given)
Radius of largest weight function in scale lengths

WTFNRES = INTEGER (Given)
Number of values per scale length in the supplied weight function

WEIGHTSIZE = INTEGER (Given)
Radius of supplied weighting function

SCALE = REAL (Given)
Size of a scale length in the same units as PXSIZE

DIAMETER = REAL (Given)
Diameter of telescope in metres

WAVELENGTH = REAL (Given)
Wavelength of map in microns

PXSIZE = REAL (Given)
Pixel size in radians

NX_OUT = INTEGER (Given)
Number of pixels in X direction of output map

NY_OUT = INTEGER (Given)
Number of pixels in Y direction of output map

I_CENTRE = INTEGER (Given)
X reference pixel

J_CENTRE = INTEGER (Given)
Y reference pixel

WTFN() = REAL (Given)
The weighting function

WEIGHT(N_MAPS) = REAL (Given)
The weight of each input dataset

BOLWT (MAX_BOLS, N_MAPS) = REAL (Given)
Relative weight for each bolometer

N_BOL (N_MAPS) = INTEGER (Given)
Number of bolometers from each map

MAX_BOLS = INTEGER (Given)
Max number of bolometers allowable in BOLWT

DATA_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to REAL input data

VAR_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to REAL input variance

XPOS_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to X bolometer positions (DOUBLE)

YPOS_PTR(N_MAPS) = INTEGER (Given)
Array of pointers to Y bolometer positions (DOUBLE)

OUT_DATA (NX_OUT, NY_OUT) = REAL (Returned)
Output map

OUT_VARIANCE (NX_OUT, NY_OUT) = REAL (Returned)
Output variance

OUT_QUALITY (NX_OUT, NY_OUT) = BYTE (Returned)
Output quality

CONV_WEIGHT (NX_OUT, NY_OUT) = REAL (Returned)
Contribution to each pixel

STATUS = _INTEGER (Given & Returned)
Global status

Copyright :

SCULIB_WTFN_REGRID_1
Calculate the coverage of the data and total weight per cell

Description:

This routine performs two tasks:

- [1] Determine which output pixels contain data (effectively detecting the edge of the useful area of the output image.
- [2] For each output pixel containing data the weight of the map is added to the total weight. This TOTAL_WEIGHT is then used to modify the behaviour when two images overlap (the TOTAL_WEIGHT will be 2.0 (say) in the overlap region and 1 elsewhere.

Invocation:

```
CALL SCULIB_WTFN_REGRID_1 (DIAMETER, WAVELENGTH, WEIGHT, IN_DATA, X, Y, NPIX,
PIXSPACE, NI, NJ, ICEN, JCEN, AV_WEIGHT, SCRATCH, STATUS)
```

Arguments:

DIAMETER = REAL (Given)
Diameter of telescope

WAVELENGTH = REAL (Given)
Wavelength of the observation

WEIGHT = REAL (Given)
The weight of the input dataset.

IN_DATA (NPIX) = REAL (Given)
The input data values. (Used to define quality)

X(NPIX) = DOUBLE PRECISION (Given)
The x coordinates of the input pixels

Y(NPIX) = DOUBLE PRECISION (Given)
The y coordinates of the input pixels

NPIX = INTEGER (Given)
the number of input pixels

PIXSPACE = DOUBLE PRECISION (Given)
the pixel spacing of the output pixels

NI = INTEGER (Given)
The number of output pixels in the x direction

NJ = INTEGER (Given)
The number of output pixels in the y direction

ICEN = INTEGER (Given)
the x index of the centre of the output array

JCEN = INTEGER (Given)
the y index of the centre of the output array

AV_WEIGHT (NI, NJ) = REAL (Given and Returned)
Given as a workspace array

SCRATCH (NI, NJ) = INTEGER (Returned)
Temporary workspace.

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_WTFN_REGRID_2
Perform convolution

Description:

This routine convolves the input data with a weighting function onto a regularly spaced output grid. The weighting function is stored in a lookup table indexed by the square of the distance and is parametrized by RES, WEIGHTSIZE, SCLSZ and WTFN.

This is an image space implementation of fourier techniques. In Fourier terms the technique could be described as follows:-

- [1] Do a 2-d discrete Fourier transform of the data points. The result is a repeating pattern made up of copies of the transform of the map as a continuous function, each copied displaced from its neighbours by 1/dx, where dx is the sample spacing of the input points (assumed equal in x and y). Different copies of the 'continuous map' transforms will overlap ('alias') if there is any power in the map at frequencies greater than 1/(2dx). It is not possible to unravel aliased spectra and this constraint leads to the Nyquist sampling criterion.
- [2] We want to derive the 'continuous map' so that map values on the new grid mesh can be derived. Do this by multiplying the transform of the data by a function that has zero value beyond a radius of 0.5/dx from the origin. This will get rid of all the repeats in the pattern and leave just the transform of the 'continuous map'.
- [3] Do an inverse FT on the remaining transform for the points where you wish the resampled points to be (note: FFTs implicitly assume that the data being transformed DO repeat ad infinitum so we'd have to be careful when using them to do this).

The analogue of these process steps in image space is as follows:

- [1] Nothing.
- [2] Convolve the input data with the FT of the function used to isolate the 'continuous map' transform.
- [3] Nothing.

If the method is done properly, the rebinned map is in fact the map on the new sample mesh that has the same FT as the continuous function going through the original sample points.

Convolution Functions:-

- [Bessel] For good data and with no time constraint on reduction the best convolution function would be one whose FT is a flat-topped cylinder in frequency space, centred on the origin and of a radius such that frequencies to which the telescope is not sensitive are set to zero. Unfortunately, this function is a Bessel function, which extends to infinity along both axes and has significant power out to a large radius. To work correctly this would require an infinite map and infinite computing time. However, a truncated Bessel function should work well on a large map, except near the edges. Edge effects can be removed by pretending that the map extends further

- of course, this only works if you know what data the pretend map area should contain, i.e. zeros. Another problem with a Bessel function arises from the fact that it does truncate the FT of the map sharply. If the data are good then there should be nothing but noise power at the truncation radius and the truncation of the FT should have no serious effect. However, if the data has spikes (power at all frequencies in the FT) or suffers from seeing effects such that the data as measured DO have power beyond the truncation radius, then this will cause ringing in the rebinned map.

- [Gaussian] In fact, any function that is finite in frequency space will have infinite extent in image space (I think). As such they will all drag in some power from the aliased versions of the map transform and all suffer from edge effects and large compute times. Some are worse than others, however. For example, a Gaussian can have most of its power concentrated over a much smaller footprint than a Bessel function, so the convolution calculation will be much faster. It is also more robust in the presence of spikes and seeing problems because it does not truncate the map FT as sharply as the Bessel function - such effects give rise to smoother defects in the rebinned map though the defects will still BE there.

Invocation:

```
CALL SCULIB_WTFN_REGRID_2 (RES, IN_DATA, IN_VARIANCE, WEIGHT, USEVARWT, VARWT,
X, Y, NPIX, PIXSPACE, NI, NJ, ICEN, JCEN, TOTAL_WEIGHT, WAVELENGTH, CONV_DATA_SUM,
CONV_VARIANCE_SUM, CONV_WEIGHT, WEIGHTSIZE, SCLSZ, WTFN, STATUS)
```

Arguments:

RES = INTEGER (Given)
number of resolution elements per scale length

IN_DATA (NPIX) = REAL (Given)
The input data values.

IN_VARIANCE (NPIX) = REAL (Given)
Variance on IN_DATA.

WEIGHT = REAL (Given)
The weight of this dataset.

USEVARWT = LOGICAL (Given)
Are we using a different weight for each point

VARWT (NPIX) = REAL (Given)
The different weight for each pixel

X (NPIX) = DOUBLE PRECISION (Given)
The x coordinates of the input pixels.

Y (NPIX) = DOUBLE PRECISION (Given)
The y coordinates of the input pixels.

NPIX = INTEGER (Given)
the number of input pixels.

PIXSPACE = REAL (Given)
the pixel spacing of the output map.

NI = INTEGER (Given)
The number of output pixels in the x direction.

NJ = INTEGER (Given)
The number of output pixels in the y direction.

ICEN = INTEGER (Given)
the x index of the centre of the output array.

JCEN = INTEGER (Given)
the y index of the centre of the output array.

TOTAL_WEIGHT (NI,NJ) = REAL (Given)
the 'total weight' of each output pixel.

WAVELENGTH = REAL (Given)
the wavelength at which the maps were made (microns).

CONV_DATA_SUM (NI,NJ) = REAL (Given and returned)
the convolution sum for each output pixel.

CONV_VARIANCE_SUM (NI,NJ) = REAL (Given and returned)
the variance convolution sum for each output pixel.

CONV_WEIGHT (NI,NJ) = REAL (Given and returned)
the convolution weight for each output pixel.

WEIGHTSIZE = INTEGER (Given)
radius of weight function in scale units (SCUIP__FILTRAD for BESSEL, 1 for LINEAR)

SCLSZ = REAL
1 scale length in the same units as PIXSPACE

WTFN (RES * RES * WEIGHTSIZE * WEIGHTSIZE) = REAL (Given)
convolution weighting function

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :

SCULIB_WTFN_REGRID_3
Sets up the 'guard ring' of bolometers outside the data

Description:

Takes the regridded image and adds in data of value 'zero' for any points that are within range of the weighting function (ie if TOTAL_WEIGHT is greater than 0) but that do not already contain any data (ie TOTAL_WEIGHT is set to something becuase regrid_1 determined that the pixel was close enough to be influenced by the convolution function, BUT no data was actually found to lie in that pixel). This has the effect of enforcing zero flux density at the edges of the map or where no data was present (eg surrounding a bad pixel). This is only really necessary for the bessel function regrid since that function is extremely sensitive to edge effects. Once the 'guard ring' has been processed, the output image is normalised by dividing by the convolution weight (ie the actual weight used for each pixel). Any point that has zero convolution weight is set to bad.

Invocation:

```
CALL SCULIB_WTFN_REGRID_3 (METHOD,RES,PIXSPACE, NI, NJ, ICEN, JCEN, TOT_WEIGHT_IN,
WAVELENGTH, CONV_DATA_SUM, CONV_VARIANCE_SUM, CONV_QUALITY_SUM, CONV_WEIGHT, WEIGHTSIZE,
SCLSZ, WTFN, STATUS)
```

Arguments:

USEGUARD = LOGICAL (Given)
Logical to determine whether the gueard ring should be used.

RES = INTEGER (Given)
number of resolution elements per scale length

PIXSPACE = REAL (Given)
the pixel spacing of the output pixels (radians).

NI = INTEGER (Given)
The number of output pixels in the x direction.

NJ = INTEGER (Given)
The number of output pixels in the y direction.

ICEN = INTEGER (Given)
the x index of the centre of the output array.

JCEN = INTEGER (Given)
the y index of the centre of the output array.

TOT_WEIGHT_IN (NI, NJ) = REAL (Given)
the 'total weight' of each output pixel.

WAVELENGTH = REAL (Given)
the wavelength at which the maps was made (microns).

CONV_DATA_SUM (NI,NJ) = REAL (Given and returned)
the convolution sum for each output pixel.

CONV_VARIANCE_SUM (NI,NJ) = REAL (Given and returned)
the convolved variance sum for each output pixel.

CONV_QUALITY_SUM (NI,NJ) = BYTE (Given)
the quality on the convolution sum for each output pixel.

CONV_WEIGHT (NI,NJ) = REAL (Given and returned)
the convolution weight for each output pixel.

WEIGHTSIZE = INTEGER (Given)
radius of weight function in scale units (SCUIP__FILTRAD for BESSEL, 1 for LINEAR)

SCLSZ = REAL (Given)
1 scale length in the same units as pixspace

WTFN (RES ∗ RES ∗ WEIGHTSIZE ∗ WEIGHTSIZE) = REAL (Given)
convolution weighting function

STATUS = INTEGER (Given and Returned)
The global status.

Copyright :