SMA Technical Memo #162

Subject: **A TOOL OF SEPARATING SPECTRAL WINDOWS IN MIRIAD -**
Pre-processing SMA data taken from the Hybrid SWARM+ASIC correlator for further calibrations and imaging
Date: September 4, 2015[1]
From: Jun-Hui Zhao

[1]Updated from the versions issued on July 15, 16, 20, 23, 24, August 03, 15, 28, 2015

- Abstract -

As required by SMA Miriad users, we have implemented a Miriad task *swarmsplt*. This task is useful for Miriad users to separate SMA visibility data in various spectral chunks generated via different IFs from either ASIC or SWARM correlator components. Given two independent backends for the ASIC and SWARM data streamers and opposite sign of the channel increments for the two spectral chunks in a SWARM quadrant given a sideband, the separation is necessary in calibrations of the SMA hybrid correlator data using the existing routines in Miriad. In addition, *swarmsplt* provides a function to average the spectral resolution. Using the new SMA-Miriad pipeline task *smalod* along with *swarmsplt*, a pre-processing C-shell script for the new SMA data generated from the hybrid SWARM+ASIC correlator has been implemented. The products of the pre-processing data are compatible with the latest version of *SMA-Miriad 1.5.1*[2] that has been globally distributed. The pre-processing has provided a quick and useful way for Miriad users inside or outside CfA in handling the interferometer data produced from the SWARM correlator during SMA backend upgrading. RTDC can use the scripts to distribute SMA data upon request from Miriad users outside CfA.

The users'interface program in Fortran, *swarmsplt.for*, is given to demonstrate how to make SMA SWARM data be adapted generally to the tasks in Miriad. The new Miriad task has been implemented in *Miriad SWARM3.1.0* or newer versions.

[2]The version SMA-Miriad 1.5.* was configured and developed for the SMA ASIC correlator with a total bandwidth (BW) of 4 GHz. The 4 GHz mode of the legacy ASIC correlator is also referred as *doubleband*, versus the early version of ASIC correlator with the first IF component in BW of 2 GHz. This software package has been well tested and is stabilized for global distribution.

- 1. Why *swarmsplt* -

There are good reasons to separate between the SWARM and ASIC correlator data prior to calibrating and imaging in Miriad. The main concerns for handling SWARM data with general Miriad tasks are summarized in the following aspects:

1) The SWARM and ASIC data streamers are produced from independent electronic and digital backend systems; the error sources are different. In particular, one needs to pay attention to the variations of the complex gains due to instrumental issues during an observation track although the results of the preliminary analysis show that the phase drifting of the SWRAM and ASIC data appeared to track each other in an observing run of 8 hours. (see Thechincal Highlights in SMA Newsletter Number 20[3], July 13, 2015)

2) Spectral resolutions across the data chunks produced from SMA hybrid correlator are not uniform. In particular, the channel resolution of the SWARM chunks is about an order of magnitude finer than those of ASIC chunks. Each of SWARM chunks contains 16384 channels including those low amplitude guard-band channels near the edges of the spectra[4]. Extra programming efforts in the specific treatment for low S/N and the implementation for large memory buffers in the Miriad code are needed in further calibrating the SWARM data in high spectral resolution.

3) In Miriad convention, the signs of spectral channel increments from various spectral windows (chunks) are assumed to be the same given a sideband. However, the signs of spectral channel increment are opposite for the two chunks in a SWARM quadrant given a sideband. The new SWARM data structure is confusing. In particular, for the calibration and continuum imaging routines, averaging is inevitable. The resultant frequency from averaging without separating the two SWARM chunks leads to an incorrect reference frequency, which could become problems in further reduction of the SWARM data, and

could cause a failure in imaging.

4) This task can be used to pre-process the new SMA data produced from the hybrid SWARM+ASIC correlator. It is necessary for users to separate the data between ASIC and SWARM chunks in order to use the existing Miriad routines and example scripts for the reduction of the SMA SWARM data in Miriad. By separating the ASIC and SWARM chunks, and reducing a factor of two in the spectral resolution for the SWARM data, *swarmsplt* can furnish users a set of pre-processed ASIC and SWARM data that is compatible with Miriad software in the previous distributions.

Finally, Ch0 contains only 1 channel of pseudo continuum visibility data that is designed for the SMA scheduler to make a quick image with Miriad in the data quality assurance routine. This channel window is recommended to be eliminated prior to further data processing in Miriad in order to avoid problems related to the single-channel window.

[3] SMA Newsletter Number 20, July 13, 2015

[4] SMA Operations Logs #29874 and #29990

- 2. Help deck -

The Miriad help deck provides a description for the task *swarmsplt*, including the programmer who wrote the code and who is responsible to modification and further implementation. The description also gives the Keywords that are used in the task and can be supplied by users via the input deck or a Miriad command line in a shell environment.

```
Task:   swarmsplt
Responsible: Jun-Hui Zhao

         SWARMSPLT special for SMA data produced from the hybrid correlators
         ASIC and SWARM, splitting the uv dataset into either the ch0 or
         the 48 spectral chunks produced from the ASIC correlator, or the
         broadband spectral chunks produced from the SWARM correlator.

Keyword: vis
         The name of the input uv data sets. The input data must be in the
         original format generated with SMALOD in converting SMA data produced
         from the hybrid correlator SWARM and ASIC; i.e. the
         data with 51 spectral windows (ch0 + 48 ASIC chunks and 2 SWARM
         chunks). No default.

Keyword: out
         The name of the output uv data set. No default.

Keyword: options
         This gives extra processing options. Eight options given below can be
         used one at a time. No multiple options are allowed:
            ch0     Means taking the first channel (ch0) data for continuum.
            asic    Means all the 48 spectral chunk data produced from the
                    legacy ASIC correlator including two IF chunks.
            if1     Means the first IF in the sideband--the first 24 spectral
                    chunk data (S1,S2,...S24) produced from the legacy ASIC
                    correlator.
            if2     Means the second IF in the sideband--the rest 24 spectral
                    chunk data (S25,S26,...S48) produced from the legacy ASIC
                    correlator.
            swarm   Means two SWARM spectral chunks (S49, S50).
            sw1     the first  SWARM chunk (S49).
            sw2     the second SWARM chunk (S50).
            sw3     the third  SWARM chunk (S51)[5].
            sw4     the third  SWARM chunk (S52)[5].
            sw5     the third  SWARM chunk (S53)[5].
            sw6     the third  SWARM chunk (S54)[5].
            sw7     the third  SWARM chunk (S55)[5].
            sw8     the third  SWARM chunk (S56)[5].
                    Default is the ASIC N (48 or less) spectral chunks' data and
                    the two (eight eventually) SWARM chunks, excluding the ch0.
```

```
Keyword: line
         The normal uv linetype in the form:
                line,nchan,start,width,step
         The default is all channels.
         To reduce down spectral resolution for swarm data,
         one may set the keyword - line as:
             line = channel,8192,1,2
             for optiions = sw1 or sw2 or other SWARM chunks; or
             line = channel,16384,1,2
             for optiions = swarm.
```

[5] To be implemented.

- 3. Usage -

```
Examples are given for usage in the current version implemented for the first SWARM
quadrant or two SWARM chunks:

1) for all spectral chunks excluding ch0, the pseudo continuum
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.spc
  options  =

2) for all swarm chunk  data
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.swarm
  options  = swarm

3) for 1st swarm chunk (S49) data
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.sw1
  options  = sw1

4) for 2nd swarm chunk (S50) data
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.sw2
  options  = sw2

5) for all swarm chunk data, averaging every two channels
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.swarm
  options  = swarm
  line     = channel,16384,1,2

6) for 1st swarm chunk (S49) data, averaging every two channels
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.sw1
  options  = sw1
  line     = channel,8192,1,2

7) for 2nd swarm chunk (S50) data, averaging every two channels
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
```

```
  out      = 150519_rx0.usb.sw2
  options  = sw2
  line     = channel,8192,1,2

8) for all asic chunk data, doubleband data
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.asic
  options  = asic

9) for the 1st IF chunk data from the ASIC correlator
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.if1
  options  = if1

10) for the 2nd IF chunk data from the ASIC correlator
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.if2
  options  = if2

11) for ch0 data
swarmsplt% inp
  Task:   swarmsplt
  vis      = 150519_rx0.usb
  out      = 150519_rx0.usb.ch0
  options  = ch0
```

- 4. Pre-processing and compatibility -

A C-shell script swarmload.csh has been implemented to pre-process the new SMA data produced from the hybrid SWARM+ASIC correlator. The pre-processing with the C-shell script involves two Miriad tasks, smalod and swarmsplt. The former is used to load the new SMA data packaged in a new MIR/IDL format and write them in Miriad format. To date (August 3, 2015), the new SWARM correlator has been implemented for two chunks of spectral data; in the process of converting to Miriad format, the SMA spectral data are separated with swarmsplt into six subsets of the SWARM and ASIC data for the two sidebands. Each sideband consists of one ASIC subset and two SWARM subsets. The spectral resolution of the two SWARM subsets can be reduced by a factor of two. These subsets are fully compatible with the latest version of the SMA Miriad software package that has been globally distributed, SMA Miriad 1.5.1 (RH 6 or CentOS 6). Then users can carry out further calibrations and imaging to follow the example scripts that have been posted in a link accesible to public, SMA MIRIAD: Data reduction scripts.

The application appears to be quite useful to the SMA users outside CfA who are not accessible to the new Miriad-SWARM software before the new system of duplication and distribution to be built up for handling SWARM data in a full spectral resolution. A guideline to process the new SWARM and ASIC data using SMA Miriad 1.5.1 (RH 6 or CentOS 6) has been posted in both the SMA data reduction page and the SMA Miriad web site, MIRIAD-SWARM Software for Users Outside CfA. With a minimum assistance from RTDC or SMA staff, SMA users who are not able to acess to the CfA computer systems will be able to process the new hybrid SWARM+ASIC data in Miriad.

As more SWARM hardware components to be implemented, the pre-processing will be implemented for handling more subsets of SWARM data with the globally-distributed SMA Miriad package.

- 5. Source code -

The relevant Fortran code that is used for user's interface is given below along with the header file. This is a special program implemented for pre-processing the new SMA data generated from the SWARM+ASIC hybrid correlator. This new Miriad task is developed based on *smasplt*. The current version distributed in Miriad-SWARM 3.1.1 has been implemented for the first SWARM quadrant (two SWARM chunks). Eight options are support to separate the new SMA data generated from the different correlator and IF components.

Also, a function of linetype has been implemented; with averaging down the spectral resolution of the SWARM chunks, the pre-processed SWARM data are compatible with the SMA-Miriad package that was specially or originally configured and developed for reduction of the SMA ASIC correlator data with a BW of 4GHz, or *doubleband*.

5.1 *swarmsplt.for*

```
c************************************************************************
        program swarmsplt
        implicit none
c
c= swarmsplt - split uv dataset into either ch0 or ASIC or SWARM chunks.
c& jhz
c: uv handling
c+
c       SWARMSPLT is a special tool for handling SMA data produced from
c       the hybrid correlators ASIC and SWARM, splitting the uv dataset
c       into either the ch0 or the ASIC (48 or smaller) spectral chunks
c       or the two SWARM chunks produced from the SMA hybrid correlator
c       (ASIC + SWARM).
c@ vis
c       The name of the input uv data sets. The input data must be in the
c       original format generated with SMALOD in converting SMA data produced
c       from the hybrid correlator SWARM and ASIC; i.e. the
c       data with 51 spectral windows (ch0 + 48 ASIC chunks and 2 SWARM
c       chunks). No default.
c@ out
c       The name of the output uv data set. No default.
c@ options
c       This gives extra processing options. Eight options given below can be
c       used one at a time. No multiple options are allowed:
c          ch0     Means taking the first channel (ch0) data for continuum.
c          asic    Means all the 48 spectral chunk data produced from the
c                  legacy ASIC correlator including two IF chunks.
c          if1     Means the first IF in the sideband--the first 24 spectral
c                  chunk data (S1,S2,...S24) produced from the legacy ASIC
c                  correlator.
c          if2     Means the second IF in the sideband--the rest 24 spectral
c                  chunk data (S25,S26,...S48) produced from the legacy ASIC
c                  correlator.
c          swarm   Means two SWARM spectral chunks (S49,S50).
c          sw1     the first  SWARM chunk (S49).
c          sw2     the second SWARM chunk (S50).
c                  Default is the ASIC N (48 or less) spectral chunks data and
c                  the two SWARM chunks, excluding the ch0.
c@ line
c       The normal uv linetype in the form:
c              line,nchan,start,width,step
c       The default is all channels.
c       To reduce down spectral resolution for swarm data,
c       one may set the keyword - line as:
c           line = channel,8192,1,2
c           for optiions = sw1 or sw2; or
c           line = channel,16384,1,2
c           for optiions = swarm.
c
c--
c  History:
c    jhz  25Nov14 borrowed setup for ASIC correlator from
c                 smasplit
c    jhz  15Feb15 re-config uvflags index for SWARM chunks in a uv handling
c                 subroutine
c    jhz  20Feb15 tested for ASIC data only
c    jhz  15Jun15 implemented select functions for spectral chunks from SWARM
c                 correlator for two spectral chunks (1st quadrant)
c    jhz  01jul15 implemented for SWARM+ASIC hybrid correlator
c    jhz  14jul15 increased MAXVIS and updated inline doc
c    jhz  20jul15 implement function of linetype for swarm data
c                 to reduce down spectral resolution
```

```
c
c  To do:
c                To implement more SWARM chunks.
c-----------------------------------------------------------------------
        include 'maxdim.h'
        character version*(*)
        parameter(version='SwarmSplt: version1.0.6 20-July-2015')
        character uvflags*12,ltype*16,out*64
        integer npol,Snpol,pol,tIn,tOut,vupd,nread,nrec,i,nbin
        real inttime,jyperk
        logical dotaver,doflush,buffered,PolVary,first
        logical ok,donenpol,doasic,doch0,doif1,doif2,dohyb
        logical doswarm,dosw1,dosw2,doasicsw
        double precision preamble(5),Tmin,Tmax,Tprev,interval
        complex data(MAXCHAN)
        logical flags(MAXCHAN)
        logical ampsc,vecamp,relax,doprt
        integer nspect, s2m(2,2)
        integer nschan(MAXWIN)
        double precision sfreq(MAXWIN)

c        character in*80
c        integer tno
c
c  Externals.
c
        logical uvDatPrb,uvDatOpn,uvVarUpd
        logical doSMAorg
c
        common/nswindows/nspect
        common/SMAIFS/s2m

c  Get the input parameters.
c
        nspect = 51
        dohyb = .false.
        doprt = .false.
        call output(version)
        call keyini
        uvflags = 'slr3'
        call uvDatInp('vis',uvflags)
        if(uvDatOpn(tIn)) then
        call uvDatRd(preamble,data,flags,maxchan,nread)
        dowhile(nread.gt.0)
        call uvrdvri(tin,'nspect',nspect,0)
        call uvgetvri(tin,'nschan',nschan,nspect)
        call uvgetvrd(tin,'sfreq',sfreq,nspect)
        call SpecSMA(tin,
     *   nspect,nschan,sfreq,MAXWIN,doprt)
        call uvDatRd(preamble,data,flags,maxchan,nread)
         if (nspect.gt.2) nread=0
         end do
         if (nspect.lt.51) dohyb = .true.
         end if
        call uvDatCls
        doSMAorg=.false.
        if(nschan(1).eq.1) doSMAorg=.true.
        if(.not.doSMAorg) call bug('f',
     *  'SWARMSPLT: does not work for this dataset')
        call keyini
        uvflags = 'dslr3'
        call
     *  getopt(uvflags,doasic,doch0,doif1,doif2,doswarm,
     *  dosw1,dosw2,doasicsw)
        call uvDatInp('vis',uvflags)
        call keyd('interval',interval,0.d0)
        call keya('out',out,' ')
        call keyfin
c
```

```
c check the logic conflict in options
        if(doif2.and.doif1)
     *  call bug ('f','Hybrid-mode: do one IF at a time.')
c
c make a certain no averaging in the splitting
c
        interval = 0.d0
           ampsc = .false.
          vecamp = .false.
           relax = .false.
c
c  Check the input parameters.
c
        if(doasic.and.(.not.dohyb))
     *  call bug ('w',
     *  'Split the SMA ASIC data into 48 spectral chunks data.')
        if(doch0)
     *  call bug ('w',
     *  'Split the first channel (Ch0) from the SMA ASIC data.')
        if(doif1)
     *  call bug ('w',
     *  'Split the first IF data from the SMA ASIC data.')
        if(doif2)
     *  call bug ('w',
     * 'Split the second IF data from the SMA ASIC data.')
        if(dohyb)
     *  call bug ('w',
     *  'Handling the hybrid-resolution SMA ASIC data,')
        if(dohyb)
     *  call bug ('w', 'split all spectral chunks data.')
        if(doswarm)
     *  call bug ('w', 'Split all SWARM chunks.')
        if(dosw1)
     *  call bug ('w', 'Split 1st SWARM chunk S49.')
        if(dosw2)
     *  call bug ('w', 'Split 2nd SWARM chunk S50.')
        if(doasicsw)
     *  call bug ('w', 'Split all SWARM and ASIC chunks.')

        if(out.eq.' ')call bug('f','Output file must be specified')
        if(interval.lt.0)call bug('f','Illegal value for interval')
c
c  Various initialisation.
c
        interval = interval/(24.*60.)
        npol = 0
        Snpol = 0
        first = .true.
        PolVary = .false.
        doflush = .false.
        buffered = .false.
        donenpol = .false.
        dotaver = interval.gt.0.or.uvDatPrb('polarization?',0.d0)
        call BufIni
        nrec = 0
c
c  Open the input and the output files.
c
        dowhile(uvDatOpn(tIn))
          nbin = 1
          if(dotaver)then
            call uvrdvri(tIn,'nbin',nbin,1)
            if(nbin.gt.1)then
              call bug('w',
     *        'Time averaging or pol''n selection of bin-mode data')
              call bug('w',
     *        'This will average all bins together')
            endif
          endif
```

```
          call uvDatGta('ltype',ltype)
          call VarInit(tIn,ltype)
          call uvVarIni(tIn,vupd)
          call uvVarSet(vupd,'dra')
          call uvVarSet(vupd,'ddec')
          call uvVarSet(vupd,'source')
          call uvVarSet(vupd,'on')
c
c Special processing the first time around.
c
          if(first)then
            call uvopen(tOut,out,'new')
            call uvset(tOut,'preamble','uvw/time/baseline',0,0.,0.,0.)
            call hdcopy(tIn,tOut,'history')
            call hisopen(tOut,'append')
            call hiswrite(tOut,'SWARMSPLT: Miriad '//version)
            call hisinput(tOut,'SWARMSPLT')
            call hisclose(tOut)
            first = .false.
          endif
          call VarOnit(tIn,tOut,ltype)
c
c  Loop over the data.
c
          call uvDatRd(preamble,data,flags,maxchan,nread)
          Tprev = preamble(4)
          Tmin = Tprev
          Tmax = Tmin
          dowhile(nread.gt.0)
c
c  Count the number of records read.
c
            nrec = nrec + 1
c
c  Do we want to keep this record.
c
            ok = relax.or.donenpol
            if(.not.ok)then
              do i=1,nread
                ok = ok.or.flags(i)
              enddo
            endif
c
c  Determine if we need to flush out the averaged data.
c
            doflush = ok.and.dotaver
            if(doflush)then
              doflush = uvVarUpd(vupd)
              doflush = (doflush.or.preamble(4)-Tmin.gt.interval.or.
     *                             Tmax-preamble(4).gt.interval)
     *                  .and.buffered
            endif
c
c  Flush out the accumulated data -- the case of time averaging.
c
            if(doflush)then
              call bufflush(tOut,ampsc,vecamp,npol)
              PolVary = PolVary.or.npol.eq.0.or.
     *          (Snpol.ne.npol.and.Snpol.gt.0)
              Snpol = npol
              Tmin = preamble(4)
              Tmax = Tmin
              buffered = .false.
c
c  Flush out the accumulated data -- the case of no time averaging.
c
            else if(.not.dotaver)then
              if(npol.le.0)call uvDatGti('npol',npol)
              if(ok)then
```

```
                if(.not.donenpol)then
                  call uvputvri(tOut,'npol',npol,1)
                  PolVary = PolVary.or.
     *              (Snpol.ne.npol.and.Snpol.gt.0)
                  Snpol = npol
                endif
                call uvDatGti('pol',pol)
                call uvputvri(tOut,'pol',pol,1)
                call VarCopy(tIn,tOut)
                call uvDatGtr('jyperk',jyperk)
                call uvputvrr(tOut,'jyperk',jyperk,1)
                call uvwrite(tOut,preamble,data,flags,nread)
                donenpol = npol.gt.1
              endif
              npol = npol - 1
            endif
c
c  Accumulate more data, if we are time averaging.
c
            if(dotaver.and.ok)then
              call uvrdvrr(tIn,'inttime',inttime,10.)
              call bufacc(preamble,inttime,data,flags,nread)
              buffered = .true.
              call VarCopy(tIn,tOut)
              if(nbin.gt.1)call uvputvri(tOut,'nbin',1,1)
            endif
c
c  Keep on going. Read in another record.
c
            if(ok)then
              Tprev = preamble(4)
              Tmin = min(Tmin,Tprev)
              Tmax = max(Tmax,Tprev)
            endif
            call uvDatRd(preamble,data,flags,maxchan,nread)
          enddo
c
c  Flush out anything remaining.
c
          if(buffered)then
            call bufflush(tOut,ampsc,vecamp,npol)
            PolVary = PolVary.or.npol.le.0.or.
     *        (Snpol.ne.npol.and.Snpol.gt.0)
            Snpol = npol
            buffered = .false.
          endif
          call uvDatCls
        enddo
c
c  Write things to the header which describe the data as a whole.
c
        if(first)call bug('f','Error opening input')
        if(nrec.eq.0)call bug('f','No data found')
        if(.not.PolVary)call wrhdi(tOut,'npol',Snpol)
c
c  Update the history and close up files.
c
        call uvclose(tOut)
        if(doasic.or.doasicsw.or.doswarm.or.
     *  doif1.or.doif2.or.dosw1.or.dosw2)
     *  call output('The program ends successfully.')
        end
c
c************************************************************************
c
        subroutine getopt(uvflags,doasic,doch0,doif1,doif2,
     * doswarm,dosw1,dosw2,doasicsw)
        implicit none
        logical doasic,doch0,doif1,doif2
```

```
        logical doswarm,dosw1,dosw2,doasicsw
        character uvflags*(*)
c
c  Determine the flags to pass to the uvdat routines.
c
c  Output:
c    uvflags    Flags to pass to the uvdat routines.
c    doasic  Handling the SMA DB data for the 48 spectral windows.
c    doch0      Handling the SMA DB data for the ch0, the 1st window.
c    doif1      Handling the SMA DB data for the first 24 windows
c               (2,3,...,25).
c    doif2      Handling the SMA DB data for the next 24 windows
c               (26,27,...,49).
c    doswarm    Handling the SMA SWARM data for 50 (S50) and 51 (S49)
c    dosw1      Handling the SMA SWARM data for 51 (S49), the 1st swarm chunk
c    dosw2      Handling the SMA SWARM data for 50 (S50), the 2nd swarm chunk
c    doasicsw   Handling all the spectral windows from both ASIC and SWARM
c               but excluding the single-channel window Ch0
c-----------------------------------------------------------------------
        integer nopts
        parameter(nopts=10)
        character opts(nopts)*8
        integer l, nopt
        logical present(nopts),docal,dopol,dopass
        data opts/'nocal    ','nopol    ','nopass   ', 'ch0      ',
     *            'asic     ','if1      ','if2      ', 'swarm    ',
     *            'sw1      ','sw2      '/
c
        call options('options',opts,present,nopts)
        nopt=0
        doch0     = present(4)
        doasic    = present(5)
        doif1     = present(6)
        doif2     = present(7)
        doswarm   = present(8)
        dosw1     = present(9)
        dosw2     = present(10)
        if(doch0)     nopt=nopt+1
        if(doasic) nopt=nopt+1
        if(doif1)     nopt=nopt+1
        if(doif2)     nopt=nopt+1
        if(doswarm)   nopt=nopt+1
        if(dosw1)     nopt=nopt+1
        if(dosw2)     nopt=nopt+1
        if(nopt.gt.1) call bug('f', 'One options at a time.')
        nopt=0
        if(.not.doch0)     nopt=nopt+1
        if(.not.doasic) nopt=nopt+1
        if(.not.doif1)     nopt=nopt+1
        if(.not.doif2)     nopt=nopt+1
        if(.not.doswarm)   nopt=nopt+1
        if(.not.dosw1)     nopt=nopt+1
        if(.not.dosw2)     nopt=nopt+1
        if(nopt.eq.7) doasicsw = .true.
        docal  = .false.
        dopol  = .false.
        dopass = .false.
c
c Set up calibration flags
c
        uvflags = 'dslr3'
        l = 5
        if(docal)then
          l = l + 1
          uvflags(l:l) = 'c'
        endif
        if(dopass)then
          l = l + 1
          uvflags(l:l) = 'f'
```

```
        endif
        if(dopol)then
          l = l + 1
          uvflags(l:l) = 'e'
        endif
        if(doasic) then
          l = l + 1
          uvflags(l:l) = 'z'
        endif
        if(doch0) then
          l = l + 1
          uvflags(l:l) = '0'
        endif
        if(doif1) then
          l = l + 1
          uvflags(l:l) = 'y'
        endif
        if(doif2) then
          l = l + 1
          uvflags(l:l) = '2'
        endif
        if(doswarm) then
          l = l + 1
          uvflags(l:l) = '4'
        endif
        if(dosw1) then
          l = l + 1
          uvflags(l:l) = '5'
        endif
        if(dosw2) then
          l = l + 1
          uvflags(l:l) = '6'
        endif
        if(doasicsw) then
          l = l + 1
          uvflags(l:l) = '7'
        endif
        end
c
c************************************************************************
c
        subroutine bufini
        implicit none
c
c  Initialise the routines which do the buffering and averaging of
c  the visibility data.
c  All the buffering/averaging is performed in arrays stored in a
c  common block.
c
c------------------------------------------------------------------------
        include 'swarmsplt.h'
        free = 1
        mbase = 0
        end
c
c************************************************************************
c
        subroutine bufflush(tOut,ampsc,vecamp,npol)
c
        implicit none
        integer tOut,npol
        logical ampsc,vecamp
c
c  This writes out the averaged data. The accumulated data is in common.
c  This starts by dividing the accumulated data by "N", and then writes
c  it out.
c
c  Inputs:
c    tOut       The handle of the output file.
```

```
c    ampsc      True for amp scalar averaging
c    vecamp     True for amp scalar averaging of only parallel hand
c               amplitudes.
c  Output:
c    npol       The number of polarisations in the output. If this
c               varies, a zero is returned.
c-----------------------------------------------------------------------
        include 'swarmsplt.h'
        complex data(MAXCHAN)
        real amp,inttime
        double precision preambl(5),time(MAXBASE)
        logical flags(MAXCHAN)
        integer i,j,jd,k,ngood,nbp,p,idx1(MAXBASE),idx2(MAXBASE)
        logical PolVary,doamp
c
c  Externals.
c
        logical PolsPara
        PolVary=.false.
c
c  Determine the number of good baselines, and sort them so we have an
c  index of increasing time.
c
        ngood = 0
        do j=1,mbase
          if(cnt(j).gt.0)then
            ngood = ngood + 1
            time(ngood) = preamble(4,j) / cnt(j)
            idx2(ngood) = j
          endif
        enddo
        if(ngood.le.0)return
        call sortidxd(ngood,time,idx1)
c
c  Now loop through the good baselines, writing them out.
c
        nbp = 0
        npol = 0
        do jd=1,ngood
          j = idx2(idx1(jd))
          if(npols(j).ne.npol)then
            call uvputvri(tOut,'npol',npols(j),1)
            PolVary = npol.gt.0
            npol = npols(j)
          endif
          preambl(1) = preamble(1,j) / cnt(j)
          preambl(2) = preamble(2,j) / cnt(j)
          preambl(3) = preamble(3,j) / cnt(j)
          preambl(4) = preamble(4,j) / cnt(j)
          preambl(5) = preamble(5,j) / cnt(j)
          inttime    = preamble(6,j) / npols(j)
          call uvputvrr(tOut,'inttime',inttime,1)
c
c  Average the data in each polarisation. If there is only one scan in the
c  average, not bother to average it.
c
          do i=1,npol
            p = pnt(i,j) - 1
            call uvputvri(tOut,'pol',pols(i,j),1)
            doamp = ampsc.or.(vecamp.and.PolsPara(pols(i,j)))
            nbp = nbp + 1
c
c  Loop over the channels. If we are doing amp-scalar averaging, and
c  the average visibility is zero, flag the data. Otherwise just
c  depend on whether we have good data or not.
c
            do k=1,nchan(i,j)
              if(doamp.and.
     *          abs(real(buf(k+p)))+abs(aimag(buf(k+p))).eq.0)
```

```
     *          count(k+p) = 0
                flags(k) = count(k+p).gt.0
                if(.not.flags(k))then
                  data(k) = 0
                else if(doamp)then
                  amp = abs(buf(k+p))
                  data(k) = (bufr(k+p) / count(k+p)) *
     *                       (buf(k+p) / amp)
                else
                  data(k) = buf(k+p) / count(k+p)
                endif
              enddo
              call uvwrite(tOut,preambl,data,flags,nchan(i,j))
            enddo
          enddo
c
c  Reset the counters.
c
          free = 1
          mbase = 0

c  If the number of polarisations varied, zero npol.
c
          if(PolVary) npol = 0
          end
c
c***********************************************************************
c
          subroutine bufacc(preambl,inttime,data,flags,nread)
c
          implicit none
          integer nread
          double precision preambl(5)
          real inttime
          complex data(nread)
          logical flags(nread)
c
c  This accumulates the visibility data. The accumulated data is left
c  in common.
c
c  Input/Output:
c    preambl    Preamble. Destroyed on output.
c    data       The correlation data to be averaged. Destroyed on output.
c    flags      The data flags.
c    nread      The number of channels.
c-----------------------------------------------------------------------
          include 'swarmsplt.h'
          integer i,i1,i2,p,bl,pol
c
c  Determine the baseline number, and conjugate the data if necessary.
c
          call BasAnt(preambl(5),i1,i2)
          bl = ((i2-1)*i2)/2 + i1
          if(bl.gt.MAXBASE)
     *    call bug('f','Too many baselines for me to handle, in BUFACC')
c
c  Zero up to, and including, this baseline.
c
          do i=mbase+1,bl
            cnt(i) = 0
          enddo
          mbase = max(mbase,bl)
c
c  Add in this visibility.
c
         if(cnt(bl).eq.0)then
            cnt(bl) = inttime
            npols(bl) = 0
            preamble(1,bl) = inttime * preambl(1)
```

```
            preamble(2,bl) = inttime * preambl(2)
            preamble(3,bl) = inttime * preambl(3)
            preamble(4,bl) = inttime * preambl(4)
            preamble(5,bl) = inttime * preambl(5)
            preamble(6,bl) = inttime
          else
            cnt(bl) = cnt(bl) + inttime
            preamble(1,bl) = preamble(1,bl) + inttime * preambl(1)
            preamble(2,bl) = preamble(2,bl) + inttime * preambl(2)
            preamble(3,bl) = preamble(3,bl) + inttime * preambl(3)
            preamble(4,bl) = preamble(4,bl) + inttime * preambl(4)
            preamble(5,bl) = preamble(5,bl) + inttime * preambl(5)
            preamble(6,bl) = preamble(6,bl) + inttime
          endif
c
c  Determine the polarisation.
c
          call uvDatGti('pol',pol)
          p = 0
          do i=1,npols(bl)
            if(pols(i,bl).eq.pol) p = i
          enddo
c
c  A new baseline. Set up the description of it.
c
          if(p.eq.0)then
            npols(bl) = npols(bl) + 1
            p = npols(bl)
            if(p.gt.MAXPOL) call bug('f',
     *        'Too many polarizations, in BufAcc')
            pols(p,bl) = pol
            nchan(p,bl) = nread
            pnt(p,bl) = free
            free = free + nread
            if(free.gt.MAXAVER)call bug('f',
     *        'Too much data to accumulate, in BufAcc')
c
c  Copy across the new data.
c
            p = pnt(p,bl) - 1
            do i=1,nread
              if(flags(i))then
                buf(i+p) = inttime * data(i)
                bufr(i+p) = inttime * abs(data(i))
                count(i+p) = inttime
              else
                buf(i+p) = (0.0,0.0)
                bufr(i+p) = 0.0
                count(i+p) = 0
              endif
            enddo
c
c  Else accumulate new data for old baseline.
c
          else
            nread = min(nread,nchan(p,bl))
            nchan(p,bl) = nread
            p = pnt(p,bl) - 1
            do i=1,nread
              if(flags(i))then
                buf(i+p) = buf(i+p) + inttime * data(i)
                bufr(i+p) = bufr(i+p) + inttime * abs(data(i))
                count(i+p) = count(i+p) + inttime
              endif
            enddo
          endif
c
          end
```

5.2 *swarmsplt.h*

```
c************************************************************************
c  jhz:  2014-11-25
c  jhz:  2015-07-14
c  Include file for swarmsplt.for
c
c  Buf          Buffer used to accumulate the data.
c  Bufr         Buffer used to accumulate amplitudes
c               for amp-scalar averaging
c  Count(i)     Weights of the correlations added into Data(i).
c  free         Points to the first unused location in Data and Count.
c  pnt          For a baseline, points to location of data in Data and Count.
c  nchan        Number of channels for a given baseline.
c  npols                Number of polarisations.
c  pols         The polarisation codes.
c  preamble     The accumulated preambles.
c  cnt          Weights of the things accumulated into the preambles.
c
        include 'maxdim.h'
        integer MAXAVER,MAXPOL
        parameter(MAXAVER=20000000,MAXPOL=4)
        complex buf(MAXAVER)
        real    bufr(MAXAVER)
        double precision count(MAXAVER),cnt(MAXBASE)
        integer pnt(MAXPOL,MAXBASE),nchan(MAXPOL,MAXBASE),free,mbase
        integer npols(MAXBASE),pols(MAXPOL,MAXBASE)
        double precision preamble(6,MAXBASE)
        common/uvavcom/preamble,count,cnt,buf,bufr,pnt,nchan,npols,
     *    pols,free,mbase
```