Section Contents

U320.010 Introduction

A non-CHIL histogram generating procedure has been defined. This is called the UCUS (for User CUStomized) procedure and the associated tape scan package is called scanu. I started to call this the Fully User Customized procedure, since CHIL is partially user customizable, but then thought better of it.

scanu is essentially the same (with a few minor changes) as Jim Beene's scanvdm which was derived from the VAX version of SCANM. Jim re-wrote all tape and disk I/O routines, tape control routines and all tape and file opening routines for UNIX compatibility. He also added several useful features and produced a user document which I have encorporated into this document. The objective here is to "standardize things" by placing all required libraries in the /usr/hhirf/ directory and assuming responsibility for software maintenance and user documentation.

Filename Extensions (.drr & .his)

I have chosen to make lower case filename extensions (.drr & .his) the standard for UNIX systems rather than (.DRR & .HIS) as used in scanvdm. I can't make a STRONG case for this but it seems a bit easier; 1) most names in UNIX are lower case, you don't have to shift to type the name extension; 2) when you copy from the VAX filenames become lower case; 3) damm can handle it either way.

U320.020 General Features

(1)....The scanu main package provides for tape reading & control, command processing, user command processing, etc. in the usual way.

(2)....All histograms are generated in memory which is allocated via a request to the "system", therefore, the maximum histogram size is determined by the amount of memory available and will vary with the DECstation being used, the time of day, etc.

(3)....UCUS & CHIL procedures use the same .his and .drr file structures.

(4)....A proper .drr file may either exist or you must provide a routine, DRRSUB, to create it at run time. See SEC# U320.030 thru U320.050.

(5)....The .his file may either exist or you will be asked for permission to create it at run time.

(6)....The main program calls user-supplied routine, HISSUB(IBUF,N), with an INTEGER*2 event buffer IBUF which may contain a raw buffer from tape (default) or a single packed event (requested by the CMD SEBU). If a single event buffer is chosen, N is the number of words in IBUF (the FFFF is not passed). For the raw tape buffer case, N is the buffer length (normally 4096).

(7)....histogramming is accomplished by user-supplied calls, normally from routine HISSUB, to one of the following standard routines:

```
COUNT1(  ID,IX,JY)   ;For no limit-checking, ranging, compression
COUNT1C( ID,IX,JY)   ;For    limit-checking, ranging
COUNT1CC(ID,IX,JY)   ;For    limit-checking, ranging, compression
```

Where, ID is a histogram ID-number in the range 1 to 8000. IX and JY are X- and Y-channel numbers in the range 0 to whatever. For 1-D histograms, JY is ignored but must be supplied. See SEC# U320.070.

U320.025 Using Shared Histogram Memory Segments (DECstations only)

Program scanu can be requested to generate histograms in either a shared or a local memory segment as indicated below:

```
scanu name        ;Starts scanu using a shared memory segment (default)

scanu name local  ;Starts scanu using a local  memory segment
```

The advantage of using the shared segment is that damm can access histograms (in memory) as they are being generated without waiting for an end or a hup.

WARNING!! if scanu should terminate abnormally (core dump), the shared segment will not be released as it normally would. You will need to perform the cleanup operation shown below, otherwise the system will eventually be eaten up with "abandoned memory segments".

Type: /usr/hhirf/shm_fixup name

Where, name is the his-file name prefix that you used in starting scanu.

IMPORTANT!!

If you do not run shm_fixup at the time of the "abnormal termination" and the machine becomes almost inoperable due to the memory tied up with abandoned segments, do the following: Find these abandoned segments by displaying all files with the .shm name extension. If you find a file name.shm and are not currently running scanu or scan with name.his, then you have found an abandoned segment and should run shm_fixup as described above.

Under certain conditions, abandoned segments may not have an associated shm-file. To find and remove these do the following.

```
Type: ipcs           ;To display shared mmemory segments & IDs

Type: ipcrm -m ID    ;To remove  shared memory  segment    ID
```

U320.030 Discussion of User-supplied Routine DRRSUB

The basic function of a DRRSUB is to define histogram specifications and write a HHIRF form directory file (.drr file). It is called ONCE by a scanu task, during the initialization process, at the beginning of a scan, and builds a HHIRF format histogram directory file (.drr file). The same information is used to allocate a .his file, and to determine how much memory to request from the OS for histogram storage. You can make it as elaborate as you like, with user prompting, etc.(for example you could test for the existence of a directory file, and if it exists ask the user if he wants to recreate it or not). The call sequence of a DRRSUB is:

```
      SUBROUTINE DRRSUB(IEXIST)
      INTEGER IEXIST
C
C     At entry  IEXIST will be 1 if a .drr file of the proper name already
C     exists and 0 otherwise. The basic structure of a DRRSUB is:
C
      SUBROUTINE DRRSUB(IEXIST)
      INTEGER IEXIST
      ...
C                     !tests and/or user interaction
C
      CALL DRRMAKE    !make new or overwrite existing DRR
C                     !calls to HDEF, HD1D or HD2D (up to 2008,
C                     !one for each histogram being defined.
C                     !See call sequence below)
      CALL ENDRR
      ...
      RETURN
      END
```

A DRRSUB is not required. A dummy DRRSUB is included in scanmlib and will be linked if you don't include one in your link list. It is a no-op which writes a message to the screen to the effect that it hasn't done anything. Using the dummy DRRSUB means that you are depending on the .drr file to already exist, whose name matches that specified on your start command line (e.g. "scanu name"). A directory file produced by CHIL works just as well as one generated by fortran calls (they're functionally identical).

DRRMAKE should not be called if you want to reuse an existing DRR, since it destroys the information in the old .drr. The calls to DRRMAKE and ENDRR are both required to generate a valid new .drr file and to finalize the generation of histogram specs. There is no harm in regenerating a .drr every time you run scanu, even though it's always the same. This does not result in a loss of information in the .his file. (Renewing a .drr file doesn't zero a .his file.)

U320.040 Histogram Specification Routines (called by DRRSUB)

There are three basic histogram definition routines, a general one HDEF which allows up to 4 dimensional histograms, and HD1D and HD2D which are specific to 1 and 2 dimensional histograms respectively. A DRRSUB should consist of a series of calls to HDxx family routines (one call per histogram) followed by a call to ENDRR (no arguments) which tells the system you're finished. A histogram directory is then listed for you on stderror and on your log file.

U320.040 Histogram Specification Routines (continued)

Note that there are some philosophical differences between a histogram specification in DRRSUB and a histogram definition in CHIL: some examples of these differences will be mentioned here to illustrate basic concepts. In CHIL, gates, conditions, etc. are directly associated with a histogram. A call to a HDxx family routine only reserves space and assigns an ID number, minimum and maximum parameter values and a compression factor to the histogram. How you increment the histogram depends of the logic of the fortran in your HISSUB. In CHIL, histograms are associated with parameter numbers, and multiple H() and OH() specs are required to generate an overlay of many parameters in to one HID. In a DRRSUB a single HDxx call is all that is necessary. The HISSUB does the overlaying.

Call Sequence for HDEF - General Histogram Specification Routine

HDEF is really too general for most purposes and requires dimensioned arrays, etc. So, read about HDEF but use routines HD1D and HD2D described on following pages.

```
      SUBROUTINE HDEF(
     &      HID,     ![INT] HIST. ID
     &      HDIM,    ![INT] HIST. DIMENSIONALITY (1-4)
     &      NHWPC,   ![INT] NO. OF HALF WORDS PER CHANNEL (1 or 2)
     &      RAWL_LST,![INT ARR] LIST OF HDIM RAW PARAM LENGTHS
     &      HSTL_LST,![INT ARR] LIST OF HDIM HISTOG. PARAM LENG.
     &      MIN_LST, ![INT ARR] LIST OF HDIM "RANGE" MINIMA
     &      MAX_LST, ![INT ARR] LIST OF HDIM "RANGE" MAXIMA
     &      LABX,    ![CHAR*N] (N.LE.12) X-LABEL
     &      LABY,    ![CHAR*N] (N.LE.12) Y-LABEL
     &      TIT)     ![CHAR*N] (N.LE.40) TITLE FOR HISTOGRAM
C     ****************************************************************
      IMPLICIT INTEGER*4 (A-Z)
      INTEGER*4 RAWL_LST(HDIM),HSTL_LST(HDIM),MIN_LST(HDIM),MAX_LST(HDIM)
      CHARACTER*(*) LABX,LABY,TIT
c     ****************************************************************
c        o   Reserves 1 histogram per call.
c        o   A call to DRRMAKE is required before first HDxx call.
c        o   A call to ENDRR is required after last HDxx call.
c        o   The four dummy arguments with suffix "_LST" (i.e. list)
c            in the HDEF call are (and the corresponding real arguments
c            must also be) arrays of dimension HDIM. The corresponding
c            dummy arguments (similar names, no "_LST" suffix) in
c            HD1D and HD2D are scalars. I leave off the "_LST" in the
c            following discussion.
c        o   RAWL and HSTL are both automatically adjusted to the
c            nearest power  of 2. working in powers of 2 is
c            traditional at HHIRF.
c        o   A value of MIN different from 0 and/or a value of MAX
c            different from HSTL has the same effect as a range
c            specification in CHIL....e.g. the corresponding CHIL is
c            $LPR 15 = RAWL
c            ..
c            $HID HID
c            H(15) L(HSTL) R(MIN,MAX).
```

U320.040 Histogram Specification Routines (continued)

```
c        o   A compression (given by RAWL/HSTL) is associated with
c            each parameter of each HID. This compression is applied for
c            you if you increment with COUNT1CC. COUNT1 and
c            COUNT1C ignore it. A compression is always a power of 2.
c        o   Some of the arguments are "optional" in the sense that
c            that if the value is 0 in the call (not missing)
c            a sensible default will be taken:
c                RAWL  IF 0 SET TO 2048.
c                HSTL  IF 0 SET TO RAWL
c                MIN   IF 0 LEFT AS 0
c                MAX   IF 0 SET TO HSTL-1
c        o   Two special entries HD1D and HD2D are provided for
c            1d and 2d histograms. They are all you will ever need.
            Forget HDEF.
```

Call Sequence for HD1D - 1D Histogram Specification Routine

```
C      ************************************************************
C      ENTRY HD1D -  ENTRY FOR 1D HISTOGRAM
C      ************************************************************
       ENTRY HD1D(
     &      HID,     ! [INT] HIST. ID
     &      NHWPC,   ! [INT] NO. OF HALF WORDS PER CHANNEL (1 or 2)
     &      RAWL,    ! [INT] RAW PARAMETER LENGTH
     &      HSTL,    ! [INT] HISTOGRAMED PARAM LENGTH
     &      MN,      ! [INT] MINIMUM PARAM "RANGE" VALUE
     &      MX,      ! [INT] MAXIMUM PARAM "RANGE" VALUE
     &      TIT)     ! [CHAR*N] (N<40) TITLE FOR HISTOGRAM
C     ************************************************************
C     Same as HDEF but only for 1D. Note all numeric arguments are scalars
C     whereas in HDEF many are arrays.
C     See HDEF for values assumed when arguments are 0.
```

Call Sequence for HD2D - 2D Histogram Specification Routine

```
C     ************************************************************
C     ENTRY HD2D - ENTRY FOR 2D HISTOGRAM
C     ************************************************************
      ENTRY HD2D(
     &      HID,    ! [INT] HIST. ID
     &      NHWPC,  ! [INT] NO. OF HALF WORDS PER CHANNEL (1 or 2)
     &      RAWLX,  ! [INT] RAW X PARAM LENGTH
     &      HSTLX,  ! [INT] HISTOGRAMED X PARAM LENGTH
     &      MNX,    ! [INT] MINIMUM X PARAM "RANGE" VALUE
     &      MXX,    ! [INT] MAXIMUM X PARAM "RANGE" VALUE
     &      RAWLY,  ! [INT] RAW Y PARAM LENGTH
     &      HSTLY,  ! [INT] HISTOGRAMED Y PARAM LENGTH
     &      MNY,    ! [INT] MINIMUM Y PARAM "RANGE" VALUE
     &      MXY,    ! [INT] MAXIMUM Y PARAM "RANGE" VALUE
     &      TIT)    ! [CHAR*N] (N<40) TITLE FOR HISTOGRAM
C     ************************************************************
C     Same as HDEF but only for 2D. Note all numeric arguments are scalars
C     whereas in HDEF many are arrays.
C     See HDEF for values assumed when arguments are 0.
```

U320.050 Example User-supplied DRRSUB routine

A very simple DRRSUB, which books 7 1D histograms and one 2D histogram. The ranging feature is illustrated by the 2D histogram. The H1D calls depend on the defaulting of MAX to HSTL-1 (i.e. 511).

```
      SUBROUTINE DRRSUB(IXST)
C
C     Very simple example of a DRRSUB.
C
      IMPLICIT INTEGER*4 (A-Z)
C
      CALL DRRMAKE
C
      DO I=1,7
      CALL HD1D(I,2,2048,512,0,0,'RAW PARAM. HIST.')
      ENDDO
C
      CALL HD2D(100,1,2048,2048,0,1000,2048,2048,27,2047,'2D HIST.')
C
      CALL ENDRR
      END
```

****

Hisogram spec list produced by running scanu with the DRRSUB above.

8 HISTOGRAMS, 2030189 HALF-WORDS

ID-LIST:

1 2 3 4 5 6 7 100

HID	DIM	HWPC	LEN(CH)	COMPR	MIN	MAX	OFFSET	TITLE
1	1	2	512	4	0	511	0	RAW PARAM. HIST.
2	1	2	512	4	0	511	1024	RAW PARAM. HIST.
3	1	2	512	4	0	511	2048	RAW PARAM. HIST.
4	1	2	512	4	0	511	3072	RAW PARAM. HIST.
5	1	2	512	4	0	511	4096	RAW PARAM. HIST.
6	1	2	512	4	0	511	5120	RAW PARAM. HIST.
7	1	2	512	4	0	511	6144	RAW PARAM. HIST.
100	2	1	2023021	1	0	1000	7168	2D HIST.
				1	27	2047		

****

U320.060 Dicussion of User-supplied Routine HISSUB

The main program calls user-supplied routine, HISSUB(IBUF,N), with an INTEGER*2 event buffer IBUF which may contain a raw buffer from tape (default) or a single packed event (requested by the CMD SEBU). If a single event buffer is chosen, N is the number of words in IBUF (the FFFF is not passed). For the raw tape buffer case, N is the buffer length (normally 4096).

You can and will, of course, do many complicated things in HISSUB and the simple example below is intended to illustrate the procedure only. In this example, IBUF is assumed to contain a full tape buffer.

```
C$PROG HISSUB
      SUBROUTINE HISSUB(IBUF,NB)
C
C     ****************************************************************
C     EXAMPLE HISSUB - WHICH HISTOGRAMS ALL RAW PARMS AT 1/4 RES
C                    - WITH HIS-ID = PARM-ID
C     ****************************************************************
C
      INTEGER*2 IBUF(*)
      DATA ID/0/
C
      DO 100 I=1,NB
      IF(IBUF(I).GE.0) GO TO 50
      IF(IBUF(I).EQ.'FFFF'X) THEN
                             ID=0
                             GO TO 100
                             ENDIF
      ID=IAND(IBUF(I),'00FF'X)-1
      GO TO 100
   50 ID=ID+1
      IX=IBUF(I)/4
      CALL COUNT1(ID,IX,0)
  100 CONTINUE
      RETURN
      END
```

U320.070 Discussion of Histogram Incrementing Routines

There are three histogram incrementing routines which differ only in the amount of checking and manipulating they do for you. The routines are COUNT1, COUNT1C and COUNT1CC. All perform the same "add one to histogram HID" function but COUNT1, which is the fastest routine does the incrementing without any checking or modification of parameter values. COUNT1C (add 1 and check) does no compression does do "ranging" and limit checking. COUNT1CC (add 1 check and compress) increments with compression (divide input by power of 2) as well as ranging and limit checking. The compression factor used by COUNT1CC and the range limits used by COUNT1CC and COUNT1C are associated with a HID at the time the histogram is defined with the HDxx family call in DRRSUB. Call sequences are given below.

```
      SUBROUTINE COUNT1(ID,IX,IY)
      INTEGER*4 ID,IX,IY

      ****************************************************************
      Increments histogram with HID ID at IX and IY. No limit
      checking or "ranging" is done. No compression is applied.
      You're on your own. This call is provided to let you do things
      the fastest way you know how. IY is ignored for a 1D histogram.
      NOTE!!! IX and IY are NOT HALF WORDS!!!

      SUBROUTINE COUNT1C(ID,IX,IY)
      INTEGER*4 ID,IX,IY

      ****************************************************************
      Increments histogram with HID ID at IX and IY. Limit
      checking is done, and CHIL type "ranging" is done, but
      no compression is applied. This lets you do your own
      compressing or save a few instructions if you know it's
      not needed.
      NOTE!!! IX and IY are NOT HALF WORDS!!!

      SUBROUTINE COUNT1CC(ID,IX,IY)
      INTEGER*4 ID,IX,IY

      ****************************************************************
      Increments histogram with HID ID at IX and IY. Limit
      checking is done, and CHIL type "ranging" is done, and
      the compression implied by the RAWL/HSTL ratio in your
      HDxx call is applied.
      NOTE!!! IX and IY are NOT HALF WORDS!!!
```

Note that no calls are provided to increment histograms with dimensionality .GT. 2, so HDEF isn't all that useful (that's what I meant by too general). You can actually increment a 3 or 4D histogram with COUNT1 if you think about it a bit.

U320.080 Discussion of 2-D Free-Form Gating

2-D free-form gate testing must be carried out by user-supplied code executed by routine HISSUB. This is also true for 1-D gate testing. However, there are some intrinsic aids for 2-D gate testing and ban-file processing. This is how it goes:

Reading in ban-files----------

One or more ban-files are opened and read in via the commands:

```
ban file1.ban
ban file2.ban
ban file3.ban
     .
     .
```

All entries (bananas) in each file are read in and stored in memory in the order in which they occur in the file. Subsequent references to individual bananas (stored in memory) may be via the stacking ordinal (sequence number as read in) or via the ban-ID from the ban-file. If more than one file is read in, one must make sure that all IDs are unique or one must reference bananas by sequence number rather than ID-number. Non-unique IDs will generate a warning at read-in time but it is not a fatal error. Also, the ID numbers of bananas to be referenced by ID number must be limited to the range of 1 to 8000. Any out-of-range IDs will generate a warning at read-in time but, again, the error is not fatal.

Zeroing the in-memory bananas ----------

You may use the command banz to clear all in-memory bananas. Subsequently, a new set may be read in as described above.

Functions which test 2-D Gates ----------

After the appropriate ban-files have been read in, one or both of the following logical functions may be used by routine HISSUB to test X and Y parameters against individual bananas.

```
----------------------------------------
      LOGICAL FUNCTION BANTESTN(NG,IX,IY)
C
      INTEGER*4 NG,IX,IY
----------------------------------------
      LOGICAL FUNCTION BANTESTI(IG,IX,IY)
C
      INTEGER*4 IG,IX,IY
----------------------------------------
```

BANTESTN references bananas via sequence numbers NG and BANTESTI references bananas via ID-numbers IG. If the the IX,IY point is contained within the specified banana, the value of the function is .TRUE., otherwise it .FALSE. In NG or IG do not exist, .FALSE. is returned. There are no error messages.

(continued on next page)

U320.080 Discussion of Free-Form Gating (continued)

The following list summarizes the features and limitations of this type of free-form gating support.

(1)....Multiple ban-files may be read in. All bananas in each file are stored in memory.

(2)....A ban-ID directory is built as files are read in. If more than one banana has the same ID, only the last one read will be entered in the directory. A warning will be displayed at read-in time but the error is not fatal (you can always reference via sequence # via routine BANTESTN).

(3)....Up to 3000 bananas may be stored by the standard support routines.

(4)....Up to 8000 banana IDs can be accomodated by the standard routines. ID numbers must be in he range of 1 through 4000 in order to be entered into the ID-directory. If any IDs are out of range, a warning will be given at read-in time but the error is not fatal (you can always reference via sequence# via routine BANTESTN).

(5)....Up to 102400 banana channels (200 512-channel bananas) may be stored in memory by the default routines from scanulib.

(6)....Up to 512000 banana channels (1000 512-channel bananas) may be stored in memory by linking in the bansup1 support package (see below for how to do it).

(7)....Up to 1024000 banana channels (2000 512-channel bananas) may be stored in memory by linking in the bansup2 support package (see below for how to do it).

The make file shown below illustrates the linking of an executable containing the bansup1 banana support package. In this example, mydir and mysubdir are intended to represent the directory and sub-directory containing user-supplied routines.

```
--------------------------------------------------------------------------
DIRA= /usr/hhirf/
DIRB= /usr/users/mydir/mysubdir/
OBJS= $(DIRA)scanu.o $(DIRA)bansup1.o $(DIRB)scanusubs.o
LIBS= $(DIRA)scanulib.a $(DIRA)milib.a $(DIRA)jblibf1.a
      $(DIRA)jblibc1.a  $(DIRA)milib.a
scanu: $(OBJS) $(LIBS)
 f77 -O2 $(OBJS) $(LIBS) -o scanu
--------------------------------------------------------------------------
```

U320.090 Comments on 1-D Gating

At the present time there is no support package for 1-D gating - the user has to do everything - open files, read in files, etc., etc. I do anticipate that there will be some sort of 1-D gating support in the future - possibly based on the use of GAF-files (see 1987 handbook SEC# 345.040). If this is done, damm will be upgraded to support the interactive creation of such files.

U320.100 How to Create and Run Customized Scanu Programs

The make-files listed below (found in /usr/hhirf/scanu.make and in &/home/upak/scanu.make) can be used as template for creating your own customized make file.

For users of HHIRF DECstations

```
-------------------------------------------------------------------------
DIRA= /usr/hhirf/
DIRB= /usr/users/mydir/mysubdir/
OBJS= $(DIRA)scanu.o $(DIRB)scanusubs1.o $(DIRB)scanusubs2.o
LIBS= $(DIRA)scanulib.a $(DIRA)milib.a $(DIRA)jblibf1.a
      $(DIRA)jblibc1.a  $(DIRA)milib.a
scanu: $(OBJS) $(LIBS)
 f77 -O2 $(OBJS) $(LIBS) -o scanu
-------------------------------------------------------------------------
```

For SUNPAK users

```
-------------------------------------------------------------------------
DIRA= /home/upak/
DIRB= /home/mydir/mysubdir/
OBJS= $(DIRA)scanu.o $(DIRB)scanusubs1.o $(DIRB)scanusubs2.o
LIBS= $(DIRA)scanulib.a $(DIRA)milib.a $(DIRA)jblibf1.a
      $(DIRA)jblibc1.a  $(DIRA)milib.a
scanu: $(OBJS) $(LIBS)
 f77 -O2 $(OBJS) $(LIBS) -o scanu
-------------------------------------------------------------------------
```

Bold face entries in the make file examples above indicate places where you will probably need to make changes. In this example, all customizing routines are contained in two files from one directory, namely:

```
/usr/users/mydir/mysubdir/scanusubs1.o  and  ;For HHIRF DECstation users
/usr/users/mydir/mysubdir/scanusubs2.o
or
/home/mydir/mysubdir/scanusubs1.o       and  ;For SUNPAK users
/home/mydir/mysubdir/scanusubs2.o
```

Your routines may be located in more than one directory but I think you will see how to do it. Assuming the name of the make-file is scanu.make and the name of the executable is to be scanu, make it by typing:

```
make -f scanu.make scanu
```

Run it by typing:

```
scanu filnam
```

Where, filnam denotes the filename prefix of the his- & drr-files to be used or created.

U320.110 Example DRRSUB

```
C$PROG DRRSUB
      SUBROUTINE DRRSUB(IEXST)
C
C     ************************************************************************
C     Example DRRSUB which reserves space for:
C     30 256-channel 1-D histograms at 32-bits/channel and
C     3  256*256-channel (range 0,199 0,199) 2-D histograms at 32-bits/chan
C     ************************************************************************
C
      IMPLICIT INTEGER*4 (A-Z)
C
      CALL DRRMAKE
C
      DO 10 HID=1,30
C
      CALL HD1D(
     &          HID,    !HIST ID
     &          2,      !# HWDS/CHANNEL
     &          2048,   !RAW  PARAM LENGTH
     &          256,    !HIST PARAM LENGTH
     &          0,      !MIN  PARAM "RANGE" VALUE
     &          255,    !MAX  PARAM "RANGE" VALUE
     &      '1D HIS')   !TITLE
C
   10 CONTINUE
C
      DO 20 HID=31,33
      CALL HD2D(
     &          HID,    !HIST ID
     &          2,      !# HWDS/CHANNEL
     &          2048,   !RAW  PARAM X-LENGTH
     &          256,    !HIST PARAM X-LENGTH
     &          0,      !MIN  X-PAR "RANGE" VALUE
     &          199,    !MAX  X-PAR "RANGE" VALUE
     &          2048,   !RAWW PARAM Y-LENGTH
     &          256,    !HIST PARAM Y-LENGTH
     &          0,      !MIN  Y-PAR "RANGE" VALUE
     &          199,    !MAX  Y-PAR "RANGE" VALUE
     &     '2D HIST')   !TITLE (40 BYTES MAX)
C
   20 CONTINUE
C
      CALL ENDRR
      RETURN
      END
```

U320.120 Example HISSUB

```
C$PROG HISSUB
      SUBROUTINE HISSUB(IBUF,NB)
C
C     ************************************************************************
C     Example HISSUB
C     Processes full tape buffers IBUF (NB = number of words in IBUF).
C     Unpacks events into JBUF but only uses/tests/clears params 15,16,17)
C     Histograms all raw parameters at resolution specified by DRRSUB
C     with his-ID = parameter number.
C     Generates ungated   2-D histograms of 15,16 & 15,17 (HID=31,32)
C     Generates ban-gated 2-D histogram  of 15,17         (HID=33)
C     ************************************************************************
C
      IMPLICIT INTEGER*4 (A-Z)
C
      LOGICAL BANTESTI
C
      INTEGER*2 IBUF(*)
      INTEGER*4 JBUF(100)
      DATA ID,JBUF/0,100*-1/
C
      DO 100 I=1,NB
      IF(IBUF(I).GE.0) GO TO 50
C
      IF(IBUF(I).EQ.'FFFF'X) THEN
      J15=JBUF(15)
      J16=JBUF(16)
      J17=JBUF(17)
                             CALL COUNT1CC(31,J15,J16)
                             CALL COUNT1CC(32,J15,J17)
      IF(BANTESTI(1,J15,J17)) CALL COUNT1CC(33,J15,J17)
      IF(BANTESTI(2,J15,J17)) CALL COUNT1CC(33,J15,J17)
      ID=0
      JBUF(15)=-1
      JBUF(16)=-1
      JBUF(17)=-1
      GO TO 100
                            ENDIF
C
      ID=IAND(IBUF(I),'0FFF'X)-1
      GO TO 100
   50 ID=ID+1
      IF(ID.LE.0.OR.ID.GT.30) GO TO 100
      IF(IBUF(I).GT.2047)     GO TO 100
      IX=IBUF(I)
      JBUF(ID)=IX
C
      CALL COUNT1CC(ID,IX,0)
C
  100 CONTINUE
      RETURN
      END
```

U320.130 List of Commands

The standard scanu commands, which are also available in the form of run-time help are listed below. Note: all commands may be entered in lower case.

Commands Related Setup and Initialization .....

```
UCOM TEXT       Send TEXT to USERCMP
BAN  file.ban   Open and read in bananas from file.ban
BANZ            Zero (reset) in-memory banana list
SWAB            Turn byte-swap ON
SWOF            Turn byte-swap OFF (default)
SEBU            Single-event   buffers passed to HISSUB
MEBU            Multiple-event buffers passed to HISSUB (default)
RECL N          Set tape data record length to N-bytes (dflt=8192)
L001 NSKIP,NPPE Specify L001 input format (see 87 Handbook, SEC# 300.060)
L002            Specify L002 input format (default)
LON             Turns log-file output ON (default)
LOF             Turns log-file output OFF
MSG  TEXT       Display TEXT (44 bytes) on (VDT)
```

Commands Related to Tape Assignment, Control, etc. .....

```
TAPE rxxx  Assign tape unit  (rxxx = rmt0, rmt1, rst0, etc)
REW        Rewind tape-unit
CLOT       Close  tape-unit
CLUN       Close and unload tape-unit
BR   N     Backspace N-records on tape-unit
FR   N     Forward   N-records on tape-unit
BF   N     Backspace N-files   on tape-unit
FF   N     Forward   N-files   in tape-unit
FIND ID    Find HEADER # ID
```

Commands Related to Running (actual processing) .....

```
ZERO       Zero his-file & reset all POINTERS
ZBUC       Zero BUFFER-counter (record counter)
GO         START/CONT process (stops on EOF,EOM or ERROR)
GO   N     Process N-files (skip bad records)
GO   N,M   Process N-files or M-recs - 1st to occur
GOEN       START/CONT process - unloads tape and ends
GOEN N     START/CONT process - unloads tape and ends
GOEN N,M   Process N-files or M-recs - 1st to occur
HUP        Updates his-file but does not terminate
CTRL/C     Interrupts tape processing and waits for next typed CMD
END        END gracefully - update his-file & END
KILL       Abort program  - no update!!
```

Commands Related Command File Operation .....

```
CMDF fil   Assign fil.cmd as cmd-file (not read yet)
CCMD       Continue reading instructions from fil.cmd
CLCM       Continue with last CMD from fil.cmd (backspaces)
CCON       Continue (reading instructions from CON:)
```