HDS.TXT

I have been trying to understand how to use the HDS facility to construct a structured data file which would be the equivalent of the files currently used in SPECX. The following section summarizes my understanding of the HDS calls necessary to create a header structure, and to place data in the 'primitive' objects.

a. To start running HDS
```
CALL HDS_START(STATUS)
```

b. To open a new HDS container file
```
CALL HDS_NEW('container_file_name', 'top-level-object_name',
             'object_type', #-dimensions, array_of_dimensions,
             'locator_of_top-level-object', STATUS)
```
or to open an already existing container file
```
CALL HDS_OPEN(.....)
```

c. To add a component to a HDS structure
- (if the structure being added to is an array element)
```
CALL DAT_CELL('locator_of_whole_structure',#_of_subscripts,
              array_of_subscripts,'locator_of_cell',
              STATUS)
```
- (if structure being created is an array)
```
CALL HDS_TUNE(...see manual...)
```
- (for all objects)
```
CALL DAT_NEW('locator' or 'locator_of_cell',
             'new_component_object_name', 'object_type',
             #_of_dimensions, array_of_dimensions,STATUS)
CALL DAT_FIND('locator' or 'locator_of_cell',
              'new_component_object_name',
              'locator_of_new_object', STATUS)
```

d. To put a value in a data cell
```
CALL DAT_PUT('locator_of_new_object',...,&c)
```

This appears fairly horrifying to me. There are a minimum of three, and up to five, subroutine calls required to create a component of a structure and place a value in it. Of course, if the hierarchical part of HDS is used, so that intermediate level structures are created, then even more calls are required. For example, consider the header and data structure defined by the INCLUDE file for the spectral line data reduction program SPECX.

```
C                       STACKCOMM.SAV
C                       -------------

C   This is the include file for the stack,flags and variables
C   common blocks in SPECX.
C   NOTE: TSYS,TMAX,VMAX,TOTINT and XWHM are for information only. These
C         variables are not used by the system, but can be displayed under
C         control of the IOK flag.
C         IST,IEND and LSCAN are inserted by the system when a scan is filed
C         in the standard data file by the routine SCNFIL. These cannot be
C         forced by the owner of the file.

      PARAMETER NRX=4

      REAL*4    TSYS,TMAX,VMAX,TOTINT,XWHM,CUTDK,VLSR,ZD
      INTEGER*4 INTT,JFCEN,JFINC
```

```
      INTEGER*2 ITREC,ITSKY,ITTEL,NPTS,IRA,IDEC,LSCAN,IMODE,
     &          NQUAD,IST,IEND,ICALZD,LSRFLG,IQCEN,IOK,IQUAD,
     &          IDRA,IDDEC
      CHARACTER ITITLE*26,IDATE*9,ITIME*8
      BYTE      IUTFLG

      COMMON /STACK/ TSYS(NRX),   ! System noise temp. in each quadrant/filter-bank
     &               TMAX(NRX),   ! Maximum temperature   "
     &               VMAX(NRX),   ! Velocity of maximum   "
     &               TOTINT(NRX), ! Integrated intensity  "
     &               XWHM(NRX),   ! Line width at half maximum "
     &               CUTDK(NRX),  ! Computer units per Kelvin  "
     &               VLSR,        ! Velocity of source w.r.t. local standard of rest.
     &               ZD,          ! Zenith angle at start of observation
     &               JFCEN(NRX),  ! Centre frequency of each quadrant ( kHz )
     &               JFINC(NRX),  ! Channel spacing   "  ( Hz )
     &               INTT,        ! Integration time at any frequency ( ms )
     &               ITREC(NRX),  ! Receiver noise temperature ( K )
     &               ITSKY(NRX),  ! Apparent sky temperature ( K )
     &               ITTEL(NRX),  ! Apparent constant telescope temperature ( K )
     &               NPTS(NRX),   ! Number of data points per quadrant.
     &               IRA(4),      ! R.A. of observation ( hh,mm,ss,ss/100 )
     &               IDEC(4),     ! Dec. of observation ( oo, ', ", "/100 )
     &               LSCAN,       ! Sequence number of observation
     &               IMODE,       ! Mode of observation
     &               NQUAD,       ! Number of quadrants/filter-banks
     &               IST,         ! Block number of first data block
     &               IEND,        ! Block number of last data block
     &               ICALZD,      ! Zenith angle of receiver/sky calibration
     &               LSRFLG,      ! Flag: Set = 1 if JFCEN corrected for lsr.
     &               IQCEN,       ! Quadrant to which LSR correction applied.
     &               IOK,         ! = 1 if variables TSYS - XWHM defined.
     &               IQUAD,       ! Quadrant to which these apply if only one
     &               IDRA,        ! Offset if any from IRA,IDEC in R.A. ( arcsec )
     &               IDDEC,       !      "    in Dec.  ( arcsec )
     &               ITITLE,      ! Character title of observation.
     &               IDATE,       ! Character date of observation dd-MON-yy
     &               IUTFLG,      ! Time is U.T. if = 1
     &               ITIME,       ! Character time of start of observation hh:mm:ss
     &               DATA(1024),  ! Data buffer for current spectrum
     &               STK(7616),   ! Remainder of stack space.
     &               STORE(1088,9) ! Storage for gain and STORE arrays.

      INTEGER*2   HEADER(128)
      EQUIVALENCE (HEADER(1),TSYS(1))
```

Now clearly there would be advantages to being able to use data structures to make more sense of this. On the other hand, this is itself fairly structured, and the use of INCLUDE files makes it possible to describe the data structure to any subroutine without difficulty.

In this particular program the 'stack' is 'filled'; that is, it contains the actual header and data for each entry, rather than just a pointer. This saves having to use scratch files on disc, gives faster program response, and allows a high level of interactive processing, with the spectrum (and header) being processed very much as dynamical entities, by which I mean that they actually change as you process them. This has two implications for HDS. First, writing to disc: When the spectra have been processed in SPECX they can be written to either the same or a new data file, using (essentially) two random access WRITE statements - one for the Header array (128 I*2 words) and one for

the data array (up to 1024 (in this implementation) R*4 words).

The header (for each spectrum) contains 75 different entities, including array elements as separate entities. Part of this multiplicity is because of the desire to include separate spectra taken simultaneously as separate 'receivers' under blanket cover of one ID, DATE, TIME etc. Examples of this would be when taking data from a dual polarization receiver, from several 'quadrants' of the autocorrelator backend, or from a dual frequency receiver.

A possible HDS structure for the data file is shown in Figure 1. There are a couple of minor inconsistencies between this structure and the INCLUDE file above, which are noted on the figure, but all in all it's pretty much the same thing. By my current understanding of HDS this would require ABOUT 250 SEPARATE SUBROUTINE CALLS to write to a file, where it used to take 2 WRITE statements. (This is based on an average of 3 calls per primitive object, plus a few to create the intermediate-level objects such as 'POSITION_OFFSET' - see the HDS-11 manual.) To be fair, I should also point out that the calls for each 'receiver' are identical, and so would normally go inside a separate routine, but even so it would be necessary to write code to make about 80 HDS calls.

The second problem caused by this sort of program architecture is that it is impossible to use the MAP feature of HDS in any sensible way. In order to have access to a MAPped variable it is necessary to pass the variable to a lower level subroutine using the %VAL() construct to pass the (mapped) address. Now this implies that sub-programs using the data have to be called by the routine which sets up the map. This type of philosophy is quite foreign to me - I prefer to get the data as one operation, and then think about what to do with it. It is arguable whether or not it is right for the data structure to dictate the program structure to the extent that HDS-11 does, but I feel it is absolutely wrong that the data-structure-handler should.

Now whether this is the fault of HDS-11 itself or of the documentation I don't know. However it is rather frightening. Things I don't understand about HDS-11 are many, and include the following:

- I don't see any way to copy the actual structure of the data. What I would like to do is to generate the structure (for a standard type of file) once only, and use it again and again, but apparently I can't. It is possible that DAT_CLONE is designed for this purpose, but this is not what the manual says - it says it clones the locator, which as far as I understand gives different access to the SAME item, not the same access to a DIFFERENT item. Again, DAT_MOVE may possibly be designed for this purpose, but all I can see is that it actually moves the whole structure, and doesn't replicate it. It would help if all elements of an array were identical! However there is nothing to say that they are.

- What are the penalties associated with extending an array using DAT_ALTER? Can I do this with impunity, or is it wise to declare the array to be maximum length when the (array) object is created?

- How does DAT_INDEX work? It says that it enables you to

"index into a structure's component list". Quite what this means I'm not sure. On p2.6 of the manual we are advised to avoid placing any importance on the position in the component list, as this is just an implementation strategy and not a design feature. It rather appears that DAT_INDEX performs a function which will not be supported.

My conclusion at this stage is that it is not worth the bother of trying to use HDS-11. It dictates absolutely the structures of programs which use it, and requires significant extra programming effort. It might also be predicted that it will increase access times to small data sets by an inordinate amount, although the whole process looks so tedious that I haven't bothered trying it. On the other hand the idea of structured data IS appealing. The HDS-11 facility as it stands also DOES offer a solution to the problem of uncalibrated, mixed, data, but only at the cost of the complexity in the data-writer noted above. VAX/VMS FORTRAN V4.0 is apparently going to offer data structures as a language extension. This is supposed to arrive around Christmas, and I would like to wait until I see the documentation for this facility before making a final decision on data structures for the MT programs.

Rachael Padman
10-DEC-84.