# VLBA Data Structures

T.J. Pearson,
Caltech

December 11, 1982

This note is a contribution to discussion in the Computer Group meetings about data base formats for the VLBA (visibility data and images). We agreed that VLBA data should be treated in a similar way to VLA data in the AIPS package. The data structures described below are suggestions as to how AIPS might be improved. The suggestions are derived from my discussions with Keith Shortridge, who has recently begun a major overhaul of the data reduction system used by Caltech optical astronomers. In fact much of the following is extracted bodily from a memorandum written by Shortridge for another purpose. Where the suggestions are concrete, they are based closely on STARLINK practice (or intended practice). (STARLINK is the British network of VAXes, used for astronomical image processing at several different observatories and university departments.)

How should VLBA data be be stored on disk? i.e., what data format should be used? This is the wrong question to ask: the correct question being how do we hide the way the data is actually stored on disk, by providing access routines that are sufficiently comprehensive and efficient that nobody need know the actual data format. This allows us the luxury of having a complex format, probably using some form of linked-list structure, which will allow us to add new data to the format in a way that is transparent to the higher level programs. This may not appear to be a problem when one thinks only of the raw data, but the operations associated with data reduction can add many forms of additional data to the simple set of correlation coefficients that comes from the correlator. It is not adequate to just leave some amount of unused space in the file and hope that this will be enough.

Thus our requirement is for data structures with a dynamic format that allows parts of the data to be missing and allows new data to be added at will. The price we shall have to pay for this is that the actual data format on disk will be too intricate to allow uncontrolled access to it (either by users writing their own programs, or by systems experts who get tempted to take short cuts!). So there will have to be a set of standard routines which everyone can use to access these data structures.

## Goals

(1) Simplicity. All the data handling we ever want to do should be available through a small set of subroutines accessing the data structure.

(2) Real expandability. In the sense that any new calibration or other information fits into the scheme without any modification.

(3) Self documenting. The data format should be self-documenting at the software level. An enquiry program can be written which will describe exactly what is in any file produced by these routines, and programs can be written to make use of this. (None of the "if it's set to zero it probably isn't really an hour angle" stuff, which turns up all to often in many formats.)

(4) Use ordinary files of the host operating system. Each logically separate structure is a separate file and all the usual host operations (backup, copy, protect etc) can be used.

(5) Program driven. Probably the most important point of all. The programs should be in control, and not constrained by the data formats that have been chosen.

(6) Efficiency. Data should not occupy more space than necessary. Access to data should be rapid when this matters, for example when accessing a bulky image; efficiency can be sacrificed to the other goals when accessing ancillary parameters.

## The STARLINK scheme

All data are treated using the data structures, and an open-ended, self-documenting data format is used. A set of FORTRAN-callable subroutines are available to access any part of the data, without the higher-level routine having to know anything about the actual data format. For example: suppose the data you are working on is called SPECTRUM1. It has associated with it all sorts of things you don't even necessarily know about, but as a minimum, something like:

SPECTRUM1.XDATA contains the wavelengths of each element of the data,
SPECTRUM1.YDATA contains the flux at each wavelength,
SPECTRUM1.XUNITS is 'Ångstrom',
SPECTRUM1.YUNITS is 'erg/(cm²sÅ)',
SPECTRUM1.XLABEL is 'Wavelength',
SPECTRUM1.YLABEL is 'Flux Density'.

You can get any of these by calling a subroutine, giving it the name of the field you want. You can easily write a program that will display not only this spectrum, but also a pretty large variety of other data structures, given the basic name of the structure, and the knowledge that it contains fields called XDATA, YDATA, XUNITS, YUNITS, XLABEL, and YLABEL. Routines are available to find out the details of the structure. If this looks a little like FITS, it is: it is easy to translate an arbitrary FITS file into a data structure of this sort.

There are two parts to the design of such a data structure system: the subroutine interface, and the actual record structure of the files. As an example, the basic subroutines needed include the following:

1. Associate a structure name with a file: "I want to use file 2DFPIX.001 and I want to call the structure in it INPUT".

2. Create an entry for a variable in a structure: "Please make space in the structure for a floating-point element called OUTPUT.X.DATA".

3. Write a variable: "Please copy 17 elements from array X into the structure element OUTPUT.CALIB".

4. Read a variable: "Please copy the structure element INPUT.OBS.TELESCOPE into character variable NEYE".

5. Map a variable (for input and output): this involves mapping part of the data structure into virtual memory and returning a pointer to the calling program. It is more efficient than using read or write for large elements.

6. Unmap a variable (release the virtual memory).

7. Enquiry routines: "Is there an element called OUTPUT.HENRY?", "What is the format of entry OUTPUT.HENRY?" (answer: floating-point, character, array dimension 400, etc).

8. Copy a data structure.

9. Close the file associated with the data structure.

The read and write routines translate the format of the data into that required by the program, if necessary; so that for example a general contouring program which expects images to be 32-bit floating point numbers can work on a 16-bit integer image. The repertoire of data types includes integer, character, floating-point, arrays, etc. Note that elements can themselves be structures, creating a hierarchical database.

The disk file format is the most critical part of the design as it cannot be changed later. Each element will have to have associated with it at least the following:

Name,
Up pointer to higher-level structure,
Down pointer to next level (only for structures),
Forward pointer to next on same level,
Back pointer to last on same level,
Type code (byte, integer, etc),
Location of data in file,
Number of dimensions,
Size in each dimension.