Exchange of Model Data:
A Proposal

Peter Teuben, University of Maryland

`teuben@astro.umd.edu`

*** DRAFT 5 July 1994 ***

ABSTRACT

We propose to define a common data format with which particle data (N-body, SPH, etc.) can be exchanged, and archived on national datacenters. Our prime concern in this is archival integrity.

The FITS format has recently been extended with "*binary tables*", which we would like to propose as a base for the standard. FITS binary tables are a flexible and efficient means of transmitting a wide variety of data structures. Table rows may be a mixture of a number of numerical, logical and character data entries, called fields. In addition, each field is allowed to be a single dimensioned array (with an obvious proposed way to extend to multiple dimensions). Numeric data are kept in floating point IEEE-754 and integer twos complement machine independant formats.

1 Introduction

The difficulties that astronomers have exchanging data have been reduced a lot since the conception of FITS (Wells et al. 1981). Various improvements have been made to the data format, to facilitate more modern datastructures (those beyond the classical "*image*"). These days, observers can freely exchange complex images and tables in FITS format (cf. IRAF, MIDAS, AIPS), with new FITS extensions appearing on the horizon. However there is one golden rule to fits: *once fits, always fits*, i.e. a FITS file will never become obsolete.

For theoreticians the situation is a far cry from this. We've had to describe the data (often nothing more than a fairly simple table!) to our collegues in order to be understood at the other end. For data in binary form we depend on either having the same machine, or understanding each others machines fairly well. Ever converted double precision floating point numbers from a VAX to a SUN?

An attempt to get ourselves organized done by Barnes et al. (1988) in the framework of NEMO[1] package were only partially successfull. A fairly simple ASCII format was used to transfer data between different machines, though all data were kept in a hierarchical data format, which was machine independant to a large degree.

It has been argued that there are (at least) three different applications for data formats: **access, interchange, and archival storage**. It is not at all obvious that any single format can meet the needs of all 3 simultaneously, even less obvious that any single format currently does meet the needs of all 3.

In general, **access formats** emphasize access efficiency. **Interchange formats** need storage efficiency and standardized numeric representations. **Archival formats** need to be self-documenting, externally documented and simple. What we need to define here, is a standard for interchange, and it would be nice if that could also serve as an archive format. There is little hope, and need, to aim for a standard access format.

2 Requirements

What would be the requirements for such an interchange/archive format?

- The data must be stored in a machine independant format, and to some extend be self-describing.

[1] An N-body/Dynamics Toolbox; *The rumors of my death have been greatly exaggerated...*

- The dataformat must be flexible and extendible in order to store a wide variety of data structures.
- Data must be stored efficiently. This eliminates virtually any ASCII format, particularly if precision is important.

3 Formats

We will quickly go over a few possible formats, although it should be obvious that we argue FITS should be used. We briefly describe HDF, (net)CDF and HDS as other potential candidates.

3.1 HDF

Hierarchical Data Format (HDF) is a multi-object file format for the transfer of graphical and floating point data between machines. The design allows self-definition of data content, and easy extensibility for future enhancements or compatibility with other data formats. HDF has been created by NCSA (National Center for Supercomputer Applications, Urbana, Illinois), in order to solve their problem of interchanging data between their Cray, Alliant, Sun, Iris, Macintosh and IBM PC.

Both Fortran and C calling interfaces are available, and some utilities exists to import and export data to and from HDF format. It has mainly been used to store objects like images, and associated color palettes and annotations.

At this writing the current version of the NCSA software is `HDF3.2r4` and

3.2 CDF

Common Data Format (CDF) is a conceptual data abstraction for storing multi-dimensional data sets. The basic component of CDF is a software programming interface that is a device independent view of the CDF data model. The application developer is insulated from the actual physical file format for reasons of conceptual simplicity, device independence, and future expandability. Version 2 of CDF has been rewritten to be portable across a wide variety of platforms. CDF files created on any given platform can be transported to any other platform on to which CDF is ported and used with any CDF tools or layered applications. A more detailed introduction to CDF can be found in the CDF User's Guide (see `ftp` list below where to obtain this).

The CDF library and toolkit programs are written in C. A C compiler (ANSI standard preferred) is necessary to build the CDF distribution. The CDF distribution also contains several FORTRAN source files which support the FORTRAN and OBSOLETE interfaces.

At this writing the current version stored on NSSDCA is `CDF2.3`.

3.3 HDS

The Hierarchical Data Structure (HDS) is used by the Starlink project.

3.4 FITS

FITS format (see also Appendix A) is an accepted and widely known astronomical standard. It can store data in binary as well as ascii format, and is mostly known for archiving (and data interchange) images and tables. Recently binary tables have been added, which greatly improves the efficiency. These binary tables are an excellent way to store N-body data.

For a variety of reasons this FITS format has become a viable option:

- The BINTABLE standard FITS extension has been defined (although all local FITS committees have ratified them, it still needs to be ratified by the IAU in The Hague)
- The FITSIO library (Pence, 1992) is available for a wide number of machines, and is written in FORTRAN. A C calling interface is now available.
- NSSDC (through NASA, operating out of Goddard) is making a large collection of data available to the community. It seems quite appropriate to store a number of standard simulation results within that framework, and make it available to the public.

We would propose to use FITS binary tables as standard to store N-body results for export and archive purposes.

4 Problems

Some random thoughts:

- each particle may have it's own time associated with it (common in variable timestep integrators, e.g. in globular cluster simulations). Although we could allow the `TIME` parameter to be an array parameter, as well as global, we could also agree on the "snapshot" concept, i.e. all particles be taken (extrapolated by online software if need be) to the same time. If this format is to be used to exchange data, and your collegue be able to continue the integration, it often means associated tables are needed to describe *such situations*. For example, a data structure describing hierarchical double and triple stars could be stored in an accompanying table, as sketched in Table 1.

- Similarly to agreements on the naming conventions of keywords that give meaning to the NBODY BINTABLE, for each newly-named extension we need to define the standard. Since FITS data can be made self-described to a large extent, it is also a good idea to add these as comments to the actual BINTABLE.

  There is however a MINOR point (so minor that it does NOT have to be counted as detrimental to the usage of FITS for archiving), which is the fact that the semantics of the usage of a particular FITS file are not necessarily (usually not) subject to the formal standardization described above. They will be at best published in "internal reports".

  This is particularly true with the binary table stuff (after all an image is an image is an image z=f(x,y), and once you've filled the mandatory and recommended keywords you know what is on x,y and z). Binary tables allow lots more of flexibility, you'd always know what are the names of the various columns ... but may not necessarily know what those names mean ... unless you look up in some project related documentation.

- datasets with different kinds of particles, which have different attributes. Storing them in one table would waste space, and we would have to tag the type of the particle in another column. For example SPH particles may have a temperature and density, which pure N-body particles don't have. The "variable length array" construct, proposed within the BINTABLE proposal could solve this problem, but we could also decide to store such "irregular" data in separate HDU's, and agree that HDUs with the same time must be co-added. The latter solution is a source of all kinds of other problems (easily loosing data being only one of them).

- In some simulations new particles may be created, and that particles may change their nature (e.g. gas particles may become star particles, which may become dark particles) would argue for a somewhat wasteful format where all the sorts of particles are stored in one table, and some kind of method is used to categorize the particle. For example, a particle with NaN (or BLANK) density is a star or dark particle. Obviously good rules need to be given for this, otherwise the format become anarchistic in nature and interpretation ambiguous. Coordination is needed here.

- Coordinate systems: if the coordinate system is specified by a single keyword in the header (e.g. CARTESIAN, POLAR etc.), we also need to describe what the NDIM axes in this coordinate system respresent. X,Y,Z or R,THETA,PHI, and in which order. More complicated are parameterized coordinate systems, such as the "perfect" ellipsoidal coordinate systems. In addition to tagging them using `COORDSYS`, a convention is needed to allow parameters be set. Some experience in astronomical coordinate systems exist, and are being working on in the FITS coordinate system proposal (still to be ratified).

- Units: do we follow the OGIP conventions. They seem to make the most sense. The current FITS practice is still too ambiguous (partially due to an ambiguous definition in the current "standard" papers).

- authorship and literature reference should also be standardized. FITS has standard keywords `AUTHOR REFERENC` for this.

- Archiving: to aid archiving, a number of descriptors should be present. This can be done through standard keywords, that an archiver can extract and store for quick online access, since it likely that the data is going to be large, and possibly stored on media with low-speed access.

5 Proposed Standard Keywords and Field Names

As we've seen, it seem we must allow a few Field names to be all the same, and present as a standard FITS keyword in the header also. The following table lists all Field names and standard keyword:

6 Example

6.1 Example Binary Table Header

First we will list the header output of the first HDU and second HDU

The following is an example of a binary table header which has 6 columns using a number of different data types and dimensions. Columns labeled "POS", "VEL", and "ACC" are arrays of dimension 2. The column labeled "CODE"

Primary HDU (no data, only header)
NBODY Standard BINTABLE Extension HDU (main particle data)
NBODY-XXX Standard BINTABLE Extension HDU (optional associated table)
...
...

Table 1: FITS file components

is a character string of length 4. The nonstandard keywords “TIME”, “COORDSYS”, etc. also appear at the end of the header, and identify global parameters of the dataset. Although copied by any FITS reading program, they meaning is only understood in the context of an “`NBODY BINTABLE`”. Note the units are not SI, which is what the IAU recommends to store data in FITS format. The first two lines of numbers are only present to show card columns and are not part of the table header.

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890

SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                    0 / length of data axis   1
NAXIS2  =                    0 / length of data axis   2
EXTEND  =                    T / FITS dataset may contain extensions
DATE    = '19/08/93'           / FITS file creation date (dd/mm/yy)
COMMENT   This file was written by a sample program.
VERSION =             3.31E+00 / FITSIO version
COMMENT   It was merely produced to show how to write an NBODY BINTABLE
COMMENT   Sample NEMO history:
HISTORY   mkplummer t1 10
HISTORY   hackforce t1 t2
HISTORY   snapdens  t2 t3
HISTORY   snapmask  t3 t4 3:9
HISTORY   snapprint t4 m,x,y,z,vx,vy,vz,phi,ax,ay,az,aux,key > nbody.tab
END
```

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
--------------------------------------------------------------------------------

XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   56 / width of table in bytes
NAXIS2  =                    7 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    7 / number of fields in each row
TTYPE1  = 'MASS    '           / label for field   1
TFORM1  = '1E      '           / data format of the field: 4-byte REAL
TTYPE2  = 'POS     '           / label for field   2
TFORM2  = '3E      '           / data format of the field: 4-byte REAL
TTYPE3  = 'VEL     '           / label for field   3
TFORM3  = '3E      '           / data format of the field: 4-byte REAL
TTYPE4  = 'PHI     '           / label for field   4
TFORM4  = '1E      '           / data format of the field: 4-byte REAL
TTYPE5  = 'ACC     '           / label for field   5
TFORM5  = '3E      '           / data format of the field: 4-byte REAL
TTYPE6  = 'AUX     '           / label for field   6
TFORM6  = '1D      '           / data format of the field: DOUBLE PRECISION
TTYPE7  = 'KEY     '           / label for field   7
TFORM7  = '1J      '           / data format of the field: 4-byte INTEGER
EXTNAME = 'NBODY   '           / name of this binary table extension
DATE    = '19/08/93'           / FITS file creation date (dd/mm/yy)
EXTVER  =                    1 / NBODY Version
NDIM    =                    3 / Space dimensionality
TIME    =       0.000000000E+00 / Snapshot time
GRAVCON =              1.0E+00 / Gravitational constant
UNITS   = 'VIRIAL  '           / Virial units
COORDSYS= 'CARTES  '           / Coordinate system
END
```

6.2 Example Program

The following program created the fits file of which we just listed the ASCII headers. The program is fully self-contained (although you need the HEASARC *fitsio* library. Linking in UNIX can normally be done as follows:

```
% f77 -g nbody.for -o nbody -lfitsio
```

```
      PROGRAM nbody
C
C  This sample program writes an NBODY binary table extension using
C  the FITSIO library  You should
C  be able to compile as follows:
C
C          f77 -O -o nbody nbody.for -lfitsio
C
C  (FITSIO: William Pence, HEASARC/NASA, V3.31 - but tested through
C   V3.415 - june 1994)
C
C  The main program is designed as to hide most of the FITSIO specific
C  details, and merely show where the astronomy isdefined
C
C  The input table is read from standard input.  Within NEMO one
C  can make a sample table as follows:
C
C    mkplummer tmp1 10           # create mass,pos,vel
C    hackforce tmp1 tmp2         # add phi and acc
C    snapdens  tmp2 tmp3         # add aux
C    snapmask  tmp3 tmp4 3:9     # add key
C    snapprint tmp4 m,x,y,z,vx,vy,vz,phi,ax,ay,az,aux,key > nbody.tab
C
C  Conversion from this simple table to FITS is as follows:
C
C    nbody < nbody.tab
C
C  The output file, named 'nbody.fits', created by 'nbody' must not exist yet.
C
c       INTEGER MAXBODY
c       PARAMETER (MAXBODY=1000)
      include 'maxbody.inc'

      REAL    mass(MAXBODY), pos(3,MAXBODY), vel(3,MAXBODY),
     *        phi(MAXBODY), acc(3,MAXBODY)
      DOUBLE PRECISION aux(MAXBODY)
      INTEGER idx(MAXBODY), nbody, lun
      REAL    time

C -- get the data from a table

      nbody = MAXBODY
      CALL tab_get(nbody,mass,pos,vel,phi,acc,aux,idx)

C -- open up file and initialize

      CALL nbf_open(lun,'nbody.fits')

C -- write header and data seperately

      time = 0.0

      CALL nbf_whead(lun, nbody, time)
      CALL nbf_wdata(lun, nbody, mass, pos, vel, phi, acc, aux, idx)
```

```
C -- close up the shop

      CALL nbf_close(lun)

      END
C***********************************************************************
      SUBROUTINE tab_get(nbody,mass,pos,vel,phi,acc,aux,idx)
      INTEGER nbody
      REAL mass(*), pos(3,*), vel(3,*), phi(*), acc(3,*)
      DOUBLE PRECISION aux(*)
      INTEGER idx(*)
C
      INTEGER i, maxbody
C
C -- read from stdin until no more
C
      maxbody = nbody
      i = 1
 100  CONTINUE
         READ(5,*,ERR=200,END=200) mass(i),
     *                      pos(1,i),pos(2,i),pos(3,i),
     *                      vel(1,i),vel(2,i),vel(3,i),phi(i),
     *                      acc(1,i),acc(2,i),acc(3,i),aux(i),idx(i)
         i=i+1
         IF (i.LE.maxbody) GOTO 100
         TYPE *,'Warning: bodies buffers exhausted'
 200  CONTINUE
      nbody = i - 1
      TYPE *,'Read ',nbody,' particles from input'

      END

C***********************************************************************

      SUBROUTINE nbf_open(lun,fname)
      INTEGER lun
      CHARACTER fname*(*)
C-
C  Write the primary HDU - only needed once
C-

      INTEGER status, bitpix, naxis, naxes(2), pcount, gcount
      LOGICAL simple, extend, exist
      REAL version
C
C       sanity checks
C

      CALL ftvers(version)
      TYPE *,'Using FITSIO Version ',version
      INQUIRE(FILE=fname,EXIST=exist)
      IF (exist) TYPE *,'### Warning: output file exists.'
C
C       open the new FITS file
C
```

```
      status=0
      lun=21
      CALL ftinit(lun,fname,2880,status)
C
C       write a dummy primary header before the BINTABLE extension
C       this will only be a header, there will be no data
C       FITS requires this to be present. We're only wasting 2880 bytes
C
      simple=.TRUE.
      bitpix=32
      naxis=2
      naxes(1)=0
      naxes(2)=0
      pcount=0
      gcount=0
      extend=.TRUE.
C       write the required primary array keywords
      CALL ftphpr(lun,simple,bitpix,naxis,naxes,pcount,gcount,
     *           extend,status)
C       add today's date and some comments about the origin
      CALL ftpdat(lun,status)
      CALL ftpcom(lun,
     *  'This file was written by a sample program.',
     *  status)
      CALL ftpkye(lun,'VERSION',version,3,'FITSIO version',status)
      CALL ftpcom(lun,
     *  'It was merely produced to show how to write an NBODY BINTABLE',
     *  status)
      CALL ftpcom(lun,'Sample NEMO history: ',status)
      CALL ftphis(lun,'mkplummer t1 10',status)
      CALL ftphis(lun,'hackforce t1 t2',status)
      CALL ftphis(lun,'snapdens  t2 t3',status)
      CALL ftphis(lun,'snapmask  t3 t4 3:9',status)
      CALL ftphis(lun,
     * 'snapprint t4 m,x,y,z,vx,vy,vz,phi,ax,ay,az,aux,key > nbody.tab',
     * status)

C       define the primary array structure
      CALL ftpdef(lun,bitpix,naxis,naxes,pcount,gcount,status)

C       call error checker
      CALL nbf_error(status)

      END

C***********************************************************************

      SUBROUTINE nbf_whead(lun, nbody, time)
      INTEGER lun, nbody
      REAL time
C-
      INTEGER MAXCOL
      PARAMETER (MAXCOL=16)
      INTEGER status
      INTEGER tfields,vardat,extver
```

```
      CHARACTER*8 tform(MAXCOL),ttype(MAXCOL),tunit(MAXCOL),extnam

C
C       now create a binary table extension
C

C       create a new empty extension

      CALL ftcrhd(lun,status)

C       define some global properties for this table
C
      extnam='NBODY'
      extver=1
      vardat=0

C       define the columns (at this point you need to know how many, and
C       which columns to write. We've choosen not to to clutter the subroutine
C       argument list with this, but to make this routine general it
C       would be needed.
C Note this in this case we didn't define tunit(), since UNITS='VIRIAL'
C does it all.

      ttype(1)='MASS'
      tform(1)='1E'
      ttype(2)='POS'
      tform(2)='3E'
      ttype(3)='VEL'
      tform(3)='3E'
      ttype(4)='PHI'
      tform(4)='1E'
      ttype(5)='ACC'
      tform(5)='3E'
      ttype(6)='AUX'
      tform(6)='1D'
      ttype(7)='KEY'
      tform(7)='1J'
      tfields=7

C       write the required keywords to the binary table extension
      CALL ftphbn(lun,nbody,tfields,ttype,tform,tunit,extnam,
     *              vardat,status)
      CALL ftpdat(lun,status)
      CALL ftpkyj(lun,'EXTVER',extver,
     *     'NBODY Version (required)',status)
      CALL ftpkyj(lun,'NDIM',3,
     *     'Space dimensionality (required)',status)
      CALL ftpkye(lun,'TIME',time,9,
     *     'Snapshot time (optional [0])',status)
      CALL ftpkye(lun,'GRAVCON',1.0,1,
     *     'Gravitational constant (optional [1])',status)
      CALL ftpkys(lun,'UNITS','VIRIAL',
     *     'Unit Type (optional [VIRIAL])',status)
      CALL ftpkys(lun,'COORDSYS','CARTES',
     *     'Coordinate system (optional [CARTES])',status)
```

```fortran
C       initialize the FITSIO parameters defining the structure of the table

      CALL ftbdef(lun,tfields,tform,vardat,nbody,status)

C       call error checker
      CALL nbf_error(status)

      END

C***********************************************************************

      SUBROUTINE nbf_wdata(lun,nbody,mass,pos,vel,phi,acc,aux,idx)
      INTEGER lun, nbody
      REAL mass(*), pos(3,*), vel(3,*), phi(*), acc(3,*)
      DOUBLE PRECISION aux(*)
      INTEGER idx(*)
C-
      INTEGER status

      CALL ftpcle(lun,1,1,1,nbody,mass,status)
      CALL ftpcle(lun,2,1,1,nbody*3,pos,status)
      CALL ftpcle(lun,3,1,1,nbody*3,vel,status)
      CALL ftpcle(lun,4,1,1,nbody,phi,status)
      CALL ftpcle(lun,5,1,1,nbody*3,acc,status)
      CALL ftpcld(lun,6,1,1,nbody,aux,status)
      CALL ftpclj(lun,7,1,1,nbody,idx,status)

C       call error checker
      CALL nbf_error(status)

      END

C***********************************************************************

      SUBROUTINE nbf_close(lun)
      INTEGER lun
C-
      INTEGER status

C       close the table
      CALL ftclos(lun,status)

C       call error checker
      CALL nbf_error(status)

      TYPE *,'Program appears to have run successfully'
      END
C***********************************************************************

      SUBROUTINE nbf_error(status)
      INTEGER status
c
      CHARACTER errtxt*40
```

```
      IF (status .GT. 0)THEN
         CALL ftgerr(status,errtxt)
         TYPE *,'*** ERROR - program did not run successfully ***'
         TYPE *,'status =',status,': ',errtxt
         STOP
      END IF

      END
C***********************************************************************
```

7 References

1. Barnes, J. E. et al. (1988), *BAAS* vol 20, no 2, p706.

2. Cotton, W.B., and Tody, D. 1992, "Binary Table Extensions to FITS: A Proposal".

3. Greisen E. W. and Harten R. H., 1981, "An Extension of FITS for Small Arrays of Data", *Astron. Astrophys. Suppl*, vol. 44, pp 371 - 374.

4. Grosbøl, P., Harten, R. H., Greisen, E. W., and Wells, D. C. 1988, "Generalized Extension and Blocking Factors for FITS", *Astron. Astrophys. Suppl.*, vol. 73, pp 359-364.

5. Harten, Grosbøl, Greisen and Wells 1988, "The FITS tables Extension", *Astron. Astrophys. Suppl.*, vol. 73, pp 365-372.

6. Wells, D. C., Greisen, E. W., and Harten R. H. 1981, "FITS: A Flexible Image Transport System", *Astron. Astrophys. Suppl*, vol. 44, pp 363 - 370.

7. NCSA "HDF Calling Interfaces and Utilities", Version 3.1 - July 1990.

8. Pence, W.D. 1992, "FITSIO - A Subroutine Interface to FITS Format Files", Version 3.21 July 1992.

9. Anonymous ftp sites with more information and software:

   - `astro.umd.edu`: see `pub/nemo` for this document and it's updates, and related follow-up.
   - `fits.cv.nrao.edu`: see `FITS/Documents`
   - `nssdca.gsfc.nasa.gov`: see `[.ACC]` for the Astronomical Data Center catalogues. See `[.CDF]` for CDF related material for VMS. A postscript document is hidden somewhere.
   - `zaphod.ncsa.uiuc.edu`: see directory `HDF`

Appendix A

FITS format

FITS files on disk can be thought of as a set of header-data units (HDU), each consisting of a header, and optionally data. Both have a size which is a multiple of 2880 bytes. The header is ascii, but contains only printable (so-called ASCII text) characters, i.e. the newline formally does not appear in the header. The header consists of cards of 80 characters each. Information in the header is of the form

```
KEYWORD = VALUE / COMMENT
```

where `KEYWORD` is up to 8 characters, always starting in column 1, the `=` sign hence appearing in column 9. There are some formal rules where and how `VALUE` is formatted, which depends on the the keywords type (character string, integer, floating point, complex etc.) After the (optional) "/" symbol, optional comments can follow, for which no rules exist (though they cannot go beyond column 80 of course).

FITS has defined a number of standard keywords (such as SIMPLE, BITPIX, NAXIS etc.) which **must** appear in some order in the beginning of each header. It describes the size of the data, which follows the header. This is to ensure that readers can **always** compute the size of the upcoming data (might be 0) and be able to skip it if it's not understood by the reader software.

Name	Key/ FIeld	dim	Comment	Examples
MASS	K/F	1	Mass of a particle	
TIME	K/F	1	Time	
POS	F	NDIM	Position	
VEL	F	NDIM	Velocity	
PHASE	F	NDIM*2	Phase space coordinates	
ACC	F	NDIM	Accelerations/Forces	
PHI	F	1	Potential	
DENS	F	1	Local Density (SPH)	
TEMP	F	1	Temperature (SPH)	
PRES	F	1?	Pressure (gradients?) (SPH)	
SPHRAD	F	1	SPH radius	
COORDSYS	K	-	Coordinate description[3]	CARTES, POLAR, CYLIN, SPHER
UNITS	K	-	Units convention [2]	VIRIAL, RMSVEL, STRUCT,
NDIM	K	-	Dimensionality	1, 2, 3, (4... ok too)
DATE-OBS	K	-	when data observed [1] 24/12/93	
DATE	K	-	when file was written [1] 09/07/94	
TELESCOP	K	-	name of the computer	e.g. Sparc-10/52
INSTRUME	K	-	name of the code	e.g.
OBJECT	K	-	model name ?	e.g. run201e, A10
REFERENC	K	-	Bibliographic reference	e.g. ApJ 555, 123 (2001)
AUTHOR	K	-	Author of data/paper ?	

[1] date strings are in format `dd/mm/yy`.
[2] for `UNITS` see also the `TUNITnnn` field descriptor.
[3] still need to describe the individual ordering (x,y,z).

Table 2: Proposed standard keywords and field names