

Proposed Keywords for SOHO

Russ Howard, William Thompson
27 February 1995

1 Overview

Keywords will be used during the SOHO operations in several places: in the FITS headers of the distribution data, in the PVL descriptions of the distribution data, in the online relational-database catalog, in the permanent catalog, and in the key-parameters files. It would be most desirable if the same keywords were used in all of these places.

Since FITS is somewhat restricted in the keywords that it uses—the keywords can only be up to eight characters long, and some keywords are already reserved—it would be best if the keywords are initially designed with the needs of FITS files foremost. It is anticipated that other routines would then use the same keywords with no problems.

The general rules for keywords are then as follows:

- Keywords can be no more than eight characters in length.
- Keywords can be made up of only the alphabetic characters, “A” through “Z”, the numerals “0” through “9”, and the underscore (“_”) character.
- Keywords must start with one of the alphabetic characters “A” through “Z”.
- Keywords will be considered to be case-insensitive, i.e. “keyword”, “Keyword”, and “KEYWORD” are all considered to be the same keyword. In FITS files, keywords are always written in uppercase.

Although the FITS standard also allows for dash characters in the keywords, other software packages may have a problem with this. Hence, the above rules do not allow for dashes in the keywords.

2 Reserved FITS keywords

The following keywords are used by the FITS [7] and FITS binary table [2] file standards to describe the format of the file. These keywords will be reserved for the purposes of storing data in FITS files only.

SIMPLE	XTENSION	TFORM1,...,TFORM999
BITPIX	GROUPS	TDIM1,...,TDIM999
NAXIS	PCOUNT	THEAP
NAXIS1,...,NAXIS999	GCOUNT	END
BLOCKED	TFIELDS	
EXTEND	TBCOL1,...,TBCOL999	

There are also some other standard optional keywords described by the above file standards. These are listed below.

AUTHOR	DATE-OBS	PTYPE1,...,PTYPE999
BLANK	DATE	PZERO1,...,PZERO999
BSCALE	EPOCH	REFERENC
BUNIT	EQUINOX	TDISP1,...,TDISP999
BZERO	EXTLEVEL	TELESCOP
CDEL1,...,CDEL1999	EXTNAME	TNULL1,...,TNULL999
COMMENT	EXTVER	TSCAL1,...,TSCAL999
CROTA1,...,CROTA999	HISTORY	TTYPE1,...,TTYPE999
CRPIX1,...,CRPIX999	INSTRUME	TUNIT1,...,TUNIT999
CRVAL1,...,CRVAL999	OBJECT	TZERO1,...,TZERO999
CTYPE1,...,CTYPE999	OBSERVER	UUUUUUUUUU
DATAMAX	ORIGIN	
DATAMIN	PSCAL1,...,PSCAL999	

If these keywords are used, then their use will be limited to a manner consistent with the definitions of these keywords as described in the FITS and FITS/BINTABLE documentation. Not all of these keywords are expected to be used.

There is also a proposed standard [9] for storing compressed data in FITS format. The keywords used by this standard are listed below.

COMPRES	LBITPIX
COMPRES1,...,COMPRES9	LXTEND
LAXIS	LXTENSIN
LAXIS1,...,LAXIS999	

3 Standard FITS keywords as used by SOHO.

Some of the above standard FITS keywords are discussed below. These keywords are considered to be of particular importance to the SOHO mission. In some cases, the standard definitions are further refined to meet the needs of the SOHO program. These additional restrictions are emphasized in the text.

DATE	(string)	The date on which the FITS structure (header and data unit) was defined, in the form DD/MM/YY. <i>The date will be expressed in Universal Time (UT).</i>
ORIGIN	(string)	Character string identifying the organization creating the FITS file. Possible values could be “SOHO”, “SOHO-EOF”, “SOHO-Univ. of Maryland”, etc.
TELESCOP	(string)	<i>The value of this field will be “SOHO”.</i>
INSTRUME	(string)	Character string identifying the instrument used to acquire the data, e.g. “CDS”, “SUMER”, etc.

OBJECT	(string)	Character string containing the name of the object observed, e.g. “CORONAL HOLE”, etc.
COMMENT		May contain any ASCII text. Any number of COMMENT records can appear in the header. <i>These records will appear in the header just before the HISTORY records, in chronological order. If there are no HISTORY records, then the COMMENT records will appear just before the END record.</i>
HISTORY		May contain any ASCII text. Any number of HISTORY records can appear in the header. Will be reserved specifically for entering records describing the processing steps applied to the data in the file (or extension). <i>The HISTORY records will appear in the header just after the COMMENT records (if any), and just before the END record. They will be in chronological order.</i>
BSCALE	(float)	Scale factor (true = value*BSCALE + BZERO).
BZERO	(float)	Offset applied to true pixel values.
BUNIT	(string)	Character string describing the units of the data.
BLANK	(integer)	Undefined pixels are set to this value. This keyword is used only for integer arrays. Floating point arrays use the IEEE standard NaN (not a number) to signal pixels with undefined values.
CTYPE $_{nnn}$	(string)	Type of physical coordinate on axis “ nnn ”.
CRPIX $_{nnn}$	(float)	Array location of a reference pixel along axis “ nnn ”, indexed starting with pixel 1.
CRVAL $_{nnn}$	(float)	Value of physical coordinate axis “ nnn ” at the reference pixel identified by the corresponding CRPIX $_{nnn}$ keyword. <i>Specifically, this corresponds to the center of the reference pixel.</i>
CDEL $_{nnn}$	(float)	Increment in physical coordinates along axis “ nnn ”.
CROT $_{nnn}$	(float)	Rotation of axis “ nnn ” from a standard coordinate system, in degrees. <i>The rotation will be counter-clockwise about the center of the reference pixel given by CRPIX$_{nnn}$</i>

3.1 Keywords for ASCII or binary tables

Note that the letters “nnn” in the following keyword names refers to the column number that the keywords apply to. In other words, the notation “TSCALnnn” refers to the series of keywords TSCAL1 to TSCAL999.

TSCALnnn	(float)	Equivalent of BSCALE keyword for column “nnn” in an ASCII or binary table extension.
TZEROnnn	(float)	Equivalent of BZERO keyword for column “nnn” in an ASCII or binary table extension.
TNULLnnn	(integer)	Equivalent of BLANK keyword for column “nnn” in an ASCII or binary table extension.
TUNITnnn	(string)	Equivalent of BUNIT keyword for column “nnn” in an ASCII or binary table extension.
TTYPEnnn	(string)	The label of column “nnn” in an ASCII or binary table. <i>There will be instances where the data represented by a standard or instrument-specific SOHO keyword will be stored as a column in a table. In these cases the value of the TTYPEnnn keyword will be the appropriate SOHO keyword describing the data, e.g.</i>

TTYPE10 = 'DATE_OBS'

Two other keywords that would be useful, although they are not actually listed in the ASCII or binary table extension descriptions, are:

TDMINnnn	(float)	Equivalent of DATAMIN keyword for a column in an ASCII or binary table.
TDMAXnnn	(float)	Equivalent of DATAMAX keyword for a column in an ASCII or binary table.

3.2 Keywords for binary tables

The formal FITS binary table proposal does not allow for defining an array stored in a binary table as being multidimensional. However, there are several ways to pass the dimensioning information to the FITS reader. One such approach is the “Multidimensional Array Facility”, which uses the following keyword:

TDIMnnn (string) The value is a character string containing the dimensions of a multidimensional array stored in column “nnn” in a binary table, with the format

TDIMnnn='(dim1,dim2,...)'

For instance, a 20×30 array stored in column 5 would be described by TDIM5='(20,30)'.

If one does use the “TDIMnnn” approach, then it becomes necessary to store other information about the dimensions of the data. However, the “Multidimensional Array Facility” does not address this issue. The following keywords were devised by William Thompson of the NASA Goddard Space Flight Center and Applied Research Corporation, and are intended to fill in this gap. (However, see also Appendix G.) Each of these keywords has the format

TXXXXnnn='(val1,val2,...)'

i.e. a character string containing a list of values with a one-to-one correspondence with the dimensions enumerated in the associated “TDIMnnn” keyword. If the “TDIMnnn” keyword is *not* present for that column (i.e., the array is one-dimensional), then these keywords have the simpler form

TXXXXnnn='value'

In either case, these keywords are *always* of type string.

TDESCnnn (string) Extension of CTYPE_n keywords to binary tables. Has the format

TDESCnnn='(desc1,desc2,...)'

where each element in the list corresponds to one of the dimensions in the associated TDIMnnn keyword. For example,

TDESC5='(WAVELNTH,SOLAR_Y)'

As yet there are no standards for the values of the descriptions to be applied to the axes of a data array, although the use of the keywords WAVELNTH, SOLAR_X, and SOLAR_Y are highly recommended.

TROTAnn (string) Extension of CROT_n keywords to binary tables. Has the format

TROTAnn='(rota1,rota2,...)'

where each element in the list corresponds to one of the dimensions in the associated TDIMnnn keyword. *The rotation will be counter-clockwise about the center of the reference pixel given by CRPIXnnn.* The rotation will be expressed in degrees.

TRPIXnnn	(string)	Extension of CRPIXn keywords to binary tables.
TRVALnnn	(string)	Extension of CRVALn keywords to binary tables.
TDELTnnn	(string)	Extension of CDELTn keywords to binary tables.
TCUNInnn	(string)	The units of the various dimentsions, e.g. TCUNI5='(NM,ARCSEC)'

It is evident from Appendix H that there are several possible schemes for storing structured datasets in FITS binary tables. Although there is a tentative plan for doing this, i.e. using the above keywords together with the TDIMnnn keyword, we should recognize that our understanding and desires may change over time. To allow for expansion, the following keyword will be used:

CONVENTN	(string)	Convention used to describe how the binary tables are used to store data. This keyword will appear in the binary table extension header. <i>The value of this keyword will be "TDIM".</i>
----------	----------	---

4 Additional SOHO keywords

Finally, we come to those keywords specifically associated with the SOHO program. To date, these are:

SCLOBJ	(string)	The science objective.
SCLSPEC	(string)	A more specific scientific objective, modifying or clarifying SCLOBJ. For example, SCLOBJ may be "Coronal hole study", while SCLSPEC might be "Density diagnostics".
OBS_PROG	(string)	The observing program.
DATE_OBS	(string)	Starting date and time of data acquisition, in UT. Shall be in CCSDS format (see Appendix E), e.g.

1994-06-28T12:34:56.789Z

(Note the distinction between this and the standard FITS keyword DATE-OBS. The standard keyword has a dash in the name and only supplies the calendar date; this keyword has an underscore and supplies both the date and the time.)

DATE_END	(string)	Ending date and time of data acquisition, in UT. Shall be in the same format as DATE_OBS.
OBT_TIME	(double)	On-board value for the starting time of data acquisition.
OBT_END	(double)	On-board value for the time of the end of data acquisition.
DEL_TIME	(double)	A time difference, in seconds, from the time given by DATE_OBS. Used to express times within an observation.
OBS_MODE	(string)	Observing mode.
DETECTOR	(string)	Detector used to acquire the data.
OBS_SEQ	(string)	The name of the observing sequence used.
EXPTIME	(float)	Exposure time in seconds, to millisecond accuracy.
OBJ_ID	(string)	Object identifier, e.g. active region number.
SOLAR_P0	(float)	The solar P_0 angle.
SOLAR_B0	(float)	The solar B_0 angle.
WAVELNTH	(float)	The wavelength of observation, in nm. Also, TWAVEnnn for binary tables. When used for observations that cover a range of wavelengths, then this keyword represents the wavelength of interest, not necessarily the central wavelength.
WAVEMIN	(float)	Minimum wavelength of observation, in nm. Also, TWMIN for binary tables.
WAVEMAX	(float)	Maximum wavelength of observation, in nm. Also, TWMAX for binary tables.

4.1 Data management keywords

The following keywords are used to describe the organization the data. In particular, these keywords will be used in the on-line catalogs.

FILENAME	(string)	The name of the data file. The use of this keyword will not be construed to mean that the FITS header will need to be changed if the file is copied from one computer to another, or to another media such as tape.
FILEORIG	(string)	Original filename as collected.
SFDUADID	(string)	Standard Formatted Data Unit (SFDU) Authority and Description Identifier (ADID) for a data object. Corresponds to bytes 1–4 and 9–12 of the SFDU label associated with the data object (file). Associating this keyword with the data maintains this information even if the SFDU labels are separated from it.
CMP_NAME	(character)	Name of coordinated observing program (e.g. campaign).
CMP_NO	(integer)	Unique identifier for a coordinated observing program.
CMP_TYPE	(character)	Type of coordinated observing program. Possible values are “JOP” or “Campaign”, although the type of campaign might also be specified.
CMP_DESC	(character)	A description of the coordinated observing program, in particular giving the scientific objectives.
INSTITUT	(character)	Name of an instrument team or institution taking part in a coordinated observing program.
PROG_ID	(integer)	Observing program identification number. PROG_ID is used to associate individual observations together as being part of the same observing program. In particular, an observing program that is not contiguous in time, for instance an observation made at the same time each day, or once every solar rotation, could be identified by the same value of PROG_ID.

PROG_NUM	(integer)	Unique identifier for the observing program. Each observing program is assigned a unique identifier number each time it's run, even if it's a continuation of an earlier program.
COUNT	(integer)	Number of repeated observing sequences within an observing program.
SEQ_NUM	(integer)	Unique number assigned to an execution of an observing sequence. Depending on how an instrument team organizes their data, this number may represent an individual observing sequence or a repeated series of the same observing sequence.
SEQ_FROM	(integer)	First value of SEQ_NUM in a series of observing sequences.
SEQ_TO	(integer)	Last value of SEQ_NUM in a series of observing sequences.
SEQ_IND	(integer)	Index of an observing sequence within an observing program, from one to the value of COUNT.
EXPCOUNT	(integer)	Number of exposures within an observing sequence.
EXP_IND	(integer)	Index of an exposure within an observing sequence, from one to the value of EXPCOUNT.
SEQVALID	(character)	Either "Y" or "N", denoting whether or not the data from an observing sequence has scientific value.
SEQONLIN	(character)	Either "Y" or "N", denoting whether or not the data from an observing sequence is online or not.

The following keywords specifically support the SOHO online catalogs, and are related to the mechanism of maintaining the database. They are not intended to be used anywhere else.

COMMENTS	(character)	Either "Y" or "N", depending on whether or not an entry in a database is commented or not. <i>For use in the online catalogs only.</i>
COMM_NO	(integer)	Comment number. <i>For use in the online catalogs only.</i>
DATE_MOD	(date)	Date/time that an entry in a catalog was last modified. <i>For use in the online catalogs only.</i>

4.2 Pointing keywords

Five levels of pointing information are envisioned. Each of these coordinate systems has the X-axis pointing to the “right”, and the Y-axis pointing “up”. The exact definitions of “right” and “up” will change depending on the pointing level being considered, but would ordinarily be associated with solar west and north respectively.

The first pointing level would represent the position of a subarray within the coordinate system of the instrument’s detector. The division of a detector array into subarrays might occur on the spacecraft, or might occur on the ground as part of the data reduction process. The keywords for this are:

DET_X	(integer)	Position of pixel within a detector along the X axis.
DET_Y	(integer)	Position of pixel within a detector along the Y axis.

The second level is the pointing of the instrument relative to its own origin. In other words, there may be scan mirror movements, etc., which change the pointing of the instrument. This situation is covered by the keywords:

INS_X	(float)	Pointing of instrument along the <i>instrument</i> X-axis.
INS_Y	(float)	Pointing of instrument along the <i>instrument</i> Y-axis.

The third level is the pointing of the instrument when it’s at its origin relative to the spacecraft. I.e., instead of changing the pointing by moving scan mirrors, the pointing of the entire instrument is changed. Alternatively, if the instrument has it’s own sun sensors, then except for roll angle this level will be relative to the solar disk. Both of these situations are covered by the keywords:

INS_X0	(float)	Pointing of instrument origin along the <i>spacecraft/solar</i> X-axis.
INS_Y0	(float)	Pointing of instrument origin along the <i>spacecraft/solar</i> Y-axis.
INS_ROLL	(float)	Instrument roll angle relative to the spacecraft, i.e. the angle between the instrument X-axis and the spacecraft X-axis, measured in a counter-clockwise direction from the spacecraft X-axis toward the spacecraft Y-axis.

Whether or not INS_X0 and INS_Y0 are relative to the spacecraft or to the sun will depend on the instrument involved.

The fourth level is the pointing of the spacecraft relative to the sun. The keywords covering this situation are:

SC_X0	(float)	Spacecraft pointing along the <i>solar</i> X-axis as defined by the keyword SOLAR_X.
SC_Y0	(float)	Spacecraft pointing along the <i>solar</i> Y-axis as defined by the keyword SOLAR_Y.
SC_ROLL	(float)	Spacecraft roll angle relative to solar coordinates.

The final level of pointing information is when all the information from the above four intermediate levels are put together to determine where a data array is pointing to on the Sun. This will be controlled by the keywords:

SOLAR_X	(float)	General pointing relative to the solar X axis, which is defined as being in the plane of the sky as seen by the instrument, perpendicular to the projection of the solar north pole, and towards the west solar limb.
SOLAR_Y	(float)	General pointing relative to the solar Y axis, which is defined as being the projection of the solar north pole axis on the plane of the sky as seen by the instrument.

These keywords are designed to be compatible with the inter-instrument coordinate systems described in the Science Operations Plan. However, they differ in that they are always aligned with the solar north pole axis, whereas it appears that this is only true for the inter-instrument axes when the spacecraft roll angle is zero.

SWAN is an exception, using Euler angles in an ecliptic coordinate system, rather than Y and Z offsets from Sun center, to describe its pointing. The keywords to describe this are (after consultation with Walter Schmidt):

AZ_START	(float)	Starting ecliptic azimuth of an observation, in degrees.
EL_START	(float)	Starting ecliptic elevation of an observation, in degrees.
AZ_END	(float)	Ending azimuth.
EL_END	(float)	Ending elevation.
AZ	(float)	The azimuth of a data point.
EL	(float)	The elevation of a data point.

4.2.1 Image coordinates

Once we have all this information, we can calculate the pointing of an image array, or set of image arrays, relative to the Sun. This information would be particularly useful in the data catalogs. It is assumed here that the instrument has a field-of-view which fits within a rectangular box at some angle to the solar north (although it may not completely fill it). The proposed keywords which describe this box are:

XCEN	(float)	Center of the instrument field-of-view along the solar X-axis.
YCEN	(float)	Center of the instrument field-of-view along the solar Y-axis.
ANGLE	(float)	Angle of rotation of the vertical axis of the instrument field-of-view relative to solar north.
IXWIDTH	(float)	Maximum width of the instrument field-of-view in the instrument X axis, i.e. the direction perpendicular to the vertical axis as used in keyword ANGLE.
IYWIDTH	(float)	Maximum width of the instrument field-of-view in the instrument Y axis, i.e. the direction along the vertical axis as used in keyword ANGLE.

4.3 Synoptic data keywords

SOHO will use synoptic data from a large variety of ground and spacecraft based sources to assist in the planning for the daily operations. Many of these sources are already making their data available in FITS format—others are not. For those who have not yet finalized their FITS format, we suggest that the FITS headers of these files contain the following information:

DATE_OBS	(string)	The date and time associated with the data, in UT. Shall be in CCSDS format (see Appendix E), e.g.
----------	----------	--

1994-06-28T12:34:56.789Z

(Note the distinction between this and the standard FITS keyword DATE-OBS. The standard keyword has a dash in the name and only supplies the calendar date; this keyword has an underscore and supplies both the date and the time.)

CENTER_X	(double)	The coordinate of the center of the Sun in pixels along the first dimension, where the center of the first pixel in the image has the coordinate value 1 along each axis.
----------	----------	---

CENTER_Y	(double)	The coordinate of the center of the Sun in pixels along the second dimension, where the center of the first pixel in the image has the coordinate value 1 along each axis.
XSCALE	(double)	The scale of the image, in arcsec/pixel, along the first dimension. If the image is left-right reversed, then this value will be negative.
YSCALE	(double)	The scale of the image, in arcsec/pixel, along the second dimension. If the image is up-down reversed, then this value will be negative.
ANGLE	(double)	Angle of rotation of the vertical axis of the instrument field-of-view relative to solar north. In other words, one would need to rotate the image clockwise by ANGLE degrees to orient the image with solar north in the vertical direction. This rotation would occur <i>after</i> applying any image reversals to restore XSCALE and YSCALE to positive values.

There are also some optional keywords, as follows:

SOLAR_R	(double)	The apparent radius of the solar disk, in pixels. This keyword can be supplied in place of XSCALE and YSCALE, assuming both would be positive.
SOLAR_P0	(string)	The solar P_0 angle.
SOLAR_B0	(string)	The solar B_0 angle.

4.4 Orbital parameters keywords

This section is devoted to those keywords related to the orbital elements and attitude of the spacecraft. These keywords are as yet **TBD**.

4.5 File processing keywords

This section is devoted to those keywords related to the state of processing of data files (i.e., state of calibration, etc.). These keywords are as yet **TBD**.

References

- [1] ESO science archive rev 1.3. Technical report, March 26 1992. ARC-SPE-ESO-0000-0001/1.3.
- [2] W. D. Cotton and D. B. Tody. Binary table extension to FITS: A proposal. preprint, 1991.
- [3] Consultative Committee for Space Data Systems. Time code formats. Technical Report CCSDS 301.0-B-2, CCSDS Secretariate, Communications and Data Systems Division, (Code-OS), National Aeronautics and Space Administration, Washington, DC 20546, USA, April 1990.
- [4] Consultative Committee for Space Data Systems. Parameter value language—a tutorial. Technical Report CCSDS 641.0-G-1, CCSDS Secretariate, Communications and Data Systems Division, (Code-OS), National Aeronautics and Space Administration, Washington, DC 20546, USA, May 1992.
- [5] Consultative Committee for Space Data Systems. Parameter value language specification (ccsd0006). Technical Report CCSDS 641.0-B-1, CCSDS Secretariate, Communications and Data Systems Division, (Code-OS), National Aeronautics and Space Administration, Washington, DC 20546, USA, May 1992.
- [6] W. H. Mish. International Solar-Terrestrial Physics (ISTP) Key Parameter Generation Software (KPGS) Standards & Conventions. Technical report, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA, April 1992.
- [7] NOST. Implementation of the Flexible Image Transport System (FITS). Technical Report NOST 100-0.3b, NASA/OSSA Office of Standards and Technology, Code 933, NASA Goddard Space Flight Center, Greenbelt MD 20771, USA, November 6 1991.
- [8] W. T. Thompson. ISTP file formats and keywords. SOHO SOWG internal memo, August 1992.
- [9] A. Warnock III, R. S. Hill, B. B. Pfarr, and D. C. Wells. An extension of FITS for data compression. preprint, 1991.

APPENDICES

A Questions

The following need to be decided.

- Should time keywords be referenced to UT at Earth, or should the time delay between the Earth and the spacecraft be taken into account?
- Should a unique number be assigned each time an observing sequence is executed (identified by keyword SEQ_NUM), or should sequences be identified only by the program number and the index within the observing program?
- Which keywords in sections 3 and 4 should be considered as required?

B Changes from previous versions

This appendix lists the changes made from earlier versions of this document. Changes are listed in chronological order.

8 March 1992 Split PROG_ID keyword into PROG_ID (observing program one might want to come back to) and PROG_NUM (unique ID for database).

14 March 1992 Changed keyword SEQ_ID to SEQ_NUM, to avoid confusion with OBS_SEQ.

14 March 1992 Changed SEQVAL to SEQVALID, and SEQONL to SEQONLIN.

14 March 1992 Changed OBS_IND to SEQ_IND, for greater consistency.

14 March 1992 Added keywords EXPCOUNT and EXP_IND.

15 March 1992 Added keyword OBS_MODE.

15 March 1992 Added keywords COMMENTS, DATE_MOD, and TIME_MOD.

22 March 1992 Changed way that columns in binary tables refer to other columns (Appendix F). Referencing is now by name rather than by number.

5 April 1992 Added INSTITUT and COMM_NO keywords.

21 August 1992 Added appendices C and E.

10 September 1992 Added appendix D.

7 June 1994 Numerous changes:

- Changed from a Y-Z coordinate system to an X-Y coordinate system, to be consistent with the inter-instrument coordinate system given in the science operations plan. This affects the following keywords:

Old keyword	New keyword
DET_Y	DET_X
DET_Z	DET_Y
INS_Y	INS_X
INS_Z	INS_Y
INS_Y0	INS_X0
INS_Z0	INS_Y0
SC_Y0	SC_X0
SC_Z0	SC_Y0
YCEN	XCEN
ZCEN	YCEN

- Also changed the following keywords:

Old keyword	New keyword
WIDTH	IXWIDTH
HEIGHT	IYWIDTH
AIT_TIME	OBT_TIME
AIT_END	OBT_END

- Added the keywords SCI_SPEC, CMP_DESC, SOLAR_X, SOLAR_Y, DEL_TIME, DET_X, DET_Y, and the various SWAN pointing keywords.
- Changed the format of DATE_OBS and DATE_END to that recommended by the CCSDS. Removed the keywords TIME_OBS, TIME_END, TIME_MOD.
- Added Appendices D.1 and G.

13 June 1994 Added keyword TCUNInnn.

8 February 1995 Added Section 4.3

27 February 1995 Used phrase “coordinated observing program” rather than campaign. Added CMP_TYPE.

C Other keyword systems

C.1 Known ISTEP keywords

The following keywords are known to be used by the ISTEP CDHF for storing data in CDF files, and in SFDU headers [6]:

CATDESC	FILLVAL	REFERENCETYPE	TEXT
COMMENT	FORMAT	LABEL	TIME_PB5
DATA_TYPE	FORM_PTR _{<i>i</i>}	REFERENCE	TIMETAG
DATA_VERSION	GENERATION_DATE	SCALEFAC	TITLE
DEPEND _{<i>i</i>}	INPUT_FILE	SCALEMAX	VALIDMAX
DESCRIPTOR	LABLAXIS	SCALEMIN	VALIDMIN
DISCIPLINE	LABL_PTR	SCALETYP	UNITS
EPOCH	MODS	SCAL_PTR	UNIT_PTR
FIELDNAM	MONOTON	SOURCE_NAME	
FILE_ID	PROJECT	START_DATE	

None of these keywords represents a conflict with SOHO keywords. However, the following correspondences must be made

SOHO	ISTP
BLANK	FILLVAL
BUNIT	UNITS
CTYPE	LABLAXIS
DATE_OBS & TIME_OBS	START_DATE
FILENAME	FILE_ID
HISTORY	MODS
TDISP _{<i>n</i>}	FORMAT
TELESCOP	SOURCE_NAME
TTYPE _{<i>n</i>}	FIELDNAM

See the document “ISTP File Formats and Keywords” [8] for more information.

Some of these keywords may be useful for the SOHO project. Others may be useful with appropriate FITS-compatible equivalents, e.g.

SOHO	ISTP
DATATYPE	DATA_TYPE
DESCRIPT	DESCRIPTOR
VERSION	DATA_VERSION

C.2 PVL reserved keywords

The following keywords are listed as reserved in the Parameter Value Language (PVL) [5] used in Standard Formatted Data Units:

BEGIN_GROUP
BEGIN_OBJECT
END_GROUP
END_OBJECT
END
GROUP
OBJECT

Of these, the only keyword which conflicts with those discussed in this document is **OBJECT**. However, **OBJECT** is a standard FITS keyword, and it seems undesirable to use something else in its place. It is therefore recommended that **OBJECT** be used everywhere except in PVL, where the equivalent keyword **OBJECT_OBSERVED** be used.

The keyword **GROUP** also represents a potential conflict. However, this keyword is not currently referenced in this document. All other keywords are either longer than the eight characters allowed by FITS, except for **END** which is also a reserved keyword in the FITS standard.

D Storing other keyword conventions in FITS files

As already noted, a number of different keyword systems will be used at the SOHO EOF. Two in particular are FITS headers, and SFDU/PVL descriptors. It is therefore inevitable that there will be keywords used by these other systems which are inconsistent with FITS in one way or another. Such inconsistencies could consist of, for example,

- Keywords longer than eight characters.
- Keywords which can appear multiple times. In FITS this is allowed only for the HISTORY, COMMENT, and `UUUUUUUU` (blank) keywords.
- Keywords which are grouped together in a hierarchal structure, e.g. delimited with BEGIN/END statements.

All data which is necessary for reading, or processing, scientific data stored in FITS files should and must be stored using conventional FITS structures. If this data would otherwise be stored in keywords that don't satisfy the FITS convention in one way or another, then either a proper FITS alias keyword must be assigned (c.f. appendix C.1), or the data must be stored in some other way, such as in an ASCII or binary table (c.f. appendix F).

However, it is also sometimes desirable to store auxilliary information pertaining to the main data, which comes from a non-FITS system. I have researched how other facilities handle this problem, and it is clear that the most appropriate thing to do is to store the data in a series of FITS HISTORY records. A FITS HISTORY record can contain any ASCII text appropriate to describing the origin and processing of the data stored in the file; however there is an additional step that can be taken which should greatly increase the usefulness of the data.

I was given this method by Eric Greisen of the National Radio Astronomical Observatory, where it is used. I reproduce his message verbatim below

```
The solution we have used at NRAO (at it is even quietly present in the original FITS paper!) is to have HISTORY cards. As the first word in the HISTORY we put an identifier that says that we have written the card, e.g. NRAO or AIPS. Then we put one or more keyword=value pairs, with any old arbitrary choice of keywords. Our FITS readers check each HISTORY card's next word and if it is one of a few magic ones they parse the rest of the card. Otherwise, they ignore the card as non-NRAO readers are expected to do with all our cards as well.
```

I would take Eric Greisen's suggestion one step further, and say that the syntax of what follows the "magic word" should depend only on the environment referenced. For example, suppose that we wanted to include a series of SFDU/PVL statements; we might have (taken from "Parameter Value Language—A Tutorial" [4]):

```
HISTORY SFDU BEGIN_GROUP = ELEMENT_DEFINITION;
HISTORY SFDU      NAME = SPACECRAFT_ID;
HISTORY SFDU      DEFINITION = "Space craft identifiers";
HISTORY SFDU      DATA_SYNTAX_ID = C;
HISTORY SFDU      DOMAIN_LIST =      {WIND, POLAR, GEOTAIL, CLUSTER, SOHO};
HISTORY SFDU END_GROUP = ELEMENT_DEFINITION;
```

This demonstrates the power of this approach for storing non-FITS like data in a FITS header. The statements follow the SFDU/PVL syntax, which differs from the FITS syntax in a number of ways, e.g. semi-colons, a BEGIN_GROUP and END_GROUP hierarchy, different rules for using quotation marks, keywords longer than eight characters, and a list of values surrounded by { } brackets. Using the word SFDU after the HISTORY statement delineates them from ordinary history records, and from other non-FITS keyword groups that might also be present (e.g. instrument-specific engineering keywords that don't match the FITS standard).

D.1 Hierarchical keywords

A related approach is that of hierarchical keywords [1]. This allows one to specify a heirarchy of parameters that is very similar to structures in C or IDL. The (non-standard) keyword HIERARCH is used to denote that what follows is in this hierarchical format. For example, one might have

```

HIERARCH LASCO GEN ID          = 'ARC-0001/1.0'
HIERARCH LASCO GEN OBSP NAME = 'POLZ '
HIERARCH LASCO GEN OBSP TYPE = 'B '
HIERARCH LASCO GEN OBSP NO    =                278
HIERARCH LASCO GEN EXPO TYPE = 'SCI '
HIERARCH LASCO GEN EXPO NO    =                34510
HIERARCH LASCO GEN TEMP-1     =                14.5
HIERARCH LASCO GEN TEMP-2     =                16.5

```

If we were looking at this as a structure, then the structure LASCO would contain the single element GEN. This would in turn be a structure with the elements ID, OBSP, EXPO, TEMP-1 and TEMP2. OBSP would be a structure with elements NAME, TYPE, and NO, and EXPO would contain simply TYPE and NO.

However, it appears that the status of this proposal is in a very uncertain state.

E CCSDS time standard

There has been a discussion of how time values should be represented in the SOHO EOF. I have come across a document which describes an international standard for representing time in space data sets [3]. This standard has been adopted by the ISTP project [6].

The following is taken verbatim from the CCSDS document "Time Code Formats" [3]:

CCSDS RECOMMENDATION FOR TIME CODE FORMATS

2.5 CCSDS ASCII CALENDAR SEGMENTED TIME CODE (ASCII)

2.5.1 T-FIELD

The CCSDS ASCII segmented time code is composed of a variable number of ASCII characters forming the T-field.

Both ASCII time code variations are UTC-based and leap second corrections must be made. The time represented is intended to match civil time usage. Therefore, the epoch is taken to be the usual Gregorian calendar epoch of 1 AD, and the time is that of the prime meridian.

The ASCII time code Recommendations are Level 1 time code formats.

2.5.1.1 ASCII TIME CODE A, Month/Day of Month Calendar Variation:

The format for ASCII Time Code A is as follows:

YYYY-MM-DDThh:mm:ss.d->dZ

where each character is an ASCII character using one octet with the following meanings:

YYYY	=	Year in four-character subfield with values 0001-9999
MM	=	Month in two-character subfield with values 01-12
DD	=	Day of month in two-character subfield with values 01-28, -29, -30, or -31
"T"	=	Calendar-Time separator
hh	=	Hour in two-character subfield with values 00-23
mm	=	Minute in two-character subfield with values 00-59
ss	=	Second in two-character subfield with values 00-59 (-58 or -60 during leap seconds)
d->d	=	Decimal fraction of second in one- to n-character subfield where each d has values 0-9
"Z"	=	time code terminator (optional)

Note that the hyphen (-), colon (:), letter "T" and period (.) are used as specific subfield separators, and that all subfields must include leading

zeros.

As many "d" characters to the right of the period as required may be used to obtain the required precision.

An optional terminator consisting of the ASCII character "Z" may be placed at the end of the time code.

EXAMPLE: 1988-01-18T17:20:43.123456Z

2.5.1.2 ASCII TIME CODE B, Year/Day of Year Calendar Variation:

The format for ASCII Time Code B is as follows:

YYYY-DDDThh:mm:ss.d->dZ

where each character is an ASCII character using one octet with the following meanings:

YYYY	=	Year in four-character subfield with values 0001-9999
DDD	=	Day of year in three-character subfield with values 001-365 or -366
"T"	=	Calendar-Time separator
HH	=	Hour in two-character subfield with values 00-23
MM	=	Minute in two-character subfield with values 00-59
SS	=	Second in two-character subfield with values 00-59 (-58 or -60 during leap seconds)
d->d	=	Decimal fraction of second in one- to n-character subfield where each d has values 0-9
"Z"	=	time code terminator (optional)

Note that the hyphen (-), colon (:), letter "T" and period (.) are used as specific subfield separators, and that all subfields must include leading zeros.

As many "d" characters to the right of the period as required may be used to obtain the required precision.

An optional terminator consisting of the ASCII character "Z" may be placed at the end of the time code.

EXAMPLE: 1988-018T17:20:43.123456Z

2.5.1.3 SUBSETS OF THE COMPLETE TIME CODES:

When it is desired to use SUBSETS of each of the TWO ASCII time code format

variations described above, the following rules must be observed:

a. The "calendar" subset (all subfields to the left of the "T") and the "time" subset (all subfields to the right of the "T") may be used independently as separate "calendar" or "time" formats, provided the context in which each subset is used makes its interpretation unambiguous.

b. When calendar or time subsets are used alone, the "T" separator is omitted.

c. Calendar or time subsets may contain all the defined subfields, or may be abbreviated to the span of interest by deleting the unneeded subfields, either on the left or on the right. However, when subfields are deleted on the LEFT, all separators that had delimited the deleted subfields must be retained (except for the "T" which, by rule b, is dropped if the subset is used alone.) When subfields are deleted on the RIGHT, the separators that had delimited the deleted subfields are dropped.

d. Subsets may NOT consist of partial subfields (e.g., must use "ss", not "s"). In particular, consistent use of the complete four-character YYYY subfield is required (e.g., "1989" instead of "89") because of the need to accommodate the upcoming century rollover in only 1 1 years. Note, however, that each fractional second ("d" character) is considered to be a complete subfield, and so any number of fractional seconds may be used.

e. If calendar and time SUBSETS are then brought together to form a single time code format (joined with the "T" separator) the CALENDAR subset may NOT have been truncated from the RIGHT, and the TIME subset may NOT have been truncated from the LEFT. That is, the format must be integral around the "T".

f. Standardization on the use of these time code formats for purposes OTHER than identifying an instant of calendar or time in UTC (e.g., unconventional use as a counter or tool for measuring arbitrary intervals) is not recommended. It is felt such a specialized application can best be viewed not as a time code format but rather as an engineering measurement format. Any such application of these time code formats is considered beyond the scope of this recommendation.

F Notes on storing keyword data in ASCII and binary tables

It should be evident from the above that the keywords for simple FITS files and for ASCII and binary tables do not always agree. Keywords used to describe the data arrays must be modified for tabular data to include the column number that the keyword applies to. This limits the actual name of the table keyword to only five characters. However, simply because a keyword parameter is to be entered into a binary table does not mean that the keyword has to be limited to five characters. There is a very simple way to enter a keyword parameter into a binary table with its full eight character name.

Think of a binary (or ASCII) table as being made up of rows and columns. The most useful way to organize data into a binary table is to consider the columns as representing different parts of a single observation, and the rows as representing different observations with the same observing mode. For example, the data from the CDS Normal Incidence Spectrometer will be organized as a series of windows selecting out different parts of the CCD for readout. Each exposure of the CCD is a separate observation, and the individual windows are each stored in a column of the table. (*Note: It appears most likely that CDS observations shall be organized with all the data in a single row of the binary table. However, this does not invalidate the general principal involved here.*)

If a keyword parameter is associated with the observation as a whole, rather than a piece (e.g. one of the CCD windows) then this parameter would be stored as an additional column in the binary table. The TTYPE_n keyword for this column would then take the value of the name of the keyword associated with this parameter. For instance, if one wanted to store the time of each exposure in the binary table, then one would store this in a column (e.g. 5), and the extension header would include the line

```
TTYPE5 = 'DATE_OBS'
```

This technique could also be used for keyword parameters that give information about a part of an observation. All that is necessary is that its TTYPE_n keyword includes the column that the parameter refers to. For instance, suppose that column 3 contains one of the windows on a CCD. One could then store the keyword parameter DATAMAX for column 3 in another column (e.g. 4). The TTYPE_n keyword for this column would then look like

```
TTYPE3 = 'FE335'  
TTYPE4 = 'DATAMAX(FE335)'
```

This technique would not be appropriate for those keyword parameters which apply to individual pieces of an observation, but which do not change from exposure to exposure in the same observing mode. This would lead to an unnecessary duplication of information. However, it is anticipated that this would not be common, and that the keywords already defined, i.e.,

```
TCUNInnn  TDISPnnn  TNULLnnn  TSCALnnn  
TDELTnnn  TDMAXnnn  TROTAnnn  TTYPEnnn  
TDESCnnn  TDMINnnn  TRPIXnnn  TUNITnnn  
TDIMnnn   TFORMnnn  TRVALnnn  TZEROnnn
```

should be adequate for most purposes. A few exceptions have been noted above, namely,

TWAVE_{nnn} TWMIN_{nnn} TWMAX_{nnn}

G The HEASARC approach to annotating columns in binary tables.

One of the shortcomings of the FITS binary table approach is that multidimensional arrays are not a formal part of the standard. There is, an appendix that describes a way to specify multidimensionality using a keyword called TDIM. However, it doesn't describe any mechanism to annotate the dimensions of an array, as one can do in ordinary FITS files.

To overcome this shortcoming, I came up with a set of keywords to replicate the standard FITS keywords relating to dimensions in binary tables. These are listed in the SoHO keywords document. However, since that time, the High Energy Astrophysics group at Goddard (HEASARC) has come up with an alternate plan to do the same thing.

To illustrate the differences, suppose that one had a two-dimensional array with dimensions representing energy and time. First of all, we would have the following keywords, regardless of which convention was being used:

```
TTYPE2 = 'RATE      '           /Observed flux count rate
TFORM2 = '80E       '           /Array has 80 data values, type Real*4
TUNIT2 = 'counts/s'           /Units of the RATE values
TDIM2  = '(8,10)   '           /Array is two-dimensional
```

In my system, the annotation for the individual dimensions are modelled after the form of TDIM, and would appear in the FITS binary table header this way:

```
TDESC2 = '(ENERGY,TIME)'       /Dimension labels
TRPIX2 = '(1.0,1.0)'           /Index of reference pixel
TRVAL2 = '(2.5,0.0)'           /Axis values at reference pixel
TDEL2  = '(0.1,4.0)'           /Pixel spacings along each axis
TCUNI2 = '(keV,s) '           /Axis units
```

However, in the HEASARC system, the same information would appear as follows:

```
1CTYP2 = 'ENERGY  '           /Dimension labels
2CTYP2 = 'TIME    '
1CRPX2 =                1.0 /Index of reference pixel
2CRPX2 =                1.0
1CRVL2 =                2.5 /Axis values at reference pixel
2CRVL2 =                0.0
1CDLT2 =                0.1 /Pixel spacings along each axis
2CDLT2 =                4.0
1CUNI2 = 'keV     '           /Axis units
2CUNI2 = 's       '           /Axis units
```

The HEASARC people were aware of my proposal, but were concerned that the length of the strings would be too long. Also, they preferred that a single FITS keyword point to a single value, rather than an array of values.

The downside of the HEASARC approach is that it takes more space than mine.

What we, the SoHO community, need to do is to decide whether or not we want to continue with the approach we started with, or convert over to HEASARC's system in the expectation that it will become a standard.

H The “Green Bank Convention”, and the relational approach

In researching how one would store complex data structures in an IDL file, I’ve come across two main ways of doing it. One, which I currently prefer, is to use binary tables with the TDIMnnn keywords as outlined above. The other approach, which has a lot of support in the FITS community, is to use something called the “Green Bank Convention” (GBC).

This convention works by storing the IDL keywords that would ordinarily describe a data array, NAXIS, NAXIS1, etc. in columns in a binary table, with one column containing the actual data arrays. In other words, the binary table would look to some extent like a series of ordinary FITS files. The advantage of this is that one would not have to have different keywords for binary tables (e.g. WAVELNTH vs. TWAVE5), or keywords that point to specific columns (e.g. “DATAMAX” vs. “DATAMAX(FE335)”). The disadvantage, as I see it, is that it makes it difficult to treat data that pertains to an observation as a whole, as opposed to a piece of an observation.

A more detailed look at the GBC points out the following:

- The binary table keyword TTYPEnnn gives the label of the keyword being stored in column “nnn”.
- The column containing the actual data arrays is labeled by TTYPEnnn=’DATA’ (**TBR**).
- The names of the keywords NAXIS, NAXIS1, etc., are modified to be MAXIS, MAXIS1, etc. (**TBR**). All other keywords (e.g. BSCALE, CTYPEnnn, etc.) are unaffected.
- There is no column corresponding to BITPIX. The data type is defined in the description of the binary table. This means that one can’t mix data types in the data column.
- Parameters which would be the same for each row can be substituted with a keyword. For instance, if all the data arrays were two-dimensional, then one could have MAXIS=2 in the header, rather than having a column for MAXIS.
- The optional variable length array facility should be used if one wants to minimize file size. Otherwise, padding would be required.

It would be instructive to consider an example. The CDS Normal Incidence Spectrograph detector is an intensified CCD array that will have the capability of selecting out a series of windows of interest on the CCD representing different spectral lines to be read out. Let us then suppose that a series of observations are made with the same series of lines being read out. Hence, we would have for our data set a series of M windows repeated N times. The size of the M windows would be different from each other, but would not vary during the N observations.

To store this data using the TDIMnnn approach, the M windows would be stored in M columns in the binary table, and there would be N rows for the N observations.

However, if one uses the GBC, then only one column would actually store all the windows, and there would be $M \times N$ rows in the table. There would be a column storing the observation number, and a column storing the window number. The dimensions would be stored in other columns, and would form a repeating pattern (window 1, window 2, ..., window M , window 1, ...).

Now let us consider where the data that would be stored that would pertain to an entire observation regardless of which window one was looking at. In the TDIMnnn approach, this would

simply be another column in the table. In the GBC approach, however, this would most efficiently be another FITS extension, and would be *related* to the extension containing the windows. In other words, one would be taking a relational database approach, joining two tables together by using common key fields. One extension would contain a row for each window from each observation ($M \times N$ rows), and the other extension would contain one row for each observation (N rows).

So far, I've been considering this relational approach as being more complex to implement than the TDIMnnn approach. However, there are some advantages to this approach:

- There would be only one form for keywords, rather than the three forms (e.g. “DATAMIN”, “TDMINnnn”, and “DATAMIN(FE335)”) required for the TDIMnnn approach.
- It does allow for more complex interrelationships than can be expressed in a single binary table; for instance, cases where the observing mode changes from exposure to exposure. (On the other hand, I've also felt that it tends to confuse the situation when the observations are a repetitive sequence)
- It groups similar pieces from the same exposure together into one column, so that operations could be performed on them simultaneously. In the above example, the TDIMnnn approach treats different windows on the CCD basically as apples and oranges, while the GBC approach treats them as slightly different kinds of apples.

My main worry about the relational approach is that the software to open up multiple extensions within a FITS file, and relate them together, would have to be developed. The above example consisted of two extensions, one for the window arrays ($M \times N$ rows), and one for the exposures (N rows). Another extension could be added with M rows that would contain information about the windows that would be the same for all exposures (e.g., the dimensions, the wavelength, etc.), and other extensions could also be imagined. All of these would have to be integrated together in software.