NOST

Definition of the Flexible Image Transport System (*FITS*)

June 18, 1993

Standard

NOST 100-1.0

NASA/Science Office of Standards and Technology
Code 633.2
NASA Goddard Space Flight Center
Greenbelt MD 20771
USA

The NASA/Science Office of Standards and Technology (NOST) has been established to serve the space science communities in evolving cost effective, interoperable data systems. The NOST performs a number of functions designed to facilitate the recognition, development, adoption, and use of standards by the space science communities.

Approval of a NOST standard requires verification by the NOST that the following requirements have been met: consensus of the technical panel, proper adjudication of the comments received from the targeted space and Earth science community, and conformance to the accreditation process.

A NOST standard represents the consensus of the technical panel convened by the NASA/Science Office of Standards and Technology (NOST) of the National Space Science Data Center (NSSDC) of the National Aeronautics and Space Administration (NASA). Consensus is established when the NOST Accreditation Panel determines that substantial agreement has been reached by the Technical Panel. However, consensus does not necessarily imply that all members were in full agreement with every item in the standard. NOST standards are not binding as published; however, they may serve as a basis for mandatory standards when adopted by NASA or other organizations.

A NOST standard may be revised at any time, depending on developments in the areas covered by the standard. Also, within five years from the date of its issuance, this standard will be reviewed by the NOST to determine whether it should 1) remain in effect without change, 2) be changed to reflect the impact of new technologies or new requirements, or 3) be retired or canceled.

The Technical Panel that developed this standard consisted of the following members:

Robert J. Hanisch, Chair	Space Telescope Science Institute
Barry M. Schlesinger, Secretary	Hughes STX
Lee E. Brotzman	Hughes STX
Edward Kemper	Hughes STX
Peter J. Teuben	University of Maryland
Michael E. Van Steenberg	NASA Goddard Space Flight Center
Wayne H. Warren Jr.	Hughes STX
Richard A. White	NASA Goddard Space Flight Center

This standard is published and maintained by the NOST. Send comments and orders for NOST documents to:

NOST, Code 633.2, NASA Goddard Space Flight Center
Greenbelt MD 20771
USA
Internet: nost@nssdca.gsfc.nasa.gov
DECNET: NSSDCA::NOST
301-286-3575

Contents

Appendixes

List of Tables

List of Figures

Introduction

The Flexible Image Transport System (*FITS*) evolved out of the recognition that a standard format was needed for transferring astronomical data from one installation to another. The original form, or Basic *FITS* [1], was designed for the transfer of images and consisted of a binary array, usually multidimensional, preceded by an ASCII text header with information describing the organization and contents of the array. The *FITS* concept was later expanded to accommodate more complex data formats. A new format for image transfer, *random groups*, was defined [2] in which the data would consist of a series of arrays, with each array accompanied by a set of associated parameters. These formats were formally endorsed by the International Astronomical Union (IAU) in 1982 [3]. Provisions for data structures other than simple arrays or groups were made later. These structures appear in *extensions*, each consisting of an ASCII header followed by the data whose organization it describes. A set of general rules governing such extensions [4] and a particular extension, *ASCII Tables* [5], were endorsed by the IAU General Assembly in 1988 [6]. At the same General Assembly, an IAU *FITS* Working Group was formed with the mandate to maintain the existing *FITS* standards and to review, approve, and maintain future extensions to *FITS*, recommended practices for *FITS*, implementations, and the thesaurus of approved *FITS* keywords [7]. In 1989, the IAU Commission 5 *FITS* Working Group approved a formal agreement [8] for the representation of floating point numbers. *FITS* was originally designed and defined for 9-track half-inch magnetic tape. However, as improvements in technology have brought forward other data storage and data distribution media, it has generally been agreed that the *FITS* format is to be understood as a logical format and not defined in terms of the physical characteristics of any particular data storage medium or media.

Section 1

Overview

1.1 Purpose

This standard formally defines the *FITS* format for data structuring and exchange that is to be used where applicable as defined in Section 1.3. It is intended as a formal codification of the *FITS* format that has been endorsed by the IAU for transfer of astronomical data, fully consistent with all actions and endorsements of the IAU and the IAU Commission 5 *FITS* Working Group. Minor ambiguities and inconsistencies in *FITS* as described in the original papers are eliminated. The eventual goal is to submit this document to the IAU Commission 5 *FITS* Working Group for endorsement as a universal standard for *FITS*.

1.2 Scope

This standard specifies the organization and content of *FITS* data sets, including the header and data, for all standard *FITS* formats: Basic *FITS*, the random groups structure, and the ASCII tables extension. It also specifies minimum structural requirements for new extensions and general principles governing the creation of new extensions, giving as an example the draft proposal for a Binary Table Extension. For headers, it specifies the proper syntax for card images and defines required and reserved keywords. For data, it specifies character and value representations and the ordering of contents within the byte stream. It defines the general rules to which new extensions are required to conform.

1.3 Applicability

The IAU has recommended that all astronomical computer facilities support *FITS* for the interchange of binary data. All spacecraft projects and astrophysics data archives under the management of the Astrophysics Division of the National Aeronautics and Space Administration are required to make processed data available to users in the *FITS* format defined by this standard, unless the Astrophysics Division specifically determines otherwise. This standard may also be used to define the format for data transport in other disciplines, as may be determined by the appropriate authorities.

1.4 Organization and Recommendations

Following the definitions in Section 3, this document describes the overall organization of a *FITS* file, the contents of the first (primary) header and data, and the rules for creating new *FITS* extensions in Section 4. The next two sections provide additional details on the header and data, with a particular focus on the primary header. Section 5 provides details about header card image syntax and specifies those keywords required and reserved in a primary header. Section 6 describes how different data types are represented in *FITS*. The following sections describe the headers and data of two standard *FITS* structures, the now to be deprecated random groups records (Section 7) and the only current standard extension, ASCII Tables (Section 8). Throughout the document, deprecation of structures or syntax is noted where relevant. Files containing deprecated features are valid *FITS*, but these features should not be used in new files; the old files using them remain standard because of the principle that no change in *FITS* shall cause a valid *FITS* file to become invalid.

The Appendixes contain material that is not part of the standard. The first two provide illustrations of *FITS* practice. Appendix A provides an example of a conforming extension, the draft proposal for the *Binary Table* Extension [9]. The generally accepted recommendations for the expression of the logical *FITS* format on various physical media are provided in Appendix B as a guide to *FITS* practices. Appendix C lists the differences between this standard and the specifications of prior publications; it also identifies those ambiguities in the documents endorsed by the IAU on which this standard provides specific rules. The next four provide reference information: a tabular summary of the *FITS* keywords (Appendix D), a list of the ASCII character set and a subset designated ASCII text (Appendix E), the bit representation of the IEEE special values (Appendix F), and a list of the reserved extension type names (Appendix G).

Section 2

References

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

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

3. IAU. 1983, *Information Bulletin* No. 49.

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

5. Harten, R. H., Grosbøl, P., Greisen, E. W., and Wells, D. C. 1988, "The *FITS* Tables Extension," *Astron. Astrophys. Suppl.*, **73**, 365–372.

6. IAU. 1988, *Information Bulletin* No. 61.

7. McNally, D., ed. 1988, Transactions of the IAU, *Proceedings of the Twentieth General Assembly* (Dordrecht: Kluwer).

8. Wells, D. C. and Grosbøl, P. 1990, "Floating Point Agreement for *FITS*." (available from the NOST *FITS* Support Office)

9. Cotton, W. D. and Tody, D. B. 1991, "Binary Table Extension to *FITS*: A Proposal," preprint. (access instructions available from the NOST *FITS* Support Office)

10. ANSI. 1978, "American National Standard for Information Processing: Programming Language FORTRAN," ANSI X3.9–1978 (ISO 1539) (New York: American National Standards Institute, Inc.).

11. ANSI. 1977, "American National Standard for Information Processing: Code for Information Interchange," ANSI X3.4–1977 (ISO 646) (New York: American National Standards Institute, Inc.).

12. IEEE. 1985, "American National Standard – IEEE Standard for Binary Floating Point Arithmetic". ANSI/IEEE 754-1985 (New York: American National Standards Institute, Inc.).

13. ANSI. 1976, "American National Standard for Information Processing: Unrecorded Magnetic Tape," ANSI X3.40–1976 (New York: American National Standards Institute, Inc.).

14. ANSI. 1978, "American National Standard for Information Processing: Magnetic Tape Labels and File Structure," ANSI X3.27–1978 (New York: American National Standards Institute, Inc.).

15. "Going AIPS," 1990, National Radio Astronomy Observatory, Charlottesville, VA.

16. Ponz, J. D., Thompson, R. W., and Muñoz, J. R. 1992, "The *FITS* IMAGE Extension. A Proposal," preprint. (available from the NOST *FITS* Support Office)

17. Muñoz, J. R. "IUE data in FITS Format," 1989, ESA IUE Newsletter, **32**, 12-45.

Section 3

Definitions, Acronyms, and Symbols

␣ Used to designate an ASCII blank.

AIPS Abbreviation of Astronomical Image Processing System.

ANSI Abbreviation of American National Standards Institute.

Array A sequence of data values, of zero or more dimensions.

Array value The value of an element of an array in a *FITS* file, without the application of the associated linear transformation to derive the physical value.

ASCII Abbreviation of American National Standard Code for Information Interchange.

ASCII blank Hexadecimal 20.

ASCII character Any member of the 7-bit ASCII character set.

ASCII text ASCII characters hexadecimal 20-7E.

Basic FITS The *FITS* structure consisting of the primary header followed by a single primary data array.

Bit A single binary digit.

Byte A string of eight bits treated as a single entity.

Card image A sequence of 80 bytes containing ASCII text, treated as a logical record.

CfA Abbreviation of Harvard-Smithsonian Center for Astrophysics.

Conforming extension An extension whose keywords and organization adhere to the requirements for conforming extensions defined in Section 4.4.1 of this standard.

Deprecate To express earnest disapproval of. This term is used to refer to obsolete structures that ought not to be used but remain valid.

Entry A value or set of values (i. e., a vector) associated with a specific row or column in a table.

ESO Abbreviation of European Southern Observatory.

Extension A *FITS* HDU appearing after the primary HDU in a *FITS* file.

Extension name The identifier used to distinguish a particular extension HDU from others of the same type, appearing as the value of the `EXTNAME` keyword.

Extension type An extension format.

Field A set of zero or more table entries collectively described by a single format.

File A sequence of one or more records terminated by an end-of-file indicator appropriate to the medium.

FITS Abbreviation of Flexible Image Transport System.

FITS file A file with a format that conforms to the specifications in this document.

FITS logical record A record of 23040 bits corresponding to 2880 8-bit bytes within a *FITS* file.

FITS structure One of the components of a *FITS* file: the primary HDU, the random groups records, an extension, or, collectively, the special records following the last extension.

Floating point A number whose bit structure is composed of a mantissa and exponent, whose ASCII representation contains an explicit decimal point and may include a power-of-ten exponent.

Group parameter value The value of one of the parameters preceding a group in the random groups structure, without the application of the associated linear transformation.

GSFC Abbreviation of Goddard Space Flight Center.

Header A series of card images organized within one or more *FITS* Logical Records which describes structures and/or data which follow it in the *FITS* file.

Header and Data Unit (HDU) A data structure consisting of a Header and the data the Header describes. Note that an HDU may consist entirely of a header with no data records.

IAU Abbreviation of International Astronomical Union.

IUE Abbreviation of International Ultraviolet Explorer.

IEEE Abbreviation of Institute of Electrical and Electronic Engineers.

IEEE NaN Abbreviation of IEEE Not-a-Number value.

IEEE special values (-0, $\pm\infty$, NaN, denormalized).

Indexed keyword A keyword that is of the form of a fixed root with an appended integer count.

Keyword The first eight bytes of a header card image.

Mandatory keyword A keyword that must be used in all *FITS* files or a keyword required in conjunction with particular *FITS* structures.

Matrix A data array of two or more dimensions.

MIDAS Abbreviation of ESO-MIDAS, the European Southern Observatory – Munich Image Data Analysis System.

NOAO Abbreviation of National Optical Astronomy Observatories.

NOST Abbreviation of NASA/Science Office of Standards and Technology.

NRAO Abbreviation of National Radio Astronomy Observatory.

Physical value The value in physical units represented by a member of an array and possibly derived from the array value using the associated, but optional, linear transformation.

Picture element A single location within an image array.

Pixel Abbreviation of "picture element".

Primary data array The data array contained in the Primary HDU.

Primary header The first header in a *FITS* file, containing information on the overall contents of the file as well as on the primary data array.

Record A sequence of bits treated as a single logical entity.

Reference point The point along a given coordinate axis, given in units of pixel number, at which a value and increment are defined.

Reserved keyword An optional keyword that may be used only in the manner defined in this standard.

Special records A series of 23040-bit (2880 8-bit byte) records, following the primary HDU, whose internal structure does not otherwise conform to that for the primary HDU or to that specified for a conforming extension in this standard.

Standard extension A conforming extension whose header and data content are specified explicitly in this standard.

STScI Abbreviation of Space Telescope Science Institute.

Type name The value of the `XTENSION` keyword used to identify the type of the extension in the data following.

Valid value A member of a data array or table corresponding to an actual physical quantity.

Section 4

FITS File Organization

4.1 Overall

A *FITS* file shall be composed of the following *FITS* structures, in the order listed:

- Primary HDU
- Random Groups structure (optional; allowed only if there is no primary data array)
- Conforming Extensions (optional)
- Other special records (optional)

Each *FITS* structure shall consist of an integral number of *FITS* logical records. The primary HDU shall start with the first record of the *FITS* file. The first record of each subsequent *FITS* structure shall be the record immediately following the last record of the preceding *FITS* structure. The size of a *FITS* logical record shall be 23040 bits, corresponding to 2880 8-bit bytes.

4.2 Individual FITS Structures

The primary HDU and every extension HDU shall consist of an integral number of header records consisting of ASCII text, which may be followed by an integral number of data records. The first record of data shall be the record immediately following the last record of the header.

4.3 Primary Header and Data Array

The first component of a *FITS* file shall be the primary header. The primary header may, but need not be, followed by a primary data array. The presence or absence of a primary data array shall be indicated by the values of the NAXIS or NAXISn keywords in the primary header (Section 5.2.1.1).

4.3.1 Primary Header

The header of a primary HDU shall consist of a series of card images in ASCII text. All header records shall consist of 36 card images. Card images without information shall be filled with ASCII blanks (hexadecimal 20).

4.3.2 Primary Data Array

In *FITS* format, the primary data array shall consist of a single data array of 0-999 dimensions. The data values shall be a byte stream with no embedded fill or blank space. The first value shall be in the first position of the first primary data array record. The first value of each subsequent row of the array shall be in the position immediately following the last value of the previous row. Arrays of more than one dimension shall consist of a sequence such that the index along axis 1 varies most rapidly, that along axis 2 next most rapidly, and those along subsequent axes progressively less rapidly, with that along axis m, where m is the value of NAXIS, varying least rapidly; i.e., the elements of an array $A(x_1, x_2, \ldots, x_m)$ shall be in the order shown in Figure 4.1. The index count along each axis shall begin with 1 and increment by 1 up to the value of the NAXISn keyword (Section 5.2.1.1). If the data array does not fill the final record, the remainder of the record shall be filled with zero values with the same data representation as the values in the array. For IEEE floating point data, values of +0. shall be used to fill the remainder of the record.

4.4 Extensions

4.4.1 Requirements for Conforming Extensions

All extensions, whether or not further described in this standard, shall fulfill the following requirements to be in conformance with this *FITS* standard.

4.4.1.1 Identity

Each extension type shall have a unique type name, specified in the header according to the syntax codified in Section 5.2.1.2. To preclude conflict, extension type names must be registered with the IAU Commission 5 FITS Working Group. The NOST shall maintain and provide a list of the registered extensions.

$$
\begin{array}{c}
\mathrm{A}(1, 1, \ldots, 1), \\
\mathrm{A}(2, 1, \ldots, 1), \\
\vdots, \\
\mathrm{A}(\mathtt{NAXIS1}, 1, \ldots, 1), \\
\mathrm{A}(1, 2, \ldots, 1), \\
\mathrm{A}(2, 2, \ldots, 1), \\
\vdots, \\
\mathrm{A}(\mathtt{NAXIS1}, 2, \ldots, 1), \\
\vdots, \\
\mathrm{A}(1, \mathtt{NAXIS2}, \ldots, \mathtt{NAXISm}), \\
\vdots, \\
\mathrm{A}(\mathtt{NAXIS1}, \mathtt{NAXIS2}, \ldots, \mathtt{NAXISm})
\end{array}
$$

Figure 4.1: Arrays of more than one dimension shall consist of a sequence such that the index along axis 1 varies most rapidly and those along subsequent axes progressively less rapidly. Except for the location of the first element, array structure is independent of record structure.

4.4.1.2 Size Specification

The total number of bits in the data of each extension shall be specified in the header for that extension, in the manner prescribed in Section 5.2.1.2.

4.4.1.3 Compatibility with Existing FITS Files

No extension shall be constructed that invalidates existing *FITS* files.

4.4.2 Standard Extensions

A standard extension shall be a conforming extension whose organization and content are completely specified in this standard. Only one *FITS* format shall be approved for each type of data organization. Each standard extension shall have a unique type name.

4.4.3 Order of Extensions

An extension may follow the primary HDU (or random groups records if present) or another conforming extension. Standard extensions and other conforming extensions may appear in any order in a *FITS* file.

4.5 Special Records

The first 8 bytes of special records must not contain the string "XTENSION". It is recommended that they not contain the string "SIMPLE␣␣". The records must have the standard *FITS* 23040-bit record length. The contents of special records are not otherwise specified by this standard.

Section 5

Headers

5.1 Card Images

5.1.1 Syntax

Header card images shall consist of a keyword, an optional value, and an optional comment. Except where specifically stated otherwise in this standard, keywords may appear in any order.

5.1.2 Components

5.1.2.1 Keyword (bytes 1-8)

The keyword shall be a left justified, 8-character, blank filled, ASCII string with no embedded blanks. All digits (hexadecimal 30 to 39,“0123456789”) and upper case Latin alphabetic characters (hexadecimal 41 to 5A, “ABCDEFG HIJKLMN OPQRST UVWXYZ”) are permitted; no lower case characters shall be used. The underscore (hexadecimal 5F, “_”) and hyphen (hexadecimal 2D, “-”) are also permitted. No other characters are permitted. For indexed keywords, the counter shall not have leading zeroes.

5.1.2.2 Value Indicator (bytes 9-10)

If this field contains an ASCII “=␣”, the keyword must have an associated value field, unless it is a commentary keyword as defined in Section 5.2.2.4.

5.1.2.3 Value/Comment (bytes 11 - 80)

This field, when used, shall contain the value, if any, of the keyword, followed by optional comments. Separation of the value and comments by a slash (hexadecimal 2F, “/”), and a space between the value and the slash are strongly recommended. The value shall be the ASCII text representation of a string or constant, in the format specified in Section 5.3. The comment may contain any ASCII text.

5.2 Keywords

5.2.1 Mandatory Keywords

Mandatory keywords are required as described in the remainder of this subsection. They may be used only as described in this standard.

5.2.1.1 Principal

Principal mandatory keywords other than `SIMPLE` are required in all *FITS* headers. The `SIMPLE` keyword is required in all primary headers. The card images of any primary header must contain the keywords shown in Table 5.1 in the order given.

1	`SIMPLE`
2	`BITPIX`
3	`NAXIS`
4	`NAXISn`, `n` = 1, ..., `NAXIS`
	⋮
	(other keywords)
	⋮
last	`END`

Table 5.1: Principal mandatory keywords.

The total number of bits in the primary data array, exclusive of fill that is needed after the data to complete the last record (Section 4.1), must be given by the following expression:

$$\begin{aligned} \mathtt{NBITS} &= |\mathtt{BITPIX}| \times \\ &\quad (\mathtt{NAXIS1} \times \mathtt{NAXIS2} \times \cdots \times \mathtt{NAXISm}), \end{aligned} \tag{5.1}$$

where `NBITS` is non-negative and the number of bits excluding fill, `m` is the value of `NAXIS`, and `BITPIX` and the `NAXISn` represent the values associated with those keywords.

`SIMPLE` Keyword The value field shall contain a logical constant with the value `T` if the file conforms to this standard. This keyword is mandatory only for the primary header. A value of F signifies that the file does not conform to this standard in some significant way.

BITPIX Keyword The value field shall contain an integer. The absolute value is used in computing the sizes of data structures. It shall specify the number of bits that represent a data value. The only valid values of BITPIX are given in Table 5.2.

Value	Data Represented
8	Character or unsigned binary integer
16	16-bit twos-complement binary integer
32	32-bit twos-complement binary integer
-32	IEEE single precision floating point
-64	IEEE double precision floating point

Table 5.2: Interpretation of valid BITPIX value.

NAXIS Keyword The value field shall contain a non-negative integer no greater than 999, representing the number of axes in an ordinary data array. A value of zero signifies that no data follow the header in the HDU.

NAXISn Keywords The value field of this indexed keyword shall contain a non-negative integer, representing the number of positions along axis n of an ordinary data array. The NAXISn must be present for all values n = 1,...,NAXIS. A value of zero for any of the NAXISn signifies that no data follow the header in the HDU. If NAXIS is equal to 0, there should not be any NAXISn keywords.

END Keyword This keyword has no associated value. Columns 9-80 shall be filled with ASCII blanks.

5.2.1.2 Conforming Extensions

The use of extensions necessitates a single additional keyword in the primary header of the *FITS* file.

EXTEND Keyword If the *FITS* file may contain extensions, a card image with the keyword EXTEND and the value field containing the logical value T must appear in the primary header immediately after the last NAXISn card image, or, if NAXIS=0, the NAXIS card image. The presence of this keyword with the value `T` in the primary header does not require that extensions be present.

The card images of any extension header must use the keywords defined in Table 5.3 in the order specified. This organization is required for any conforming extension, whether or not further specified in this standard.

1	XTENSION
2	BITPIX
3	NAXIS
4	NAXISn, n = 1, ..., NAXIS
	⋮
	(other keywords, including ...)
	PCOUNT
	GCOUNT
	⋮
last	END

Table 5.3: Mandatory keywords in conforming extensions.

The total number of bits in the extension data array exclusive of fill that is needed after the data to complete the last record (Section 4.1) such as that for the primary data array (Section 4.3.2) must be given by the following expression:

$$\begin{aligned} \mathtt{NBITS} &= |\mathtt{BITPIX}| \times \mathtt{GCOUNT} \times \\ &\quad (\mathtt{PCOUNT} + \mathtt{NAXIS1} \times \mathtt{NAXIS2} \times \cdots \times \mathtt{NAXISm}), \end{aligned} \tag{5.2}$$

where `NBITS` is non-negative and the number of bits excluding fill, `m` is the value of `NAXIS`, and `BITPIX`, `GCOUNT`, `PCOUNT`, and the `NAXISn` represent the values associated with those keywords.

`XTENSION` Keyword The value field shall contain a character string giving the name of the extension type. This keyword is mandatory for an extension header and must not appear in the primary header. For an extension that is not a standard extension, the type name must not be the same as that of a standard extension.

The IAU Commission 5 FITS Working Group may specify additional type names that must be used only to identify specific types of extensions; the full list shall be available from the NOST.

PCOUNT Keyword The value field shall contain an integer that shall be used in any way appropriate to define the data structure, consistent with equation 5.2.

GCOUNT Keyword The value field shall contain an integer that shall be used in any way appropriate to define the data structure, consistent with equation 5.2.

5.2.2 Other Reserved Keywords

These keywords are optional but may be used only as defined in this standard. These keywords apply to any *FITS* structure except where specifically further restricted.

5.2.2.1 Keywords Describing the History or Physical Construction of the HDU

DATE Keyword The value field shall contain a character string giving the date on which the HDU was created, in the form DD/MM/YY, where DD shall be the day of the month, MM the month number, with January given by 01 and December by 12, and YY the last two digits of the year. Specification of the date using Universal Time is recommended. Copying of a *FITS* file does not require changing any of the keyword values in the file's HDUs.

ORIGIN Keyword The value field shall contain a character string identifying the organization creating the *FITS* file.

BLOCKED Keyword This keyword may be used only in the primary header. It shall appear within the first 36 card images of the *FITS* file. (Note: This keyword thus cannot appear if NAXIS is greater than 31, or if NAXIS greater than 30 and the EXTEND keyword is present.) Its presence with the required logical value of T advises that the physical block size of the *FITS* file on which it appears may be an integral multiple of the logical record length, and not necessarily equal to it. Physical block size and logical record length may be equal even if this keyword is present or unequal if it is absent. It is reserved primarily to prevent its use with other meanings. The issuance of this standard deprecates the BLOCKED keyword.

5.2.2.2 Keywords Describing Observations

DATE-OBS Keyword The value field shall contain a character string giving the day on which the observations represented by the array were made, in the form DD/MM/YY, where DD shall be the day of the month, MM the month number, with January given by 01 and December by 12, and YY the last two digits of the year. Specification of the date using Universal Time is recommended.

TELESCOP Keyword The value field shall contain a character string identifying the telescope used to acquire the data contained in the array.

INSTRUME Keyword The value field shall contain a character string identifying the instrument used to acquire the data contained in the array.

OBSERVER Keyword The value field shall contain a character string identifying who acquired the data associated with the header.

OBJECT Keyword The value field shall contain a character string giving the name of the object observed.

EQUINOX Keyword The value field shall contain a floating point number giving the equinox in years for the celestial coordinate system in which positions given in either the header or data are expressed.

EPOCH Keyword The value field shall contain a floating point number giving the equinox in years for the celestial coordinate system in which positions given in either the header or data are expressed. This document deprecates the use of the EPOCH keyword and thus it shall not be used in *FITS* files created after the adoption of this standard; rather, the EQUINOX keyword shall be used.

5.2.2.3 Bibliographic Keywords

AUTHOR Keyword The value field shall contain a character string identifying who compiled the information in the data associated with the header. This keyword is appropriate when the data originate in a published paper or are compiled from many sources.

REFERENC Keyword The value field shall contain a character string citing a reference where the data associated with the header are published.

5.2.2.4 Commentary Keywords

COMMENT Keyword This keyword shall have no associated value; columns 9-80 may contain any ASCII text. Any number of COMMENT card images may appear in a header.

HISTORY Keyword This keyword shall have no associated value; columns 9-80 may contain any ASCII text. The text should contain a history of steps and procedures associated with the processing of the associated data. Any number of HISTORY card images may appear in a header.

Keyword Field is Blank Columns 1-8 contain ASCII blanks. Columns 9-80 may contain any ASCII text. Any number of card images with blank keyword fields may appear in a header.

5.2.2.5 Array Keywords

These keywords are used to describe the contents of an array, either alone or in a series of random groups. They are optional, but if they appear in the header describing an array or groups, they must be used as defined in this section of this standard. They shall not be used in headers describing other structures unless the meaning is the same as that for a primary or groups array.

`BSCALE` **Keyword** This keyword shall be used, along with the `BZERO` keyword, when the array pixel values are not the true physical values, to transform the primary data array values to the true physical values they represent, using equation 5.3. The value field shall contain a floating point number representing the coefficient of the linear term in the scaling equation, the ratio of physical value to array value at zero offset. The default value for this keyword is 1.0.

`BZERO` **Keyword** This keyword shall be used, along with the `BSCALE` keyword, when the array pixel values are not the true physical values, to transform the primary data array values to the true values. The value field shall contain a floating point number representing the physical value corresponding to an array value of zero. The default value for this keyword is 0.0.

The transformation equation is as follows:

$$\text{physical value} = \mathtt{BZERO} + \mathtt{BSCALE} \times \text{array value} \tag{5.3}$$

`BUNIT` **Keyword** The value field shall contain a character string, describing the physical units in which the quantities in the array, after application of `BSCALE` and `BZERO`, are expressed. Use of the units defined in the IAU Style Manual [7] is recommended.

`BLANK` **Keyword** This keyword shall be used only in headers with positive values of `BITPIX` (i.e., in arrays with integer data). Columns 1-8 contain the string, "`BLANK␣␣␣`" (ASCII blanks in columns 6-8). The value field shall contain an integer that specifies the representation of array values whose physical values are undefined.

`CTYPEn` **Keywords** The value field shall contain a character string, giving the name of the coordinate represented by axis `n`. Where this coordinate represents a physical quantity, units defined in the IAU Style Manual [7] are recommended.

CRPIXn Keywords The value field shall contain a floating point number, identifying the location of a reference point along axis n, in units of the axis index. This value is based upon a counter that runs from 1 to NAXISn with an increment of 1 per pixel. The reference point value need not be that for the center of a pixel nor lie within the actual data array. Use comments to indicate the location of the index point relative to the pixel.

CRVALn Keywords The value field shall contain a floating point number, giving the value of the coordinate specified by the CTYPEn keyword at the reference point CRPIXn.

CDELTn Keywords The value field shall contain a floating point number, giving the partial derivative of the coordinate specified by the CTYPEn keywords with respect to the pixel index, evaluated at the reference point CRPIXn, in units of the coordinate specified by the CTYPEn keyword.

CROTAn Keywords This keyword is used to indicate a rotation from a standard coordinate system described by the CTYPEn to a different coordinate system in which the values in the array are actually expressed. Rules for such rotations are not further specified in this standard; the rotation should be explained in comments. The value field shall contain a floating point number, giving the rotation angle in degrees between axis n and the direction implied by the coordinate system defined by CTYPEn.

DATAMAX Keyword The value field shall always contain a floating point number, regardless of the value of BITPIX. This number shall give the maximum valid physical value represented in the array, exclusive of any special values.

DATAMIN Keyword The value field shall always contain a floating point number, regardless of the value of BITPIX. This number shall give the minimum valid physical value represented in the array, exclusive of any special values.

5.2.2.6 Extension Keywords

These keywords are used to describe an extension.

EXTNAME Keyword The value field shall contain a character string, to be used to distinguish among different extensions of the same type, i.e., with the same value of XTENSION, in a *FITS* file.

EXTVER Keyword The value field shall contain an integer, to be used to distinguish among different extensions in a *FITS* file with the same type and name, i.e., the same values for XTENSION and EXTNAME. The values need not start with 1 for the first extension with a particular value of EXTNAME and need not be in sequence for subsequent values. If the EXTVER keyword is absent, the file should be treated as if the value were 1.

EXTLEVEL Keyword The value field shall contain an integer, specifying the level in a hierarchy of extension levels of the extension header containing it. The value shall be 1 for the highest level; levels with a higher value of this keyword shall be subordinate to levels with a lower value. If the EXTLEVEL keyword is absent, the file should be treated as if the value were 1.

5.2.3 Additional Keywords

5.2.3.1 Requirements

New keywords may be devised in addition to those described in this standard, so long as they are consistent with the generalized rules for keywords and do not conflict with mandatory or reserved keywords.

5.2.3.2 Restrictions

No keyword in the primary header shall specify the presence of a specific extension in a *FITS* file; only the EXTEND keyword described in Section 5.2.1.2 shall be used to indicate the possible presence of extensions. No keyword in either the primary or extension header shall explicitly refer to the physical block size, other than the BLOCKED keyword of Section 5.2.2.1.

5.3 Value

5.3.1 General Format Requirements

Unless otherwise specified, the value field must be written in a notation consistent with the list-directed read operations in ANSI FORTRAN-77 [10]. The structure shall be determined by the type of the variable. The fixed format is required for values of mandatory keywords and recommended for values of all others. This standard imposes no requirements on case sensitivity of character strings other than those explicitly specified.

5.3.2 Fixed Format

5.3.2.1 Character String

If the value is a character string, column 11 shall contain a single quote (hexadecimal code 27, "'"); the string shall follow, starting in column 12, followed by a closing single quote (also hexadecimal code 27) that should not occur before column 20 and must occur in or before column 80. Reading the data values in a *FITS* file should not require decoding any more than the first eight characters of a character string value of a keyword. The character string shall be composed only of ASCII text. A single quote is represented within a string as two successive single quotes, e.g., O'HARA = `'O''HARA'`. Leading blanks are significant; trailing blanks are not.

5.3.2.2 Logical Variable

If the value is a logical constant, it shall appear as a `T` or `F` in column 30.

5.3.2.3 Integer

If the value is an integer, the ASCII representation shall appear right justified in columns 11-30. For a complex integer, the imaginary part shall be right justified in columns 31-50. This format for complex integers does not correspond to ANSI FORTRAN-77 list-directed read.

5.3.2.4 Real Floating Point Number

If the value is a real floating point number, the ASCII representation shall appear in columns 11-30. Letters in the exponential form shall be upper case. The value shall be right justified, and the decimal point must appear. Note: The full precision of 64-bit values can not be expressed as a single value using the fixed format.

5.3.2.5 Complex Floating Point Number

If the value is a complex floating point number, the ASCII representation of the real part shall appear in the same manner as a real floating point number (see above). The ASCII representation of the imaginary part shall appear in columns 31 - 50. Letters in the exponential form shall be upper case. The value shall be right justified, and the decimal point must appear. This format for complex floating point numbers does not correspond to ANSI FORTRAN-77 list-directed read. Note: The full precision of 64-bit values can not be expressed as a single value using the fixed format.

Section 6

Data Representation

Primary and extension data shall be represented in one of the formats described in this section. *FITS* data shall be interpreted to be a byte stream. Bytes are in order of decreasing significance. The byte that includes the sign bit shall be first, and the byte that has the ones bit shall be last.

6.1 Characters

Each character shall be represented by one byte. A character shall be represented by its 7-bit ASCII [11] code in the low order seven bits in the byte. The high-order bit shall be zero.

6.2 Integers

6.2.1 Eight-bit

Eight-bit integers shall be unsigned binary integers, contained in one byte.

6.2.2 Sixteen-bit

Sixteen-bit integers shall be twos-complement signed binary integers, contained in two bytes.

6.2.3 Thirty-two-bit

Thirty-two-bit integers shall be twos-complement signed binary integers, contained in four bytes.

6.3 IEEE-754 Floating Point

Transmission of 32- and 64-bit floating point data within the *FITS* format shall use the ANSI/IEEE-754 standard [12]. `BITPIX = -32` and `BITPIX = -64` signify 32- and 64-bit IEEE floating point numbers, respectively; the absolute value of `BITPIX` is used for computing the sizes of data structures. The full IEEE set of number forms is allowed for *FITS* interchange, including all special values (e.g., the "Not-a-Number" cases). The order of the bytes will be sign and exponent first, followed by the mantissa bytes in order of decreasing significance. The `BLANK` keyword should not be used when `BITPIX = -32` or `-64`. Use of the `BSCALE` and `BZERO` keywords is not recommended.

6.3.1 Thirty-two-bit Floating Point

6.3.1.1 Structure

Table 6.1 describes the bit structure of 32-bit floating point standard numeric values.

Bit Positions (first to last)	Content
1	sign
2 - 9	exponent
10 - 32	mantissa

Table 6.1: Content of 32-bit floating point bit positions.

6.3.1.2 Interpretation

Standard numeric values of IEEE 32-bit floating point numbers are interpreted according to the following rule:

$$value = (-1)^{\mathrm{sign}} \times 2^{(\mathrm{exponent}-127)} \times \mathrm{mantissa} \qquad (6.1)$$

The IEEE NaN (Not-a-Number) values shall be used to represent undefined values. All IEEE special values are recognized.

6.3.2 Sixty-four-bit Floating Point

6.3.2.1 Structure

Table 6.2 describes the bit structure of 64-bit floating point standard numeric values.

Bit Positions (first to last)	Content
1	sign
2 - 12	exponent
13 - 64	mantissa

Table 6.2: Content of 64-bit floating point bit positions.

6.3.2.2 Interpretation

Standard numeric values for IEEE 64-bit floating point numbers are interpreted according to the following rule:

$$value = (-1)^{\text{sign}} \times 2^{(\text{exponent}-1023)} \times \text{mantissa} \qquad (6.2)$$

The IEEE NaN (Not-a-Number) values shall be used to represent undefined values. All IEEE special values are recognized.

Section 7

Random Groups Structure

Although it is standard *FITS*, the random groups structure has been used almost exclusively for applications in radio interferometry; outside this field, few *FITS* readers can read data in random groups format. A proposed binary tables extension will eventually be able to accommodate the structure described by random groups. While existing *FITS* files use the format, and it is therefore included in this standard, its use for future applications is deprecated by this document.

7.1 Keywords

7.1.1 Mandatory Keywords

If the random groups format records follow the primary header, the card images of the primary header must use the keywords defined in Table 7.1 in the order specified.

The total number of bits in the random groups records exclusive of the fill described in Section 7.2 must be given by the following expression:

$$\begin{aligned} \texttt{NBITS} &= |\texttt{BITPIX}| \times \texttt{GCOUNT} \times \\ &\quad (\texttt{PCOUNT} + \texttt{NAXIS2} \times \texttt{NAXIS3} \times \cdots \times \texttt{NAXISm}), \end{aligned} \tag{7.1}$$

where `NBITS` is non-negative and the number of bits excluding fill, `m` is the value of `NAXIS`, and `BITPIX`, `GCOUNT`, `PCOUNT`, and the `NAXISn` represent the values associated with those keywords.

7.1.1.1 `SIMPLE` Keyword

The card image containing this keyword is structured in the same way as if a primary data array were present (Section 5.2.1).

1	SIMPLE
2	BITPIX
3	NAXIS
4	NAXIS1
5	NAXISn, n=2, ..., value of NAXIS
	⋮
	(other keywords, which must include ...)
	GROUPS
	PCOUNT
	GCOUNT
	⋮
last	END

Table 7.1: Mandatory keywords in primary header preceding random groups.

7.1.1.2 BITPIX Keyword

The card image containing this keyword is structured as prescribed in Section 5.2.1.

7.1.1.3 NAXIS Keyword

The value field shall contain an integer ranging from 1 to 999, representing one more than the number of axes in each data array.

7.1.1.4 NAXIS1 Keyword

The value field shall contain the integer 0, a signature of random groups format indicating that there is no primary data array.

7.1.1.5 NAXISn Keywords (n=2, ..., value of NAXIS)

The value field shall contain an integer, representing the number of positions along axis n-1 of the data array in each group.

7.1.1.6 GROUPS Keyword

The value field shall contain the logical constant T. The value T associated with this keyword implies that random groups records are present.

7.1.1.7 `PCOUNT` Keyword

The value field shall contain an integer equal to the number of parameters preceding each group.

7.1.1.8 `GCOUNT` Keyword

The value field shall contain an integer equal to the number of random groups present.

7.1.1.9 `END` Keyword

The card image containing this keyword is structured as described in Section 5.2.1.

7.1.2 Reserved Keywords

7.1.2.1 `PTYPEn` Keywords

The value field shall contain a character string giving the name of parameter `n`. If the `PTYPEn` keywords for more than one value of `n` have the same associated name in the value field, then the data value for the parameter of that name is to be obtained by adding the derived data values of the corresponding parameters. This rule provides a mechanism by which a random parameter may have more precision than the accompanying data array members; for example, by summing two 16-bit values with the first scaled relative to the other such that the sum forms a number of up to 32-bit precision.

7.1.2.2 `PSCALn` Keywords

This keyword shall be used, along with the `PZEROn` keyword, when the n^{th} *FITS* group parameter value is not the true physical value, to transform the group parameter value to the true physical values it represents, using equation 7.2. The value field shall contain a floating point number representing the coefficient of the linear term in equation 7.2, the scaling factor between true values and group parameter values at zero offset. The default value for this keyword is 1.0.

7.1.2.3 `PZEROn` Keywords

This keyword shall be used, along with the `PSCALn` keyword, when the n^{th} *FITS* group parameter value is not the true physical value, to transform the group parameter value to the physical value. The value field shall contain a floating point number, representing the true value corresponding to a group parameter value of zero. The default value for this keyword is 0.0. The transformation equation is as follows:

$$\text{physical value} = \texttt{PZEROn} + \texttt{PSCALn} \times groupparametervalue \qquad (7.2)$$

7.2 Data Sequence

Random groups data shall consist of a set of groups. The number of groups shall be specified by the GCOUNT keyword in the associated header record. Each group shall consist of the number of parameters specified by the PCOUNT keyword followed by an array with the number of members GMEM given by the following expression:

$$\mathtt{GMEM} = (\mathtt{NAXIS2} \times \mathtt{NAXIS3} \times \cdots \times \mathtt{NAXISm}). \qquad (7.3)$$

where GMEM is the number of members in the data array in a group, m is the value of NAXIS, and the NAXISn represent the values associated with those keywords.

The first parameter of the first group shall appear in the first location of the first data record. The first element of each array shall immediately follow the last parameter associated with that group. The first parameter of any subsequent group shall immediately follow the last member of the array of the previous group. The arrays shall be organized internally in the same way as an ordinary primary data array. If the groups data do not fill the final record, the remainder of the record shall be filled with zero values in the same way as a primary data array (Section 4.3.2). If random groups records are present, there shall be no primary data array.

7.3 Data Representation

Permissible data representations are those listed in Section 6. Parameters and members of associated data arrays shall have the same representation. Should more precision be required for an associated parameter than for a member of a data array, the parameter shall be divided into two or more addends, represented by the same value for the PTYPEn keyword. The value shall be the sum of the physical values, which may have been obtained from the group parameter values using the PSCALn and PZEROn keywords.

Section 8

Standard Extensions

8.1 ASCII Tables Extension

Data shall appear as an ASCII Tables extension if the primary header of the *FITS* file has the keyword `EXTEND` set to `T` and the first keyword of that extension header has `XTENSION=␣'TABLE␣␣␣'`.

8.1.1 Mandatory Keywords

The card images in the header of an ASCII Tables Extension must use the keywords defined in Table 8.1 in the order specified.

`XTENSION` **Keyword** The value field shall contain the character string `'TABLE␣␣␣'`.

`BITPIX` **Keyword** The value field shall contain the integer 8, denoting that the array contains ASCII characters.

`NAXIS` **Keyword** The value field shall contain the integer 2, denoting that the included data array is two-dimensional: rows and columns.

`NAXIS1` **Keyword** The value field shall contain a non-negative integer, giving the number of ASCII characters in each row of the table.

`NAXIS2` **Keyword** The value field shall contain a non-negative integer, giving the number of rows in the table.

`PCOUNT` **Keyword** The value field shall contain the integer 0.

1	XTENSION
2	BITPIX
3	NAXIS
4	NAXIS1
5	NAXIS2
6	PCOUNT
7	GCOUNT
8	TFIELDS
	⋮
	(other keywords, which must include ...)
	TBCOLn, n=1,2,...,k where k is the value of TFIELDS
	TFORMn, n=1,2,...,k where k is the value of TFIELDS
	⋮
last	END

Table 8.1: Mandatory keywords in ASCII tables extensions.

GCOUNT Keyword The value field shall contain the integer 1; the data records contain a single table.

TFIELDS Keyword The value field shall contain a non-negative integer representing the number of fields in each row. The maximum permissible value is 999.

TBCOLn Keywords The value field of this indexed keyword shall contain an integer specifying the column in which field n starts. The first column of a row is numbered 1.

TFORMn Keywords The value field of this indexed keyword shall contain a character string describing the FORTRAN-77 [10] format in which field n is coded. The formats in Table 8.2 are permitted for encoding.

Repetition of a format from one field to the next must be indicated by using separate pairs of TBCOLn and TFORMn keywords for each field; format repetition may not be indicated by prefixing the format by a number.

END Keyword This keyword has no associated value. Columns 9-80 shall contain ASCII blanks.

Field Value	Data Type
Aw	Character
Iw	Integer
Fw.d	Single precision real
Ew.d	Single precision real, exponential notation
Dw.d	Double precision real, exponential notation

Table 8.2: Valid `TFORMn` format values in `TABLE` extensions.

8.1.2 Other Reserved Keywords

In addition to the mandatory keywords defined in section 8.1.1, these keywords may be used to describe the structure of an ASCII Tables data array. They are optional, but if they appear within an ASCII Tables extension header, they must be used as defined in this section of this standard.

`TSCALn` Keywords This indexed keyword shall be used, along with the `TZEROn` keyword, when the quantity in field `n` does not represent a true physical quantity. The value field shall contain a floating point number representing the coefficient of the linear term in equation 8.1, which must be used to compute the true physical value of the field. The default value for this keyword is 1.0. This keyword may not be used for A-format fields.

`TZEROn` Keywords This indexed keyword shall be used, along with the `TSCALn` keyword, when the quantity in field `n` does not represent a true physical quantity. The value field shall contain a floating point number representing the zero point for the true physical value of field `n`. The default value for this keyword is 0.0. This keyword may not be used for A-format fields.

The transformation equation used to compute a true physical value from the quantity in field `n` is

$$\text{physical value} = \texttt{TZEROn} + \texttt{TSCALn} \times \text{field value}. \quad (8.1)$$

`TNULLn` Keywords The value field for this indexed keyword shall contain the character string that represents an undefined value for field `n`. The string is implicitly blank filled to the width of the field.

`TTYPEn` Keywords The value field for this indexed keyword shall contain a character string, giving the name of field `n`. It is recommended that only letters, digits, and underscore (hexadecimal code 5F, "_") be used in the name. However, string comparisons with the values of `TTYPEn` keywords should not be case sensitive. The use of identical names for different fields should be avoided.

`TUNITn` Keywords The value field shall contain a character string describing the physical units in which the quantity in field `n`, after any application of `TSCALn` and `TZEROn`, is expressed. Use of the units defined in the IAU Style Manual [7] is recommended.

8.1.3 Data Sequence

The table is constructed from a two-dimensional array of ASCII characters. The row length and the number of rows shall be those specified, respectively, by the `NAXIS1` and `NAXIS2` keywords of the associated header records. The number of characters in a row and the number of rows in the table shall determine the size of the character array. Every row in the array shall have the same number of characters. The first character of the first row shall be at the start of the record immediately following the last header record. The first character of subsequent rows shall follow immediately the character at the end of the previous row, independent of the record structure. The positions in the last data record after the last character of the last row of the data array shall be filled with ASCII blanks.

8.1.4 Fields

Each row in the array shall consist of a sequence of fields, with one entry in each field. For every field, the FORTRAN-77 format of the information contained, location in the row of the beginning of the field and (optionally) the field name, shall be specified in keywords of the associated header records. A separate format keyword must be provided for each field. The location and format of fields shall be the same for every row. Fields may overlap. There may be characters in a table row that are not included in any field.

8.1.5 Entries

All data in an ASCII tables extension record shall be ASCII text in formats that conform to the rules for fixed field input in ANSI FORTRAN-77 [10] format, including implicit decimal points. The only possible formats shall be those specified in Table 8.2. If values of -0 and +0 must be distinguished, then the sign character should appear in a separate field in character format. `TNULLn` keywords may be used to specify a character string that represents an undefined value in each field. The characters representing an undefined value may differ from field to field but must be the same within a field. ASCII Table extension data should be decoded as though the FORTRAN-77 OPEN statement specifier BLANK is set to NULL. That is, blanks within the fields are not to be interpreted as zeroes; zeroes must be given explicitly.

8.2 Other Standard Extensions

At the effective date of this standard there are no other standard extensions.

Section 9

Restrictions on Changes

Any structure that is a valid *FITS* structure shall remain a valid *FITS* structure at all future times. Use of certain valid *FITS* structures may be deprecated by this or future *FITS* standard documents.

Appendix A

Draft Proposal for Binary Table Extension

(This Appendix is not part of the NOST *FITS* Standard but is included for informational purposes only.)

This appendix contains a draft proposal for a Binary Table extension, type name "BINTABLE", developed by W. D. Cotton (NRAO) and D. Tody (NOAO), dated September 20, 1991. With their permission, that proposal [9] is reproduced nearly verbatim; the only changes are those required for stylistic consistency with the rest of this document. The BINTABLE extension has been developed from the earlier A3DTABLE extension implemented in AIPS by NRAO. It supports all features of the earlier, more limited extension. Binary tables files have been successfully exchanged between a number of organizations, among them NRAO, ESO, STScI, NASA/GSFC and CfA. However, some features of the binary tables extension, such as vector table elements, are not supported by all *FITS* readers, and other *FITS* readers may not support binary tables at all. The extension is now being considered by the regional *FITS* committees as part of the process leading to formal approval by the IAU FITS Working Group. Because it is becoming widely used, and because it illustrates an application of the rules for conforming extensions, the text of the proposal is included as the remainder of this Appendix.

A.1 Abstract

This paper describes the *FITS* binary tables which 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. In addition, each entry is allowed to be a single dimensioned array. Numeric data are kept in IEEE formats.

A.2 Introduction

The Flexible Image Transport System (*FITS*) [1], [2] has been used for a number of years both as a means of transporting data between computers and/or processing systems and as an archival format for a variety of astronomical data. The success of this system has resulted in the introduction of enhancements. In particular, considerable use has been made of the records following the "main" data file. Grosbøl *et al.* [4] introduced a generalized header format for extension "files" following the "main" data file, but in the same physical file. Harten *et al.* [5] defined an ASCII table structure which could convey information that could be conveniently printed as a table. This paper generalizes the ASCII tables and defines an efficient means for conveying a wide variety of data structures as "extension" files.

A.3 Binary Tables

The binary tables are tables in the sense that they are organized into rows and columns. They are multi-dimensional since an entry, or set of values associated with a given row and column, can be an array of arbitrary size. These values are represented in a standardized binary form. Each row in the table contains an entry for each column. This entry may be one of a number of different data types, 8 bit unsigned integers, 16 or 32 bit signed integers, logical, character, bit, 32 or 64 bit floating point or complex values. The datatype and dimensionality are independently defined for each column but each row must have the same structure. Additional information associated with the table may be included in the table header as keyword/value pairs.

The binary tables come after the "main" data file, if any, in a *FITS* file and follow the standards for generalized extension tables defined in [4].

The use of the binary tables requires the use of a single additional keyword in the main header:

`EXTEND` (logical) if true (ASCII `'T'`) indicates that there *may* be extension files following the data records and, if there are, that they conform to the generalized extension file header standards.

A.4 Table Header

The table header begins at the first byte in the first record following the last record of main data (if any) or following the last record of the previous extension file. The format of the binary table header is such that a given *FITS* reader can decide if it wants (or understands) it and can skip the table if not.

A table header consists of one or more 2880 8-bit byte logical records each containing 36 80-byte "card images" in the form:

```
keyword = value    / comment
```

where the keyword begins in column 1 and contains up to eight characters and the value begins in column 10 or later. Keyword/value pairs in binary table headers conform to standard *FITS* usage.

The number of columns in the table is given by the value associated with keyword `TFIELDS`. The type, dimensionality, labels, units, blanking values, and display formats for entries in column `nnn` may be defined by the values associated with the keywords `TFORMnnn`, `TTYPEnnn`, `TUNITnnn`, `TNULLnnn`, and `TDISPnnn`. Of these only `TFORMnnn` is required but the use of `TTYPEnnn` is strongly recommended. An entry may be omitted from the table, but still defined in the header, by using a zero element count in the `TFORMnnn` entry.

The required keywords `XTENSION`, `BITPIX`, `NAXIS`, `NAXIS1`, `NAXIS2`, `PCOUNT`, `GCOUNT` and `TFIELDS` must be in order; other keywords follow these in an arbitrary order. The required keywords in a binary table header record are:

`XTENSION` **(character)** indicates the type of extension file, this must be the first keyword in the header. This is `'BINTABLE'` for the binary tables.

`BITPIX` **(integer)** gives the number of bits per "pixel" value. For binary tables this value is 8.

`NAXIS` **(integer)** gives the number of "axes"; this value is 2 for binary tables.

`NAXIS1` **(integer)** gives the number of 8 bit bytes in each "row". This should correspond to the sum of the values defined in the `TFORMnnn` keywords.

`NAXIS2` **(integer)** gives the number of rows in the table.

`PCOUNT` **(integer)** is used to tell the number of bytes *following* the regular portion of the table. These bytes are allowed but no meaning is attached to them in this document. `PCOUNT` should normally be 0 for binary tables (see however Section A.9.2).

`GCOUNT` **(integer)** gives the number of groups of data defined as for the random group main data records. This is 1 for binary tables.

`TFIELDS` **(integer)** gives the number of fields (columns) present in the table.

TFORMnnn[1] **(character)** gives the size and data type of field **nnn**. Allowed values of **nnn** range from 1 to the value associated with **TFIELDS**. Allowed values of **TFORMnnn** are of the form rL, rX, rI, rJ, rA, rE, rD, rB, rC, rM, or rP (logical, bit, 16-bit integers, 32-bit integers, characters, single precision, double precision, unsigned bytes, complex pair of single precision values, double complex pair of double precision values and variable length array descriptor [64 bits]) where r=number of elements. If the element count is absent, it is assumed to be 1. A value of zero is allowed. Note: additional characters may follow the datatype code character but they are not defined in this document.

The number of bytes determined from summing the **TFORMnnn** values should equal **NAXIS1** but **NAXIS1** should be used as the definition of the actual length of the row.

END is always the last keyword in a header. The remainder of the *FITS* logical (2880–byte) record following the **END** keyword is blank filled.

The optional standard keywords are:

EXTNAME (character) can be used to give a name to the extension file to distinguish it from other similar files. The name may have a hierarchical structure giving its relation to other files (e.g., "map1.cleancomp")

EXTVER (integer) is a version number which can be used with **EXTNAME** to identify a file.

EXTLEVEL (integer) specifies the level of the extension file in a hierarchical structure. The default value for **EXTLEVEL** should be 1.

TTYPEnnn (character) gives the label for field **nnn**.

TUNITnnn (character) gives the physical units of field **nnn**.

TSCALnnn (floating) gives the scale factor for field **nnn**. True_value = *FITS*_value * **TSCAL** + **TZERO**. Note: **TSCALnnn** and **TZEROnnn** are not defined for A, L, P, or X format fields. Default value is 1.0.

TZEROnnn (floating) gives the offset for field **nnn**. (See **TSCALnnn**.) Default value is 0.0.

TNULLnnn (integer) gives the undefined value for integer (B, I, and J) field **nnn**. Section A.6 discusses the conventions for indicating invalid data of other data types.

[1] The "nnn" in keyword names indicates an integer index in the range 1 - 999. The integer is left justified with no leading zeroes, e.g. TFORM1, TFORM19, etc.

`TDISPnnn` **(character)** gives the Fortran 90 format suggested for the display of field `nnn`. Each byte of bit and byte arrays will be considered to be a signed integer for purposes of display. The allowed forms are Aw, Lw, Iw.m, Bw.m (Binary, integers only), Ow.m (Octal, integers only), Zw.m (Hexadecimal, integers only) Fw.d, Ew.dEe, ENw.d, ESw.d, Gw.dEe, and Dw.dEe where w is the width of the displayed value in characters, m is the minimum number of digits possibly requiring leading zeroes, d is the number of digits to the right of the decimal, and e is the number of digits in the exponent. All entries in a field are displayed with a single, repeated format. If Fortran 90 formats are not available to a reader which prints a table then equivalent FORTRAN 77 formats may be substituted. Any `TSCALnnn` and `TZEROnnn` values should be applied before display of the value. Note that characters and logical values may be null (zero byte) terminated.

`TDIMnnn` **(character)** This keyword is reserved for use by the convention described in Section A.9.1.

`THEAP` **(integer)** This keyword is reserved for use by the convention described in Section A.9.2.

`AUTHOR` **(character)** gives the name of the author or creator of the table.

`REFERENC` **(character)** gives the reference for the table.

Nonstandard keyword/value pairs adhering to the *FITS* keyword standards are allowed although a reader may chose to ignore them.

A.5 Conventions for Multidimensional Arrays

There is commonly a need to use data structures more complex than the one dimensional definition of the table entries defined for this table format. Multidimensional arrays, or more complex structures, may be implemented by passing dimensions or other structural information as either column entries or keywords in the header. Passing the dimensionality as column entries has the advantage that the array can have variable dimension (subject to a fixed maximum size and storage usage). A convention is suggested in Section A.9.1.

A.6 Table Data Records

The binary table data records begin with the next logical record following the last header record. If the intersection of a row and column is an array then the elements of this array are contiguous and in order of increasing array index. Within a row, columns are stored in order of increasing column number. Rows are given in order of increasing row number. All 2880–byte logical records are completely filled with no extra bytes between columns or rows. Columns and rows do not necessarily begin in the first byte of a 2880–byte record. Note that this implies that a given word may not be aligned in the record along word boundaries of its type; words may even span 2880–byte records. The last 2880–byte record should be zero byte filled past the end of the valid data.

If word alignment is ever considered important for efficiency considerations then this may be accomplished by the proper design of the table. The simplest way to accomplish this is to order the columns by data type (M, D, C, P, E, J, I, B, L, A, X) and then add sufficient padding in the form of a dummy column of type B with the number of elements such that the size of a row is either an integral multiple of 2880 bytes or an integral number of rows is 2880 bytes.

The data types are defined in the following list (r is the number of elements in the entry):

rL A logical value consists of an ASCII "T" indicating true and "F" indicating false. A null character (zero byte) indicates an invalid value.

rX A bit array will start in the most significant bit of the byte and the following bits in the order of decreasing significance in the byte. Bit significance is in the same order as for integers. A bit array entry consists of an integral number of bytes with trailing bits zero.

No explicit null value is defined for bit arrays but if the capability of blanking bit arrays is needed it is recommended that one of the following conventions be adopted: 1) designate a bit in the array as a validity bit, 2) add an L type column to indicate validity of the array or 3) add a second bit array which contains a validity bit for each of the bits in the original array. Such conventions are beyond the scope of this general format design and in general readers will not be expected to understand them.

rB Unsigned 8-bit integer with bits in decreasing order of significance. Signed values may be passed with appropriate values of `TSCALnnn` and `TZEROnnn`.

rI A 16-bit twos-complement integer with the bits in decreasing order of significance. Unsigned values may be passed with appropriate values of `TSCALnnn` and `TZEROnnn`.

rJ A 32-bit twos-complement integer with the bits in decreasing order of significance. Unsigned values may be passed with appropriate values of `TSCALnnn` and `TZEROnnn`.

rA Character strings are represented by ASCII characters in their natural order. A character string may be terminated before its explicit length by an ASCII NULL character. An ASCII NULL as the first character will indicate an undefined string i.e. a NULL string. Legal characters are printable ASCII characters in the range ' ' (hex 20) to '~' (hex 7E) inclusive and ASCII NULL after the last valid character. Strings the full length of the field are not NULL terminated.

rE Single precision floating point values are in IEEE 32-bit precision format in the order: sign bit, exponent and mantissa in decreasing order of significance. The IEEE NaN (not a number) values are used to indicate an invalid number; a value with all bits set is recognized as a NaN. All IEEE special values are recognized.

rD Double precision floating point values are in IEEE 64-bit precision format in the order: sign bit, exponent and mantissa in decreasing order of significance. The IEEE NaN values are used to indicate an invalid number; a value with all bits set is recognized as a NaN. All IEEE special values are recognized.

rC A Complex value consists of a pair of IEEE 32-bit precision floating point values with the first being the real and the second the imaginary part. If either word contains a NaN value the complex value is invalid.

rM Double precision complex values. These consist of a pair of IEEE 64-bit precision floating point values with the first being the real and the second the imaginary part. If either word contains a NaN value the complex value is invalid.

rP Variable length array descriptor. An element is equal in size to a pair of 32-bit integers (i.e., 64 bits). The anticipated use of this data type is described in Section A.9.2. Arrays of type P are not defined; the r field is permitted, but values other than 0 or 1 are undefined. For purposes of printing, an entry of type P should be considered equivalent to 2J.

A.7 Example Binary Table Header

The following is an example of a binary table header which has 19 columns using a number of different data types and dimensions. Columns labeled "IFLUX", "QFLUX", "UFLUX", "VFLUX", "FREQOFF", "LSRVEL" and "RESTFREQ" are arrays of dimension 2. Columns labeled "SOURCE" and "CALCODE" are character strings of length 16 and 4 respectively. The nonstandard keywords "NO_IF", VELTYP", and "VELDEF" also appear at the end of the header. 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
12345678901234567890123456789012345678901234567890123456789012345678
901234
XTENSION= 'BINTABLE'           / Extension type
BITPIX  =                    8 / Binary data
NAXIS   =                    2 / Table is a matrix
NAXIS1  =                  168 / Width of table row in bytes
NAXIS2  =                    5 / Number of rows in table
PCOUNT  =                    0 / Random parameter count
GCOUNT  =                    1 / Group count
TFIELDS =                   19 / Number of columns in each row
EXTNAME = 'AIPS SU '           / AIPS source table
EXTVER  =                    1 / Version number of table
TFORM1  = '1I      '           / 16-bit integer
TTYPE1  = 'ID. NO.         '   / Type (label) of column  1
TUNIT1  = '        '           / Physical units of column  1
TFORM2  = '16A     '           / Character string
TTYPE2  = 'SOURCE          '   / Type (label) of column  2
TUNIT2  = '        '           / Physical units of column  2
TFORM3  = '1I      '           / 16-bit integer
TTYPE3  = 'QUAL            '   / Type (label) of column  3
TUNIT3  = '        '           / Physical units of column  3
TNULL3  = 32767                / Undefined value for column 3
TFORM4  = '4A      '           / Character string
TTYPE4  = 'CALCODE         '   / Type (label) of column  4
TUNIT4  = '        '           / Physical units of column  4
TFORM5  = '2E      '           / Single precision array
TTYPE5  = 'IFLUX           '   / Type (label) of column  5
TUNIT5  = 'JY      '           / Physical units of column  5
TFORM6  = '2E      '           / Single precision array
TTYPE6  = 'QFLUX           '   / Type (label) of column  6
TUNIT6  = 'JY      '           / Physical units of column  6
TFORM7  = '2E      '           / Single precision array
TTYPE7  = 'UFLUX           '   / Type (label) of column  7
TUNIT7  = 'JY      '           / Physical units of column  7
TFORM8  = '2E      '           / Single precision array
TTYPE8  = 'VFLUX           '   / Type (label) of column  8
TUNIT8  = 'JY      '           / Physical units of column  8
TFORM9  = '2D      '           / Double precision array.
TTYPE9  = 'FREQOFF         '   / Type (label) of column  9
TUNIT9  = 'HZ      '           / Physical units of column  9
TSCAL9  = 1.0D9                / Scaling factor of column 9
TZERO9  = 0.0                  / Offset of column 9
TFORM10 = '1D      '           / Double precision
TTYPE10 = 'BANDWIDTH       '   / Type (label) of column 10
TUNIT10 = 'HZ      '           / Physical units of column 10
```

```
TFORM11 = '1D      '          / Double precision
TTYPE11 = 'RAEPO           '  / Type (label) of column 11
TUNIT11 = 'DEGREES '          / Physical units of column 11
TFORM12 = '1D      '          / Double precision
TTYPE12 = 'DECEPO          '  / Type (label) of column 12
TUNIT12 = 'DEGREES '          / Physical units of column 12
TFORM13 = '1D      '          / Double precision
TTYPE13 = 'EPOCH           '  / Type (label) of column 13
TUNIT13 = 'YEARS   '          / Physical units of column 13
TFORM14 = '1D      '          / Double precision
TTYPE14 = 'RAAPP           '  / Type (label) of column 14
TUNIT14 = 'DEGREES '          / Physical units of column 14
TFORM15 = '1D      '          / Double precision
TTYPE15 = 'DECAPP          '  / Type (label) of column 15
TUNIT15 = 'DEGREES '          / Physical units of column 15
TFORM16 = '2D      '          / Double precision array
TTYPE16 = 'LSRVEL          '  / Type (label) of column 16
TUNIT16 = 'M/SEC   '          / Physical units of column 16
TFORM17 = '2D      '          / Double precision array
TTYPE17 = 'RESTFREQ        '  / Type (label) of column 17
TUNIT17 = 'HZ      '          / Physical units of column 17
TDISP17 = 'D17.10'            / Display format of column 17
TFORM18 = '1D      '          / Double precision array
TTYPE18 = 'PMRA            '  / Type (label) of column 18
TUNIT18 = 'DEG/DAY '          / Physical units of column 18
TFORM19 = '1D      '          / Double precision array
TTYPE19 = 'PMDEC           '  / Type (label) of column 19
TUNIT19 = 'DEG/DAY '          / Physical units of column 19
NO_IF   =      2
VELTYP  = 'LSR     '
VELDEF  = 'OPTICAL '
END
```

A.8 Acknowledgments

The authors would like to thank E. Greisen, D. Wells, P. Grosbøl, B. Hanisch, E. Mandel, E. Kemper, S. Voels, B. Schlesinger, W. Pence and many others for invaluable discussions and suggestions.

A.9 Appendixes to Draft Proposal for Binary Tables Extension

A.9.1 "Multidimensional Array" Convention

It is anticipated that binary tables will need to contain data structures more complex that those describable by the basic notation. Examples of these are multidimensional arrays and nonrectangular data structures. Suitable conventions may be defined to pass these structures using some combination of keyword/value pairs and table entries to pass the parameters of these structures.

One case, multidimensional arrays, is so common that it is prudent to describe a simple convention. The "Multidimensional array" convention consists of the following: any column with a dimensionality of 2 or larger will have an associated character keyword TDIMnnn='(l, m, n, ...)' where l, m, n, ... are the dimensions of the array. The data is ordered such that the array index of the first dimension given (l) is the most rapidly varying and that of the last dimension given is the least rapidly varying. The size implied by the TDIMnnn keyword will equal the element count specified in the TFORMnnn keyword. The adherence to this convention will be indicated by the presence of a TDIMnnn keyword in the form described above.

A character string is represented in a binary table by a one-dimensional character array, as described in section A.6. For example, a FORTRAN CHARACTER*20 variable could be represented in a binary table as a character array declared as TFORMn = '20A '. Arrays of character strings, i.e., multidimensional character arrays, may be represented using the TDIMnnn notation. If a column is an array of strings then each string may be null terminated. For example, if TFORMn='20A' and TDIMn='(5, 4)' then the entry consists of 4 strings of up to 5 characters each of which may be null terminated.

This convention is optional and will not preclude other conventions. This convention is not part of the proposed binary table definition.

A.9.2 "Variable Length Array" Facility

One of the most attractive features of binary tables is that any field of the table can be an array. In the standard case this is a fixed size array, i.e., a fixed amount of storage is allocated in each record for the array data - whether it is used or not. This is fine so long as the arrays are small or a fixed amount of array data will be stored in each record, but if the stored array length varies for different records, it is necessary to impose a fixed upper limit on the size of the array that can be stored. If this upper limit is made too large excessive wasted space can result and the binary table mechanism becomes seriously inefficient. If the limit is set too low then it may become impossible to store certain types of data in the table.

The "variable length array" construct presented here was devised to deal with this problem. Variable length arrays are implemented in such a way that, even if a table contains such arrays, a simple reader program which does not understand variable length arrays will still be able to read the main table (in other words a table containing variable length arrays conforms to the basic binary table standard). The implementation chosen is such that the records in the main table remain fixed in size even if the table contains a variable length array field, allowing efficient random access to the main table.

Variable length arrays are logically equivalent to regular static arrays, the only differences being 1) the length of the stored array can differ for different records, and 2) the array data is not stored directly in the table records. Since a field of any datatype can be a static array, a field of any datatype can also be a variable length array (excluding type P, the variable length array descriptor itself, which is not a datatype so much as a storage class specifier). Conventions such as `TDIMnnn` apply equally to both to variable length and static arrays.

A variable length array is declared in the table header with a special field datatype specifier of the form

```
rPt(maxelem)
```

where the "P" indicates the amount of space occupied by the array descriptor in the data record (64 bits), the element count "r" should be 0, 1, or absent, `t` is a character denoting the datatype of the array data (L, X, B, I, J, etc., but not P), and `maxelem` is a quantity guaranteed to be equal to or greater than the maximum number of elements of type `t` actually stored in a table record. There is no built-in upper limit on the size of a stored array; `maxelem` merely reflects the size of the largest array actually stored in the table, and is provided to avoid the need to preview the table when, for example, reading a table containing variable length elements into a database that supports only fixed size arrays.

For example,

```
TFORM8  = 'PB(1800)'          / Variable length byte array
```

indicates that field 8 of the table is a variable length array of type byte, with a maximum stored array length not to exceed 1800 array elements (bytes in this case).

The data for the variable length arrays in a table is not stored in the actual data records; it is stored in a special data area, the heap, following the last fixed size data record. What is stored in the data record is an *array descriptor*. This consists of two 32 bit integer values: the number of elements (array length) of the stored array, followed by the zero-indexed byte offset of the first element of the array, measured from the start of the heap area. Storage for the array is contiguous. The array descriptor for field N as it would appear embedded in a data record is illustrated symbolically below.

```
field N-1  ;  { nelem  offset }  ;  field N+1
```

If the stored array length is zero there is no array data, and the offset value is undefined (it should be set to zero). The storage referenced by an array descriptor must lie entirely within the heap area; negative offsets are not permitted.

A binary table containing variable length arrays consists of three main segments, as follows:

```
table header
record storage area (data records)
heap area (variable array data)
```

The table header consists of one or more 2880 byte *FITS* logical records with the last record indicated by the keyword `END` somewhere in the record. The record storage area begins with the next 2880 byte logical record following the last header record and is `NAXIS1*NAXIS2` bytes in length. The zero indexed byte offset of the heap measured from the start of the record storage area is given by the `THEAP` keyword in the header. If this keyword is missing the heap is assumed to begin with the byte immediately following the last data record, otherwise there may be a gap between the last stored record and the start of the heap. If there is no gap the value of the heap offset is `NAXIS1*NAXIS2`. The total length in bytes of the area following the last stored record (gap plus heap) is given by the `PCOUNT` keyword in the table header.

For example, suppose we have a table containing 5 168 byte records, with a heap area 2880 bytes long, beginning at an offset of 2880, thereby aligning the record storage and heap areas on *FITS* record boundaries (this alignment is not necessarily recommended but is useful for our example). The data portion of the table consists of 2 2880 byte *FITS* records, 840 bytes of which are used by the 5 table records, hence `PCOUNT` is 2*2880-840, or 4920 bytes.

```
NAXIS1  =                  168 / Width of table row in bytes
NAXIS2  =                    5 / Number of rows in table
PCOUNT  =                 4920 / Random parameter count
   ...
THEAP   =                 2880 / Byte offset of heap area
```

While the above description is sufficient to define the required features of the variable length array implementation, some hints regarding usage of the variable length array facility may also be useful.

Programs which read binary tables should take care to not assume more about the physical layout of the table than is required by the specification. For example, there are no requirements on the alignment of data within the heap. If efficient runtime access is a concern one may want to design the table so that data arrays are aligned to the size of an array element. In another case one might want to minimize storage and forgo any efforts at alignment (by careful design it is often possible to achieve both goals). Variable array data may be stored in the heap in any order, i.e., the data for record N+1 is not necessarily stored at a larger offset than that for record N. There may be gaps in the heap where no data is stored. Pointer aliasing is permitted, i.e., the array descriptors for two or more arrays may point to the same storage location (this could be used to save storage if two or more arrays are identical).

Byte arrays are a special case because they can be used to store a "typeless" data sequence. Since *FITS* is a machine independent storage format, some form of machine specific data conversion (byte swapping, floating point format conversion) is implied when accessing stored data with types such as integer and floating, but byte arrays are copied to and from external storage without any form of conversion.

An important feature of variable length arrays is that it is possible that the stored array length may be zero. This makes it possible to have a column of the table for which, typically, no data is present in each stored record. When data is present the stored array can be as large as necessary. This can be useful when storing complex objects as records in a table.

Accessing a binary table stored on a random access storage medium is straightforward. Since the data records in the main table are fixed in size they may be randomly accessed given the record number, by computing the offset. Once the record has been read in, any variable length array data may be directly accessed using the element count and offset given by the array descriptor stored in the data record.

Reading a binary table stored on a sequential access storage medium requires that a table of array descriptors be built up as the main table records are read in. Once all the table records have been read, the array descriptors are sorted by the offset of the array data in the heap. As the heap data is read, arrays are extracted sequentially from the heap and stored in the affected records using the back pointers to the record and field from the table of array descriptors. Since array aliasing is permitted, it may be necessary to store a given array in more than one field or record.

Variable length arrays are more complicated than regular static arrays and imply an extra data access per array to fetch all the data for a record. For this reason, it is recommended that regular static arrays be used instead of variable length arrays unless efficiency or other considerations require the use of a variable array.

This facility is still undergoing trials and is not currently part of the main binary table definition.

Appendix B

Implementation on Physical Media

(This Appendix is not part of the NOST *FITS* Standard, but is included as a guide to recommended practices.)

B.1 Block Size

The block size (physical record length) for transport of data should, where possible, equal the logical record length or an integer blocking factor times this record length. Standard values of the blocking factor may be specified for each medium; if not otherwise specified, the expected value is unity.

B.1.1 Nine-Track, Half-Inch Magnetic Tape

For nine-track half-inch magnetic tapes conforming to the ANSI X3.40–1983 specifications [13], there should be from one to ten logical records per physical block. The `BLOCKED` keyword (section 5.2.2.1) may be used to warn that there may be more than one logical record per physical block. The last physical block of a *FITS* file should be truncated to the minimum number of *FITS* logical records required to hold the remaining data, in accordance with ANSI X3.27–1978 specifications [14]. With the issuance of this standard, the `BLOCKED` keyword is deprecated by this document.

B.1.2 Other Media

For media where the physical block size cannot be equal to or an integral multiple of the *FITS* logical record length of 23040-bits (2880 8-bit bytes), records should be written over multiple blocks as a byte stream. Conventions regarding the relation between physical block size and logical record length of *FITS* files have not been otherwise established for other media.

B.2 Physical Properties of Media

The arrangement of digital bits and other physical properties of any medium should be in conformance with the relevant national and/or international standard for that medium.

B.3 Labeling

B.3.1 Tape

Tapes may be either ANSI standard labeled or unlabeled. Unlabeled tapes are preferred.

B.3.2 Other Media

Conventions regarding labels for physical media containing *FITS* files have not been established for other media.

B.4 FITS File Boundaries

B.4.1 Magnetic Reel Tape

Individual *FITS* files are terminated by a tape-mark.

B.4.2 Other Media

For media where the physical record size cannot be equal to or an integral multiple of the standard *FITS* logical record length, a logical record of fewer than 23040 bits (2880 8-bit bytes) immediately following the end of the primary header, data, or an extension should be treated as an end-of-file. Otherwise, individual *FITS* files should be terminated by a delimiter appropriate to the medium, analogous to the tape end-of-file mark. If more than one *FITS* file appears on a physical structure, the appropriate end-of-file indicator should immediately precede the start of the primary headers of all files after the first.

B.5 Multiple Physical Volumes

Storage of a single *FITS* file on more than one unlabeled tape or on multiple units of any other medium is not universally supported in *FITS*. One possible way to handle multivolume unlabeled tape was suggested in [1].

Appendix C

Differences from IAU-endorsed Publications

(This Appendix is not part of the NOST *FITS* Standard but is included for informational purposes only.)

Note: In this discussion, the term *the FITS papers* refers to [1], [2], [4], and [5], collectively, and the term *Floating Point Agreement (FPA)* refers to [8].

1. Section 3 – Definitions, Acronyms, and Symbols

Array value – This precise definition is not used in the original *FITS* papers.

ASCII text – This permissible subset of the ASCII character set, used in many contexts, is not precisely defined in the *FITS* papers.

Basic FITS – This definition includes the possibility of floating point data arrays, while the terminology in the *FITS* papers refers to *FITS* as described in [1], where only integer arrays were possible.

Conforming Extension – This terminology is not used in the *FITS* papers.

Deprecate – The concept of deprecation does not appear in the *FITS* papers.

FITS structure – This terminology is not used in the *FITS* papers in the precise way that it is in this standard.

Header and Data Unit – This terminology is not used in the *FITS* papers.

Indexed keyword – This terminology is not used in the original *FITS* papers.

Physical value – This precise definition is not used in the original *FITS* papers.

Reference point – This term replaces the *reference pixel* of the *FITS* papers. The new terminology is consistent with the fact that the array need not represent a digital image and that the reference point (or *pixel*) need not lie within the array.

Reserved keyword – The *FITS* papers describe optional keywords but do not say explicitly that they are reserved.

Standard Extension – This precise definition is new. The term *standard extension* is used in some contexts in the *FITS* papers to refer to what this standard defines as a *standard extension* and in others to refer to what this standard defines as a *conforming extension*.

2. Section 4.3.2 Primary Data Array
   Fill format – This specification is new. The *FITS* papers and the FPA do not precisely specify the format of data fill for the primary data array.

3. Section 4.4.1.1 Identity (of conforming extensions)
   The *FITS* papers specify that creators of new extension types should check with the *FITS* standards committee. This standard identifies the committee specifically, introduces the role of the NOST as its agent, and mandates registration.

4. Section 5.1.2.1 Keyword (as header component)
   The specification of permissible keyword characters is new. The *FITS* papers do not precisely define the permissible characters for keywords.

5. Section 5.2.1.1 Principal (mandatory keywords)

   (a) `NAXIS` keyword – The requirement that the `NAXIS` keyword may not be negative is not explicitly specified in the *FITS* papers.

   (b) `NAXISn` keyword – The requirement that the `NAXISn` keyword may not be negative is not explicitly specified in the *FITS* papers.

6. Section 5.2.1.2 Conforming Extensions

   (a) `NBITS` – The requirement that `NBITS` may not be negative is not explicitly specified in the *FITS* papers.

   (b) `XTENSION` keyword – That this keyword may not appear in the primary header is only implied by the *FITS* papers; the prohibition is explicit in this standard. The *FITS* papers name a *FITS* standards committee as the keeper of the list of accepted extension type names. This standard specifically identifies the committee and introduces the role of the NOST as its agent.

7. Section 5.2.2 Other Reserved Keywords
   That the optional keywords defined in the *FITS* papers are to be reserved with the meanings and usage defined in those papers, as in the standard, is not explicitly stated in them.

8. Section 5.2.2.1 Keywords Describing the History...

(a) `DATE` Keyword – The recommendation for use of Universal Time is not in the *FITS* papers.

(b) `BLOCKED` keyword – The *FITS* papers require the `BLOCKED` keyword to appear in the first record of the primary header even though it cannot when the value of `NAXIS` exceeds the values described in the text. They do not address this contradiction. Deprecation of the `BLOCKED` keyword is new with this standard.

9. Section 5.2.2.2 Keywords Describing Observations

   (a) `DATE-OBS` Keyword – The recommendation for use of Universal Time is not in the *FITS* papers.

   (b) `EQUINOX` and `EPOCH` keywords – This standard replaces the `EPOCH` keyword with the more appropriately named `EQUINOX` keyword and deprecates the `EPOCH` name.

10. Section 5.2.2.4 Commentary keywords
    Keyword field is blank – Reference [1] contains the text "BLANK" to represent a blank keyword field. The standard clarifies the intention.

11. Section 5.2.2.5 Array keywords

    (a) `BUNIT` Keyword – The *FITS* papers recommend the use of SI units and identify other units standard in astronomy. This standard makes the recommendation more specific by referring to the IAU Style Manual [7].

    (b) `CTYPEn` Keywords – This standard extends the recommendations on units to coordinate axes.

    (c) `CRPIXn` Keywords – This standard explicitly notes the ambiguity in the location of the index number relative to an image pixel.

    (d) `CDELTn` Keywords – The definition in the standard differs from that in the *FITS* papers in that it provides for the case where the spacing between index points varies over the grid. For the case of constant spacing, it is identical to the specification in the *FITS* papers.

    (e) `DATAMAX` and `DATAMIN` Keywords – The standard clarifies that the value refers to the physical value represented by the array, after any scaling, not the array value before scaling. The standard also notes that special values are not to be considered when determining the values of `DATAMAX` and `DATAMIN`, an issue not specifically addressed by the *FITS* papers or the FPA.

12. Section 5.3.1 General Format Requirements
    The *FITS* papers specify that the value field is to be written following the rules

of ANSI FORTRAN list-directed input, with some restrictions. The standard incorporates such restrictions by explicitly noting that formats may be otherwise specified in the Standard.

13. Section 5.3.2.1 Character String (fixed format)
    The standard explicitly describes how single quotes are to be coded into keyword values, a rule only implied by the FORTRAN-77 list-directed read requirements of the *FITS* papers.

14. Section 5.3.2.3 Integer (fixed format) The standard explicitly notes that the fixed format for complex integers does not conform to the rules for ANSI FORTRAN list-directed read.

15. Section 5.3.2.4 Real Floating Point Number (fixed format)
    The standard explicitly notes that the full precision of 64-bit values cannot be expressed as a single value using the fixed format.

16. Section 5.3.2.4 Complex Floating Point Number (fixed format)
    The standard explicitly notes that the fixed format for complex floating point numbers does not conform to the rules for ANSI FORTRAN list-directed read. It notes also that the full precision of 64-bit values cannot be expressed as a single value using the fixed format.

17. Section 7 Random Groups Structure
    The standard deprecates the Random Groups structure.

18. Section 7.1.2 Reserved keywords (random groups)
    That the optional keywords defined in the *FITS* papers are to be reserved with the meanings and usage defined in those papers, as in the standard, is not explicitly stated in them.

19. Section 7.1.2.2 `PSCALn` Keywords – The default value is explicitly specified in the standard, whereas in the *FITS* papers it is assumed by analogy with the `BSCALE` keyword.

20. Section 7.1.2.3 `PZEROn` Keywords – The default value is explicitly specified in the standard, whereas in the *FITS* papers it is assumed by analogy with the `BZERO` keyword.

21. Section 8.1 ASCII Tables Extension
    The name *ASCII Tables* is given to the Tables extension discussed in the *FITS* papers to distinguish it from binary tables.

22. Section 8.1.1 Mandatory Keywords (ASCII tables)

(a) `NAXIS1` keyword – The requirement that the `NAXIS1` keyword may not be negative in an ASCII table header is not explicitly specified in the *FITS* papers.

(b) `NAXIS2` keyword – The requirement that the `NAXIS2` keyword may not be negative in an ASCII table header is not explicitly specified in the *FITS* papers.

(c) `TFIELDS` keyword – The requirement that the `TFIELDS` keyword may not be negative is not explicitly specified in the *FITS* papers.

23. Section 8.1.2 Other Reserved Keywords (ASCII tables)
That the optional keywords defined in the *FITS* papers are to be reserved with the meanings and usage defined in those papers, as in the standard, is not explicitly stated in them.

(a) `TUNITn` Keyword – The *FITS* papers do not explicitly recommend the use of any particular units for this keyword, although the reference to the `BUNIT` keyword may be considered an implicit extension of the recommendation for that keyword. This standard makes the recommendation more specific for the `TUNITn` keyword by referring to the IAU Style Manual [7].

(b) `TSCALn` Keyword – The prohibition against use in A-format fields is stronger than the statement in the *FITS* papers that the keyword "is not relevant".

(c) `TZEROn` Keywords – The prohibition against use in A-format fields is stronger than the statement in the *FITS* papers that the keyword "is not relevant".

24. Section 9 Restrictions on Changes
The concept of deprecation is not provided for in the *FITS* papers.

25. Appendix B Implementation on Physical Media
Material in the *FITS* papers specifying the expression of *FITS* on specific physical media is not part of this Standard.

Appendix D

Summary of Keywords

(This Appendix is not part of the NOST *FITS* Standard, but is included for convenient reference).

Principal HDU	Conforming Extension	ASCII Table Extension	Random Groups Records	Proposed Binary Table Extension
SIMPLE	XTENSION	XTENSION[1]	SIMPLE	XTENSION[2]
BITPIX	BITPIX	BITPIX = 8	BITPIX	BITPIX = 8
NAXIS	NAXIS	NAXIS = 2	NAXIS	NAXIS = 2
NAXISn	NAXISn	NAXIS1	NAXIS1 = 0	NAXIS1
EXTEND[3]	PCOUNT	NAXIS2	NAXISn	NAXIS2
END	GCOUNT	PCOUNT = 0	GROUPS = T	PCOUNT
	END	GCOUNT = 1	PCOUNT	GCOUNT = 1
		TFIELDS	GCOUNT	TFIELDS
		TBCOLn	END	TFORMn
		TFORMn		END
		END		

[1] XTENSION=␣'TABLE␣␣␣' for the ASCII Table extension.
[2] XTENSION=␣'BINTABLE' for the proposed binary table extension.
[3] Required only if extensions are present.

Table D.1: Mandatory *FITS* keywords for the structures described in this document.

Principal HDU		Conforming	ASCII Table	Random Groups	Binary Table
General	Array	Extension	Extension	Records	Extension
DATE	BSCALE	EXTNAME	TSCALn	PTYPEn	TSCALn
ORIGIN	BZERO	EXTVER	TZEROn	PSCALn	TZEROn
BLOCKED	BUNIT	EXTLEVEL	TNULLn	PZEROn	TNULLn
AUTHOR	BLANK		TTYPEn		TTYPEn
REFERENC	CTYPEn		TUNITn		TUNITn
COMMENT	CRPIXn				TDISPn
HISTORY	CROTAn				TDIMn
␣␣␣␣␣␣␣␣	CRVALn				THEAP
DATE-OBS	CDELTn				
TELESCOP	DATAMAX				
INSTRUME	DATAMIN				
OBSERVER					
OBJECT					
EQUINOX					
EPOCH					

Table D.2: Reserved *FITS* keywords for the structures described in this document. Note that the EPOCH and BLOCKED keywords are deprecated by this document.

Production	Bibliographic	Commentary	Observation	Array
DATE	AUTHOR	COMMENT	DATE-OBS	BSCALE
ORIGIN	REFERENC	HISTORY	TELESCOP	BZERO
BLOCKED		␣␣␣␣␣␣␣␣	INSTRUME	BUNIT
			OBSERVER	BLANK
			OBJECT	CTYPEn
			EQUINOX	CRPIXn
			EPOCH	CROTAn
				CRVALn
				CDELTn
				DATAMAX
				DATAMIN

Table D.3: General Reserved *FITS* keywords described in this document. Note that the EPOCH and BLOCKED keywords are deprecated by this document.

Appendix E

ASCII Text

(This appendix is not part of the NOST *FITS* standard; the material in it is taken from the ANSI standard for ASCII [11] and is included here for informational purposes.)

In the following table, the first column is the decimal and the second column the hexadecimal value for the character in the third column. The characters hexadecimal 20 to 7E (decimal 32 to 126) constitute the subset referred to in this document as ASCII text.

ASCII Control			ASCII Text								
dec	hex	char	dec	hex	char	dec	hex	char	dec	hex	char
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL[1]

[1] Not ASCII Text

Table E.1: ASCII character set

Appendix F

IEEE Special Formats

(The material in this Appendix is not part of this standard; it is taken from the IEEE-754 floating point standard [12], for informational purposes. It is not intended to be a comprehensive description of the IEEE special formats; readers should refer to the IEEE standard.)

Table F.1 displays the hexadecimal contents, most significant byte first, of the double and single precision IEEE special values.

IEEE special value	Double Precision	Single Precision
-0	8000000000000000	80000000
$+\infty$	7FF0000000000000	7F800000
$-\infty$	FFF0000000000000	FF800000
NaN[1]	7FF0000000000001	7F800001
	to	to
	7FFFFFFFFFFFFFFF	7FFFFFFF
	and	and
	FFF0000000000001	FF800001
	to	to
	FFFFFFFFFFFFFFFF	FFFFFFFF
positive overflow	7FEFFFFFFFFFFFFF	7F7FFFFF
negative overflow	FFEFFFFFFFFFFFFF	FF7FFFFF
positive underflow	0010000000000000	00800000
negative underflow	8010000000000000	80800000
denormalized	0000000000000001	00000001
	to	to
	000FFFFFFFFFFFFF	007FFFFF
	and	and
	8000000000000001	80000001
	to	to
	800FFFFFFFFFFFFF	807FFFFF

[1] Certain values may be designated as *quiet* NaN (no diagnostic when used) or *signaling* (produces diagnostic when used) by particular implementations.

Table F.1: IEEE special floating point formats

Appendix G

Reserved Extension Type Names

(This Appendix is not part of the NOST *FITS* Standard, but is included for informational purposes. It describes the extension type names registered as of the date this standard was issued. A current list is available from the NOST.)

Type Name	Status	Reference	Sponsor	Comments
'A3DTABLE'	L	[15]	NRAO	Prototype binary table design supported in AIPS; superseded by BINTABLE, which supports all A3DTABLE features.
'BINTABLE'	D	[9]	IAU NRAO NOAO	Draft proposal for binary table design.
'DUMP␣␣␣␣'	R	none	none	Intended for binary dumps.
'FILEMARK'	R	none	NRAO	Intended for structure to represent equivalent of tape mark on other media.
'IMAGE␣␣␣'	D	[16]	IUE	Draft proposal extension containing a multidimensional matrix.
'IUEIMAGE'	L	[17]	IUE	Prototype matrix extension used for archiving IUE products, superseded by IMAGE.
'TABLE␣␣␣'	S	[5]	IAU	ASCII Tables.

Table G.1: Reserved Extension Type Names

Code	Significance
D	Draft extension proposal for discussion by regional FITS committees.
L	Local FITS extension.
P	Proposed FITS extension approved by regional FITS committees but not by IAU FITS Working Group.
R	Reserved type name for which a full draft proposal has not been submitted.
S	Standard extension approved by IAU FITS Working Group and endorsed by the IAU.

Table G.2: Status Codes

Appendix H

NOST Publications

Document	Title	Date	Status
NOST 100-0.1	FITS Standard	December, 1990	Draft Standard
NOST 100-0.2	FITS Implementation Standard	June, 1991	Revised Draft Standard
NOST 100-0.3	FITS Implementation Standard	December, 1991	Revised Draft Standard
NOST 100-1.0	FITS Definition Standard	March, 1993	Proposed Standard
NOST 100-1.0	FITS Definition Standard	June, 1993	NOST Standard

Table H.1: NOST Publications

Index