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

December 9, 2005

FITS Standard

Version 2.1b

IAU FITS Working Group
http://fits.gsfc.nasa.gov/iaufwg/

Preface to Version 2.1b

This Version 2.1b of the *FITS* Standard is an interium release that incorporates 3 changes that were approved by the IAU *FITS* Working Group on 8 December 2005:

- added support for 64-bit integer columns in binary tables, with TFORMn = 'K', and
- added support for 64-bit descriptor columns in binary tables, with TFORMn = 'Q'.
- added conditionally support for *FITS* primary arrays and image extension with 64-bit integer pixels (the final wording of this change is still under negotiation).

The other recent change in this version (and in the previous 2.1 version) is that the descriptions of the "Variable-Length Array" facility and the "Multidimensional Array" convention have been moved from the unofficial Appendix B into section 8.3 of the *FITS* Standard itself. The IAU *FITS* Working Group officially approved these changes to the *FITS* Standard on 7 April, 2005.

Other than these changes, this version of the *FITS* Standard is very similar to the NOST 100-2.0 version that was developed in 1999, approved by the IAU *FITS* Working Group in October 2000, and published in the *Astronomy & Astrophysics* scientific journal in 2001 (volume 376, page 359).

Further information about the membership and role of the IAU FITS Working Group can be found on the `http://fits.gsfc.nasa.gov/iaufwg/` web site.

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) 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 NOST. 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 version of the standard consisted of the following members:

Robert J. Hanisch, Chair	Space Telescope Science Institute
William D. Pence, Secretary	NASA Goddard Space Flight Center
Barry M. Schlesinger, Past Secretary	Raytheon STX
Allen Farris	Space Telescope Science Institute
Eric W. Greisen	National Radio Astronomy Observatory
Peter J. Teuben	University of Maryland
Randall W. Thompson	Computer Sciences Corporation
Archibald Warnock	A/WWW Enterprises

Members of the previous Technical Panels also included:

Lee E. Brotzman	Hughes STX
Edward Kemper	Hughes STX
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
electronic mail: nost@nssdca.gsfc.nasa.gov
+1-301-286-3575
http://ssdoo.gsfc.nasa.gov/nost/

Other information about *FITS* can be obtained from the *FITS* Support Office. The *FITS* Support Office can be contacted as follows:

FITS Support Office
Code 662, NASA Goddard Space Flight Center
Greenbelt MD 20771
USA
electronic mail: fits@fits.gsfc.nasa.gov
+1-301-286-4599
http://fits.gsfc.nasa.gov/

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 [3] by the International Astronomical Union (IAU) in 1982. 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 table [5], were endorsed by the IAU General Assembly [6] in 1988. At the same General Assembly, an IAU *FITS* Working Group (IAUFWG) was formed [7] under IAU Commission 5 (Astronomical Data) 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. In 1989, the IAUFWG approved a formal agreement [8] for the representation of floating point numbers. In 1994, the IAUFWG endorsed two additional extensions, the image extension [9] and the binary table extension [10]. *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. In 1994, the IAUFWG adopted a set of rules [11] governing the relation between the *FITS* logical record size and the physical block size for sequential media and bitstream devices. The IAUFWG also approved in 1997 an agreement [12] defining a new format for encoding the date and time in the `DATE`, `DATE-OBS`, and other related `DATExxxx` keywords to correct the ambiguity in the original DATE keyword format beginning in the year 2000.

Section 1

Overview

> *An* archival format *must be utterly portable and self-describing, on the assumption that, apart from the transcription device, neither the software nor the hardware that wrote the data will be available when the data are read.* "Preserving Scientific Data on our Physical Universe," p. 60. Steering Committee for the Study on the Long-Term Retention of Selected Scientific and Technical Records of the Federal Government, [US] National Research Council, National Academy Press 1995.

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 §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 *FITS* Working Group (IAUFWG). Minor ambiguities and inconsistencies in *FITS* as described in the original papers are eliminated.

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, the ASCII table extension, the image extension, and the binary table extension. It also specifies minimum structural requirements for new extensions and general principles governing the creation of new extensions. It specifies the relation between physical block sizes and logical records for *FITS* files on bitstream devices and sequential media. 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

This standard describes an extensible data interchange format particularly well suited for transport and archiving of arrays and tables of astronomical data. The IAU has recommended that all astronomical computer facilities support *FITS* for the interchange of binary data. It has been NASA policy for its astrophysics projects to make their data available in *FITS* format. 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 of This Document

§3 is a glossary of definitions, acronyms, and symbols. In §4, this document describes the overall organization of a *FITS* file, the contents of the first (primary) header and data, the rules for creating new *FITS* extensions, and the relation between physical block sizes and logical records for *FITS* files on bitstream devices and sequential media. The next two sections provide additional details on the header and data, with a particular focus on the primary header. §5 provides details about header card image syntax and specifies those keywords required and reserved in a primary header. §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 deprecated random groups records (§7) and the current standard extensions: ASCII table, image, and binary table (§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, Appendix A, provides a formal expression of the keyword/value syntax for header card images described in §5.2. Appendix B describes a convention for handling arrays of substrings in binary table extensions. Appendix C describes aspects of the implementation of *FITS* on physical media not covered by the blocking agreement. Appendix D is the appendix to the agreement endorsed by the IAUFWG for a new format for keywords expressing dates. The new format uses a four-digit value for the year, and thus eliminates any ambiguity in dates from the year 2000 and after. This appendix is not part of the formal agreement. It contains a detailed discussion of time systems. It has been slightly reformatted for stylistic compatibility with the remainder of this document. Appendix E 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 appendixes provide reference information: a tabular summary of the *FITS* keywords (Appendix F), a list of the ASCII character set and a subset designated *ASCII text* (Appendix G), a description of the IEEE floating point format (Appendix H), and a list of the extension type names that have been reserved as of the date this document was issued (Appendix I). Appendix J is a list of NOST documents, including earlier versions of this standard.

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 electronically from `ftp://nssdc.gsfc.nasa.gov/pub/fits/fp_agree.ps`).

9. Ponz, J. D., Thompson, R. W., and Muñoz, J. R. 1994, "The *FITS* Image Extension," *Astron. Astrophys. Suppl.*, **105**, 53–55.

10. Cotton, W. D., Tody, D. B., and Pence, W. D. 1995, "Binary Table Extension to *FITS*," *Astron. Astrophys. Suppl.*, **113**, 159–166.

11. Grosbøl, P. and Wells, D. C. 1994, "Blocking of Fixed-block Sequential Media and Bitstream Devices," (available electronically from FITS Support Office at `ftp://nssdc.gsfc.nasa.gov/pub/fits/blocking94.txt`).

12. Bunclark, P. and Rots, A. 1997, "Precise re-definition of DATE-OBS Keyword encompassing the millennium," (available electronically from `ftp://nssdc.gsfc.nasa.gov/pub/fits/year2000_agreement.txt`).

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

14. 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.).

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

16. Jennings, D. G., Pence, W. D., Folk, M., and Schlesinger, B. M, 1997, "A Hierarchical Grouping Convention for *FITS*," preprint, available electronically from `http://fits.gsfc.nasa.gov/group.html` .

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

18. 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.

ANSI American National Standards Institute.

Array A sequence of data values. This sequence corresponds to the elements in a rectilinear, n-dimension matrix ($0 \leq n \leq 999$).

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 American National Standard Code for Information Interchange.

ASCII blank The ASCII character for blank which is represented by hexadecimal 20 (decimal 32).

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

ASCII NULL Hexadecimal 00.

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 An ordered sequence of eight consecutive bits treated as a single entity.

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

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

DAT 4mm Digital Audio Tape.

Deprecated This term is used to refer to obsolete structures that should not be used for new applications but remain valid.

Entry A single value in a table.

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 Flexible Image Transport System.

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

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 computer representation of a real number.

Fraction The field of the mantissa (or significand) of a floating point number that lies to the right of its implied binary point.

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 Goddard Space Flight Center.

HDU Header and Data Unit. 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.

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

Heap A supplemental data area, currently defined to follow the table in a binary table extension.

IAU International Astronomical Union.

IAUFWG International Astronomical Union *FITS* Working Group.

IUE International Ultraviolet Explorer.

IEEE Institute of Electrical and Electronic Engineers.

IEEE NaN IEEE Not-a-Number value.

IEEE special values Floating point number byte patterns that have a special, reserved meaning, such as -0, $\pm\infty$, $\pm$underflow, $\pm$overflow, $\pm$denormalized, $\pm$ NaN. (See Appendix H).

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

Keyword The first eight bytes of a header card image.

Logical record A record comprising 2880 8-bit bytes.

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

Mantissa Also known as significand. The component of an IEEE floating point number consisting of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

Matrix A data array of two or more dimensions.

NOST NASA/Science Office of Standards and Technology.

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

Picture element A single location within an array.

Pixel Picture element.

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

Primary HDU The first HDU in a *FITS* file.

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.

Repeat count The number of values represented in a binary table field.

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.

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
- 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 (§5.4.1.1).

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

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.

4.3.1 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 random groups convention in the primary data array is a more complicated structure (see §7). 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 (§5.4.1.1).

If the data array does not fill the final record, the remainder of the record shall be filled by setting all bits to zero.

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 §5.4.1.2. To preclude conflict, extension type names must be registered with the IAUFWG. The *FITS* Support Office shall maintain and provide a list of the registered extensions.

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 §5.4.1.2.

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 name given by the value of the `XTENSION` keyword (see Appendix I)

4.4.3 Order of Extensions

An extension may follow the primary HDU 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.

4.6 Physical Blocking

4.6.1 Bitstream Devices

For bitstream devices, including but not restricted to logical file systems, *FITS* files shall be written with fixed blocks of a physical block size equal to the 23040-bit *FITS* logical record size.

4.6.2 Sequential Media

4.6.2.1 Fixed Block

For fixed block length sequential media, including but not restricted to optical disks (accessed as a sequential set of records), QIC format 1/4-inch cartridge tapes, and local area networks, *FITS* files shall be written as a bitstream, using the fixed block size of the medium. If the end of the last logical record does not coincide with the end of a physical fixed block, all bits in the remainder of the physical block containing the last logical record shall be set to zero. After an end-of-file mark has been detected in the course of reading a *FITS* file, subsequent incomplete *FITS* logical records should be disregarded.

4.6.2.2 Variable Block

For variable block length sequential media, including but not restricted to 1/2-inch 9-track tapes, DAT 4 mm cartridge tapes, and 8 mm cartridge tapes, *FITS* files may be written with an integer blocking factor equal to 1–10 logical records per physical block.

Section 5

Headers

5.1 Card Images

5.1.1 Syntax

Header card images shall consist of a keyword, a value indicator (optional unless a value is present), a value (optional), and a comment (optional). Except where specifically stated otherwise in this standard, keywords may appear in any order.

A formal syntax, giving a complete definition of the syntax of *FITS* card images, is given in Appendix A. It is intended as an aid in interpreting the text defining the standard.

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 with a single index the counter shall not have leading zeroes.

5.1.2.2 Value Indicator (bytes 9–10)

If this field contains the ASCII characters "`=␣`", it indicates the presence of a value field associated with the keyword, unless it is a commentary keyword as defined in §5.4.2.4. If the value indicator is not present or if it is a commentary keyword then columns 9–80 may contain any ASCII text.

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. The value field may be a null field; i.e., it may consist entirely of spaces. If the value field is null, the value associated with the keyword is undefined. If a comment is present, it must be preceded by a slash (hexadecimal 2F, "/"). A space between the value and the slash is strongly recommended. The value shall be the ASCII text representation of a string or constant, in the format specified in §5.2. The comment may contain any ASCII text.

5.2 Value

The structure of the value field shall be determined by the type of the variable. The value field represents a single value and not an array of values. The value field must be in one of two formats: fixed or free. 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.

5.2.1 Character String

If the value is a fixed format 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. 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.

Free format character strings follow the same rules as fixed format character strings except that the starting and closing single quote characters may occur anywhere within columns 11–80. Any columns preceding the starting quote character and after column 10 must contain the space character.

Note that there is a subtle distinction between the following 3 keywords:

```
KEYWORD1= ''                   / null string keyword
KEYWORD2= '    '               / blank keyword
KEYWORD3=                      / undefined keyword
```

The value of `KEYWORD1` is a null, or zero length string whereas the value of the `KEYWORD2` is a blank string (nominally a single blank character because the first blank in the string is significant, but trailing blanks are not). The value of `KEYWORD3` is undefined and has an indeterminate datatype as well, except in cases where the data type of the specified keyword is explicitly defined in this standard.

The maximum allowed length of a keyword string is 68 characters (with the opening and closing quote characters in columns 11 and 80, respectively). In general, no length limit less than 68 is implied for character-valued keywords.

5.2.2 Logical

If the value is a fixed format logical constant, it shall appear as a `T` or `F` in column 30. A logical value is represented in free format by a single character consisting of `T` or `F`. This character must be the first non-blank character in columns 11–80. The only characters that may follow this single character are spaces, or a slash followed by an optional comment (see §5.1.2.3).

5.2.3 Integer Number

If the value is a fixed format integer, the ASCII representation shall be right justified in columns 11–30. An integer consists of a '+' (hexadecimal 2B) or '−' (hexadecimal 2D) sign, followed by one or more ASCII digits (hexadecimal 30 to 39), with no embedded spaces. The leading '+' sign is optional. Leading zeros are permitted, but are not significant. The integer representation described here is always interpreted as a signed, decimal number.

A free format integer value follows the same rules as fixed format integers except that it may occur anywhere within columns 11–80.

5.2.4 Real Floating Point Number

If the value is a fixed format real floating point number, the ASCII representation shall appear, right justified, in columns 11–30.

A floating point number is represented by a decimal number followed by an optional exponent, with no embedded spaces. A decimal number consists of a '+' (hexadecimal 2B) or '-' (hexadecimal 2D) sign, followed by a sequence of ASCII digits containing a single decimal point ('.'), representing an integer part and a fractional part of the floating point number. The leading '+' sign is optional. At least one of the integer part or fractional part must be present. If the fractional part is present, the decimal point must also be present. If only the integer part is present, the decimal point may be omitted. The exponent, if present, consists of an exponent letter followed by an integer. Letters in the exponential form ('E' or 'D') shall be upper case. Note: The full precision of 64-bit values cannot be expressed over the whole range of values using the fixed format.

A free format floating point value follows the same rules as fixed format floating point values except that it may occur anywhere within columns 11–80.

5.2.5 Complex Integer Number

There is no fixed format for complex integer numbers.

If the value is a complex integer number, the value must be represented as a real part and an imaginary part, separated by a comma and enclosed in parentheses. Spaces may precede and follow the real and imaginary parts. The real and imaginary parts are represented as integers (§5.2.3). Such a representation is regarded as a single value for the complex integer number. This representation may be located anywhere within columns 11–80.

5.2.6 Complex Floating Point Number

There is no fixed format for complex floating point numbers.

If the value is a complex floating point number, the value must be represented as a real part and an imaginary part, separated by a comma and enclosed in parentheses. Spaces may precede and follow the real and imaginary parts. The real and imaginary parts are represented as floating point values (§5.2.4). Such a representation is regarded as a single value for the complex floating point number. This representation may be located anywhere within columns 11–80.

5.3 Units

The units of all *FITS* header keyword values, with the exception of measurements of angles, should conform with the recommendations in the IAU Style Manual [7]. For angular measurements given as floating point values and specified with reserved keywords, degrees are the recommended units (with the units, if specified, given as `'deg'`).

5.4 Keywords

5.4.1 Mandatory Keywords

Mandatory keywords are required in every HDU as described in the remainder of this subsection. They may be used only as described in this standard. Values of the mandatory keywords must be written in fixed format.

5.4.1.1 Principal

The `SIMPLE` keyword is required to be the first keyword in the primary header of all *FITS* files. Principal mandatory keywords other than `SIMPLE` are required in all *FITS* headers. The card images of any primary header must contain the keywords shown in

Table 5.1 in the order given. No other keywords may intervene between the `SIMPLE` keyword and the last `NAXISn` keyword.

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

Table 5.1: Mandatory keywords for primary header.

The total number of bits in the primary data array, exclusive of fill that is needed after the data to complete the last record (§4.3.2), is given by the following expression:

$$N_{\text{bits}} \quad = \quad |\texttt{BITPIX}| \times (\texttt{NAXIS1} \times \texttt{NAXIS2} \times \cdots \times \texttt{NAXISm}), \tag{5.1}$$

where N_{bits} 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 for the primary header and is not permitted in extension headers. A value of `F` signifies that the file does not conform to this standard.

`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.

`NAXIS` **Keyword** The value field shall contain a non-negative integer no greater than 999, representing the number of axes in the associated 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 elements along axis `n` of a data array. The `NAXISn` must be present for all values `n = 1,...,NAXIS`, and for no other values of `n`. 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.

Value	Data Represented
8	Character or unsigned binary integer
16	16-bit twos-complement binary integer
32	32-bit twos-complement binary integer
64	64-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.

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

5.4.1.2 Conforming Extensions

All conforming extensions must use the keywords defined in Table 5.3 in the order specified. No other keywords may intervene between the `XTENSION` keyword and the last `NAXISn` keyword. 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 such as that for the primary data array (§4.3.2) is given by the following expression:

$$N_{\text{bits}} = |\texttt{BITPIX}| \times \texttt{GCOUNT} \times$$

$$(\texttt{PCOUNT} + \texttt{NAXIS1} \times \texttt{NAXIS2} \times \cdots \times \texttt{NAXISm}), \quad (5.2)$$

where N_{bits} 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 IAUFWG may specify additional type names that must be used only to identify specific types of extensions; the full list shall be available from the *FITS* Support Office.

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

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

5.4.2 Other Reserved Keywords

These keywords are optional but may be used only as defined in this standard. They apply to any *FITS* structure with the meanings and restrictions defined below. Any *FITS* structure may further restrict the use of these keywords.

5.4.2.1 Keywords Describing the History or Physical Construction of the HDU

`DATE` **Keyword** Starting January 1, 2000, the following format shall be used. *FITS* writers should commence writing the value of the `DATE` keyword in this format starting January 1, 1999 and before January 1, 2000. The value field shall contain a character string giving the date on which the HDU was created, in the form `YYYY-MM-DD`, or the date and time when the HDU was created, in the form `YYYY-MM-DDThh:mm:ss[.sss...]`, where `YYYY` shall be the four-digit calendar year number, `MM` the two-digit month number with January given by 01 and December by 12, and `DD` the two-digit day of the month. When both date and time are given, the literal `T` shall separate the date and time, `hh` shall be the two-digit hour in the day, `mm` the two-digit number of minutes after the hour, and `ss[.sss...]` the number of seconds (two digits followed by an optional fraction) after the minute. No fields may be defaulted and no leading zeroes omitted. The decimal part of the seconds field is optional and may be arbitrarily long, so long as it is consistent with the rules for value formats of §5.2.

The value of the `DATE` keyword shall always be expressed in UTC when in this format, for all data sets created on earth.

The following format may appear on files written before January 1, 2000. The value field contains a character string giving the date on which the HDU was created, in the form `DD/MM/YY`, where `DD` is 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, the first two digits being understood to be 19. Specification of the date using Universal Time is recommended but not assumed.

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 or institution responsible for 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` is 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. Since the issuance of version 1 of this standard, the `BLOCKED` keyword has been deprecated.

5.4.2.2 Keywords Describing Observations

`DATE-OBS` **Keyword** The format of the value field for `DATE-OBS` keywords shall follow the prescriptions for the `DATE` keyword (§5.4.2.1). Either the 4-digit year format or the 2-digit year format may be used for observation dates from 1900 through 1999 although the 4-digit format is preferred.

When the format with a four-digit year is used, the default interpretations for time shall be UTC for dates beginning 1972-01-01 and UT before. Other date and time scales are permissible. The value of the `DATE-OBS` keyword shall be expressed in the principal time system or time scale of the HDU to which it belongs; if there is any chance of ambiguity, the choice shall be clarified in comments. The value of DATE-OBS shall be assumed to refer to the start of an observation, unless another interpretation is clearly explained in the comment field. Explicit specification of the time scale is recommended. By default, times for TAI and times that run simultaneously with TAI, e.,g., UTC and TT, will be assumed to be as measured at the detector (or, in practical cases, at the observatory). For coordinate times such as TCG, TCB, and TDB which are tied to an unambiguous coordinate system, the default shall be time as if the observation had taken place at the origin of the coordinate time system. Conventions may be developed that use other time systems. Appendix D of this document contains the appendix to the agreement on a four digit year, which discusses time systems in some detail.

When the value of DATE-OBS is expressed in the two-digit year form, allowed for files written before January 1, 2000 with a year in the range 1900-1999, there is no default assumption as to whether it refers to the start, middle or end of an observation.

DATExxxx Keywords The value fields for all keywords beginning with the string DATE whose value contains date, and optionally time, information shall follow the prescriptions for the DATE-OBS keyword.

TELESCOP Keyword The value field shall contain a character string identifying the telescope used to acquire the data associated with the header.

INSTRUME Keyword The value field shall contain a character string identifying the instrument used to acquire the data associated with the header.

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 a name for 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 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 are expressed. Starting with Version 1, this standard has deprecated 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.

5.4.2.3 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.

5.4.2.4 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.

5.4.2.5 Array Keywords

These keywords are used to describe the contents of an array, either alone or in a series of random groups (§7). 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 Eq. 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:

$$\text{physical\_value} = \texttt{BZERO} + \texttt{BSCALE} \times \text{array\_value} \tag{5.3}$$

`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. These units must follow the prescriptions of §5.3.

`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`.

`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`. Units must follow the prescriptions of §5.3.

`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. These units must follow the prescriptions of §5.3.

`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 by 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 by the array, exclusive of any special values.

5.4.2.6 Extension Keywords

These keywords are used to describe an extension and should not appear in the primary header.

`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.

5.4.3 Additional Keywords

5.4.3.1 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.

5.4.3.2 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 §5.4.1.2 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 deprecated `BLOCKED` keyword of §5.4.2.1.

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 [14] 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.

6.2.4 Sixty-four-bit

Sixty-four-bit integers shall be twos-complement signed binary integers, contained in eight bytes.

6.2.5 Unsigned Integers

Unsigned sixteen-bit integers can be represented in *FITS* files by subtracting 32768 from each value (thus offsetting the values into the range of a signed sixteen-bit integer) before writing them to the *FITS* file. The `BZERO` keyword (or the `TZEROn` keyword in the case of binary table columns with `TFORMn = 'I'`) must also be included in the header with its value set to 32768 so that *FITS* reading software will add this offset to the raw values in the *FITS* file, thus restoring them to the original unsigned integer values. Unsigned thirty-two-bit integers can be represented in *FITS* files in a similar way by applying an offset of 2147483648 (2^{31}) to the data values. Unsigned sixty-four-bit integers can be represented in *FITS* files by applying an offset of 9223372036854775808 (2^{63}) to the data values.

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 [15]. `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.

The `BLANK` keyword should not be used when `BITPIX = -32` or `-64`; rather, the IEEE NaN should be used to represent an undefined value. Use of the `BSCALE` and `BZERO` keywords is not recommended.

Appendix H has additional details on the IEEE format.

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. The binary table extension (§8.3) can 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 has been deprecated since the issue of Version 1 of this standard. Use of the word "deprecated" is understood to mean that binary table extensions should be used in new astronomical application areas instead of the random groups format where either is appropriate and where there is no historical precedent for random groups. Existing applications of the random groups structure (almost exclusively interferometry) may continue to use random groups as needed indefinitely;

7.1 Keywords

7.1.1 Mandatory Keywords

The `SIMPLE` keyword is required to be the first keyword in the primary header of all *FITS* files, including those with random groups records. 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. No other keywords may intervene between the `SIMPLE` keyword and the last `NAXISn` keyword.

The total number of bits in the random groups records exclusive of the fill described in §7.2 is given by the following expression:

$$\begin{aligned} N_{\text{bits}} &= |\mathtt{BITPIX}| \times \mathtt{GCOUNT} \times \\ &\quad (\mathtt{PCOUNT} + \mathtt{NAXIS2} \times \mathtt{NAXIS3} \times \cdots \times \mathtt{NAXISm}), \qquad (7.1) \end{aligned}$$

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.

where N_{bits} 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 (§5.4.1).

7.1.1.2 `BITPIX` Keyword

The card image containing this keyword is structured as prescribed in §5.4.1.

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 array in a 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 §5.4.1.

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 elements; 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 Eq. 7.2. The value field shall contain a floating point number representing the coefficient of the linear term in Eq. 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 $\mathtt{n}^{\rm th}$ *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:

$$\text{physical\_value} = \mathtt{PZEROn} + \mathtt{PSCALn} \times \text{group\_parameter\_value} \tag{7.2}$$

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 elements $N_{\rm elem}$ given by the following expression:

$$N_{\rm elem} = (\mathtt{NAXIS2} \times \mathtt{NAXIS3} \times \cdots \times \mathtt{NAXISm}), \tag{7.3}$$

where $N_{\rm elem}$ is the number of elements 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 element 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 (§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 §6. Parameters and elements of associated data arrays shall have the same representation. Should more precision be required for an associated parameter than for an element 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 The ASCII Table Extension

Data shall appear as an ASCII table 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 header of an ASCII table extension must use the keywords defined in Table 8.1. The first keyword must be `XTENSION`; the seven keywords following `XTENSION` (`BITPIX` ...`TFIELDS`) must be in the order specified with no intervening keywords.

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

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 table extensions.

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

`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 format in which field `n` is encoded. Only the formats in Table 8.2, interpreted as ANSI FORTRAN-77 [13] input formats and discussed in more detail in §8.1.5, are permitted for encoding. Format codes must be specified in upper case. Other format editing codes common to ANSI FORTRAN-77 such as repetition, positional editing, scaling, and field termination are not permitted. All values in numeric fields have a number base of ten (i.e., they are decimal); binary, octal, hexadecimal, and other representations are not permitted.

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

Field Value	Data Type
`Aw`	Character
`Iw`	Decimal 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 §8.1.1, the following keywords may be used to describe the structure of an ASCII table data array. They are optional, but if they appear within an ASCII table 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 Eq. 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

$$\text{physical\_value} = \texttt{TZEROn} + \texttt{TSCALn} \times \text{field\_value}. \tag{8.1}$$

`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. 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. Units must follow the prescriptions in §5.3.

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 ANSI 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 table extension field shall be ASCII text in a format that conforms to the rules for fixed field input in ANSI FORTRAN-77 [13] format, as described below, 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. Writers of ASCII tables should select a format appropriate to the form, range of values, and accuracy of the data in the table.

The value of a character-formatted (Aw) field is a character string of width w containing the characters in columns TBCOLn through TBCOLn$+$w $-$ 1.

The value of an integer-formatted (Iw) field is an integer number determined by removing all blanks from columns TBCOLn through TBCOLn$+$w $-$ 1 and interpreting the remaining, right-justified characters as a signed decimal integer. A blank field has value 0. All characters other than blanks, the decimal integers ("0" through "9") and a single leading sign character ("+" and "-") are forbidden.

The value of a real-formatted field (Fw.d, Ew.d, Dw.d) is a real number determined from the w characters from columns TBCOLn through TBCOLn$+$w$-$1. The value is formed by

1. discarding all blank characters and right-justifying the non-blank characters,
2. interpreting the first non-blank characters as a numeric string consisting of a single optional sign ("+" or "-") followed by one or more decimal digits ("0" through "9") optionally containing a single decimal point ("."). The numeric string is terminated by the end of the right-justified field or by the occurrence of any character other than a decimal point (".") and the decimal integers ("0" through "9"). If the string contains no explicit decimal point, then the implicit decimal point is taken as immediately preceding the rightmost d digits of the string, with leading zeros assumed if necessary.
3. if the numeric string is terminated by a
   (a) "+" or "-", interpreting the following string as an exponent in the form of a signed decimal integer, or
   (b) "E", or "D", interpreting the following string as an exponent of the form E or D followed by an optionally signed decimal integer constant.
4. The exponent string, if present, is terminated by the end of the right-justified string.
5. Characters other than those specified above are forbidden.

The numeric value of the table field is then the value of the numeric string multiplied by ten (10) to the power of the exponent string, i.e., value = numeric_string $\times$ $10^{(\text{exponent\_string})}$. The default exponent is zero and a blankfield has value zero. There is no difference between the F, D, and E formats; the content of the string determines its interpretation. Numbers requiring more precision and/or range than the local computer can support may be represented. It is good form to specify a D format in TFORMn for a column of an ASCII table when that column will contain numbers that cannot be accurately represented in 32-bit IEEE binary format (see Appendix H).

Note that the above definitions allow for embedded blanks anywhere in integer-formatted and real-formatted fields and implicit decimal points in real-formatted fields.

FITS reading tasks will have to honor these flexibilities. However, since these flexibilities are likely to cause confusion and possible misinterpretation, it is recommended that *FITS* writing tasks write tables with explicit decimal points and no embedded or trailing blanks whenever possible.

8.2 Image Extension

Data shall appear as an image 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=␣'IMAGE␣␣␣'`.

8.2.1 Mandatory Keywords

The `XTENSION` keyword is required to be the first keyword of all image extensions. The card images in the header of an image extension must use the keywords defined in Table 8.3 in the order specified. No other keywords may intervene between the `XTENSION` and `GCOUNT` keywords.

1	`XTENSION`
2	`BITPIX`
3	`NAXIS`
4	`NAXISn`, `n` $= 1, \ldots,$ `NAXIS`
5	`PCOUNT`
6	`GCOUNT`
	⋮
	(other keywords ...)
	⋮
last	`END`

Table 8.3: Mandatory keywords in image extensions.

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

`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.

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

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

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

`GCOUNT` **Keyword** The value field shall contain the integer 1; each image extension contains a single array.

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

8.2.2 Units

The units of all header keyword values in an image extension shall follow the prescriptions in §5.3.

8.2.3 Data Sequence

The data format shall be identical to that of a primary data array as described in §4.3.2.

8.3 Binary Table Extension

Data shall appear as a binary table 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=␣'BINTABLE'`.

8.3.1 Mandatory Keywords

The `XTENSION` keyword is the first keyword of all binary table extensions. The seven keywords following (`BITPIX` ...`TFIELDS`) must be in the order specified in Table 8.4, with no intervening keywords.

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

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

Table 8.4: Mandatory keywords in binary table extensions.

BITPIX Keyword The value field shall contain the integer 8, denoting that the array is an array of 8-bit bytes.

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 8-bit bytes 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 number of bytes that follow the table in the associated extension data.

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.

TFORMn Keywords The value field of this indexed keyword shall contain a character string of the form $r\mathtt{T}a$. The repeat count r is the ASCII representation of a non-negative integer specifying the number of elements in field `n`. The default value of r is 1; the repeat count need not be present if it has the default value. A zero element count, indicating an empty field, is permitted. The data type `T` specifies the data type of the contents of field `n`. Only the data types in Table 8.5 are permitted. The format codes must be specified in upper case. For fields of type `P` or `Q`, the only permitted repeat counts are 0 and 1. The additional characters a are optional and are not further defined in this standard. Table 8.5 lists the number of bytes each data type occupies in a table row. The first field of a row is numbered 1. The total number of bytes n_{row} in a table row is given by

$$n_{\mathrm{row}} = \sum_{i=1}^{\mathtt{TFIELDS}} r_i b_i \tag{8.2}$$

where r_i is the repeat count for field i, b_i is the number of bytes for the data type in field i, and `TFIELDS` is the value of that keyword, must equal the value of `NAXIS1`.

`TFORMn` value	Description	8-bit Bytes
`L`	Logical	1
`X`	Bit	*
`B`	Unsigned byte	1
`I`	16-bit integer	2
`J`	32-bit integer	4
`K`	64-bit integer	8
`A`	Character	1
`E`	Single precision floating point	4
`D`	Double precision floating point	8
`C`	Single precision complex	8
`M`	Double precision complex	16
`P`	Array Descriptor (32-bit)	8
`Q`	Array Descriptor (64-bit)	16

* number of 8-bit bytes needed to contain all bits

Table 8.5: Valid `TFORMn` data types in `BINTABLE` extensions.

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

8.3.2 Other Reserved Keywords

In addition to the mandatory keywords defined in §8.3.1, these keywords may be used to describe the structure of a binary table data array. They are optional, but if they appear within a binary table extension header, they must be used as defined in this section of this standard.

`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. 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. Units must follow the prescriptions in §5.3.

`TNULLn` Keywords The value field for this indexed keyword shall contain the integer that represents an undefined value for field `n` of data type `B`, `I`, `J` or `K`, or `P` or `Q` array descriptor fields (§8.3.5) that point to `B`, `I`, `J` or `K` integer arrays. The keyword may not be used if field `n` is of any other data type.

`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. It may not be used if the format of field `n` is `A`, `L`, or `X`. The interpretation for fields of type `P` or `Q` is defined in §8.3.5. For fields with all other data types, the value field shall contain a floating point number representing the coefficient of the linear term in Eq. 8.1, which is used to compute the true physical value of the field, or, in the case of the complex data types C and M, of the real part of the field, with the imaginary part of the scaling factor set to zero. The default value for this keyword is 1.0.

`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. It may not be used if the format of field `n` is `A`, `L`, or `X`. The interpretation for fields of type `P` or `Q` is defined in §8.3.5. For fields with all other data types, the value field shall contain a floating point number representing the true physical value corresponding to a value of zero in field `n` of the *FITS* file, or, in the case of the complex data types `C` and `M`, in the real part of the field, with the imaginary part set to zero. The default value for this keyword is 0.0. Equation 8.1 is used to compute a true physical value from the quantity in field `n`.

`TDISPn` **Keywords** The value field of this indexed keyword shall contain a character string describing the format recommended for the display of the contents of field `n`. If the table value has been scaled, the physical value, derived using Eq. 8.1, shall be displayed. All elements in a field shall be displayed with a single, repeated format. For purposes of display, each byte of bit (type `X`) and byte (type `B`) arrays is treated a an unsigned integer. Arrays of type `A` may be terminated with a zero byte. Only the format codes in Table 8.6, discussed in §8.3.4, are permitted for encoding. The format codes must be specified in upper case. If the `Bw.m`, `Ow.m`, and `Zw.m` formats are not readily available to the reader, the `Iw.m` display format may be used instead, and if the `ENw.d` and `ESw.d` formats are not available, `Ew.d` may be used. In the case of fields of type `P` or `Q`, the `TDISPn` value applies to the data array pointed to by the array descriptor (§8.3.5), not the values in the array descriptor itself.

Field Value	Data Type
`Aw`	Character
`Lw`	Logical
`Iw.m`	Integer
`Bw.m`	Binary, integers only
`Ow.m`	Octal, integers only
`Zw.m`	Hexadecimal, integers only
`Fw.d`	Single precision real
`Ew.dEe`	Single precision real, exponential notation
`ENw.d`	Engineering; E format with exponent multiple of 3
`ESw.d`	Scientific; same as EN but nonzero leading digit if not zero
`Gw.dEe`	General; appears as F if significance not lost, else E.
`Dw.dEe`	Double precision real, exponential notation

Table 8.6: Valid `TDISPn` format values in `BINTABLE` extensions. `w` is the width in characters of displayed values, `m` is the minimum number of digits displayed, `d` is the number of digits to right of decimal, and `e` is number of digits in exponent. The `.m` and `Ee` fields are optional.

`THEAP` **Keyword** The value field of this keyword shall contain an integer providing the separation, in bytes, between the start of the main data table and the start of a supplemental data area called the heap. The default value, which is also the minimum allowed value, shall be the product of the values of `NAXIS1` and `NAXIS2`. This keyword shall not be used if the value of `PCOUNT` is zero. The use of this keyword is described in in §8.3.5.

`TDIMn` **Keywords** The value field of this indexed keyword shall contain a character string describing how to interpret the contents of field `n` as a multidimensional array with a format of `'(`l`,`m`,`n`...)'` where l, m, n, ... are the dimensions of the array. The data are 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 total number of elements in the array equals the product of the dimensions specified in the `TDIMn` keyword. The size must be less than or equal to the repeat count on the `TFORMn` keyword, or, in the case of columns that have a "`P`" or "`Q`" `TFORMn` datatype, less than or equal to the array length specified in the variable-length array descriptor (see §8.3.5). In the special case where the variable-length array descriptor has a size of zero, then the `TDIMn` keyword is not applicable. If the number of elements in the array implied by the `TDIMn` is less than the allocated size of the array in the *FITS* file, then the unused trailing elements should be interpreted as containing undefined fill values.

A character string is represented in a binary table by a one-dimensional character array, as described under "Character" in the list of datatypes in §8.3.3.1 ("Main Data Table "). For example, a Fortran 77 `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 `TDIMn` notation. For example, if `TFORMn = '60A␣␣␣␣␣'` and `TDIMn = '(5,4,3)'`, then the entry consists of a 4×3 array of strings of 5 characters each.

8.3.3 Data Sequence

The data in a binary table extension shall consist of a Main Data Table which may, but need not, be followed by additional bytes. The positions in the last data record after the last additional byte, or, if there are no additional bytes, the last character of the last row of the data array, shall be filled by setting all bits to zero.

8.3.3.1 Main Data Table

The table is constructed from a two-dimensional byte array. The number of bytes in a row shall be specified by the value of the `NAXIS1` keyword and the number of rows shall be specified by the `NAXIS2` keyword of the associated header records. Within a row, fields shall be stored in order of increasing column number, as determined from the `n` of the `TFORMn` keywords. The number of bytes in a row and the number of rows in the table shall determine the size of the byte array. Every row in the array shall have the same number of bytes. The first row shall begin at the start of the record immediately following the last header record. Subsequent rows shall begin immediately following the end of the previous row, with no intervening bytes, independent of the record structure. Words need not be aligned along word boundaries.

Each row in the array shall consist of a sequence of fields. The number of elements in each field and their data type 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 be empty, if the repeat count specified in the value of the `TFORMn` keyword of the header is 0. The following data types, and no others, are permitted.

Logical If the value of the `TFORMn` keyword specifies data type `L`, the contents of field `n` shall consist of ASCII `T` indicating true or ASCII `F`, indicating false. A 0 byte (hexadecimal 0) indicates an invalid value.

Bit Array If the value of the `TFORMn` keyword specifies data type `X`, the contents of field `n` shall consist of a sequence of bits starting with the most significant bit; the bits following shall be in order of decreasing significance, ending with the least significant bit. A bit array shall be composed of an integral number of bytes, with those bits following the end of the data set to zero. No null value is defined for bit arrays.

Character If the value of the `TFORMn` keyword specifies data type `A`, field `n` shall contain a character string of zero or more members, composed of ASCII text. This character string may be terminated before the length specified by the repeat count by an ASCII NULL (hexadecimal code 00). Characters after the first ASCII NULL are not defined. A string with the number of characters specified by the repeat count is not NULL terminated. Null strings are defined by the presence of an ASCII NULL as the first character.

Unsigned 8-Bit Integer If the value of the `TFORMn` keyword specifies data type `B`, the data in field `n` shall consist of unsigned 8-bit integers, with the most significant bit first, and subsequent bits in order of decreasing significance. Null values are given by the value of the associated `TNULLn` keyword.

16-Bit Integer If the value of the `TFORMn` keyword specifies data type `I`, the data in field `n` shall consist of twos-complement signed 16-bit integers, contained in two bytes. The most significant byte shall be first. Within each byte the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance. Null values are given by the value of the associated `TNULLn` keyword. Unsigned integers can be represented using the convention described in § 6.2.5.

32-Bit Integer If the value of the `TFORMn` keyword specifies data type `J`, the data in field `n` shall consist of twos-complement signed 32-bit integers, contained in four bytes. The most significant byte shall be first, and subsequent bytes shall be in order of decreasing significance. Within each byte, the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance. Null values are given by the value of the associated `TNULLn` keyword. Unsigned integers can be represented using the convention described in § 6.2.5.

64-Bit Integer If the value of the `TFORMn` keyword specifies data type `K`, the data in field `n` shall consist of twos-complement signed 64-bit integers, contained in eight bytes. The most significant byte shall be first, and subsequent bytes shall be in order of decreasing significance. Within each byte, the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance. Null values are given by the value of the associated `TNULLn` keyword. Unsigned integers can be represented using the convention described in § 6.2.5.

Single Precision Floating Point If the value of the `TFORMn` keyword specifies data type `E`, the data in field `n` shall consist of ANSI/IEEE-754 [15] 32-bit floating point numbers, as described in Appendix H. All IEEE special values are recognized. The IEEE NaN is used to represent invalid values.

Double Precision Floating Point If the value of the `TFORMn` keyword specifies data type `D`, the data in field `n` shall consist of ANSI/IEEE-754 [15] 64-bit double precision floating point numbers, as described in Appendix H. All IEEE special values are recognized. The IEEE NaN is used to represent invalid values.

Single Precision Complex If the value of the `TFORMn` keyword specifies data type `C`, the data in field `n` shall consist of a sequence of pairs of 32-bit single precision floating point numbers. The first member of each pair shall represent the real part of a complex number, and the second member shall represent the imaginary part of that complex number. If either member contains a NaN, the entire complex value is invalid.

Double Precision Complex If the value of the `TFORMn` keyword specifies data type `M`, the data in field `n` shall consist of a sequence of pairs of 64-bit double precision floating point numbers. The first member of each pair shall represent the real part of a complex number, and the second member of the pair shall represent the imaginary part of that complex number. If either member contains a NaN, the entire complex value is invalid.

Array Descriptor If the value of the `TFORMn` keyword specifies data type `P`, the data in field `n` shall consist of not more than one pair of 32-bit integers. If the value of the `TFORMn` keyword specifies data type `Q`, the data in field `n` shall consist of not more than one pair of 64-bit integers. The meaning of these integers is defined in §8.3.5.

8.3.3.2 Bytes Following Main Table

The main data table shall be followed by an additional data area containing zero or more bytes, as specified by the value of the PCOUNT keyword. One use for this data area is described in §8.3.5. This does not preclude other uses for these bytes.

8.3.4 Data Display

Character data are encoded under format code Aw. If the character datum has length less than or equal to w, it is represented on output right-justified in a string of w characters. If the character datum has length greater than w, the first w characters of the datum are represented on output in a string of w characters. Character data are not surrounded by single or double quotation marks unless those marks are themselves part of the data value.

Logical data are encoded under format code Lw. Logical data are represented on output with the character T for true or F for false right justified in a blank-filled string of w characters. A null value may be represented by a completely blank string of w characters.

Integer data (including bit X and byte B type fields) are encoded under format codes Iw.m, Bw.m, Ow.m, and Zw.m. The default value of m is one and the ".m" is optional. The first letter of the code specifies the number base for the encoding with I for decimal (10), B for binary (2), O for octal (8), and Z for hexadecimal (16). Hexadecimal format uses the upper-case letters A through F to represent decimal values 10 through 15. The output field consists of w characters containing zero or more leading blanks followed by a minus if the internal datum is negative followed by the magnitude of the internal datum in the form of an unsigned integer constant in the specified number base with only as many leading zeros as are needed to have at least m numeric digits. Note that $\mathtt{m} \leq \mathtt{w}$ is allowed if all values are positive, but $\mathtt{m} < \mathtt{w}$ is required if any values are negative. If the number of digits required to represent the integer datum exceeds w, then the output field consists of a string of w asterisk (*) characters.

Real data are encoded under format codes Fw.d, Ew.dEe, Dw.dEe, ENw.d, and ESw.d. In all cases, the output is a string of w characters including the decimal point, any sign characters, and any exponent including the exponent's indicators, signs, and values. If the number of digits required to represent the real datum exceeds w, then the output field consists of a string of w asterisk (*) characters. In all cases, d specifies the number of digits to appear to the right of the decimal point. The F format code output field consists of $\mathtt{w} - \mathtt{d} - 1$ characters containing zero or more leading blanks followed by a minus if the internal datum is negative followed by the absolute magnitude of the internal datum in the form of an unsigned integer constant. These characters are followed by a decimal point (".") and d characters giving the fractional part of the internal datum, rounded by the normal rules of arithmetic to d fractional digits. For the E and D format codes, an exponent is taken such that the fraction $0.1 \leq |\mathrm{datum}|/10^{\mathrm{exponent}} < 1.0$. The fraction (with appropriate sign) is output with an F format of width $w - e - 2$ characters with d characters after the decimal followed by an E or D followed by the exponent as a signed $e + 1$ character integer with leading zeros as needed. The default value of e is 2 when the Ee portion of the format code is omitted. If the exponent value will not fit in $e + 1$ characters but will fit in $e + 2$ then the E (or D) is omitted and the wider field used. If the exponent value will not fit (with a sign character) in $e + 2$ characters, then the entire w-character output field is filled with asterisks (*). The ES format code is processed in the same manner as the E format code except that the exponent is taken so that $1.0 \leq \text{fraction} < 10$. The EN format code is processed in the same manner as the E format code except that the exponent is taken to be an integer multiple of 3 and so that $1.0 \leq \text{fraction} < 1000.0$. All real format codes have number base 10. There is no difference between E and D format codes on input other than an implication with the latter of greater precision in the internal datum.

The `Gw.dEe` format code may be used with data of any type. For data of type integer, logical, or character, it is equivalent to `Iw`, `Lw`, or `Aw`, respectively. For data of type real, it is equivalent to an F format (with different numbers of characters after the decimal) when that format will accurately represent the value and is equivalent to an E format when the number (in absolute value) is either very small or very large. Specifically, for real values outside the range $0.1 - 0.5\times10^{-d-1} \leq \text{value} < 10^{d} - 0.5$, it is equivalent to `Ew.dEe`. For real values within the above range, it is equivalent to $\mathtt{Fw}'.\mathtt{d}'$ followed by $2+e$ blanks, where $w' = w - e - 2$ and $d' = d - k$ for $k = 0, 1, \ldots, d$ if the real datum value lies in the range $10^{k-1}\,(1 - 0.5\times10^{-d}) \leq \text{value} \leq 10^{k}\,(1 - 0.5\times10^{-d})$.

Complex data are encoded with any of the real data formats as described above. The same format is used for the real and imaginary parts. It is recommended that the 2 values be separated by a comma and enclosed in parentheses with a total field width of $2w + 3$.

8.3.5 Variable-Length Arrays

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 are 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 the type `P` and `Q` variable-length array descriptors themselves, which are not a datatype so much as a storage class specifier). Other established *FITS* conventions that apply to static arrays will generally apply as well to variable-length arrays.

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

$$\mathtt{rPt}(\mathtt{e}_{\mathrm{max}}) \text{ or } \mathtt{rQt}(\mathtt{e}_{\mathrm{max}})$$

where the "`P`" or "`Q`" indicates the presence of an array descriptor (described below), 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`, `K`, etc., but not `P` or `Q`), and $\mathtt{e}_{\mathrm{max}}$ 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 (other than the fundamental limit imposed by the range of the array descriptor, defined below); $\mathtt{e}_{\mathrm{max}}$ 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. There may be additional characters in the `TFORMn` keyword following the $\mathtt{e}_{\mathrm{max}}$.

For example,

```
TFORM8 = 'PB(1800)' / Variable 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 are not stored in the actual data records; they are 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 signed integer values in the case of "`P`" array descriptors, or two 64-bit signed integer values in the case of "`Q`" array descriptors: 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. The meaning of a negative value for either of these integers is not defined by this standard. 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:

$$\ldots \text{[field } N\text{–1] [(nelem,offset)] [field } N\text{+1]} \ldots$$

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 principal segments, as follows:

[table_header] [record_storage_area] [heap_area]

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` $\times$ `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` $\times$ `NAXIS2`. The total length in bytes of the heap 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 rows each 168 bytes long, with a heap area 3000 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 3 2880-byte *FITS* records: the first record contains the 840 bytes from the 5 rows of the main table followed by 2040 fill bytes; the heap completely fills the second record; the third record contains the remaining 120 bytes of the heap followed by 2760 filll bytes. `PCOUNT` gives the total number of bytes from the end of the main table to the end of the heap and in this example has a value of $2040 + 2880 + 120 = 5040$. This is expressed in the table header as:

```
NAXIS1  =    168 / Width of table row in bytes
NAXIS2  =      5 / Number of rows in table
PCOUNT  =   5040 / Random parameter count
  ...
THEAP   =   2880 / Byte offset of heap area
```

The values of `TSCALn` and `TZEROn` for variable-length array column entries are to be applied to the values in the data array in the heap area, not the values of the array descriptor. These keywords can be used to scale data values in either static or variable-length arrays.

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-length array data may be stored in the heap in any order, i.e., the data for record $N+1$ are not necessarily stored at a larger offset than that for record N. There may be gaps in the heap where no data are 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 are present in each stored record. When data are 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 are 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 may not be supported by some software systems. The producers of *FITS* data products should consider the capabilities of the likely recipients of their files when deciding whether or not to use this format, and as a general rule should use it only in cases where it provides significant advantages over the simpler fixed-length array format. In particular, the use of variable-length arrays may present difficulties for applications that ingest the *FITS* file via a sequential input stream because the application cannot fully process any rows in the table until after the entire fixed-length table and potentially the entire heap has been transmitted as outlined in the previous paragraph.

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

Formal Syntax of Card Images

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

The following notation is used in defining the formal syntax.

:=	means "is defined to be"
X | Y	means one of X or Y (no ordering relation is implied)
[X]	means that X is optional
X...	means X is repeated 1 or more times
'B'	means the ASCII character B
'A'–'Z'	means one of the ASCII characters A through Z
\0xnn	means the ASCII character associated with the hexadecimal code nn
{...}	expresses a constraint or a comment (it immediately follows the syntax rule)

The following statements define the formal syntax used in *FITS* free format card images.

FITS_card_image :=
    FITS_commentary_card_image | FITS_value_card_image

FITS_commentary_card_image :=
    COMMENT_keyword [ascii_text_char...] |
    HISTORY_keyword [ascii_text_char...] |
    BLANKFIELD_keyword [ascii_text_char...] |
    keyword_field anychar_but_equal [ascii_text_char...] |
    keyword_field '=' anychar_but_space [ascii_text_char...]
{Constraint: The total number of characters in a FITS_commentary_card_image must be exactly equal to 80.}

FITS_value_card_image :=
    keyword_field value_indicator [space...] [value] [space...] [comment]

{Constraint: The total number of characters in a FITS_value_card_image must be exactly equal to 80.}

{Comment: If the value field is not present, the value of the *FITS* keyword is not defined.}

keyword_field :=
    [keyword_char...] [space...]

{Constraint: The total number of characters in the keyword_field must be exactly equal to 8.}

keyword_char :=
    ‘A’–‘Z’ | ‘0’–‘9’ | ‘_’ | ‘-’

COMMENT_keyword :=
    ‘C’ ‘O’ ‘M’ ‘M’ ‘E’ ‘N’ ‘T’ space

HISTORY_keyword :=
    ‘H’ ‘I’ ‘S’ ‘T’ ‘O’ ‘R’ ‘Y’ space

BLANKFIELD_keyword :=
    space space space space space space space space

value_indicator :=
    ‘=’ space

space :=
    ‘ ’

comment :=
    ‘/’ [ascii_text_char...]

ascii_text_char :=
    space–‘~’

anychar_but_equal :=
    space–‘<’ | ‘>’–‘~’

anychar_but_space :=
    ‘!’–‘~’

value :=
    character_string_value | logical_value | integer_value | floating_value |
    complex_integer_value | complex_floating_value

character_string_value :=
    begin_quote [string_text_char...] end_quote
{Constraint: The begin_quote and end_quote are not part of the character string value but only serve as delimiters. Leading spaces are significant; trailing spaces are not.}

begin_quote :=
    quote

end_quote :=
    quote
{Constraint: The ending quote must not be immediately followed by a second quote.}

quote :=
    \0x27

string_text_char :=
    ascii_text_char
{Constraint: A string_text_char is identical to an ascii_text_char except for the quote char; a quote char is represented by two successive quote chars.}

logical_value :=
    'T' | 'F'

integer_value :=
    [sign] digit [digit...]
{Comment: Such an integer value is interpreted as a signed decimal number. It may contain leading zeros.}

sign :=
    '-' | '+'

digit :=
    '0'–'9'

floating_value :=
    decimal_number [exponent]

decimal_number :=
  [sign] [integer_part] [?.? [fraction_part]]
{Constraint: At least one of the integer_part and fraction_part must be present.}

integer_part :=
  digit | [digit...]

fraction_part :=
  digit | [digit...]

exponent :=
  exponent_letter [sign] digit [digit...]

exponent_letter :=
  'E' | 'D'

complex_integer_value :=
  '(' [space...] real_integer_part [space...] ',' [space...]
  imaginary_integer_part [space...] ')'

real_integer_part :=
  integer_value

imaginary_integer_part :=
  integer_value

complex_floating_value :=
  '(' [space...] real_floating_part [space...] ',' [space...]
  imaginary_floating_part [space...] ')'

real_floating_part :=
  floating_value

imaginary_floating_part :=
  floating_value

Appendix B

Proposed Binary Table Convention

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

In the paper describing the binary table extension, type name `'BINTABLE'` [10], the authors present three conventions: one for variable length arrays, one for multidimensional arrays and one for substring arrays. The first 2 conventions were subsequently (April 2005) approved by the IAUFWG and have been incorporated into the official *FITS* standard. The draft text for the remaining appendix, available on-line in the directory `http://www.cv.nrao.edu/fits/documents/standards/`, is reproduced here nearly verbatim; the only changes are those required for stylistic consistency with the rest of this document.

B.1 "Substring Array" Convention

This appendix describes a layered convention for specifying that a character array field (`TFORMn = 'rA '`) consists of an array of either fixed-length or variable-length substrings within the field. This convention utilizes the option described in the basic binary table definition to have additional characters following the datatype code character in the `TFORMn` value field. The full form for the value of `TFORMn` within this convention is

`'rA:SSTRw/nnn'`

and a simpler form that may be used for fixed-length substrings only is

`'rAw'`

where

`r` is an integer giving the total length including any delimiters (in characters) of the field,

`A` signifies that this is a character array field,

`:` indicates that a convention indicator follows,

`SSTR` indicates the use of the "Substring Array" convention,

`w` is an integer $\leq$ `r` giving the (maximum) number of characters in an individual substring (not including the delimiter), and

`/nnn` if present, indicates that the substrings have variable-length and are delimited by an ASCII text character with decimal value `nnn` in the range 032 to 126 decimal, inclusive. This character is referred to as the delimiter character. The delimiter character for the last substring will be an ASCII NUL.

To illustrate this usage:

`'40A:SSTR8'` signifies that the field is 40 characters wide and consists of an array of 5 8-character fixed-length substrings. This could also be expressed using the simpler form as `'40A8'`

`'100A:SSTR8/032'` signifies that the field is 100 characters wide and consists of an array of variable-length substrings where each substring has a maximum length of 8 characters and, except for the last substring, is terminated by an ASCII SPACE (decimal 32) character.

Note that simple *FITS* readers that do not understand this substring convention can ignore the `TFORM` characters following the `rA` and can interpret the field simply as a single long string as described in the basic binary table definition.

The following rules complete the full definition of this convention:

1. In the case of fixed-length substrings, if `r` is not an integer multiple of `w` then the remaining odd characters are undefined and should be ignored. For example if `TFORMn ='14A:SSTR3'`, then the field contains 4 3-character substrings followed by 2 undefined characters.

2. Fixed-length substrings must always be padded with blanks if they do not otherwise fill the fixed-length subfield. The ASCII NUL character must not be used to terminate a fixed-length substring field.

3. The character following the delimiter character in variable-length substrings is the first character of the following substring.

4. The method of signifying an undefined or null substring within a fixed-length substring array is not explicitly defined by this convention (note that there is no ambiguity if the variable-length format is used). In most cases it is recommended

that a completely blank substring or other adopted convention (e.g. `'INDEF'`) be used for this purpose although general readers are not expected to recognize these as undefined strings. In cases where it is necessary to make a distinction between a blank, or other, substring and an undefined substring use of variable-length substrings is recommended.

5. Undefined or null variable-length substrings are designated by a zero-length substring, i.e., by a delimiter character (or an ASCII NUL if it is the last substring in the table field) in the first position of the substring. An ASCII NUL in the first character of the table field indicates that the field contains no defined variable-length substrings.

6. The "Multidimensional Array"convention described in §8.3.2 of this paper provides a syntax using the `TDIMn` keyword for describing multidimensional arrays of any datatype which can also be used to represent arrays of fixed-length substrings. For a one dimensional array of substrings (a two dimensional array of characters) the "Substring Array" convention is preferred over the "Multidimensional Array" convention. Multidimensional arrays of (fixed length) strings require the use of the "Multidimensional Array" convention.

7. This substring convention may be used in conjunction with the "Variable Length Array" facility described in §8.3.5. In this case, the two possible full forms for the value of the `TFORM` keyword are

$$\texttt{TFORMn = 'rPA(e}_{\mathrm{max}}\texttt{):SSTRw/nnn'}$$

and

$$\texttt{TFORMn = 'rPA(e}_{\mathrm{max}}\texttt{):SSTRw'}$$

for the variable and fixed cases, respectively.

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

Appendix C

Implementation on Physical Media

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

C.1 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.

C.2 Labeling

C.2.1 Tape

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

C.2.2 Other Media

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

C.3 FITS File Boundaries

C.3.1 Magnetic Reel Tape

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

C.3.2 Other Media

For fixed block length sequential media where the physical block 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.

C.4 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]. A convention for logically grouping on-line *FITS* HDUs that may physically be located in different sites has been proposed in [16].

Appendix D

Suggested Time Scale Specification

[Not part of formal `DATExxxx` agreement]

1. Use of the keyword `TIMESYS` is suggested as an implementation of the time scale specification. It sets the principal time system for time-related keywords and data in the HDU (i.e., it does not preclude the addition of keywords or data columns that provide information for transformations to other time scales, such as sidereal times or barycenter corrections). Each HDU shall contain not more than one `TIMESYS` keyword. Initially, officially allowed values are:

   `UTC` Coordinated Universal Time; defined since 1972.

   `UT` Universal Time, equal to Greenwich Mean Time (GMT) since 1925; the UTC equivalent before 1972; see: Explanatory Supplement, p. 76.

   `TAI` International Atomic Time; "UTC without the leap seconds"; 31 s ahead of UTC on 1997-07-01.

   `AT` International Atomic Time; deprecated synonym of TAI.

   `ET` Ephemeris Time, the predecessor of TT; valid until 1984.

   `TT` Terrestrial Time, the IAU standard time scale since 1984; continuous with ET and synchronous with (but 32.184 s ahead of) TAI.

   `TDT` Terrestrial Dynamical Time; = TT.

   `TDB` Barycentric Dynamical Time.

   `TCG` Geocentric Coordinate Time; runs ahead of TT since 1977-01-01 at a rate of approximately 22 ms/year.

   `TCB` Barycentric Coordinate Time; runs ahead of TDB since 1977-01-01 at a rate of approximately 0.5 s/year.

For reference, see: Explanatory Supplement to the Astronomical Almanac, P. K. Seidelmann, ed., University Science Books, 1992, ISBN 0-935702-68-7, or

`http://tycho.usno.navy.mil/systime.html`

Use of Global Positioning Satellite (GPS) time (19 s behind TAI) is deprecated.

2. By default, times will be deemed to be as measured at the detector (or in practical cases, at the observatory) for times that run synchronously with TAI (i.e., TAI, UTC, and TT). In the case of coordinate times (such as TCG and TCB) and TDB which are tied to an unambiguous coordinate origin, the default meaning of time values will be: time as if the observation had taken place at the origin of the coordinate time system. These defaults follow common practice; a future convention on time scale issues in *FITS* files may allow other combinations but shall preserve this default behavior. The rationale is that raw observational data are most likely to be tagged by a clock that is synchronized with TAI, while a transformation to coordinate times or TDB is usually accompanied by a spatial transformation, as well. This implies that path length differences have been corrected for. Note that the difference TDB − UTC, in that case, is approximately sinusoidal, with period one year and amplitude up to 500 s, depending on source position. Also, note that when the location is not unambiguous (such as in the case of an interferometer) precise specification of the location is strongly encouraged in, for instance, geocentric Cartesian coordinates.

3. Note that `TT` is the IAU preferred standard. It may be considered equivalent to `TDT` and `ET`, though `ET` should not be used for data taken after 1984. For reference, see: Explanatory Supplement, pp. 40-48.

4. If the `TIMESYS` keyword is absent or has an unrecognized value, the value `UTC` will be assumed for dates since 1972, and `UT` for pre-1972 data.

5. Examples. The three legal representations of the date of October 14, 1996, might be written as:

```
DATE-OBS= '14/10/96'           / Original format, means 1996 Oct 14.

TIMESYS = 'UTC     '           / Explicit time scale specification: UTC.
DATE-OBS= '1996-10-14'         / Date of start of observation in UTC.

DATE-OBS= '1996-10-14'         / Date of start of observation, also in UTC.

TIMESYS = 'TT      '           / Explicit time scale specification: TT.
DATE-OBS= '1996-10-14T10:14:36.123' / Date and time of start of obs. in TT.
```

6. The convention suggested in this Appendix is part of the mission-specific *FITS* conventions adopted for, and used in, the RXTE archive, building on existing High Energy Astrophysics *FITS* conventions. See:

   `http://heasarc.gsfc.nasa.gov/docs/xte/abc/time_tutorial.html`
   `http://heasarc.gsfc.nasa.gov/docs/xte/abc/time.html`

   The VLBA project has adopted a convention where the keyword `TIMSYS`, rather than `TIMESYS`, is used, currently allowing the values `UTC` and `IAT`. See p. 9 and p. 16 of:

   `http://www.cv.nrao.edu/fits/documents/drafts/vlba_format.ps`

Appendix E

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], [5], [9], and [10] collectively, the term *Floating Point Agreement (FPA)* refers to [8], the term *Blocking Agreement* refers to [11]; and the term *DATExxxx Agreement* refers to the redefinition of the value format for date keywords approved by the IAUFWG in 1997.

1. §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.

Fraction — This terminology and the distinction between *fraction* and *mantissa* do not appear in the Floating Point Agreement.

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.

Repeat count —This terminology is not used in the *FITS* papers.

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 *conforming extension.*

2. §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. §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 *FITS* Support Office as its agent, and mandates registration.

4. §4.6 Physical Blocking
   This material is based entirely on the Blocking Agreement. Material in the early *FITS* papers [1,4] specifying the expression of *FITS* on specific physical media is not part of this standard.

5. §4.6.1 Bitstream Devices
   The Blocking Agreement specifies that this rule applies to *FITS* files written to logical file systems. This standard applies the rule to all bitstream devices, not only logical file systems.

6. §4.6.2.1 Fixed Block
   The Blocking Agreement specifies that this rule applies to *FITS* files written to optical disks, (accessed as a sequential set of records), QIC format 1/4-inch cartridge tapes and Local Area networks. This standard extends the rule to other fixed block length sequential media.

7. §4.6.2.2 Variable Block
   The Blocking Agreement specifies that this rule applies to *FITS* files written to 1/2-inch 9 track tapes, DDS/DAT 4mm cartridge tapes and 8mm cartridge tape (Exabyte). This standard extends the rule to all variable block length sequential media and eliminates references to specific products.

8. §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.

9. §5.1.2.2 Value Indicator (bytes 9–10)
The *FITS* papers do not specifically address the permissibility of null values. This standard states explicitly that they are permitted.

10. §5.1.2.3 Value/Comment (bytes 11–80)
In the *FITS* papers, the slash between the value and comment is optional. This standard requires the slash, consistent with the prescription of FORTRAN-77 list-directed input.

11. §5.2 Value, including its subsections
The *FITS* papers specify that the value field is to be written following the rules of ANSI FORTRAN-77 list-directed input, with some restrictions. This standard explicitly describes the format of the value field. The *FITS* papers permit the value field to contain an array of values. This standard specifies that there shall be only one value in the value field. The *FITS* papers require the fixed format for the most essential parameters. This standard identifies those parameters with the values of the mandatory keywords.

12. §5.2.1 Character String
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.

    The standard states that in general, character-valued keywords can have lengths up to the maximum 68 character length.

13. §5.2.3 Integer
The standard explicitly notes that the fixed format for complex integers does not conform to the rules for ANSI FORTRAN list-directed read.

14. §5.2.4 Real Floating Point Number
The standard explicitly notes that the full precision of 64-bit values cannot be expressed as a single value using the fixed format.

15. §5.2.5 Complex Integer Number
The standard does not support the fixed format for complex integers defined in the *FITS* papers but is consistent with FORTRAN-77 list-directed read as required in the *FITS* papers for free format. Because the fixed format of the *FITS* papers did not conform to the rules for FORTRAN-77 list-directed I/O, consistency with both was impossible. There are no known *FITS* files that use the fixed format for complex integers that was defined in the *FITS* papers.

16. §5.2.6 Complex Floating Point Number
The standard does not support the fixed format for complex floating point numbers defined in the *FITS* papers but is consistent with FORTRAN-77 list-directed read as required in the *FITS* papers for free format. Because the fixed format of the *FITS* papers did not conform to the rules for FORTRAN-77 list-directed I/O, consistency with both was impossible. There are no known *FITS* files that use the fixed format for complex floating point numbers that was defined in the *FITS* papers.

17. §5.3 Units
The *FITS* papers recommend the use of SI units and identify certain other units standard in astronomy. This standard codifies the recommendation and makes it more specific by referring to the IAU Style Manual [7], while explicitly recommending degrees for angular measure and requiring degrees for celestial coordinates.

18. §5.4.1.1 Principal (mandatory keywords)

    (a) `SIMPLE` keyword — The explicit prohibition against the appearance of the `SIMPLE` keyword in extensions does not appear in the *FITS* papers.
    (b) `NAXIS` keyword — The requirement that the `NAXIS` keyword may not be negative is not explicitly specified in the *FITS* papers.
    (c) `NAXISn` keyword — The requirement that the `NAXISn` keyword may not be negative is not explicitly specified in the *FITS* papers.

19. §5.4.1.2 Conforming Extensions

    (a) N_{bits} — The requirement that N_{bits} 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 *FITS* Support Office as its agent.

20. §5.4.2 Other Reserved Keywords
That the optional keywords defined in the *FITS* papers are to be reserved for both the primary HDUs and all extensions with the meanings and usage defined in those papers, as in the standard, is not explicitly stated in all of them, although some keywords are explicitly reserved in the papers describing the image and binary table extensions.

21. §5.4.2.1 Keywords Describing the History or Physical Construction of the HDU

    (a) `DATE` Keyword — The notation for four-digit year number is `YYYY` rather than the `CCYY` of the "DATExxxx Agreement". The recommendation for use of Universal Time in the superseded format with a two-digit year 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. This standard deprecates the `BLOCKED` keyword.

22. §5.4.2.2 Keywords Describing Observations

    (a) `DATE-OBS` Keyword — The recommendation for use of Universal Time in the superseded format with a two-digit year 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.

23. §5.4.2.4 Commentary keywords
    Keyword field is blank — Reference [1] contains the text "`BLANK`" to represent a blank keyword field. The standard clarifies the intention.

24. §5.4.2.5 Array keywords

    (a) `BUNIT` Keyword — The *FITS* papers recommend the use of SI units, degrees as the appropriate unit for angles, and identify other units standard in astronomy. This standard specically applies the recommendations of §5.3 to the `BUNIT` keyword.

    (b) `CTYPEn`, `CRVALn`, `CDELTn`, and `CROTAn` Keywords — This standard extends the recommendations on units to coordinate axes, explicitly requiring decimal degrees for coordinates.

    (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.

25. §7 Random Groups Structure
The standard deprecates the Random Groups structure.

26. §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.

27. §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.

28. §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.

29. §8.1 ASCII Table Extension
The name *ASCII table* is given to the "tables" extension discussed in the *FITS* papers to distinguish it from the binary table extension.

30. §8.1.1 Mandatory Keywords (ASCII table)

    (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.

    (d) `TFORMn` keyword — The requirement that format codes must be specified in upper case is implied but not explicitly specified in the *FITS* papers.

31. §8.1.2 Other Reserved Keywords (ASCII table)
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` Keywords — 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 requiring conformance to the prescriptions in §5.3.

(b) `TSCALn` Keywords — 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".

32. §8.3.2 Other Reserved Keywords (Binary Table)
The `EXTNAME`, `EXTVER`, `EXTLEVEL`, `AUTHOR`, and `REFERENC` keywords explicitly reserved for binary tables in the defining paper are reserved in the standard under the general prescription of §5.4.2.

(a) `TUNITn` Keywords — The *FITS* papers do not explicitly recommend the use of any particular units for this keyword. This standard makes the recommendation more specific for the `TUNITn` keyword by requiring conformance to the prescriptions of §5.3.

(b) `TDISPn` Keywords — The version of the `BINTABLE` paper upon which the *FITS* committees voted stated incorrectly that the values used to display bit and byte arrays should be considered *signed*. This standard follows the text in the published `BINTABLE` paper, which specifies that these values should be *unsigned*. The `BINTABLE` paper does not specify how a `TDISPn` value for a field of type `P` is interpreted; this standard explicitly mandates no interpretation but allows conventions to provide interpretations. The requirement that format codes must be specified in upper case is implied but not explicitly specified in the `BINTABLE` paper.

(c) `THEAP` Keywords — The *FITS* papers state only that the keyword is reserved for use in the convention described in in §8.3.5. This standard makes the more specific statement that this keyword is used to provide the separation, in bytes, between the start of the main data table and the start of a supplemental data area called the heap and identifies the default value.

(d) `TDIMn` Keywords — The *FITS* papers state only that the keyword is reserved for use in the convention described in Appendix 8.3.2. This standard makes the more specific statement that the contents of the value field contain a character string describing how to interpret the contents of a field as a multidimensional array.

33. §8.3.4 Data Display
The `BINTABLE` paper suggests that the format for display suggested by the `TDISPn`

should be understood as a Fortran-90 format or, where Fortran-90 is unavailable, a FORTRAN-77 format. This standard explicitly describes the formats. The statement in the standard concerning differences between `E` and `D` format codes, which notes that the latter implies greater precision in the internal datum, does not appear in the `BINTABLE` paper.

34. §9 Restrictions on Changes
    The *FITS* papers do not provide for the concept of deprecation.

35. Appendix C Implementation on Physical Media
    Material in the *FITS* papers specifying the expression of *FITS* on specific physical media is not part of this standard; what is provided in the appendix is purely as a guide to recommended practices.

Appendix F

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	Image Extension	Binary Table Extension	Random Groups Records
`SIMPLE`	`XTENSION`	`XTENSION`[1]	`XTENSION`[2]	`XTENSION`[3]	`SIMPLE`
`BITPIX`	`BITPIX`	`BITPIX = 8`	`BITPIX`	`BITPIX = 8`	`BITPIX`
`NAXIS`	`NAXIS`	`NAXIS = 2`	`NAXIS`	`NAXIS = 2`	`NAXIS`
`NAXISn`[4]	`NAXISn`[4]	`NAXIS1`	`NAXISn`[4]	`NAXIS1`	`NAXIS1 = 0`
`EXTEND`[5]	`PCOUNT`	`NAXIS2`	`PCOUNT = 0`	`NAXIS2`	`NAXISn`[4]
`END`	`GCOUNT`	`PCOUNT = 0`	`GCOUNT = 1`	`PCOUNT`	`GROUPS = T`
	`END`	`GCOUNT = 1`	`END`	`GCOUNT = 1`	`PCOUNT`
		`TFIELDS`		`TFIELDS`	`GCOUNT`
		`TBCOLn`[6]		`TFORMn`[6]	`END`
		`TFORMn`[6]		`END`	
		`END`			

[1] `XTENSION= 'TABLE    '` for the ASCII table extension.
[2] `XTENSION= 'IMAGE    '` for the image extension.
[3] `XTENSION= 'BINTABLE'` for the binary table extension.
[4] Runs from 1 through the value of `NAXIS`.
[5] Required only if extensions are present.
[6] Runs from 1 through the value of `TFIELDS`.

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

All HDUs	Array[1] HDUs	Conforming Extension	ASCII Table Extension	Binary Table Extension	Random Groups Records
DATE	BSCALE	EXTNAME	TSCALn	TSCALn	PTYPEn
ORIGIN	BZERO	EXTVER	TZEROn	TZEROn	PSCALn
BLOCKED[2]	BUNIT	EXTLEVEL	TNULLn	TNULLn	PZEROn
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[2]					

[1] Primary HDU, image extension, user-defined HDUs with same array structure.
[2] Deprecated.

Table F.2: Reserved *FITS* keywords for the structures described in this document.

Production	Bibliographic	Commentary	Observation
DATE	AUTHOR	COMMENT	DATE-OBS
ORIGIN	REFERENC	HISTORY	TELESCOP
BLOCKED[1]		␣␣␣␣␣␣␣␣	INSTRUME
			OBSERVER
			OBJECT
			EQUINOX
			EPOCH[1]

[1] Deprecated.

Table F.3: General reserved *FITS* keywords described in this document.

Appendix G

ASCII Text

(This appendix is not part of the NOST *FITS* standard; the material in it is based on the ANSI standard for ASCII [14] 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 G.1: ASCII character set

Appendix H

IEEE Floating Point Formats

(The material in this Appendix is not part of this standard; it is adapted from the IEEE-754 floating point standard [15] and provided for informational purposes. It is not intended to be a comprehensive description of the IEEE formats; readers should refer to the IEEE standard.)

FITS recognizes all IEEE basic formats, including the special values.

H.1 Basic Formats

Numbers in the single and double formats are composed of the following three fields:

1. 1-bit sign s
2. Biased exponent $e = E + bias$
3. Fraction $f = \bullet b_1 b_2 \cdots b_{p-1}$

The range of the unbiased exponent E shall include every integer between two values E_{min} and E_{max}, inclusive, and also two other reserved values $E_{min} - 1$ to encode ± 0 and denormalized numbers, and $E_{max}+1$ to encode $\pm\infty$ and NaNs. The foregoing parameters are given in Table H.1. Each nonzero numerical value has just one encoding. The fields are interpreted as follows:

H.1.1 Single

A 32-bit single format number X is divided as shown in Fig. H.1. The value v of X is inferred from its constituent fields thus

1. If $e = 255$ and $f \neq 0$, then v is NaN regardless of s
2. If $e = 255$ and $f = 0$, then $v = (-1)^s \infty$

Parameter	Format			
	Single	Single Extended	Double	Double Extended
p	24	≥ 32	53	≥ 64
E_{max}	$+127$	$\geq +1023$	$+1023$	$\geq +16383$
E_{min}	-126	≤ -1022	-1022	≤ -16382
Exponent *bias*	$+127$	unspecified	$+1023$	unspecified
Exponent width in bits	8	≥ 11	11	≥ 15
Format width in bits	32	≥ 43	64	≥ 79

Table H.1: Summary of Format Parameters

3. If $0 < e < 255$, then $v = (-1)^s 2^{e-127}(1 \bullet f)$
4. If $e = 0$ and $f \neq 0$, then $v = (-1)^s 2^{e-126}(0 \bullet f)$ (denormalized numbers)
5. If $e = 0$ and $f = 0$, then $v = (-1)^s 0$ (zero)

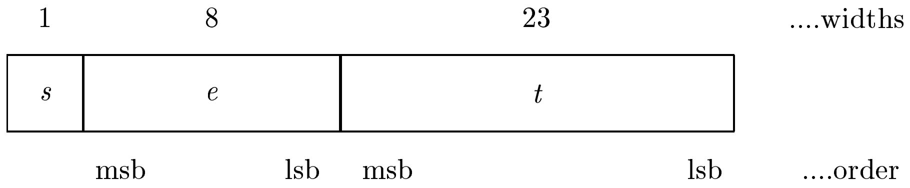

Figure H.1: Single Format. `msb` means *most significant bit*, `lsb` means *least significant bit*

H.1.2 Double

A 64-bit double format number X is divided as shown in Fig. H.2. The value v of X is inferred from its constituent fields thus

1. If $e = 2047$ and $f \neq 0$, then v is NaN regardless of s
2. If $e = 2047$ and $f = 0$, then $v = (-1)^s \infty$
3. If $0 < e < 2047$, then $v = (-1)^s 2^{e-1023}(1 \bullet f)$
4. If $e = 0$ and $f \neq 0$, then $v = (-1)^s 2^{e-1022}(0 \bullet f)$ (denormalized numbers)

5. If $e = 0$ and $f = 0$, then $v = (-1)^s 0$ (zero)

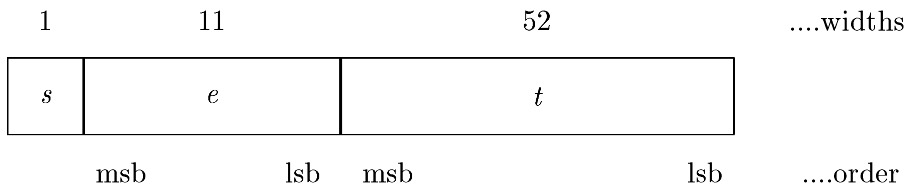

Figure H.2: Double Format. `msb` means *most significant bit*, `lsb` means *least significant bit*

H.2 Byte Patterns

Table H.2 shows the types of IEEE floating point value, whether regular or special, corresponding to all double and single precision hexadecimal byte patterns.

IEEE value	Double Precision	Single Precision
$+0$	0000000000000000	00000000
denormalized	0000000000000001	00000001
	to	to
	000FFFFFFFFFFFFF	007FFFFF
positive underflow	0010000000000000	00800000
positive numbers	0010000000000001	00800001
	to	to
	7FEFFFFFFFFFFFFE	7F7FFFFE
positive overflow	7FEFFFFFFFFFFFFF	7F7FFFFF
$+\infty$	7FF0000000000000	7F800000
NaN[1]	7FF0000000000001	7F800001
	to	to
	7FFFFFFFFFFFFFFF	7FFFFFFF
-0	8000000000000000	80000000
negative	8000000000000001	80000001
denormalized	to	to
	800FFFFFFFFFFFFF	807FFFFF
negative underflow	8010000000000000	80800000
negative numbers	8010000000000001	80800001
	to	to
	FFEFFFFFFFFFFFFE	FF7FFFFE
negative overflow	FFEFFFFFFFFFFFFF	FF7FFFFF
$-\infty$	FFF0000000000000	FF800000
NaN[1]	FFF0000000000001	FF800001
	to	to
	FFFFFFFFFFFFFFFF	FFFFFFFF

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

Table H.2: IEEE Floating Point Formats

Appendix I

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 *FITS* Support Office at

`http://fits.gsfc.nasa.gov/xtension.html`

or

`ftp://nssdc.gsfc.nasa.gov/pub/fits/xtension.lis`

Type Name	Status	Reference	Sponsor	Comments
`'A3DTABLE'`	L	[17]	NRAO	Prototype binary table design used in AIPS; subset of `BINTABLE`.
`'BINTABLE'`	S	[10]	IAU	Binary table extension. Available at *FITS* Archives in files /documents/standards/bintable.aa* of 1995-Feb-06. Note: only main document, excluding appendixes.
`'COMPRESS'`	R	none	GSFC A/WWW	Suggested extension name by A. Warnock. Preliminary proposal in *FITS* archives in the files `compress.*`.
`'DUMP␣␣␣␣'`	R	none	none	Suggested extension name for binary dumps. No full proposal submitted.
`'FILEMARK'`	R	none	NRAO	Suggested for equivalent of tape mark on other media. No full proposal submitted.
`'IMAGE␣␣␣'`	S	[9]	IAU	Image extension.
`'IUEIMAGE'`	L	[18]	IUE	Local extension originally defined for archiving special IUE data products, Identical to `IMAGE`.
`'TABLE␣␣␣'`	S	[5]	IAU	ASCII table extension.
`'VGROUP␣␣'`	R	none	GSFC	Suggested extension name for HDF Vgroups (D. Jennings) No formal proposal; not used in current HDF-*FITS* conversion proposals

Table I.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 I.2: Status Codes

Acronym	Meaning
NRAO	National Radio Astronomy Observatory
AIPS	Astronomical Image Processing System
A/WWW	A/WWW Enterprises
HDF	Hierarchical Data Format

Table I.3: Acronyms in List of Registered Extensions

Appendix J

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
NOST 100-1.1	*FITS* Definition Standard	June, 1995	Proposed Standard
NOST 100-1.1	*FITS* Definition Standard	September, 1995	NOST Standard
NOST 100-1.2	*FITS* Definition Standard	April, 1998	Draft Standard
NOST 100-2.0	*FITS* Definition Standard	March, 1999	NOST Standard

Table J.1: NOST Publications

Index