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

DRAFT

The FITS*FITS* Standard

Version 4.0: updated 2016 July 22 by the IAUFWG

Document Original document publication date: 2016 July 22
Language-edited document publication date: 2018 August xx

FITS*FITS* Working Group [1]
Commission 5: Documentation and Astronomical Data [2]
International Astronomical Union
`http://fits.gsfc.nasa.gov/iaufwg/`

[1] to be absorbed in Data Representation Working Group under new Commission B2
[2] now Commission B2 Data and Documentation

DRAFT

Contents

Contents **i**

1 Introduction **1**
1.1 Brief history of FITS*FITS* . . . 1
1.2 Version history of this document . . . 1
1.3 Acknowledgments . . . 2

2 Definitions, acronyms, and symbols **3**
2.1 Conventions used in this document . . . 3
2.2 Defined terms . . . 3

3 FITS*FITS* file organization **4**
3.1 Overall file structure . . . 4
3.2 Individual FITS*FITS* Structures . . . 5
3.3 Primary header and data unit . . . 5
3.3.1 Primary header . . . 5
3.3.2 Primary data array . . . 5
3.4 Extensions . . . 5
3.4.1 Requirements for conforming extensions . . . 5
3.4.2 Standard extensions . . . 6
3.4.3 Order of extensions . . . 6
3.5 Special records (restricted use) . . . 6
3.6 Physical blocking . . . 6
3.6.1 Bitstream Bit-stream devices . . . 6
3.6.2 Sequential media . . . 6
3.7 Restrictions on changes . . . 6

4 Headers **6**
4.1 Keyword records . . . 6
4.1.1 Syntax . . . 6
4.1.2 Components . . . 6
4.2 Value . . . 7
4.2.1 Character string . . . 7
4.2.2 Logical . . . 8
4.2.3 Integer number . . . 8
4.2.4 Real floating-point number . . . 9
4.2.5 Complex integer number . . . 9
4.2.6 Complex floating-point number . . . 9
4.2.7 Date . . . 9
4.3 Units . . . 9
4.3.1 Construction of units strings . . . 10
4.3.2 Units in comment fields . . . 11
4.4 Keywords . . . 11
4.4.1 Mandatory keywords . . . 11
4.4.2 Other reserved keywords . . . 13
4.4.3 Additional keywords . . . 17

5 Data Representationrepresentation **17**
5.1 Characters . . . 17
5.2 Integers . . . 17
5.2.1 Eight-bit . . . 17
5.2.2 Sixteen-bit . . . 17
5.2.3 Thirty-two-bit . . . 17
5.2.4 Sixty-four-bit . . . 17
5.2.5 Unsigned integers . . . 17
5.3 IEEE-754 floating point . . . 17
5.4 Time . . . 18

DRAFT

6 Random groups Random-groups structure **18**
6.1 Keywords . . . 18
6.1.1 Mandatory keywords . . . 18
6.1.2 Reserved keywords . . . 18
6.2 Data sequence . . . 19
6.3 Data representation . . . 19

7 Standard extensions **19**
7.1 Image extension . . . 19
7.1.1 Mandatory keywords . . . 19
7.1.2 Other reserved keywords . . . 20
7.1.3 Data sequence . . . 20
7.2 The ASCII table ASCII-table extension . . . 20
7.2.1 Mandatory keywords . . . 20
7.2.2 Other reserved keywords . . . 21
7.2.3 Data sequence . . . 22
7.2.4 Fields . . . 23
7.2.5 Entries . . . 23
7.3 Binary table Binary-table extension . . . 24
7.3.1 Mandatory keywords . . . 24
7.3.2 Other reserved keywords . . . 24
7.3.3 Data sequence . . . 27
7.3.4 Data display . . . 28
7.3.5 Variable-length arrays . . . 30
7.3.6 Variable-length-array guidelines . . . 31

8 World coordinate World-coordinate systems **31**
8.1 Basic concepts . . . 31
8.2 World coordinate system World-coordinate-system representations . . . 32
8.2.1 Alternative WCS axis descriptions . . . 35
8.3 Celestial coordinate system Celestial-coordinate-system representations . . . 35
8.4 Spectral coordinate system Spectral-coordinate-system representations . . . 37
8.4.1 Spectral coordinate Spectral-coordinate reference frames . . . 37
8.5 Conventional coordinate Conventional-coordinate types . . . 39

9 Representations of time coordinates **39**
9.1 Time values . . . 39
9.1.1 ISO-8601 datetime*datetime* strings . . . 39
9.1.2 Julian and Besselian epochs . . . 40
9.2 Time coordinate frame . . . 40
9.2.1 Time scale . . . 40
9.2.2 Time reference value . . . 41
9.2.3 Time reference position . . . 42
9.2.4 Time reference direction . . . 43
9.2.5 Solar System Ephemerisephemeris . . . 43
9.3 Time unit . . . 43
9.4 Time offset, binning, and errors . . . 44
9.4.1 Time offset . . . 44
9.4.2 Time resolution and binning . . . 44
9.4.3 Time errors . . . 44
9.5 Global time keywords . . . 45
9.6 Other time coordinate time-coordinate axes . . . 45
9.7 Durations . . . 46
9.8 Recommended best practices . . . 46
9.8.1 Global keywords and overrides . . . 46
9.8.2 Restrictions on alternate descriptions . . . 46
9.8.3 Image time axes . . . 46

DRAFT

10 Representations of compressed data **47**
10.1 Tiled Image Compressionimage compression . . . 47
10.1.1 Required Keywordskeywords . . . 47
10.1.2 Other Reserved Keywordsreserved keywords . . . 47
10.1.3 Table Columnscolumns . . . 48
10.2 Quantization of Floating-Point Datafloating-point data . . . 49
10.2.1 Dithering Algorithmsalgorithms . . . 50
10.2.2 Preserving undefined pixels with lossy compression . . . 51
10.3 Tiled Table Compressiontable compression . . . 51
10.3.1 Required Keywordskeywords . . . 51
10.3.2 Procedure for Table Compressiontable compression . . . 52
10.3.3 Compression Directive Keywordsdirective keywords . . . 52
10.3.4 Other Reserved Keywordsreserved keywords . . . 52
10.3.5 Supported Compression Algorithms compression algorithms for Tablestables . . . 52
10.3.6 Compressing Variable-Length Array Columnsvariable-length array columns . . . 53
10.4 Compression Algorithmsalgorithms . . . 53
10.4.1 Rice compression . . . 53
10.4.2 GZIP Gzip compression . . . 53
10.4.3 IRAF/PLIO compression . . . 54
10.4.4 H-Compress algorithm . . . 54

A Syntax of keyword records **56**

B Suggested time scale time-scale specification **57**

C Summary of keywords **58**

D ASCII text **60**

E IEEE floating-point formats **60**
E.1 Basic formats . . . 60
E.1.1 Single . . . 61
E.1.2 Double . . . 61
E.2 Byte patterns . . . 61

F Reserved extension type names **63**
F.1 Standard extensions . . . 63
F.2 Conforming extensions . . . 63
F.3 Other suggested extension names . . . 63

G MIME types **63**
G.1 MIME type ‘application/fitsapplication/fits’ . . . 63
G.1.1 Recommendations for application writers . . . 64
G.2 MIME type ‘image/fitsimage/fits’ . . . 64
G.2.1 Recommendations for application writers . . . 64
G.3 File extensions . . . 65

H Past changes or clarifications to the formal definition of FITS*FITS* **65**
H.1 Differences between the requirements in this standard Standard and the requirements in the original FITS*FITS* papers. 65
H.2 List of modification to the FITSstandard*FITS* Standard, version 3Version 3.0 . . . 65
H.3 List of modifications to the latest FITS standard*FITS* Standard . . . 66
H.4 List of modifications for language editing . . . 67

I Random Number GeneratorRandom-number generator **67**

J CHECKSUMImplementation GuidelinesCHECKSUM implementation guidelines **67**
J.1 Recommended CHECKSUMKeyword ImplementationCHECKSUM keyword implementation . . . 68
J.2 Recommended ASCII Encoding Algorithmencoding algorithm . . . 68
J.3 Encoding Exampleexample . . . 69
J.4 Incremental Updating updating of the Checksumchecksum . . . 69
J.5 Example *C* Code code for Accumulating accumulating the Checksumchecksum . . . 69
J.6 Example *C* Code code for ASCII Encodingencoding . . . 70

DRAFT

K Header inheritance convention **70**

L Green Bank convention **70**

References / Index **71**

List of Tables

1 Significant milestones in the development of FITS*FITS*. . . . 2
2 Version history of the standardStandard. . . . 2
3 IAU-recommended basic units. . . . 9
4 Additional allowed units. . . . 10
5 Prefixes for multiples and submultiples. . . . 11
6 Characters and strings allowed to denote mathematical operations. . . . 11
7 Mandatory keywords for primary header. . . . 11
8 Interpretation of valid BITPIX`BITPIX` value. . . . 11
9 Example of a primary array header. . . . 13
10 Mandatory keywords in conforming extensions. . . . 13
11 Usage of BZERO`BZERO` to represent non-default integer data types. . . . 16
12 Mandatory keywords in primary header preceding random groups. . . . 18
13 Mandatory keywords in image `IMAGE` extensions. . . . 19
14 Mandatory keywords in ASCII table ASCII-table extensions. . . . 21
15 Valid TFORMn`TFORM`*n* format values in TABLE`TABLE` extensions. . . . 21
16 Valid TDISPn`TDISP`*n* format values in TABLE`TABLE` extensions. . . . 22
18 Valid TFORMn`TFORM`*n* data types in BINTABLE`BINTABLE` extensions. . . . 24
17 Mandatory keywords in binary table binary-table extensions. . . . 25
19 Usage of TZEROn`TZERO`*n* to represent non-default integer data types. . . . 26
20 Valid TDISPn`TDISP`*n* format values in BINTABLE`BINTABLE` extensions. . . . 27
21 WCS and celestial coordinates notation. . . . 33
22 Reserved WCS keywords (continues on next page) . . . 34
23 Reserved celestial coordinate algorithm celestial-coordinate-algorithm codes. . . . 36
24 Allowed values of `RADESYS`*a*. . . . 37
25 Reserved spectral coordinate spectral-coordinate type codes. . . . 38
26 Non-linear spectral algorithm codes. . . . 38
27 Spectral reference systems. . . . 39
28 Example keywords keyword records for a 100 element 100-element array of complex values. . . . 39
29 Conventional Stokes values. . . . 39
30 Recognized Time Scale Values . . . 41
31 Standard Time Reference Position Values . . . 42
32 Compatibility of Time Scales and Reference Positions . . . 42
33 Valid solar system Solar System ephemerides . . . 44
34 Recommended time units . . . 44
35 Keywords for global time values . . . 45
36 Valid mnemonic values for the `ZCMPTYPE` and `ZCTYP`*n* keywords . . . 53
37 Keyword parameters for Rice compression . . . 53
38 PLIO Instructions . . . 54
39 Keyword parameters for H-compression . . . 55
C.1 Mandatory *FITS* keywords. . . . 58
C.2 Reserved *FITS* keywords. . . . 58
C.3 General reserved *FITS* keywords. . . . 59
D.1 ASCII character set . . . 60
E.1 Summary of format parameters. . . . 61
E.2 IEEE floating-point formats. . . . 62

DRAFT

1. Introduction

> *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.

This document, hereafter referred to as the 'standardStandard', describes the Flexible Image Transport System (FITS)*FITS*), which is the standard archival data format for astronomical data sets. Although FITS*FITS* was originally designed for transporting image data on magnetic tape (which accounts for the 'I' and 'T' in the name), the capabilities of the FITS*FITS* format have expanded to accommodate more complex more-complex data structures. The role of FITS*FITS* has also grown from simply a way to transport data between different analysis software systems into the preferred format for data in astronomical archives, as well as the on-line analysis format used by many software packages.

This standard is intended as a formal codification of the FITSformat*FITS* format, which has been endorsed by the International Astronomical Union (IAU) for the interchange of astronomical data (IAU 1983). It is fully consistent with all actions and endorsements of the IAU FITS*FITS* Working Group (IAUFWG), which was appointed by Commission 5 of the IAU to oversee further development of the FITS*FITS* format. In particular, this standard defines the organization and content of the header and data units for all standard FITS*FITS* data structures: the primary array, the random groups random-groups structure, the image extension, the ASCII table ASCII-table extension, and the binary table binary-table extension. It also specifies minimum structural requirements and general principles governing the creation of new extensions. For headers, it specifies the proper syntax for keyword records and defines required and reserved keywords. For data, it specifies character and numeric value character- and numeric-value representations and the ordering of contents within the byte stream.

One important feature of the FITS*FITS* format is that its structure, down to the bit level, is completely specified in documents (such as this standard), many of which have been published in refereed scientific journals. Given these documents, which are readily available in hard copy form in libraries around the world as well as in electronic form on the Internet, future researchers should be able to decode the stream of bytes in any FITS*FITS* format data file. In contrast, many other current data formats are only implicitly defined by the software that reads and writes the files. If that software is not continually maintained so that it can be run on future computer systems, then the information encoded in those data files could be lost.

1.1. Brief history of FITSFITS

The FITS*FITS* format evolved out of the recognition that a standard format was needed for transferring astronomical images from one research institution to another. The first prototype developments of a universal interchange format that would eventually lead to the definition of the FITS*FITS* format began in 1976 between Don Wells at KPNO and Ron Harten at the Netherlands Foundation for Research in Astronomy (NFRA). This need for an image interchange format was raised at a meeting of the Astronomy section of the U.S. National Science Foundation in January 1979, which led to the formation of a task force to work on the problem. Most of the technical details of the first basic FITS*FITS* agreement (with files consisting of only a primary header followed by a data array) were subsequently developed by Don Wells and Eric Greisen (NRAO) in March 1979. After further refinements, and successful image interchange tests between observatories that used widely different types of computer systems, the first papers that defined the FITS*FITS* format were published in 1981 (Wells et al. 1981; Greisen & Harten 1981). The FITS*FITS* format quickly became the defacto de facto standard for data interchange within the astronomical community (mostly on 9-track nine-track magnetic tape at that time), and was officially endorsed by the IAU in 1982 (IAU 1983). Most national and international astronomical projects and organizations subsequently adopted the FITS*FITS* format for distribution and archiving of their scientific data products. Some of the highlights in the developmental history of FITS*FITS* are shown in Table 1.

1.2. Version history of this document

The fundamental definition of the FITS*FITS* format was originally contained in a series of published papers (Wells et al. 1981; Greisen & Harten 1981; Grosbøl et al. 1988; Harten et al. 1988). As FITS*FITS* became more widely used, the need for a single document to unambiguously define the requirements of the FITS*FITS* format became apparent. In 1990, the NASA Science Office of Standards and Technology (NOST) at the Goddard Space Flight Center provided funding for a technical panel to develop the first version of this standard Standard document. As shown in Table 2, the NOST panel produced several draft versions, culminating in the first NOST standard document, NOST 100-1.0, in 1993. Although this document was developed under a NASA accreditation process, it was subsequently formally approved by the IAUFWG, which is the international control authority for the FITS*FITS* format. The small update to the standard Standard in 1995 (NOST 100-1.1) added a recommendation on the physical units of header keyword values.

The NOST technical panel was convened a second time to make further updates and clarifications to the standardStandard, resulting in the NOST 100-2.0 versionthat , which was approved by the IAUFWG in 1999 and published in 2001 (Hanisch et al. 2001). In 2005, the IAUFWG formally approved the variable-length array convention in binary tables, and a short time later approved support for the 64-bit integers data type. New versions of the standard Standard were released to reflect both of these changes (versions IAUFWGVersions IAUFWG 2.1 and IAUFWG 2.1b).

In early 2007 the IAUFWG appointed its own technical panel to consider further modifications and updates to the standardStandard. The changes proposed by this panel, which were ultimately approved in 2008 by the IAUFWG after a formal public review process, are shown in the Version 3.0 version of the document, published in Pence et al. (2010).

DRAFT

Table 1: Significant milestones in the development of FITS*FITS*.

Date	Milestone	Section
1979	Initial FITS*FITS* Agreement and first interchange of files	
1981	Published original (single HDU) definition (Wells et al. 1981)	
1981	Published random groups random-groups definition (Greisen & Harten 1981)	Sect. 6
1982	Formally endorsed by the IAU (IAU 1983)	
1988	Defined rules for multiple extensions (Grosbøl et al. 1988)	
1988	IAU FITS*FITS* Working Group (IAUFWG) established	
1988	Extended to include ASCII table ASCII-table extensions (Harten et al. 1988)	Sect. 7.2
1988	Formal IAU approval of ASCII tables (IAU 1988)	Sect. 7.2
1990	Extended to include IEEE floating-point data (Wells & Grosbøl 1990)	Sect. 5.3
1994	Extended to multiple IMAGEarray `IMAGE`-array extensions (Ponz et al. 1994)	Sect. 7.1
1995	Extended to binary table binary-table extensions (Cotton et al. 1995)	Sect. 7.3
1997	Adopted 4-digit year four-digit-year date format (Bunclark & Rots 1997)	Sect. 4.4.2
2002	Adopted proposals for world coordinate world-coordinate systems (Greisen & Calabretta 2002)	Sect. 8
2002	Adopted proposals for celestial coordinates (Calabretta & Greisen 2002)	Sect. 8.3
2004	Adopted MIME types for FITS*FITS* data files (Allen & Wells 2005)	App. G
2005	Extended to support variable-length arrays in binary tables	Sect. 7.3.5
2005	Adopted proposals for spectral coordinate spectral-coordinate systems (Greisen et al. 2006)	Sect. 8.4
2005	Extended to include 64-bit integer data type	Sect. 5.2.4
2006	Adopted WCS HEALPix projection (Calabretta & Roukema 2007)	Sect. 8.3
2006	Established *FITS* convention registry	
2014	Adopted proposals for time coordinates (Rots et al. 2015)	Sect. 9
2016	Adopted proposals for compressed data	Sect. 10
2016	Adopted various registered conventions	App. H.3
2018	General language editing	App. H.4

Table 2: Version history of the standardStandard.

Version	Date	Status
NOST 100-0.1	1990 December	First Draft Standard
NOST 100-0.2	1991 June	Second Revised Draft Standard
NOST 100-0.3	1991 December	Third Revised Draft Standard
NOST 100-1.0	1993 June	NOST Standard
NOST 100-1.1	1995 September	NOST Standard
NOST 100-2.0	1999 March	NOST Standard
IAUFWG 2.1	2005 April	IAUFWG Standard
IAUFWG 2.1b	2005 December	IAUFWG Standard
IAUFWG 3.0	2008 July	IAUFWG Standard
IAUFWG 4.0	2016 July	IAUFWG Standard (approved)
IAUFWG 4.0	2018 August	IAUFWG Standard (language-edited)

Since 2006 a Registry for *FITS* conventions submitted by the community was established under the care of the IAUFWG at `http://fits.gsfc.nasa.gov/fits_registry.html`. The Registry was intended as a repository of documentation of usages, which, although not endorsed as part of the *FITS* Standard, are otherwise perfectly legal usages of *FITS*. In 2014 a small team was formed to evaluate the possible incorporation of some conventions within the Standard, while another small team was in charge to update the Standard document with a summary of the WCS time representation (Rots et al. 2015), which in the meanwhile had been voted natively as part of the *FITS* Standard.

Details on the conventions that have been incorporated into this latest version of the Standard (`CONTINUE` long-string keywords, blank header space, `CHECKSUM`, column limits, tiled image and table compression) or only briefly mentioned (keyword inheritance and Green Bank conventions) are described in Appendix H.3, which also lists the corresponding affected sections of the Standard.

After the approval by the IAUFWG in July 2016 the Standard was subjected to a thorough language editing (with no impact on the technical prescriptions) before the final issue in 2018. Details about the language editing changes are provided in Appendix H.4.

The latest version of the standardStandard, as well as other information about the FITS*FITS* format, can be obtained from the FITSSupport Office web site *FITS* Support Office website at `http://fits.gsfc.nasa.gov`. This web site website also contains the contact information for the Chairman of the IAUFWG, to whom any questions or comments regarding this standard Standard should be addressed.

1.3. Acknowledgments

The members of the three technical panels that produced this standard Standard are shown below.

First technical panel, 1990 – 19931990–1993
Robert J. Hanisch (Chair) Space Telescope Science Inst.
Lee E. Brotzman Hughes STX

Edward Kemper	Hughes STX
Barry M. Schlesinger	Raytheon STX
Peter J. Teuben	University of Maryland
Michael E. Van Steenberg	NASA Goddard SFC
Wayne H. Warren Jr.	Hughes STX
Richard A. White	NASA Goddard SFC

Second technical panel, 1994 – 19991994–1999

Robert J. Hanisch (Chair)	Space Telescope Science Inst.
Allen Farris	Space Telescope Science Inst.
Eric W. Greisen	National Radio Astr. Obs.
William D. Pence	NASA Goddard SFC
Barry M. Schlesinger	Raytheon STX
Peter J. Teuben	University of Maryland
Randall W. Thompson	Computer Sciences Corp.
Archibald Warnock	A/WWW Enterprises

Third technical panel, 2007

William D. Pence (Chair)	NASA Goddard SFC
Lucio Chiappetti	IASF Milano, INAF, Italy
Clive G. Page	University of Leicester, UK
Richard Shaw	National Optical Astr. Obs.
Elizabeth Stobie	University of Arizona

Dedicated task forces, 2013-2016

Lucio Chiappetti	IASF Milano, INAF, Italy
Steve Allen	UCO Lick Observatory
Adam Dobrzycki	European Southern Observatory
William D. Pence	NASA Goddard SFC
Arnold Rots	Harvard Smithsonian CfA
Richard Shaw	National Optical Astr. Obs.
William T. Thompson	NASA Goddard SFC

Language editing, 2016-2018

Malcolm J. Currie	Rutherford Appleton Lab, UK
Lucio Chiappetti	IASF Milano, INAF, Italy

2. Definitions, acronyms, and symbols

2.1. Conventions used in this document

Terms or letters set in Courier fontCourier typeface represent literal strings that appear in FITS*FITS* files. In the case of keyword names, such as 'NAXISnNAXIS*n*', the lower case lower-case letter represents a positive integer index number, generally in the range 1 to 999. The emphasized words must, shall, should, may, recommended, and optional*must*, *shall*, *should*, *may*, *recommended*, *required*, and *optional* in this document are to be interpreted as described in IETF standard, RFC 2119 (Bradner 1997).

2.2. Defined terms

␣ Used to designate an ASCII space character.

ANSI American National Standards Institute.

Array A sequence of data values. This sequence corresponds to the elements in a rectilinear, n-dimension -dimensional matrix ($1 \le n \le 999$, or $n = 0$ in the case of a null array).

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

ASCII American National Standard Code for Information Interchange.

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

ASCII digit One of the ten ASCII characters '0' through '9', which are represented by decimal character codes 48 through 57 (hexadecimal 30 through 39).

ASCII NULL The ASCII character that has all eight bits set to zero.

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

ASCII text The restricted set of ASCII characters decimal 32 through 126 (hexadecimal 20 through 7E).

Basic FITS The FITS

Basic *FITS* The *FITS* structure consisting of the primary header followed by a single primary data array. This is also known as Single Image FITS*FITS* (SIF), as opposed to Multi-Extension FITS*FITS* (MEF) files that contain one or more extensions following the primary HDU.

Big endian The numerical data format used in FITS*FITS* files in which the most significant most-significant byte of the value is stored first followed by the remaining bytes in order of significance.

Bit A single binary digit.

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

Card image An obsolete term for an 80-character keyword record derived from the 80-column punched computer cards that were prevalent in the 1960s and 1970s.

Character string A sequence of one or more of the restricted set of ASCII text ASCII-text characters, decimal 32 through 126 (hexadecimal 20 through 7E).

Conforming extension An extension whose keywords and organization adhere to the requirements for conforming extensions defined in Sect. 3.4.1 of this standardStandard.

Data block A 2880-byte FITS*FITS* block containing data described by the keywords in the associated header of that HDU.

Deprecate To express disapproval of. This term is used to refer to obsolete structures that should not*should not* be used in new FITSfilesbut which shall*FITS* files, but which *shall* remain valid indefinitely.

Entry A single value in an ASCII table or binary table ASCII-table or binary-table standard extension.

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

Extension type name The value of the XTENSIONXTENSION keyword, used to identify the type of the extension.

Field A component of a larger entity, such as a keyword record or a row of an ASCII table or binary table ASCII-table or binary-table standard extension. A field in a table extension table-extension row consists of a set of zero or more 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

FITS Flexible Image Transport System.

FITSblock

DRAFT

***FITS* block** A sequence of 2880 8-bit eight-bit bytes aligned on 2880 byte 2880-byte boundaries in the FITS*FITS* file, most commonly either a header block or a data block. Special records are another infrequently used type of FITS*FITS* block. This block length was chosen because it is evenly divisible by the byte and word lengths of all known computer systems at the time FITS*FITS* was developed in 1979.

FITSfile

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

FITSstructure

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

FITSSupport Office The FITSinformation web site

***FITS* Support Office** The *FITS* information website that is maintained by the IAUFWG and is currently hosted at `http://fits.gsfc.nasa.gov`.

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 random-groups structure, without the application of the associated linear transformation.

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

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

Header block A 2880-byte FITS*FITS* block containing a sequence of thirty-six 80-character keyword records.

Heap The supplemental data area following the main data table in a binary table binary-table standard extension.

IAU International Astronomical Union.

IAUFWG International Astronomical Union FITS*FITS* Working Group.

IEEE Institute of Electrical and Electronic Engineers.

IEEE NaN IEEE Not-a-Number value; used to represent undefined floating-point values in FITS*FITS* arrays and binary tables.

IEEE special values Floating-point number byte patterns that have a special, reserved meaning, such as −0, ±∞, ±underflow, ±overflow, ±denormalized, ±NaN. (See Appendix E).

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

Keyword name The first eight bytes of a keyword record, which contain the ASCII name of a metadata quantity (unless it is blank).

Keyword record An 80-character record in a header block consisting of a keyword name in the first eight characters followed by an optional*optional* value indicator, value, and comment string. The keyword record shall*shall* be composed only of the restricted set of ASCII text ASCII-text characters ranging from decimal 32 to 126 (hexadecimal 20 to 7E).

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

MEF Multi-Extension FITS*FITS*, i.e., a FITS*FITS* file containing a primary HDU followed by one or more extension HDUs.

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*optional*, linear transformation.

Pixel Short for 'Picture element'; a single location within an array.

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

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

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

Random Group A FITS*FITS* structure consisting of a collection of 'groups', where a group consists of a subarray of data and a set of associated parameter values. Random groups are deprecated for any use other than for radio interferometry data.

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

Repeat count The number of values represented in a field in a binary tablebinary-table standard extension.

Reserved keyword An optionalkeyword that mustoptional keyword that *must* be used only in the manner defined in this standardStandard.

SIF Single Image FITS*FITS*, i.e., a FITS*FITS* file containing only a primary HDU, without any extension HDUs. Also known as Basic FITS*FITS*.

Special records A series of one or more FITS*FITS* blocks following the last HDU whose internal structure does not otherwise conform to that for the primary HDU or to that specified for a conforming extension in this standardStandard. Any use of special records requires approval from the IAU FITS *FITS* Working Group.

Standard extension A conforming extension whose header and data content are completely specified in Sect. 7 of this standardStandard, namely, an image extension, an ASCII table ASCII-table extension, or a binary table binary-table extension.

3. FITS*FITS* file organization

3.1. Overall file structure

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

- Primary header and data unit (HDU).
- Conforming Extensions (optional*optional*).
- Other special records (optional*optional*, restricted).

A FITS*FITS* file composed of only the primary HDU is sometimes referred to as a Basic FITS*FITS* file, or a Single Image FITS*FITS* (SIF) file, and a FITS*FITS* file containing one or more extensions following the primary HDU is sometimes referred to as a Multi-Extension FITS*FITS* (MEF) file.

DRAFT

Each FITS*FITS* structure shall*shall* consist of an integral number of FITSblocks*FITS* blocks, which are each 2880 bytes (23040 bits) in length. The primary HDU shall*shall* start with the first FITS*FITS* block of the FITS*FITS* file. The first FITS*FITS* block of each subsequent FITS*FITS* structure shallbe the FITS*shall* be the *FITS* block immediately following the last FITS*FITS* block of the preceding FITS*FITS* structure.

This standard does not impose Standard neither imposes a limit on the total size of a FITS*FITS* file, nor imposes a limit on the size of an individual HDU within a FITS*FITS* file. Software packages that read or write data according to this standard Standard could be limited, however, in the size of files that are supported. In particular, some software systems have historically only supported files up to 2^{31} bytes in size (approximately 2.1×10^9 bytes).

3.2. Individual FITSFITS Structures

The primary HDU and every extension HDU shall*shall* consist of one or more 2880-byte header blocks immediately followed by an optional*optional* sequence of associated 2880-byte data blocks. The header blocks shall*shall* contain only the restricted set of ASCIIASCII-text text characters, decimal 32 through 126 (hexadecimal 20 through 7E). The ASCII control characters with decimal values less than 32 (including the null, tab, carriage return, and line feed line-feed characters), and the delete character (decimal 127 or hexadecimal 7F) must not*must not* appear anywhere within a header block.

3.3. Primary header and data unit

The first component of a FITSfile shall*FITS* file *shall* be the primary HDU, which always contains the primary header and may*may* be followed by the primary data array. If the primary data array has zero length, as determined by the values of the NAXIS`NAXIS` and NAXISn`NAXIS`*n* keywords in the primary header (Sect. 4.4.1), then the primary HDU shall*shall* contain no data blocks.

3.3.1. Primary header

The header of a primary HDU shall*shall* consist of one or more header blocks, each containing a series of 80-character keyword records containing only the restricted set of ASCIIASCII-text text characters. Each 2880-byte header block contains 36 keyword records. The last header block mustcontain the END*must* contain the `END` keyword (defined in Sect. 4.4.1) 4.4.1), which marks the logical end of the header. Keyword records without information (e.g., following the END`END` keyword) shall*shall* be filled with ASCII spaces (decimal 32 or hexadecimal 20).

3.3.2. Primary data array

The primary data array, if present, shall*shall* consist of a single data array with from 1 to 999 dimensions (as specified by the NAXIS`NAXIS` keyword defined in Sect. 4.4.1). The random groups random-groups convention in the primary data array is a more complicated more-complicated structure and is discussed separately in Sect. 6. The entire array of data values are represented by a continuous stream of bits starting with the first bit of the first data block. Each data value shall*shall* consist of a fixed number of bits that is determined by the value of the BITPIX`BITPIX` keyword (Sect. 4.4.1). Arrays of more than one dimension shall*shall* consist of a sequence such that the index along axisAxis 1 varies most rapidly, that along axisAxis 2 next most rapidly, and those along subsequent axes progressively less rapidly, with that along axis m, where mAxis `m`, where `m` is the value of NAXIS`NAXIS`, varying least rapidly. There is no space or any other special character between the last value on a row or plane and the first value on the next row or plane of a multi-dimensional array. Except for the location of the first element, the array structure is independent of the FITS*FITS* block structure. This storage order is shown schematically in Fig. 1 and is the same order as in multi-dimensional arrays in the Fortran programming language (ISO 2004). The index count along each axis shall*shall* begin with 1 and increment by 1 up to the value of the NAXISn`NAXIS`*n* keyword (Sect. 4.4.1).

A(1, 1, . . . , 1),
A(2, 1, . . . , 1),
⋮,
A(NAXIS1`NAXIS1`, 1, . . . , 1),
A(1, 2, . . . , 1),
A(2, 2, . . . , 1),
⋮,
A(NAXIS1`NAXIS1`, 2, . . . , 1),
⋮,
A(1, NAXIS2`NAXIS2`, . . . , NAXISm`NAXIS`*m*),
⋮,
A(NAXIS1, NAXIS2`NAXIS1`, `NAXIS2`, . . . , NAXISm`NAXIS`*m*)

Fig. 1: Arrays of more than one dimension shall*shall* consist of a sequence such that the index along axis Axis 1 varies most rapidly and those along subsequent axes progressively less rapidly.

If the data array does not fill the final data block, the remainder of the data block shall*shall* be filled by setting all bits to zero. The individual data values shall*shall* be stored in big-endian byte order such that the byte containing the most significant most-significant bits of the value appears first in the FITS*FITS* file, followed by the remaining bytes, if any, in decreasing order of significance.

3.4. Extensions

3.4.1. Requirements for conforming extensions

All extensions, whether or not further described in this standard, shallStandard, *shall* fulfill the following requirements to be in conformance with this FITSstandard*FITS* Standard. New extension types should*should* be created only when the organization of the information is such that it cannot be handled by one of the existing extension types. A FITS*FITS* file that contains extensions is commonly referred to as a multi-extension FITS*FITS* (MEF) file.

3.4.1.1. Identity

Each extension type shall*shall* have a unique type name,

DRAFT

specified in the header by the XTENSIONXTENSION keyword (Sect. 4.4.1). To preclude conflict, extension type names mustmust be registered with the IAUFWG. The current list of registered extensions is given in Appendix F. An up-to-date list is also maintained on the FITSSupport Office web siteFITS Support Office website.

3.4.1.2. Size specification

The total number of bits in the data of each extension shallshall be specified in the header for that extension, in the manner prescribed in Sect. 4.4.1.

3.4.2. Standard extensions

A standard extension is a conforming extension whose organization and content are completely specified in Sect. 7 of this standardStandard. Only one extension format shallshall be approved for each type of data organization.

3.4.3. Order of extensions

An extension maymay follow the primary HDU or another conforming extension. Standard extensions and other conforming extensions maymay appear in any order in a FITSFITS file.

3.5. Special records (restricted use)

Special records are 2880-byte FITSFITS blocks following the last HDU of the FITSFITS file that have an unspecified structure that does not meet the requirements of a conforming extension. The first 8 eight bytes of the special records must notmust not contain the string '`XTENSION`'. It is recommendedrecommended that they do not contain the string '`SIMPLE␣␣`'. The contents of special records are not otherwise specified by this standardStandard.

Special records were originally designed as a way for the FITSFITS format to evolve by allowing new FITSFITS structures to be implemented. Following the development of conforming extensions, which provide a general mechanism for storing different types of data structures in FITSFITS format in a well defined manner, the need for other new types of FITSFITS data structures has been greatly reduced. Consequently, further use of special records is restricted and requires the approval of the IAU FITS FITS Working Group.

3.6. Physical blocking

3.6.1. Bitstream Bit-stream devices

For bitstream bit-stream devices, including but not restricted to logical file systems, FITSfiles shallFITS files shall be interpreted as a sequence of one or more 2880-byte FITSFITS blocks, regardless of the physical blocking structure of the underlying recording media. When writing a FITSFITS file on media with a physical block size unequal to the 2880-byte FITSFITS block length, any bytes remaining in the last physical block following the end of the FITSFITS file shouldshould be set to zero. Similarly, when reading FITSFITS files on such media, any bytes remaining in the last physical block following the end of the FITSfile shallFITS file shall be disregarded.

3.6.2. Sequential media

The FITSFITS format was originally developed for writing files on sequential magnetic tapemagnetic-tape devices. The following rules on how to write to sequential media (Grosbøl & Wells 1994) are now irrelevant to most current data storage data-storage devices.

If physically possible, FITSfiles shallFITS files shall be written on sequential media in blocks that are from one to ten integer multiples of 2880-bytes 2880 bytes in length. If this is not possible, the FITSfile shallFITS file shall be written as a bitstream bit stream using the native block size of the sequential device. Any bytes remaining in the last block following the end of the FITSfile shallFITS file shall be set to zero.

When reading FITSFITS files on sequential media, any files shorter than 2880 bytes in length (e.g., ANSI tape labels) are not considered part of the FITSfiles and shouldFITS files and should be disregarded.

3.7. Restrictions on changes

Any structure that is a valid FITSFITS structure shallshall remain a valid FITSFITS structure at all future times. Use of certain valid FITSstructures mayFITS structures may be deprecated by this or future FITSstandard FITS Standard documents.

4. Headers

The first two sections of this chapter define the structure and content of header keyword records. This is followed in Sect. 4.3 with Sect. 4.3 offers recommendations on how physical units should be expressed. The final section defines the mandatory and reserved keywords for primary arrays and conforming extensions.

4.1. Keyword records

4.1.1. Syntax

Each 80-character header keyword record shallshall consist of a keyword name, a value indicator (only required required if a value is present), an optionaloptional value, and an optionaloptional comment. Keywords maymay appear in any order except where specifically stated otherwise in this standardStandard. It is recommendedrecommended that the order of the keywords in FITSFITS files be preserved during data processing operations because the designers of the FITSFITS file may have used conventions that attach particular significance to the order of certain keywords (e.g., by grouping sequences of COMMENTCOMMENT keywords at specific locations in the header, or appending HISTORYHISTORY keywords in chronological order of the data processing steps) or using CONTINUECONTINUE keywords to generate long-string keyword values).

A formal syntax, giving a complete definition of the syntax of FITSFITS keyword records, is given in Appendix A. It is intended as an aid in interpreting the text defining the standardStandard.

In earlier versions of this standard a FITS Standard a FITS keyword, assumed as an item whose value is to be looked up by name (and presumably assigned to a variable) by a FITS reading FITS-reading program, was associated one to one to a single header keyword record. With the introduction of continued

DRAFT

(long-string) keywords (see Sect. 4.2.1.2), such FITS keywords may *FITS keywords may* span more than one header keyword record, and the value shall*shall* be created by concatenation as explained in Sect. 4.2.1.2.

4.1.2. Components

4.1.2.1. Keyword name (bytesBytes 1 through 8)

The keyword name shall*shall* be a left justified, 8-charactereight-character, space-filled, ASCII string with no embedded spaces. All digits 0 through 9 (decimal ASCII codes 48 to 57, or hexadecimal 30 to 39) and upper case Latin alphabetic characters 'A'through 'Z''A' through 'Z' (decimal 65 to 90 or hexadecimal 41 to 5A) are permitted; lower case characters shall notlower-case characters *shall not* be used. The underscore ('_''_', decimal 95 or hexadecimal 5F) and hyphen ('-''-', decimal 45 or hexadecimal 2D) are also permitted. No other characters are permitted.[3] For indexed keyword names that have a single positive integer index counter appended to the root name, the counter shall nothave leading zeroes *shall not* have leading zeros (e.g., NAXIS1, not NAXIS001`NAXIS1`, not `NAXIS001`). Note that keyword names that begin with (or consist solely of) any combination of hyphens, underscores, and digits are legal.

4.1.2.2. Value indicator (bytesBytes 9 and 10)

If the two ASCII characters '=␣' (decimal 61 followed by decimal 32) are present in bytesBytes 9 and 10 of the keyword record, this indicates that the keyword has a value field associated with it, unless it is one of the commentary keywords defined in Sect. 4.4.2 (i.e., a HISTORY`HISTORY`, COMMENT`COMMENT`, or completely blank keyword name)which by definition , which, by definition, have no value.

4.1.2.3. Value/comment (bytesBytes 11 through 80)

In keyword records that contain the value indicator in bytesBytes 9 and 10, the remaining bytesBytes 11 through 80 of the record shall*shall* contain the value, if any, of the keyword, followed by optional*optional* comments. In keyword records without a value indicator, bytesBytes 9 through 80 should*should* be interpreted as commentary text, however, this does not preclude conventions that interpret the content of these bytes in other ways.

The value field, when present, shallcontain the ASCII text *shall* contain the ASCII-text representation of a literal string constant, a logical constant, or a numerical constant, in the format specified in Sect. 4.2. The value field may*may* be a null field; i.e., it may*may* consist entirely of spaces, in which case the value associated with the keyword is undefined.

The mandatory FITS*FITS* keywords defined in this standard must notStandard *must not* appear more than once within a header. All other keywords that have a value should not*should not* appear more than once. If a keyword does appear multiple times with different values, then the value is indeterminate.

[3] This requirement differs from the wording in the original FITS*FITS* papers. See Appendix HH.

If a comment follows the value field, it must*must* be preceded by a slash ('/''/', decimal 47 or hexadecimal 2F).[3] A space between the value and the slash is strongly recommended*recommended*. The comment may*may* contain any of the restricted set of ASCIIASCII-text text characters, decimal 32 through 126 (hexadecimal 20 through 7E). The ASCII control characters with decimal values less than 32 (including the null, tab, carriage return, and line feed line-feed characters), and the delete character (decimal 127 or hexadecimal 7F) must not*must not* appear anywhere within a keyword record.

4.2. Value

The structure of the value field depends on the data type of the value. The value field represents a single value and not an array of values.[3] The value field must*must* be in one of two formats: fixed or free. The fixed-format is required *required* for values of mandatory keywords and is recommended*recommended* for values of all other keywords.

4.2.1. Character string

4.2.1.1 Single record Single-record string keywords

A character string character-string value shall*shall* be composed only of the set of restricted ASCIIASCII-text text characters, decimal 32 through 126 (hexadecimal 20 through 7E) enclosed by single quote single-quote characters ("''", decimal 39, hexadecimal 27). A single quote is represented within a string as two successive single quotes, e.g., O'HARA = 'O''HARA''O''HARA'. Leading spaces are significant; trailing spaces are not. This standard Standard imposes no requirements on the case sensitivity of character string values unless explicitly stated in the definition of specific keywords.

If the value is a fixed-format character string, the starting single quote character mustbe in bytesingle-quote character *must* be in Byte 11 of the keyword record and the closing single quote must*must* occur in or before byteByte 80. Earlier versions of this standard also requiredStandard also *required* that fixed-format characters strings must*must* be padded with space characters to at least a length of eight characters so that the closing quote character does not occur before byteByte 20. This minimum character string character-string length is no longer required*required*, except for the value of the XTENSION`XTENSION` keyword (e.g., 'IMAGE␣␣␣' and 'TABLE␣␣␣'; see Sect. 7)which must 7), which *must* be padded to a length of eight characters for backward compatibility with previous usage.

Free-format character strings follow the same rules as fixed-format character strings except that the starting single quote character mayoccur after bytesingle-quote character *may* occur after Byte 11. Any bytes preceding the starting quote character and after byteByte 10 must*must* contain the space character.

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

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

DRAFT

The value of KEYWORD1KEYWORD1 is a null, or zero length zero-length string whereas the value of the KEYWORD2KEYWORD2 is an empty string (nominally a single space character because the first space in the string is significant, but trailing spaces are not). The value of KEYWORD3KEYWORD3 is undefined and has an indeterminate data type as well, except in cases where the data type of the specified keyword is explicitly defined in this standardStandard.

The maximum length of a string value that can be represented on a single keyword record is 68 characters, with the opening and closing quote characters in bytesBytes 11 and 80, respectively. In general, no length limit less fewer than 68 is implied for character-valued keywords.

Whenever a keyword value is declared 'string' or said to 'contain a character string', the length limits in this section apply. The next section 4.2.1.2 applies when the value is declared 'long-string'.

4.2.1.2 Continued string (long-string) keywords

Earlier versions of this Standard only defined single record single-record string keywords as described in the previous section. The Standard now incorporates a convention (originally developed for use in FITS*FITS* files from high-energy astrophysics missions) for continuing arbitrarily long string values over a potentially unlimited sequence of multiple consecutive keyword records using the following procedure: .

1. Divide the long string long-string value into a sequence of smaller substrings, each of which is no longer than 67 charactersin lengthcontains fewer than 68 characters. (Note that if the string contains any literal single quote single-quote characters, then these must *must* be represented as a pair of single quote single-quote characters in the FITSkeyword *FITS*-keyword value, and these two characters must *must* both be contained within a single substring).
2. Append an ampersand character ('&''&') to the end of each substring, except for the last substring. This character serves as a flag to FITSreading *FITS*-reading software that this string value may*may* be continued on the following keyword in the header.
3. Enclose each substring with single quote single-quote characters. Non-significant space characters may *may* occur between the ampersand character and the closing quote character.
4. Write the first substring as the value of the specified keyword.
5. Write each subsequent substring, in order, to a series of keywords that all have the reserved keyword name CONTINUE(see CONTINUE (see Sect. 4.4.2) in bytes Bytes 1 through 8, and have space characters in bytes Bytes 9 and 10 of the keyword record. The substring may *may* be located anywhere in bytes Bytes 11 through 80 of the keyword record and may *may* be preceded by non-significant space characters starting in byte Byte 11. A comment string may *may* follow the substring; if present, the comment string must *must* be separated from the substring by a forward slash character ('/'forward-slash character ('/'). Also, it is *strongly recommended* that the slash character be preceded by a space character.

The CONTINUE keyword must notCONTINUE keyword *must not* be used with of any of the mandatory or reserved keywords defined in this standard Standard unless explicitly declared of type long-string.

The following keyword records illustrate a string value that is continued over multiple keyword records. (Note: the length of the substrings have been reduced to fit within the page layout): .)

```
WEATHER = 'Partly cloudy during the evening f&'
CONTINUE  'ollowed by cloudy skies overnight.&'
CONTINUE  ' Low 21C. Winds NNE at 5 to 10 mph.'
```

If needed, additional space for the keyword comment field can be generated by continuing the string value with one or more null strings, as illustrated schematically below: .

```
STRKEY  = 'This keyword value is continued &'
CONTINUE  ' over multiple keyword records.&'
CONTINUE  '&' / The comment field for this
CONTINUE  '&' / keyword is also continued
CONTINUE  ''  / over multiple records.
```

FITSreading *FITS*-reading software can reconstruct the long string long-string value by following an inverse procedure of checking if the string value ends with the '&' '&' character and is immediately followed by a conforming CONTINUECONTINUE keyword record. If both conditions are true, then concatenate the substring from the CONTINUECONTINUE record onto the previous substring after first deleting the trailing '&' '&' character. Repeat these steps until all subsequent CONTINUECONTINUE records have been processed.

Note that if a string value ends with the '&' '&' character, but is not immediately followed by a CONTINUECONTINUE keyword that conforms to all the previously described requirements, then the '&' character should '&' character *should* be interpreted as the literal last character in the string. Also, any 'orphaned' CONTINUECONTINUE keyword records (formally not invalidating the FITS *FITS* file, although likely representing an error with respect to what the author of the file meant) should *should* be interpreted as containing commentary text in bytes 9 – 80 Bytes 9–80 (similar to a COMMENTCOMMENT keyword).

4.2.2. Logical

If the value is a fixed-format logical constant, it shall*shall* appear as an uppercase Tor Fupper-case T or F in byteByte 30. A logical value is represented in free-format by a single character consisting of an uppercase Tor Fupper-case T or F as the first non-space character in bytesBytes 11 through 80.

4.2.3. Integer number

If the value is a fixed-format integer, the ASCII representation shall*shall* be right-justified in bytesBytes 11 through 30. An integer consists of a '+' '+' (decimal 43 or hexadecimal 2B) or '–' '-' (decimal 45 or hexadecimal 2D) sign, followed by one or more contiguous ASCII digits (decimal 48 to 57 or hexadecimal 30 to 39), with no embedded spaces. The leading '+' '+' sign is optional*optional*. Leading zeros are permitted, but are not

DRAFT

significant. The integer representation shall*shall* always be interpreted as a signed, decimal number. This standard Standard does not limit the range of an integer keyword value, however, software packages that read or write data according to this standard Standard could be limited in the range of values that are supported (e.g., to the range that can be represented by a 32-bit or 64-bit signed binary integer).

A free-format integer value follows the same rules as fixed-format integers except that the ASCII representation may*may* occur anywhere within bytesBytes 11 through 80.

4.2.4. Real floating-point number

If the value is a fixed-format real floating-point number, the ASCII representation shall*shall* be right-justified in bytesBytes 11 through 30.

A floating-point number is represented by a decimal number followed by an optional*optional* exponent, with no embedded spaces. A decimal number shall*shall* consist of a '+' `'+'` (decimal 43 or hexadecimal 2B) or '–' `'-'` (decimal 45 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*optional*. At least one of the integer part or fractional part must*must* be present. If the fractional part is present, the decimal point must*must* also be present. If only the integer part is present, the decimal point may*may* be omitted, in which case the floating-point number is indistinguishable from an integer. The exponent, if present, consists of an exponent letter followed by an integer. Letters in the exponential form ('E' or 'D')`'E'` or `'D'`)[4] shall*shall* be upper case. The full precision of 64-bit values cannot be expressed over the whole range of values using the fixed-format. This standard does not impose Standard neither imposes an upper limit on the number of digits of precision, nor any limit on the range of floating-point keyword values. Software packages that read or write data according to this standard Standard could be limited, however, in the range of values and exponents that are supported (e.g., to the range that can be represented by a 32-bit or 64-bit floating-point number).

A free-format floating-point value follows the same rules as a fixed-format floating-point value except that the ASCII representation may*may* occur anywhere within bytesBytes 11 through 80.

4.2.5. Complex integer number

There is no fixed-format for complex integer numbers.[5]

If the value is a complex integer number, the value must*must* be represented as a real part and an imaginary part, separated by a comma and enclosed in parentheses e.g., (123, 45). Spaces may`(123, 45)`. Spaces *may* precede and follow the real and imaginary parts. The real and imaginary parts are represented in the same way as integers (Sect. 4.2.3). Such a representation is regarded as a single value for the complex integer number. This representation may*may* be located anywhere within bytesBytes 11 through 80.

Table 3: IAU-recommended basic units.

Quantity	Unit	Meaning	Notes
SI base & supplementary units			
length	`m`	meter	
mass	`kg`	kilogram	`g` gram allowed
time	`s`	second	
plane angle	`rad`	radian	
solid angle	`sr`	steradian	
temperature	`K`	kelvin	
electric current	`A`	ampere	
amount of substance	`mol`	mole	
luminous intensity	`cd`	candela	
IAU-recognized derived units			
frequency	`Hz`	hertz	s^{-1}
energy	`J`	joule	N m
power	`W`	watt	J s^{-1}
electric potential	`V`	volt	J C^{-1}
force	`N`	newton	kg m s^{-2}
pressure, stress	`Pa`	pascal	N m^{-2}
electric charge	`C`	coulomb	A s
electric resistance	`Ohm`	ohm	V A^{-1}
electric conductance	`S`	siemens	A V^{-1}
electric capacitance	`F`	farad	C V^{-1}
magnetic flux	`Wb`	weber	V s
magnetic flux density	`T`	tesla	Wb m^{-2}
inductance	`H`	henry	Wb A^{-1}
luminous flux	`lm`	lumen	cd sr
illuminance	`lx`	lux	lm m^{-2}

4.2.6. Complex floating-point number

There is no fixed-format for complex floating-point numbers.[5]

If the value is a complex floating-point number, the value must*must* be represented as a real part and an imaginary part, separated by a comma and enclosed in parentheses, e.g., (123.23, -45.7). Spaces may`(123.23, -45.7)`. Spaces *may* precede and follow the real and imaginary parts. The real and imaginary parts are represented in the same way as floating-point values (Sect. 4.2.4). Such a representation is regarded as a single value for the complex floating-point number. This representation may*may* be located anywhere within bytesBytes 11 through 80.

4.2.7. Date

There is strictly no such thing as a data type for *date valued* keywords, however a pseudo data type of *datetime* is defined in Sect. 9.1.1 and must*must* be used to write ISO-8601 *datetime* strings as character strings.

If a keyword needs to express a *time* in JD or MJD (see Sect. 9), this can be formatted as an arbitrary precision number, optionally separating the integer and fractional part as specified in Sect. 9.2.2.

4.3. Units

When a numerical keyword value represents a physical quantity, it is recommended*recommended* that units be provided. Units shall be *shall* be represented with a string of characters com-

[4] The 'D' `'D'` exponent form is traditionally used when representing values that have more decimals of precision or a larger magnitude than can be represented by a single precision single-precision 32-bit floating-point number, but otherwise there is no distinction between 'E' `'E'` or 'D'`'D'`.

[5] This requirement differs from the wording in the original FITS*FITS* papers. See Appendix HH.

DRAFT

Table 4: Additional allowed units.

Quantity		Unit	Meaning	Notes
plane angle plane angle		`deg`	degree of arc	$\pi/180$ rad
		`arcmin`	minute of arc	1/60 deg
		`arcsec`	second of arc	1/3600 deg
		`mas`	milli-second of arc	1/3 600 000 deg
time		`min`	minute	60 s
		`h`	hour	60 min = 3600 s
		`d`	day	86 400 s
	†	`a`	year (Julian)	31 557 600 s (365.25 d), peta a(`Pa`) forbidden
	†	`yr`	year (Julian)	`a` is IAU-style
energy*	†	`eV`	electron volt	$1.6021765 \times 10^{-19}$ J
	‡	`erg`	erg	10^{-7} J
		`Ry`	rydberg	$\frac{1}{2}\left(\frac{2\pi e^2}{hc}\right)^2 m_e c^2 = 13.605692$ eV
mass*		`solMass`	solar mass	1.9891×10^{30} kg
		`u`	unified atomic mass unit	$1.6605387 \times 10^{-27}$ kg
luminosity		`solLum`	Solar luminosity	3.8268×10^{26} W
length	‡	`Angstrom`	angstrom	10^{-10} m
		`solRad`	Solar radius	6.9599×10^{8} m
		`AU`	astronomical unit	1.49598×10^{11} m
		`lyr`	light year	9.460730×10^{15} m
	†	`pc`	parsec	3.0857×10^{16} m
events		`count`	count	
		`ct`	count	
		`photon`	photon	
		`ph`	photon	
flux density	†	`Jy`	jansky	10^{-26} W m^{-2} Hz^{-1}
	†	`mag`	(stellar) magnitude	
	†	`R`	rayleigh	$10^{10}/(4\pi)$ photons m^{-2} s^{-1} sr^{-1}
magnetic field	†‡	`G`	gauss	10^{-4} T
area		`pixel`	(image/detector) pixel	
		`pix`	(image/detector) pixel	
	†‡	`barn`	barn	10^{-28} m^2
Miscellaneous units				
		`D`	debye	$\frac{1}{3} \times 10^{-29}$ C.m
		`Sun`	relative to Sun	e.g., abundances
		`chan`	(detector) channel	
		`bin`	numerous applications	(including the 1-d analogue one-dimensional analog of pixel)
		`voxel`	3-d analogue three-dimensional analog of pixel	
	†	`bit`	binary information unit	
	†	`byte`	(computer) byte	8 bit eight bits
		`adu`	Analog-to-digital converter	
		`beam`	beam area of observation	as in Jy/beam

Notes. (†)Addition of prefixes for decimal multiples and submultiples are allowed. (‡)Deprecated in IAU Style Manual (McNally 1988) but still in use. (*)Conversion factors from CODATA Internationally recommended values of the fundamental physical constants 2002 (`http://physics.nist.gov/cuu/Constants/`).

posed of the restricted ASCII text ASCII-text character set. Unit strings can be used as values of keywords (e.g., for the reserved keywords BUNIT, and TUNITn`BUNIT`, and `TUNIT`*n*), as an entry in a character string character-string column of an ASCII or binary table ASCII-table or binary-table extension, or as part of a keyword comment string (see Sect. 4.3.2, below).

The units of all FITS*FITS* header keyword values, with the exception of measurements of angles, should*should* conform with the recommendations in the IAU Style Manual (McNally 1988). For angular measurements given as floating-point values and specified with reserved keywords, the units should*should* be degrees (i.e., deg`deg`). If a requirement exists within this standard Standard for the units of a keyword, then those units must*must* be used.

The units for fundamental physical quantities recommended by the IAU are given in Table 3, and . Table 4 lists additional units that are commonly used in astronomyare given in Table 4. Further specifications for time units are given in Sect. 9.3. The recommended plain text plain-text form for the IAU-recognized *base units* are given in columnColumn 2 of both tables.[6] All base units strings may*may* be preceded, with no intervening spaces, by a single character (two for deca) taken from Table 5 and representing scale factors mostly in steps of 10^3. Compound prefixes (e.g., ZYeV `ZYeV` for 10^{45} eV) must not*must not* be used.

[6] These tables are reproduced from the first in a series of papers on world coordinate world-coordinate systems (Greisen & Calabretta 2002), which provides examples and expanded discussion.

DRAFT

4.3.1. Construction of units strings

Compound units strings may*may* be formed by combining strings of base units (including prefixes, if any) with the recommended syntax described in Table 6. Two or more base units strings (called str1and str2`str1` and `str2` in Table 6) may*may* be combined using the restricted set of (explicit or implicit) operators that provide for multiplication, division, exponentiation, raising arguments to powers, or taking the logarithm or square-root of an argument. Note that functions such as log`log` actually require dimensionless arguments, so that log(Hz)`log(Hz)`, for example, actually means log(x/1 Hz)`log(x/1 Hz)`. The final units string is the compound string, or a compound of compounds, preceded by an optional*optional* numeric multiplier of the form 10**k, 10^k, or 10`10**`k, `10^`k, or `10`$\pm k$ where kk is an integer, *optionally* surrounded by parentheses with the sign character required *required* in the third form in the absence of parentheses. Creators of FITS*FITS* files are encouraged to use the numeric multiplier only when the available standard scale factors of Table 5 will not suffice. Parentheses are used for symbol grouping and are strongly recommended*recommended* whenever the order of operations might be subject to misinterpretation. A space character implies multiplication, which can also be conveyed explicitly with an asterisk or a period. Therefore, although spaces are allowed as symbol separators, their use is discouraged. Note that, per IAU convention, case is significant throughout. The IAU style manual forbids the use of more than one slash (‘/’`'/'`) character in a units string. However, since normal mathematical precedence rules apply in this context, more than one slash may*may* be used but is discouraged.

A unit raised to a power is indicated by the unit string followed, with no intervening spaces, by the optional*optional* symbols ** or ^ followed by the power given as a numeric expression, called exprexpr in Table 6. The power may*may* be a simple integer, with or without sign, *optionally* surrounded by parentheses. It may*may* also be a decimal number (e.g., 1.5, 0.5`1.5`, `0.5`) or a ratio of two integers (e.g., 7/9`7/9`), with or without sign, which must*must* be surrounded by parentheses. Thus meters squared may*may* be indicated by m**(2), m**+2, m+2, m2, m^2, m^(+2)`m**(2)`, `m**+2`, `m+2`, `m2`, `m^2`, `m^(+2)`, etc. and per meter cubed may*may* be indicated by m**-3, m-3, m^(-3), /m3`m**-3`, `m-3`, `m^(-3)`, `/m3`, and so forth. Meters to the three-halves power may*may* be indicated by m(1.5), m^(1.5), m**(1.5), m(3/2), m**(3/2), and m^(3/2), but notby m^3/2 or m1.5`m(1.5)`, `m^(1.5)`, `m**(1.5)`, `m(3/2)`, `m**(3/2)`, and `m^(3/2)`, but *not* by ms/2 or m1.5.

4.3.2. Units in comment fields

If the units of the keyword value are specified in the comment of the header keyword, it is recommended*recommended* that the units string be enclosed in square brackets (i.e., enclosed by ‘`[`’ and ‘`]`’) at the beginning of the comment field, separated from the slash (‘/’`'/'`) comment field delimiter by a single space character. An example, using a non-standard keyword, is
EXPTIME = 1200. / sexposure time in secondsEXPTIME = 1200. / [s] exposure time in seconds
This widespread, but optional*optional*, practice suggests that square brackets should*should* be used in comment fields only for this purpose. Nonetheless, software should not*should not* depend on units being expressed in this fashion within a keyword comment, and software should not*should not* depend on any string within square brackets in a comment field containing a proper units string.

Table 5: Prefixes for multiples and submultiples.

Submult	Prefix	Char	Mult	Prefix	Char
10^{-1}	deci	d	10	deca	da
10^{-2}	centi	c	10^{2}	hecto	h
10^{-3}	milli	m	10^{3}	kilo	k
10^{-6}	micro	u	10^{6}	mega	M
10^{-9}	nano	n	10^{9}	giga	G
10^{-12}	pico	p	10^{12}	tera	T
10^{-15}	femto	f	10^{15}	peta	P
10^{-18}	atto	a	10^{18}	exa	E
10^{-21}	zepto	z	10^{21}	zetta	Z
10^{-24}	yocto	y	10^{24}	yotta	Y

Table 6: Characters and strings allowed to denote mathematical operations.

String	Meaning
`str1 str2`	Multiplication
`str1*str2`	Multiplication
`str1.str2`	Multiplication
`str1/str2`	Division
`str1**expr`	Raised to the power `expr`
`str1^expr`	Raised to the power `expr`
`str1expr`	Raised to the power `expr`
`log(str1)`	Common Logarithm (to base 10)
`ln(str1)`	Natural Logarithm
`exp(str1)`	Exponential ($e^{\texttt{str1}}$)
`sqrt(str1)`	Square root

Table 7: Mandatory keywords for primary header.

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

Table 8: Interpretation of valid BITPIX`BITPIX` value.

Value	Data represented
8 `8`	Character or unsigned binary integer
16 `16`	16-bit two’s complement binary integer
32 `32`	32-bit two’s complement binary integer
64 `64`	64-bit two’s complement binary integer
-32 `-32`	IEEE single precision single-precision floating point
-64 `-64`	IEEE double precision double-precision floating point

DRAFT

4.4. Keywords

4.4.1. Mandatory keywords

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

4.4.1.1. Primary header

The SIMPLE`SIMPLE` keyword is required *required* to be the first keyword in the primary header of all FITS*FITS* files. The primary header must*must* contain the other mandatory keywords shown in Table 7 in the order given. Other keywords must notmust not intervene between the SIMPLE`SIMPLE` keyword and the last NAXISn`NAXIS`*n* keyword.

SIMPLE`SIMPLE` keyword. The value field shall*shall* contain a logical constant with the value T`T` if the file conforms to this standardStandard. This keyword is mandatory for the primary header and must notmust not appear in extension headers.[7] A value of F`F` signifies that the file does not conform to this standardStandard.

BITPIX`BITPIX` keyword. The value field shall*shall* contain an integer. The absolute value is used in computing the sizes of data structures. It shall*shall* specify the number of bits that represent a data value in the associated data array. The only valid values of BITPIX`BITPIX` are given in Table 8. Writers of FITSarrays shouldselect a BITPIX*FITS* arrays *should* select a `BITPIX` data type appropriate to the form, range of values, and accuracy of the data in the array.

NAXIS`NAXIS` keyword. The value field shall*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`NAXIS`*n* keywords. The NAXISnkeywords must`NAXIS`*n* keywords *must* be present for all values n = 1, ..., NAXIS*n* = 1, ..., `NAXIS`, in increasing order of n*n*, and for no other values of n*n*. The value field of this indexed keyword shall*shall* contain a non-negative integer representing the number of elements along axis nAxis *n* of a data array. A value of zero for any of the NAXISn`NAXIS`*n* signifies that no data follow the header in the HDU (however, the random groups random-groups structure described in Sect. 6 has NAXIS1 6 has `NAXIS1` =0 `0`, but will have data following the header if the other NAXISn`NAXIS`*n* keywords are non-zero). If NAXIS`NAXIS` is equal to 0, there shall not`0`, there *shall not* be any NAXISn`NAXIS`*n* keywords.

END`END` keyword. This keyword has no associated value. Bytes 9 through 80 shall*shall* be filled with ASCII spaces (decimal 32 or hexadecimal 20). The END`END` keyword marks the logical end of the header and must *must* occur in the last 2880-byte FITS*FITS* block of the header.

The total number of bits in the primary data array, exclusive of fill that is needed after the data to complete the last 2880-byte data block (Sect. 3.3.2), is given by the following expression:

$N_{\rm bits}N_{\rm bits}$ = |BITPIXBITPIX| × (NAXIS1NAXIS1 × NAXIS2NAXIS2 × ·

where $N_{\rm bits}$ mustbe$N_{\rm bits}$ *must* be non-negative and is the number of bits excluding fill, m`m` is the value of NAXIS`NAXIS`, and BITPIX`BITPIX` and the NAXISn`NAXIS`*n* represent the values associated with those keywords. Note that the random groups random-groups convention in the primary array has a more complicated more-complicated structure whose size is given by Eq. 4. The header of the first FITS*FITS* extension in the file, if present, shall*shall* start with the first FITS*FITS* block following the data block that contains the last bit of the primary data array.

An example of a primary array header is shown in Table 9. In addition to the required keywords, it includes a few of the reserved keywords that are discussed in Sect. 4.4.2.

4.4.1.2. Conforming extensions

All conforming extensions, whether or not further specified in this standard, mustStandard, *must* use the keywords defined in Table 10 in the order specified. Other keywords must notmust not intervene between the XTENSION`XTENSION` keyword and the GCOUNT`GCOUNT` keyword. The BITPIX, NAXIS, NAXISn, and END`BITPIX`, `NAXIS`, `NAXIS`*n*, and `END` keywords are defined in Sect. 4.4.1.

XTENSION`XTENSION` keyword. The value field shall*shall* contain a character string giving the name of the extension type. This keyword is mandatory for an extension header and must notmust not appear in the primary header.[8] [7] To preclude conflict, extension type names must*must* be registered with the IAUFWG. The current list of registered extensions is given in Appendix F. An up-to-date list is also maintained on the FITSSupport Office web site*FITS* Support Office website.

PCOUNT`PCOUNT` keyword. The value field shall*shall* contain an integer that shall*shall* be used in any way appropriate to define the data structure, consistent with Eq. 2. In IMAGE`IMAGE` (Sect. 7.1) and TABLE`TABLE` (Sect. 7.2) extensions this keyword must*must* have the value 0; in BINTABLE`0`; in `BINTABLE` extensions (Sect. 7.3) it is used to specify the number of bytes that follow the main data table in the supplemental data area called the heap. This keyword is also used in the random groups random-groups structure (Sect. 6) to specify the number of parameters preceding each array in a group.

GCOUNT`GCOUNT` keyword. The value field shall*shall* contain an integer that shall*shall* be used in any way appropriate to define the data structure, consistent with Eq. 2. This keyword must*must* have the value 1 in the IMAGE, TABLEand BINTABLE`1` in the `IMAGE`, `TABLE`, and `BINTABLE` standard extensions defined in Sect. 7. This keyword is also used in the random groups

[7] This requirement differs from the wording in the original FITS*FITS* papers. See Appendix HH.

[8] This requirement differs from the wording in the original FITSpapers. See Appendix H.

DRAFT

Table 9: Example of a primary array header.

Keyword records
SIMPLE = T / file does conform to FITS standardSIMPLE = T / file does confor
BITPIX = 16 / number of bits per data pixel
NAXIS = 2 / number of data axes
NAXIS1 = 250 / length of data axis 1
NAXIS2 = 300 / length of data axis 2
OBJECT = 'Cygnus X-1'
DATE = '2006-10-22'
END

random-groups structure (Sect. 6) to specify the number of random groups present.

The total number of bits in the extension data array (exclusive of fill that is needed after the data to complete the last 2880-byte data block) is given by the following expression:

$$N_{\rm bits}N_{\rm bits} = |\texttt{BITPIX}\texttt{BITPIX}| \times \texttt{GCOUNT}\texttt{GCOUNT} \times (\texttt{PCOUNT}\texttt{PCOUNT} + \texttt{NAXIS1}\texttt{NAXIS1} \times \texttt{NAXIS2}\texttt{NAXIS2} \times \cdots \times \texttt{NAXIS}m\texttt{NAXIS}m)$$

where $N_{\rm bits}$ must be $N_{\rm bits}$ *must* be non-negative and is the number of bits excluding fill; mm is the value of NAXISNAXIS; and BITPIX, GCOUNT, PCOUNTBITPIX, GCOUNT, PCOUNT, and the NAXISnNAXIS*n* represent the values associated with those keywords. If $N_{\rm bits} > 0 N_{\rm bits} > 0$, then the data array shall*shall* be contained in an integral number of 2880-byte FITS*FITS* data blocks. The header of the next FITS*FITS* extension in the file, if any, shall*shall* start with the first FITS*FITS* block following the data block that contains the last bit of the current extension data array.

4.4.2. Other reserved keywords

The reserved keywords described below are optional*optional*, but if present in the header they must*must* be used only as defined in this standardStandard. They apply to any FITS*FITS* structure with the meanings and restrictions defined below. Any FITSstructure may*FITS* structure *may* further restrict the use of these keywords.

4.4.2.1. General descriptive keywords

DATEDATE keyword. The value field shall*shall* contain a character string giving the date on which the HDU was created, in the form YYYY-MM-DDYYYY-MM-DD, or the date and time when the HDU was created, in the form YYYY-MM-DDThh:mm:ss.sss..., where YYYYshallYYYY-MM-DDThh:mm:ss[.sss...], where YYYY *shall* be the four-digit calendar year number, MMMM the two-digit month number with January given by 01 and December by 12, and DDDD the two-digit day of the month. When both date and time are given, the literal TshallT *shall* separate the date and time, hhshallhh *shall* be the two-digit hour in the day, mmmm the two-digit number of minutes after the hour, and ss.sss...ss[.sss...] the number of seconds (two digits followed by an optional*optional* fraction) after the minute. Default values must not*must not* be given to any portion of the date/time string, and leading zeros must not*must not* be omitted. The decimal part of the seconds field is optionaland may*optional* and *may* be arbitrarily long, so long as it is consistent with the rules for value formats of Sect. 4.2. Otherwise said, the format for DATEDATE keywords written after January 1, 2000 shall*shall* be the ISO-8601 *datetime* form described in Sect. 9.1.1. See also Sect. 9.5.

Table 10: Mandatory keywords in conforming extensions.

# Position	Keyword
1	XTENSIONXTENSION
2	BITPIXBITPIX
3	NAXISNAXIS
4	NAXISn, nNAXIS*n*, $n = 1, \ldots,$ NAXISNAXIS
5	PCOUNTPCOUNT
6	GCOUNTGCOUNT
	⋮
	(other keywords)
	⋮
last	ENDEND

The value of the DATEkeyword shallDATE keyword *shall* always be expressed in UTC when in this format, for all data sets created on Earth.

The following format may*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 DDDD/MM/YY, where DD is the day of the month, MMMM the month number with January given by 01 and December by 12, and YYYY 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*recommended* but not assumed.

When a newly created HDU is substantially a verbatim copy of another HDU, the value of the DATEDATE keyword in the original HDU may*may* be retained in the new HDU instead of updating the value to the current date and time.

ORIGINORIGIN keyword. The value field shall*shall* contain a character string identifying the organization or institution responsible for creating the FITS*FITS* file.

EXTENDEXTEND keyword. The value field shall*shall* contain a logical value indicating whether the FITS*FITS* file is allowed to contain conforming extensions following the primary HDU. This keyword may*may* only appear in the primary header and must not*must not* appear in an extension header. If the value field is Tthen there mayT then there *may* be conforming extensions in the FITS*FITS* file following the primary HDU. This keyword is only advisory, so its presence with a value TT does not require

DRAFT

that the FITS*FITS* file contains extensions, nor does the absence of this keyword necessarily imply that the file does not contain extensions. Earlier versions of this standard Standard stated that the EXTENDkeyword mustEXTEND keyword *must* be present in the primary header if the file contained extensions, but this is no longer required*required*.

BLOCKEDBLOCKED keyword. This keyword is deprecated and should not*should not* be used in new FITS*FITS* files. It is reserved primarily to prevent its use with other meanings. As previously defined, this keyword, if used, was required*required* to appear only within the first 36 keywords in the primary header. Its presence with the required *required* logical value of T advised that the physical block size of the FITS*FITS* file on which it appears may*may* be an integral multiple of the FITS*FITS* block length and not necessarily equal to it.

4.4.2.2. Keywords describing observations

DATE-OBSDATE-OBS keyword. The format of the value field for DATE-OBSkeywords shallDATE-OBS keywords *shall* follow the prescriptions for the DATEDATE keyword (Sect. 4.4.2 and Sect. 9.1.1 Either the four-digit year format or the two-digit year format may*may* be used for observation dates from 1900 through 1999, although the four-digit format is recommended*recommended*.

When the format with a four-digit year is used, the default interpretations for time should*should* be UTC for dates beginning 1972-01-01 and UT before. Other date and time scales are permissible. The value of the DATE-OBSkeyword shallDATE-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 should*should* be clarified in comments. The value of DATE-OBSshallDATE-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*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, the default shall*shall* be to include light-time corrections to the associated spatial origin, namely the geocenter for TCG and the solar-system Solar System barycenter for the other two. Conventions may*may* be developed that use other time systems. Time scales are now discussed in detail in Sect. 9.2.1 and Table 30.

When the value of DATE-OBSDATE-OBS is expressed in the two-digit year form, allowed for files written before January 1, 2000 with a year in the range 1900-19991900–1999, there is no default assumption as to whether it refers to the start, middle or end of an observation.

DATExxxxDATExxxx keywords. The value fields for all keywords beginning with the string DATEDATE whose value contains date, and *optionally* time, information shall*shall* follow the prescriptions for the DATE-OBSDATE-OBS keyword. See also Sect. 9.1.1 for the *datetime* format, and Sect. 9.5 for further global time keywords specified by the Standard.

TELESCOPTELESCOP keyword. The value field shall*shall* contain a character string identifying the telescope used to acquire the data associated with the header.

INSTRUMEINSTRUME keyword. The value field shall*shall* contain a character string identifying the instrument used to acquire the data associated with the header.

OBSERVEROBSERVER keyword. The value field shall*shall* contain a character string identifying who acquired the data associated with the header.

OBJECTOBJECT keyword. The value field shall*shall* contain a character string giving a name for the object observed.

4.4.2.3. Bibliographic keywords

AUTHORAUTHOR keyword. The value field shall*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.

REFERENCREFERENC keyword. The value field shall*shall* contain a character string citing a reference where the data associated with the header are published. It is recommended*recommended* that either the 19-digit bibliographic identifier[8] used in the Astrophysics Data System bibliographic databases (`http://adswww.harvard.edu/`) or the Digital Object Identifier (`http://doi.org`) be included in the value string, when available (e.g., '1994A&AS..103..135A'or 'doi:10.1006/jmbi.1998.2354'`'1994A&AS..103..135A'` or `'doi:10.1006/jmbi.1998.2354'`).

4.4.2.4. Commentary keywords

These keywords provide commentary information about the contents or history of the FITSfile and may*FITS* file and *may* occur any number of times in a header. These keywords shall*shall* have no associated value even if the value indicator characters `'=␣'` appear in bytesBytes 9 and 10 (hence it is recommended*recommended* that these keywords not contain the value indicator). Bytes 9 through 80 may*may* contain any of the restricted set of ASCIIASCII-text text characters, decimal 32 through 126 (hexadecimal 20 through 7E).

In earlier versions of this standard Standard continued string keywords (see Sect. 4.2.1.2) could be handled as commentary keywords if the relevant convention was not supported. Now CONTINUEkeywords shallCONTINUE keywords *shall* be honoured as specified in Section Sect. 4.2.1.2.

COMMENTCOMMENT keyword. This keyword may*may* be used to supply any comments regarding the FITS*FITS* file.

[8] This bibliographic convention (Schmitz et al. 1995) was initially developed for use within NED (NASA/IPAC Extragalactic Database) and SIMBAD (operated at CDS, Strasbourg, France).

DRAFT

HISTORYHISTORY keyword. This keyword should*should* be used to describe the history of steps and procedures associated with the processing of the associated data.

Keyword field is blank. This keyword may*may* be used to supply any comments regarding the FITS*FITS* file. It is frequently used for aesthetic purposes to provide a break between groups of related keywords in the header.

A sequence of one or more entirely blank keyword records (consisting of 80 ASCII space characters) that immediately precede the ENDkeyword may END keyword *may* be interpreted as non-significant fill space that may *may* be overwritten when new keywords are appended to the header. This usage convention enables an arbitrarily large amount of header space to be preallocated when the FITS*FITS* HDU is first created, which can help mitigate the potentially time-consuming alternative of having to shift all the following data in the file by 2880 bytes to make room for a new FITS *FITS* header block each time space is needed for a new keyword.

4.4.2.5. Keywords that describe arrays

These keywords are used to describe the contents of an array, either in the primary array, in an image IMAGE extension (Sect. 7.1), or in a series of random groups (Sect. 6). They are optional*optional*, but if they appear in the header describing an array or groups, they must*must* be used as defined in this section of this standard. They shall notStandard. They *shall not* be used in headers describing other structures unless the meaning is the same as defined here.

BSCALEBSCALE keyword. This keyword shall*shall* be used, along with the BZEROBZERO keyword, to linearly scale the array pixel values (i.e., the actual values stored in the FITS*FITS* file) to transform them into the physical values that they represent using Eq. 3.

$$\text{physical\_value} = \text{BZERO}_{\text{BZERO}} + \text{BSCALE}_{\text{BSCALE}} \times \text{array\_value} \quad (3)$$

The value field shall*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 1.01.0. Before support for IEEE floating-point data types was added to FITS*FITS* (Wells & Grosbøl 1990), this technique of linearly scaling integer values was the only way to represent the full range of floating-point values in a FITS*FITS* array. This linear scaling technique is still commonly used to reduce the size of the data array by a factor of two by representing 32-bit floating-point physical values as 16-bit scaled integers.

BZEROBZERO keyword. This keyword shall*shall* be used, along with the BSCALEBSCALE keyword, to linearly scale the array pixel values (i.e., the actual values stored in the FITS*FITS* file) to transform them into the physical values that they represent using Eq. 3. The value field shall*shall* contain a floating-point number representing the physical value corresponding to an array value of zero. The default value for this keyword is 0.00.0.

Besides its use in representing floating-point values as scaled integers (see the description of the BSCALEBSCALE keyword), the BZEROBZERO keyword is also used when storing unsigned integer unsigned-integer values in the FITS*FITS* array. In this special case the BSCALEkeyword shallBSCALE keyword *shall* have the default value of 1.01.0, and the BZEROkeyword shallBZERO keyword *shall* have one of the integer values shown in Table 11.

Since the FITS*FITS* format does not support a native unsigned integer data type (except for the unsigned 8-bit eight-bit byte data type), the unsigned values are stored in the FITS*FITS* array as native signed integers with the appropriate integer offset specified by the BZEROBZERO keyword value shown in the table. For the byte data type, the converse technique can be used to store signed byte values as native unsigned values with the negative BZEROBZERO offset. In each case, the physical value is computed by adding the offset specified by the BZEROBZERO keyword to the native data type value that is stored in the FITSfile.*FITS* file.[9]

BUNITBUNIT keyword. The value field shall*shall* contain a character string describing the physical units in which the quantities in the array, after application of BSCALEand BZEROBSCALE and BZERO, are expressed. These units must*must* follow the prescriptions of Sect. 4.3.

BLANKBLANK keyword. This keyword shall*shall* be used only in headers with positive values of BITPIXBITPIX (i.e., in arrays with integer data). Bytes 1 through 8 contain the string `BLANK␣␣␣' (ASCII spaces in bytesBytes 6 through 8). The value field shall*shall* contain an integer that specifies the value that is used within the integer array to represent pixels that have an undefined physical value.

If the BSCALEand BZEROBSCALE and BZERO keywords do not have the default values of 1.0 and 0.01.0 and 0.0, respectively, then the value of the BLANKkeyword mustBLANK keyword *must* equal the actual value in the FITS*FITS* data array that is used to represent an undefined pixel and not the corresponding physical value (computed from Eq. 3). To cite a specific, common example, *unsigned* 16-bit integers are represented in a *signed* integer FITS*FITS* array (with BITPIXBITPIX = 1616) by setting BZERO BZERO =32768and BSCALE 32768 and BSCALE =1 1. If it is desired to use pixels that have an *unsigned* value (i.e., the physical value) equal to 0 to represent undefined pixels in the array, then the BLANKkeyword mustBLANK keyword *must* be set to the value −32768 -32768 because that is the actual value of the undefined pixels in the FITS*FITS* array.

DATAMAXDATAMAX keyword. The value field shall*shall* always contain a floating-point number, regardless of the value of BITPIXBITPIX. This number shall*shall* give the maximum valid

[9] A more computationally efficient method of adding or subtracting the BZERO BZERO values is to simply flip the most significant most-significant bit of the binary value. For example, using 8-bit eight-bit integers, the decimal value 248 minus the BZERO BZERO value of 128 128 equals 120. The binary representation of 248 is 11111000. Flipping the most significant most-significant bit gives the binary value 01111000, which is equal to decimal 120.

Table 11: Usage of BZEROBZERO to represent non-default integer data types.

BITPIX BITPIX	Native data type	Physical data type	BZERO BZERO	
8 8	unsigned	signed byte	-128-128	(-2^7)
16 16	signed	unsigned 16-bit	3276832768	(2^{15})
32 32	signed	unsigned 32-bit	21474836482147483648	(2^{31})
64 64	signed	unsigned 64-bit	92233720368547758089223372036854775808	(2^{63})

physical value represented by the array (from Eq. 3), exclusive of any IEEE special values.

DATAMINDATAMIN keyword. The value field shall*shall* always contain a floating-point number, regardless of the value of BITPIXBITPIX. This number shall*shall* give the minimum valid physical value represented by the array (from Eq. 3), exclusive of any IEEE special values.

WCS keywords. An extensive set of keywords have been defined to describe the world coordinates associated with an array. These keywords are discussed separately in Sect. 8.

4.4.2.6. Extension keywords

The next three keywords were originally defined for use within the header of a conforming extension, however they also may*may* appear in the primary header with an analogous meaning. If these keywords are present, it is recommended*recommended* that they have a unique combination of values in each HDU of the FITS*FITS* file.

EXTNAMEEXTNAME keyword. The value field shall*shall* contain a character string to be used to distinguish among different extensions of the same type, i.e., with the same value of XTENSIONXTENSION, in a FITS*FITS* file. Within this context, the primary array should*should* be considered as equivalent to an IMAGEIMAGE extension.

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

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

The following keyword is optional*optional*, but is *reserved* for use by the convention described in Appendix K. If present it shall*shall* appear in the extension header immediately after the mandatory keywords, and be used as described in the Appendixappendix.

INHERITINHERIT keyword. The value field shall*shall* contain a logical value of Tor FT or F to indicate whether or not the current extension should inherit the keywords in the primary header of the FITS *FITS* file.

4.4.2.7 Data Integrity KeywordsData-integrity keywords

The two keywords described here provide an integrity check on the information contained in FITS*FITS* HDUs.

DATASUMKeywordDATASUM keyword. The value field of the DATASUMkeyword shallDATASUM keyword *shall* consist of a character string that shouldcontain the unsigned integer *should* contain the unsigned-integer value of the 32-bit 1' s ones' complement checksum of the data records in the HDU (i.e., excluding the header records). For this purpose, each 2880-byte FITSlogical record should *FITS* logical record *should* be interpreted as consisting of 720 32-bit unsigned integers. The 4 four bytes in each integer must*must* be interpreted in order of decreasing significance where the most significant most-significant byte is first, and the least significant least-significant byte is last. Accumulate the sum of these integers using 1' s ones' complement arithmetic in which any overflow of the most significant most-significant bit is propagated back into the least significant least-significant bit of the sum.

The DATASUMDATASUM value is expressed as a character string (i.e., enclosed in single quote single-quote characters) because support for the full range of 32-bit unsigned integer unsigned-integer keyword values is problematic in some software systems. This string may *may* be padded with non-significant leading or trailing blank characters or leading zeros. A string containing only one or more consecutive ASCII blanks may *may* be used to represent an undefined or unknown value for the DATASUMDATASUM keyword. The DATASUMkeyword may DATASUM keyword *may* be omitted in HDUs that have no data records, but it is preferable to include the keyword with a value of 0. 0. Otherwise, a missing DATASUMDATASUM keyword asserts no knowledge of the checksum of the data records. Recording in the comment field the ISO-8601-formatted Datetime (ISO 2004b) when the value of this keyword record is created or updated is recommended*recommended*.

CHECKSUMKeywordCHECKSUM keyword. The value field of the CHECKSUMkeyword shallCHECKSUM keyword *shall* consist

DRAFT

of an ASCII character string whose value forces the 32-bit 1' s ones' complement checksum accumulated over the entire FITS*FITS* HDU to equal negative 0. (Note that 1ones's complement arithmetic has both positive and negative zero elements). It is recommended*recommended* that the particular 16-character string generated by the algorithm described in Appendix J be used. A string containing only one or more consecutive ASCII blanks may *may* be used to represent an undefined or unknown value for the CHECKSUM`CHECKSUM` keyword.

The CHECKSUMkeyword value must`CHECKSUM` keyword value *must* be expressed in fixed format, when the algorithm in Appendix J is used, otherwise the usage of fixed format is recommended*recommended*. Recording in the comment field the ISO-8601-formatted Datetime when the value of this keyword record is created or updated is recommended*recommended*.

If the CHECKSUM`CHECKSUM` keyword exists in the header of the HDU and the accumulated checksum is not equal to -0−`0`, or if the DATASUM`DATASUM` keyword exists in the header of the HDU and its value does not match the data checksum, then this provides a strong indication that the content of the HDU has changed subsequent to the time that the respective keyword value was computed. Such an invalid checksum may indicate corruption during a prior file copy or transfer operation, or a corruption of the physical media on which the file was stored. It may alternatively reflect an intentional change to the data file by subsequent data processing if the CHECKSUM`CHECKSUM` value was not also updated.

Normally both keywords will be present in the header if either is present, but this is not required*required*. These keywords apply *only* to the HDU in which they are contained. If these keywords are written in one HDU of a multi-HDU FITS*FITS* file then it is *strongly recommended* that they also be written to every other HDU in the file with values appropriate to each HDU in turn; in that case the checksum accumulated over the entire file will equal -0 −0 as well. The DATASUMkeyword must`DATASUM` keyword *must* be updated before the CHECKSUM`CHECKSUM` keyword. In general updating the two checksum keywords should *should* be the final step of any update to either data or header records in a FITS *FITS* HDU. It is highly recommended*recommended* that if a FITS*FITS* file is intended for public distribution, then the checksum keywords, if present, should *should* contain valid values.

4.4.3. Additional keywords

New keywords may*may* be devised in addition to those described in this standardStandard, so long as they are consistent with the generalized rules for keywords and do not conflict with mandatory or reserved keywords. Any keyword that refers to or depends upon the existence of other specific HDUs in the same or other files should *should* be used with caution because the persistence of those HDUs cannot always be guaranteed.

5. Data Representationrepresentation

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

5.1. Characters

Each character shall*shall* be represented by one byte. A character shall*shall* be represented by its 7-bit seven-bit ASCII (ANSI 1977) code in the low order low-order seven bits in the byte. The high-order bit shall*shall* be zero.

5.2. Integers

5.2.1. Eight-bit

Eight-bit integers shall*shall* be unsigned binary integers, contained in one byte with decimal values ranging from 0 to 255.

5.2.2. Sixteen-bit

Sixteen-bit integers shall*shall* be two's complement signed binary integers, contained in two bytes with decimal values ranging from -32768 to +−32768 to +32767.

5.2.3. Thirty-two-bit

Thirty-two-bit integers shall*shall* be two's complement signed binary integers, contained in four bytes with decimal values ranging from -2147483648 to +−2147483648 to +2147483647.

5.2.4. Sixty-four-bit

Sixty-four-bit integers shall*shall* be two's complement signed binary integers, contained in eight bytes with decimal values ranging from -9223372036854775808 to +−9223372036854775808 to +9223372036854775807.

5.2.5. Unsigned integers

The FITS*FITS* format does not support a native unsigned integer data type (except for the unsigned 8-bit byte data type) therefore unsigned 16-bit, 32-bit, or 64-bit binary integers cannot be stored directly in a FITS*FITS* data array. Instead, the appropriate offset must*must* be applied to the unsigned integer to shift the value into the range of the corresponding signed integer, which is then stored in the FITS*FITS* file. The BZERO keyword shall`BZERO` keyword *shall* record the amount of the offset needed to restore the original unsigned value. The BSCALE`BSCALE` keyword shall*shall* have the default value of 1.0 `1.0` in this case, and the appropriate BZERO`BZERO` value, as a function of BITPIX`BITPIX`, is specified in Table 11.

This same technique must*must* be used when storing unsigned integers in a binary table binary-table column of signed integers (Sect. 7.3.2). In this case the TSCALn`TSCAL`*n* keyword (analogous to BSCALE) shall`BSCALE`) *shall* have the default value of 1.0`1.0`, and the appropriate TZEROn`TZERO`*n* value (analogous to BZERO`BZERO`) is specified in Table 19.

5.3. IEEE-754 floating point

Transmission of 32- and 64-bit floating-point data within the FITSformat shall*FITS* format *shall* use the ANSI/IEEE-754

DRAFT

standard (IEEE 1985). BITPIXBITPIX = -32and BITPIX-32 and BITPIX = -64-64 signify 32- and 64-bit IEEE floating-point numbers, respectively; the absolute value of BITPIXBITPIX is used for computing the sizes of data structures. The full IEEE set of number forms is allowed for FITS*FITS* interchange, including all special values.

The BLANKBLANK keyword should not*should not* be used when BITPIXBITPIX = -32-32 or -64-64; rather, the IEEE NaN should*should* be used to represent an undefined value. Use of the BSCALEand BZEROBSCALE and BZERO keywords is not recommended*not recommended*.

Appendix E has additional details on the IEEE format.

5.4. Time

There is strictly no such thing as a data type for *time valued* data, but rules to encode time values are given in Sect. 9 and in more detail in Rots et al. (2015).

6. Random groups Random-groups structure

The random groups random-groups structure allows a collection of 'groups', where a group consists of a subarray of data and a set of associated parameter values, to be stored within the FITS*FITS* primary data array. Random groups have been used almost exclusively for applications in radio interferometry; outside this field, there is little support for reading or writing data in this format. Other than the existing use for radio interferometry data, the random groups random-groups structure is deprecated and should not*should not* be further used. For other applications, the binary table binary-table extension (Sect. 7.3) provides a more extensible and better documented way of associating groups of data within a single data structure.

6.1. Keywords

6.1.1. Mandatory keywords

The SIMPLESIMPLE keyword is required*required* to be the first keyword in the primary header of all FITS*FITS* files, including those with random groupsrandom-groups records. If the random groups random-groups format records follow the primary header, the keyword records of the primary header must*must* use the keywords defined in Table 12 in the order specified. No other keywords may*may* intervene between the SIMPLESIMPLE keyword and the last NAXISnNAXIS*n* keyword.

SIMPLESIMPLE keyword. The keyword record containing this keyword is structured in the same way as if a primary data array were present (Sect. 4.4.1).

BITPIXBITPIX keyword. The keyword record containing this keyword is structured as prescribed in Sect. 4.4.1.

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

Table 12: Mandatory keywords in primary header preceding random groups.

# Position	Keyword
1	SIMPLE SIMPLE = TT
2	BITPIXBITPIX
3	NAXISNAXIS
4	NAXIS1 NAXIS1 = 00
5	NAXISn, nNAXIS*n*, *n* = 2, . . . , value of NAXISNAXIS
	⋮
	(other keywords, which must*must* include . . .)
	GROUPS GROUPS = TT
	PCOUNTPCOUNT
	GCOUNTGCOUNT
	⋮
last	ENDEND

NAXIS1NAXIS1 keyword. The value field shall*shall* contain the integer 00, a signature of random groups random-groups format indicating that there is no primary data array.

NAXISnNAXIS*n* keywords (n=2*n* = 2, . . . , value of NAXISNAXIS). The NAXISnkeywords mustNAXIS*n* keywords *must* be present for all values n = 2, ..., NAXIS*n* = 2, . . . , NAXIS, in increasing order of n*n*, and for no larger values of n*n*. The value field shall*shall* contain an integer, representing the number of positions along axis n-1Axis $n-1$ of the data array in each group.

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

PCOUNTPCOUNT keyword. The value field shall*shall* contain an integer equal to the number of parameters preceding each array in a group.

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

ENDEND keyword. This keyword has no associated value. Bytes 9 through 80 shall*shall* contain ASCII spaces (decimal 32 or hexadecimal 20).

The total number of bits in the random groups random-groups records exclusive of the fill described in Sect. 6.2 is given by the following expression:

$$N_{\rm bits}N_{\rm bits} = |\text{BITPIXBITPIX}| \times \text{GCOUNTGCOUNT} \times (\text{PCOUNTPCOUNT} + \text{NAXIS2NAXIS2} \times \text{NAXIS3NAXIS3} \times \cdots$$

where $N_{\rm bits}$ is$N_{\rm bits}$ is non-negative and the number of bits excluding fill; mm is the value of NAXISNAXIS; and BITPIX, GCOUNT, PCOUNTBITPIX, GCOUNT, PCOUNT, and the NAXISnNAXIS*n* represent the values associated with those keywords.

DRAFT

6.1.2. Reserved keywords

PTYPEnPTYPE*n* keywords. The value field shall*shall* contain a character string giving the name of parameter nParameter *n*. If the PTYPEnPTYPE*n* keywords for more than one value of n*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*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.

PSCALnPSCAL*n* keywords. This keyword shall*shall* be used, along with the PZEROnPZERO*n* keyword, when the nth FITSn^{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. 5. The value field shall*shall* contain a floating-point number representing the coefficient of the linear term in Eq. 5, the scaling factor between true values and group parameter values at zero offset. The default value for this keyword is 1.0`1.0`.

PZEROnPZERO*n* keywords. This keyword shall*shall* be used, along with the PSCALnPSCAL*n* keyword, when the nth FITSn^{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*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`0.0`. The transformation equation is as follows:

$$\text{physical\_value} = \text{PZEROn}\texttt{PZERO}n + \text{PSCALn}\texttt{PSCAL}n \times \text{group\_parm\_value} \quad \text{group\_param\_val(n)} \tag{5}$$

6.2. Data sequence

Random groups data shallRandom-groups data *shall* consist of a set of groups. The number of groups shall*shall* be specified by the GCOUNTGCOUNT keyword in the associated header. Each group shall*shall* consist of the number of parameters specified by the PCOUNTPCOUNT keyword followed by an array with the number of elements $N_{\rm elem}$ given$N_{\rm elem}$ given by the following expression:

$$N_{\rm elem}N_{\rm elem} = (\text{NAXIS2}\texttt{NAXIS2} \times \text{NAXIS3}\texttt{NAXIS3} \times \cdots \times \text{NAXISm}\texttt{NAXIS}m) \tag{6}$$

where $N_{\rm elem}$ $N_{\rm elem}$ is the number of elements in the data array in a group, m*m* is the value of NAXISNAXIS, and the NAXISnNAXIS*n* represent the values associated with those keywords.

The first parameter of the first group shall*shall* appear in the first location of the first data block. The first element of each array shall*shall* immediately follow the last parameter associated with that group. The first parameter of any subsequent group shall*shall* immediately follow the last element of the array of the previous group. The arrays shall*shall* be organized internally in the same way as an ordinary primary data array. If the groups data do not fill the final data block, the remainder of the block shall*shall* be filled with zero values in the same way as a primary data array (Sect. 3.3.2). If random groups random-groups records are present, there shall*shall* be no primary data array.

6.3. Data representation

Permissible data representations are those listed in Sect. 5. Parameters and elements of associated data arrays shall*shall* have the same representation. If more precision is required for an associated parameter than for an element of a data array, the parameter shall*shall* be divided into two or more addends, represented by the same value for the PTYPEnPTYPE*n* keyword. The value shall*shall* be the sum of the physical values, which may*may* have been obtained from the group parameter values using the PSCALnand PZEROnPSCAL*n* and PZERO*n* keywords.

7. Standard extensions

A standard extension is a conforming extension whose organization and content are completely specified in this standardStandard. The specifications for the 3 three currently defined standard extensions, namely,

1. 'IMAGE'`IMAGE` extensions;
2. 'TABLE'ASCII table `TABLE` ASCII-table extensions; and
3. 'BINTABLE'binary table extensions ; `BINTABLE` binary-table extensions

are given in the following sections. A list of other conforming extensions is given in Appendix F.

7.1. Image extension

The FITSimage *FITS* `IMAGE` extension is nearly identical in structure to the the primary HDU and is used to store an array of data. Multiple image `IMAGE` extensions can be used to store any number of arrays in a single FITS*FITS* file. The first keyword in an image extension shallrecord in an `IMAGE` extension *shall* be `XTENSION=␣'IMAGE␣␣␣'` `XTENSION=␣'IMAGE␣␣␣'`.

7.1.1. Mandatory keywords

The XTENSIONkeyword is required`XTENSION` keyword is *required* to be the first keyword of all image `IMAGE` extensions. The keyword records in the header of an image extension must use the keywords defined in Table 13 in the order specified. No other keywords may*may* intervene between the XTENSIONand GCOUNT`XTENSION` and `GCOUNT` keywords.

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

BITPIX`BITPIX` keyword. The value field shall*shall* contain an integer. The absolute value is used in computing the sizes of data structures. It shall*shall* specify the number of bits that represent a data value. The only valid values of BITPIX`BITPIX` are given in Table 8. Writers of IMAGEextensions shouldselect a

DRAFT

Table 13: Mandatory keywords in image IMAGE extensions.

# Position	Keyword
1	XTENSION=␣'IMAGE␣␣␣'
2	BITPIXBITPIX
3	NAXISNAXIS
4	NAXISn, nNAXIS*n*, *n* = 1, . . . , NAXISNAXIS
5	PCOUNT PCOUNT = 00
6	GCOUNT GCOUNT = 11
	⋮
	(other keywords . . .)
	⋮
last	ENDEND

BITPIXIMAGE extensions *should* select a BITPIX data type appropriate to the form, range of values, and accuracy of the data in the array.

NAXISNAXIS keyword. The value field shall*shall* contain a non-negative integer no greater than 999, representing the number of axes in the associated data array. If the value is zero then the image extension shall notIMAGE extension *shall not* have any data blocks following the header.

NAXISnNAXIS*n* keywords. The NAXISnkeywords mustNAXIS*n* keywords *must* be present for all values n = 1, ..., NAXIS*n* = 1, . . . , NAXIS, in increasing order of n*n*, and for no other values of n*n*. The value field of this indexed keyword shall*shall* contain a non-negative integer, representing the number of elements along axis nAxis *n* of a data array. If the value of any of the NAXISnNAXIS*n* keywords is zero, then the image extension shall notIMAGE extension *shall not* have any data blocks following the header. If NAXISNAXIS is equal to 0, there should notbe any NAXISn0 there *should not* be any NAXIS*n* keywords.

PCOUNTPCOUNT keyword. The value field shall*shall* contain the integer 00.

GCOUNTGCOUNT keyword. The value field shall*shall* contain the integer 11; each image IMAGE extension contains a single array.

ENDEND keyword. This keyword has no associated value. Bytes 9 through 80 shall*shall* be filled with ASCII spaces (decimal 32 or hexadecimal 20).

7.1.2. Other reserved keywords

The reserved keywords defined in Sect. 4.4.2 (except for EXTENDand BLOCKED) mayEXTEND and BLOCKED) *may* appear in an image extension image-extension header. The keywords must*must* be used as defined in that section.

7.1.3. Data sequence

The data format shall*shall* be identical to that of a primary data array as described in Sect. 3.3.2.

7.2. The ASCII table ASCII-table extension

The ASCII table ASCII-table extension provides a means of storing catalogs and tables of astronomical data in FITS*FITS* format. Each row of the table consists of a fixed-length sequence of ASCII characters divided into fields that correspond to the columns in the table. The first keyword in an ASCII table extension shallbe record in an ASCII-table extension *shall* be XTENSION=␣'TABLE␣␣␣'.

7.2.1. Mandatory keywords

The header of an ASCII table extension mustASCII-table extension *must* use the keywords defined in Table 14. The first keyword mustbe XTENSION*must* be XTENSION; the seven keywords following XTENSION(BITPIX. . . TFIELDS) mustXTENSION (BITPIX . . . TFIELDS) *must* be in the order specified with no intervening keywords.

XTENSIONXTENSION keyword. The value field shall*shall* contain the character string value text 'TABLE␣␣␣'.

BITPIXBITPIX keyword. The value field shall*shall* contain the integer 88, denoting that the array contains ASCII characters.

NAXISNAXIS keyword. The value field shall*shall* contain the integer 22, denoting that the included data array is two-dimensional: rows and columns.

NAXIS1NAXIS1 keyword. The value field shall*shall* contain a non-negative integer, giving the number of ASCII characters in each row of the table. This includes all the characters in the defined fields plus any characters that are not included in any field.

NAXIS2NAXIS2 keyword. The value field shall*shall* contain a non-negative integer, giving the number of rows in the table.

PCOUNTPCOUNT keyword. The value field shall*shall* contain the integer 00.

GCOUNTGCOUNT keyword. The value field shall*shall* contain the integer 11; the data blocks contain a single table.

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

TBCOLnTBCOL*n* keywords. The TBCOLnkeywords mustTBCOL*n* keywords *must* be present for all values n = 1, ..., TFIELDS*n* = 1, . . . , TFIELDS and for no other values of n*n*. The value field of this indexed keyword shall*shall* contain

DRAFT

Table 14: Mandatory keywords in ASCII table ASCII-table extensions.

# Position	Keyword
1	XTENSION= 'TABLE '
2	BITPIX BITPIX = 8
3	NAXIS NAXIS = 2
4	NAXIS1 NAXIS1
5	NAXIS2 NAXIS2
6	PCOUNT PCOUNT = 0
7	GCOUNT GCOUNT = 1
8	TFIELDS TFIELDS
	:
	(other keywords, including (if TFIELDS TFIELDS is not zero) . . .)
	TTYPEn, n=1, 2, . . . , k where k TTYPE*n*, *n* = 1, 2, . . . , *k*, where *k* is the value of TFIELDS (Recommended TFIELDS *(recommended)*
	TBCOLn, n=1, 2, . . . , k where k TBCOL*n*, *n* = 1, 2, . . . , *k*, where *k* is the value of TFIELDS (Required TFIELDS *(required)*
	TFORMn, n=1, 2, . . . , k where k TFORM*n*, *n* = 1, 2, . . . , *k*, where *k* is the value of TFIELDS (Required TFIELDS *(required)*
	:
last	END END

Table 15: Valid TFORMn TFORM*n* format values in TABLE TABLE extensions.

Field value	Data type
Aw Aw	Character
Iw Iw	Decimal integer
Fw.d Fw.d	Floating-point, fixed decimal notation
Ew.d Ew.d	Floating-point, exponential notation
Dw.d Dw.d	Floating-point, exponential notation

Notes. w is the width in characters of the field and d is the number of digits to the right of the decimal.

an integer specifying the column in which field nField *n* starts. The first column of a row is numbered 1.

TFORMn TFORM*n* keywords. The TFORMn keywords must TFORM*n* keywords *must* be present for all values n = 1, ..., TFIELDS *n* = 1, . . . , TFIELDS and for no other values of n*n*. The value field of this indexed keyword shall*shall* contain a character string describing the format in which field nField *n* is encoded. Only the formats in Table 15, interpreted as Fortran (ISO 2004) input formats and discussed in more detail in Sect. 7.2.5, are permitted for encoding. Format codes must*must* be specified in upper case. Other format editing codes common to Fortran 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. The TDISPn TDISP*n* keyword, defined in Sect. 7.2.2, may 7.2.2, *may* be used to *recommend* that a decimal integer value in an ASCII table be displayed as the equivalent binary, octal, or hexadecimal value.

END END keyword. This keyword has no associated value. Bytes 9 through 80 shall*shall* contain ASCII spaces (decimal 32 or hexadecimal 20).

7.2.2. Other reserved keywords

In addition to the reserved keywords defined in Sect. 4.4.2 (except for EXTENDand BLOCKED EXTEND and BLOCKED), the following other reserved keywords may*may* be used to describe the structure of an ASCII table ASCII-table data array. They are optional*optional*, but if they appear within an ASCII table ASCII-table extension header, they must*must* be used as defined in this section of this standardStandard.

TTYPEn TTYPE*n* keywords. The value field for this indexed keyword shall*shall* contain a character string giving the name of field nField *n*. It is *strongly recommended* that every field of the table be assigned a unique, case-insensitive name with this keyword, and it is recommended*recommended* that the character string be composed only of upper and lower case upper- and lower-case letters, digits, and the underscore ('_' '_', decimal 95, hexadecimal 5F) character. Use of other characters is not recommended*not recommended* because it may be difficult to map the column names into variables in some languages (e.g., any hyphens, '*' or '+' '*' or '+' characters in the name may be confused with mathematical operators). String comparisons with the TTYPEnkeyword values should not TTYPE*n* keyword values *should not* be case sensitive (e.g., 'TIME' and 'Time' should 'TIME' and 'Time' *should* be interpreted as the same name).

TUNITn TUNIT*n* keywords. The value field shall*shall* contain a character string describing the physical units in which the quantity in field nField *n*, after any application of TSCALnand TZEROn TSCAL*n* and TZERO*n*, is expressed. Units must*must* follow the prescriptions in Sect. 4.3.

TSCALn TSCAL*n* keywords. This indexed keyword shall*shall* be used, along with the TZEROn TZERO*n* keyword, to linearly scale the values in the table field nField *n* to transform them into the physical values that they represent using Eq. 7. The value field shall*shall* contain a floating-point number representing the coefficient of the linear term in the scaling equation. The default value for this keyword is 1.0 1.0. This keyword must not*must not* be used for A-format fields.

DRAFT

Table 16: Valid TDISPnTDISP*n* format values in TABLETABLE extensions.

Field value	Data type
Aw `Aw`	Character
Iw.m `Iw.m`	Integer
Bw.m `Bw.m`	Binary, integers only
Ow.m `Ow.m`	Octal, integers only
Zw.m `Zw.m`	Hexadecimal, integers only
Fw.d `Fw.d`	Floating-point, fixed decimal notation
Ew.dEe `Ew.dEe`	Floating-point, exponential notation
ENw.d `ENw.d`	Engineering; E format with exponent multiple of three
ESw.d `ESw.d`	Scientific; same as EN but non-zero leading digit if not zero
Gw.dEe `Gw.dEe`	General; appears as F if significance not lost, else E.
Dw.dEe `Dw.dEe`	Floating-point, exponential notation

Notes. `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*.

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

$$\text{physical\_value} = \text{TZERO}n\text{TZERO}n + \text{TSCALn}\text{TSCAL}n \times \text{field\_value} \quad (7)$$

where field_valuefield_value is the value that is actually stored in that table field in the FITS*FITS* file.

TZEROnTZERO*n* keywords. This indexed keyword shall*shall* be used, along with the TSCALnTSCAL*n* keyword, to linearly scale the values in the table field nField *n* to transform them into the physical values that they represent using Eq. 7. The value field shall*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`0.0`. This keyword must not*must not* be used for A-format fields.

TNULLnTNULL*n* keywords. The value field for this indexed keyword shall*shall* contain the character string that represents an undefined value for field nField *n*. The string is implicitly space filled to the width of the field.

TDISPnTDISP*n* keywords. The value field of this indexed keyword shall*shall* contain a character string describing the format recommended for displaying an ASCII text ASCII-text representation of of the contents of field nField *n*. This keyword overrides the default display format given by the TFORMnTFORM*n* keyword. If the table value has been scaled, the physical value, derived using Eq. 7, shall*shall* be displayed. All elements in a field shall*shall* be displayed with a single, repeated format. Only the format codes in Table 16, interpreted as Fortran (ISO 2004) output formats, and discussed in more detail in Sect. 7.3.4, are permitted for encoding. The format codes must*must* be specified in upper case. If the Bw.m, Ow.m, and Zw.mBw.m, Ow.m, and Zw.m formats are not readily available to the reader, the Iw.mdisplay format mayIw.m display format *may* be used instead, and if the ENw.dand ESw.dENw.d and ESw.d formats are not available, Ew.dmayEw.d *may* be used.

The following four keywords may*may* be used to specify minimum and maximum values in numerical columns of a FITS*FITS* ASCII or binary table. These keywords must*must* have the same data type as the physical values in the associated column (either an integer or a floating point floating-point number). Any undefined elements in the column shall*shall* be excluded when determining the value of these keywords.

TDMINnTDMIN*n* keywords. The value field shall*shall* contain a number giving the minimum physical value contained in column nColumn *n* of the table. This keyword is analogous to the DATAMINDATAMIN keyword that is defined for arrays in Sect. 4.4.2.5.

TDMAXnTDMAX*n* keywords. The value field shall*shall* contain a number giving the maximum physical value contained in column nColumn *n* of the table. This keyword is analogous to the DATAMAXDATAMAX keyword that is defined for arrays in Sect. 4.4.2.5.

TLMINnTLMIN*n* keywords. The value field shall*shall* contain a number that specifies the minimum physical value in column nColumn *n* that has a valid meaning or interpretation. The column is not required *required* to actually contain any elements that have this value, and the column may *may* contain elements with physical values less than TLMINnTLMIN*n*, however, the interpretation of any such out-of-range column elements is not defined.

TLMAXnTLMAX*n* keywords. The value field shall*shall* contain a number that specifies the maximum physical value in column nColumn *n* that has a valid meaning or interpretation. The column is not required *required* to actually contain any elements that have this value, and the column may *may* contain elements with physical values greater than TLMAXnTLMAX*n*, however, the interpretation of any such out-of-range column elements is not defined.

The TLMINnand TLMAXnTLMIN*n* and TLMAX*n* keywords are commonly used when constructing histograms of the data values in a column. For example, if a table contains columns that give the X and Y *X* and *Y* pixel location of a list of photons that were detected by a photon counting photon-counting device, then the TLMINnand TLMAXnTLMIN*n* and TLMAX*n* keywords could be used respectively to specify the minimum and maximum values

DRAFT

that the detector is capable of assigning to the X and Y *X* and *Y* columns.

7.2.3. Data sequence

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

7.2.4. Fields

Each row in the array shall*shall* consist of a sequence of from 0 to 999 fields, as specified by the TFIELDS`TFIELDS` keyword, with one entry in each field. For every field, the Fortran (ISO 2004) format of the information contained (given by the TFORMn`TFORM`*n* keyword), the location in the row of the beginning of the field (given by the TBCOLn`TBCOL`*n* keyword), and (*optionally*, but *strongly recommended*) the field name (given by the TTYPEn`TTYPE`*n* keyword), shall*shall* be specified in the associated header. The location and format of fields shall*shall* be the same for every row. Fields may*may* overlap, but this usage is not recommended*not recommended*. Only a limited set of ASCII character values may*may* appear within any field, depending on the field type as specified below. There may*may* be characters in a table row that are not included in any field, (e.g., between fields, or before the first field or after the last field). Any 7-bit seven-bit ASCII character may*may* occur in characters of a table row that are not included in a defined field. A common convention is to include a space character between each field for added legibility if the table row is displayed verbatim. It is also permissible to add control characters, such as a carriage return or line feed line-feed character, following the last field in each row as a way of formatting the table if it is printed or displayed by a text editing text-editing program.

7.2.5. Entries

All data in an ASCII table extension field shallASCII-table extension field *shall* be ASCII text in a format that conforms to the rules for fixed field input in Fortran (ISO 2004) format, as described below. The only possible formats shall*shall* be those specified in Table 15. If values of −0 and +0 need to be distinguished, then the sign character should*should* appear in a separate field in character format. TNULLn`TNULL`*n* keywords may*may* be used to specify a character string that represents an undefined value in each field. The characters representing an undefined value may*may* differ from field to field but must*must* be the same within a field. Writers of ASCII tables should*should* select a format for each field that is appropriate to the form, range of values, and accuracy of the data in that field. This standard Standard does not impose an upper limit on the number of digits of precision, nor any limit on the range of numeric values. Software packages that read or write data according to this standard Standard could be limited, however, in the range of values and exponents that are supported (e.g., to the range that can be represented by 32-bit or 64-bit binary numbers).

The value of each entry shall*shall* be interpreted as described in the following paragraphs.

Character fields. The value of a character-formatted (Aw`Aw`) field is a character string of width *w* `w` containing the characters in columns TBCOLnthrough TBCOLn+`w` − 1`TBCOL`*n* through `TBCOL`*n* + `w` − 1. The character string shall*shall* be composed of the restricted set of ASCII text ASCII-text characters with decimal values in the range 32 through 126 (hexadecimal 20 through 7E).

Integer fields. The value of an integer-formatted (Iw`Iw`) field is a signed decimal integer contained in columns TBCOLnthrough TBCOLn+`w` − 1 Columns `TBCOL`*n* through `TBCOL`*n* + `w` − 1 consisting of a single optionalsign ('+'or '-'*optional* sign (`'+'` or `'-'`) followed by one or more decimal digits ('0'through '9'`'0'` through `'9'`). Non-significant space characters may *may* precede and/or follow the integer value within the field. A blank field has value 0. All characters other than leading and trailing spaces, a contiguous string of decimal digits, and a single leading sign character are forbidden.

Real fields. The value of a real-formatted field (Fw.d, Ew.d, Dw.d`Fw.d`, `Ew.d`, `Dw.d`) is a real number determined from the wcharacters from columns TBCOLnthrough TBCOLn+`w` − 1`w` characters from Columns `TBCOL`*n* through `TBCOL`*n* + `w` − 1. The value is formed by

1. discarding any trailing space characters in the field and right-justifying the remaining characters,
2. interpreting the first non-space characters as a numeric string consisting of a single optionalsign ('+'or '-'*optional* sign (`'+'` or `'-'`) followed by one or more decimal digits ('0'through '9'`'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'`'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. The use of implicit decimal points is *deprecated* and is strongly discouraged because of the possibility that FITSreading *FITS*-reading programs will misinterpret the data value. Therefore, real-formatted fields should*should* always contain an explicit decimal point.
3. If the numeric string is terminated by a

   (a) '+'or '-'`'+'` or `'-'`, interpreting the following string as an exponent in the form of a signed decimal integer, or

   (b) 'E', or 'D'`'E'`, or `'D'`, interpreting the following string as an exponent of the form E`E` or D`D` followed by an *optionally* signed decimal integer constant.

DRAFT

4. The exponent string, if present, is terminated by the end of the right-justified string.
5. Characters other than those specified above, including embedded space characters, 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 $\times$ $10^{(\text{exponent\_string})}$ $10^{(\text{exponent\_string})}$. The default exponent is zero and a blank field has value zero. There is no difference between the F, D, and E`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*may* be represented. It is good form to specify a Dformat in TFORMn`D` format in `TFORM`*n* 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 E).

7.3. *Binary table* *Binary-table* *extension*

The binary table binary-table extension is similar to the ASCII table in that it provides a means of storing catalogs and tables of astronomical data in FITS*FITS* format, however, it offers more features and provides more-efficient data storage than ASCII tables. The numerical values in binary tables are stored in more-compact binary formats rather than coded into ASCII, and each field of a binary table can contain an array of values rather than a simple scalar as in ASCII tables. The first keyword in a binary table extension shallrecord in a binary-table extension *shall* be `XTENSION= 'BINTABLE'`XTENSION= 'BINTABLE'.

7.3.1. Mandatory keywords

The XTENSION`XTENSION` keyword is the first keyword of all binary table binary-table extensions. The seven keywords following (BITPIX...TFIELDS) mustBITPIX ...TFIELDS) *must* be in the order specified in Table 17, with no intervening keywords.

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

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

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

NAXIS1`NAXIS1` keyword. The value field shall*shall* contain a non-negative integer, giving the number of 8-bit eight-bit bytes in each row of the table.

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

PCOUNT`PCOUNT` keyword. The value field shall*shall* contain the number of bytes that follow the table in the supplemental data area called the heap.

GCOUNT`GCOUNT` keyword. The value field shall*shall* contain the integer 1`1`; the data blocks contain a single table.

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

TFORMn`TFORM`*n* keywords. The TFORMnkeywords must`TFORM`*n* keywords *must* be present for all values n = 1, ..., TFIELDS*n* = 1, ..., `TFIELDS` and for no other values of n*n*. The value field of this indexed keyword shall*shall* contain a character string of the form *r*Ta`T`*a*. The repeat count *r* is the ASCII representation of a non-negative integer specifying the number of elements in field nField *n*. The default value of *r* is 1`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`T` specifies the data type of the contents of field nField *n*. Only the data types in Table 18 are permitted. The format codes must*must* be specified in upper case. For fields of type P`P` or Q`Q`, the only permitted repeat counts are 0 and 1. `0` and `1`. The additional characters *a* are optional*optional* and are not further defined in this standardStandard. Table 18 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} n_{row} in a table row is given by

$$n_{\text{row}\,\text{row}} = \sum_{i=1}^{\text{TFIELDS}\,\text{TFIELDS}} r_i b_i \qquad (8)$$

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

END`END` keyword. This keyword has no associated value. Bytes 9 through 80 shall*shall* contain ASCII spaces (decimal 32 or hexadecimal 20).

7.3.2. Other reserved keywords

In addition to the reserved keywords defined in Sect. 4.4.2 (except for EXTENDand BLOCKED`EXTEND` and `BLOCKED`), the following other reserved keywords may*may* be used to describe the structure of a binary table binary-table data array. They are optional*optional*, but if they appear within a binary table binary-table extension header, they must*must* be used as defined in this section of this standardStandard.

TTYPEn`TTYPE`*n* keywords. The value field for this indexed keyword shall*shall* contain a character string giving the name of field nField *n*. It is *strongly recommended* that every field of the table be assigned a unique, case-insensitive name with this keyword, and it is recommended*recommended* that the character string be composed only of upper and lower case upper-

DRAFT

Table 17: Mandatory keywords in binary table binary-table extensions.

# Position	Keyword
1	XTENSION=␣'BINTABLE'
2	BITPIX BITPIX = 8
3	NAXIS NAXIS = 2
4	NAXIS1 NAXIS1
5	NAXIS2 NAXIS2
6	PCOUNT PCOUNT
7	GCOUNT GCOUNT = 1
8	TFIELDS TFIELDS
	⋮
	(other keywords, including (if TFIELDS TFIELDS is not zero) . . .)
	TTYPEn, n=1, 2, . . . , k where k TTYPE*n*, *n* = 1, 2, . . . , *k*, where *k* is the value of TFIELDS (Recommended TFIELDS (*recommended*)
	TFORMn, n=1, 2, . . . , k where k TFORM*n*, *n* = 1, 2, . . . , *k*, where *k* is the value of TFIELDS (Required TFIELDS (*required*)
	⋮
last	END END

Table 18: Valid TFORMn TFORM*n* data types in BINTABLE BINTABLE extensions.

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

Notes. (†) Number of eight-bit bytes needed to contain all bits.

and lower-case letters, digits, and the underscore ('_' _, decimal 95, hexadecimal 5F) character. Use of other characters is not recommended*not recommended* because it may be difficult to map the column names into variables in some languages (e.g., any hyphens, '*' or '+' '*' or '+' characters in the name may be confused with mathematical operators). String comparisons with the TTYPEnkeyword values should not TTYPE*n* keyword values *should not* be case sensitive (e.g., 'TIME' and 'Time' should 'TIME' and 'Time' *should* be interpreted as the same name).

TUNITn TUNIT*n* keywords. The value field shall*shall* contain a character string describing the physical units in which the quantity in field nField *n*, after any application of TSCALnand TZEROn TSCAL*n* and TZERO*n*, is expressed. Units must*must* follow the prescriptions in Sect. 4.3.

TSCALn TSCAL*n* keywords. This indexed keyword shall*shall* be used, along with the TZEROn TZERO*n* keyword, to linearly scale the values in the table field nField *n* to transform them into the physical values that they represent using Eq. 7. It must not*must not* be used if the format of field nis A, L, or XField *n* is 'A', 'L', or 'X'. For fields with all other data types, the value field shall*shall* contain a floating-point number representing the coefficient of the linear term in Eq. 7, which is used to compute the true physical value of the field, or, in the case of the complex data types Cand M '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 1.0. For fields of type Por Q 'P' or 'Q', the values of TSCALnand TZEROn TSCAL*n* and TZERO*n* are to be applied to the values in the data array in the heap area, not the values of the array descriptor (see Sect. 7.3.5).

TZEROn TZERO*n* keywords. This indexed keyword shall*shall* be used, along with the TSCALn TSCAL*n* keyword, to linearly scale the values in the table field nField *n* to transform them into the physical values that they represent using Eq. 7. It must not*must not* be used if the format of field nis A, L, or XField *n* is 'A', 'L', or 'X'. For fields with all other data types, the value field shall*shall* contain a floating-point number representing the true physical value corresponding to a value of zero in field nof the FITSField *n* of the *FITS* file, or, in the case of the complex data types Cand M '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 0.0. Equation 7 is used to compute a true physical value from the quantity in field nField *n*. For fields of type Por Q 'P' or 'Q', the values of TSCALnand TZEROn TSCAL*n* and TZERO*n* are to be applied to the values in the data array in the heap area, not the values of the array descriptor (see Sect. 7.3.5).

In addition to its use in representing floating-point values as scaled integers, the TZEROn TZERO*n* keyword is also used when storing unsigned integer values in the field. In this special case the TSCALnkeyword shall TSCAL*n* keyword *shall* have the default value of 1.0 and the TZEROnkeyword shall 1.0 and the TZERO*n* keyword *shall* have one of the integer values shown in Table 19.

Since the binary table binary-table format does not support a native unsigned integer data type (except for the unsigned 8-bit 'B'eight-bit 'B' column type), the unsigned values are stored in the field as native signed integers with the appropriate integer offset specified by the TZEROn TZERO*n* keyword value shown in the table. For the byte column type, the converse technique can be used to store signed byte values as native unsigned values with the negative TZEROn TZERO*n* offset. In each case, the physical value is computed by adding the offset specified by the

DRAFT

Table 19: Usage of TZEROnTZERO*n* to represent non-default integer data types.

TFORMnTFORM*n*	Native data type	Physical data type	TZEROnTZERO*n*	
B 'B'	unsigned	signed byte	-128-128	(-2^7)
I'I'	signed	unsigned 16-bit	3276832768	(2^{15})
J'J'	signed	unsigned 32-bit	21474836482147483648	(2^{31})
K'K'	signed	unsigned 64-bit	92233720368547758089223372036854775808	(2^{63})

TZEROnTZERO*n* keyword to the native data type value that is stored in the table field.

TNULLnTNULL*n* keywords. The value field for this indexed keyword shall*shall* contain the integer that represents an undefined value for field nof data type B, I, Jor K, or Por Qarray descriptorField *n* of Data Type B, I, J or K, or P or Q array-descriptor fields (Sect. 7.3.5) that point to B, I, Jor KB, I, J, or K integer arrays. The keyword must not*must not* be used if field nField *n* is of any other data type. The value of this keyword corresponds to the table column values before applying any transformation indicated by the TSCALnand TZEROnTSCAL*n* and TZERO*n* keywords.

If the TSCALnand TZEROnTSCAL*n* and TZERO*n* keywords do not have the default values of 1.0 and 0.01.0 and 0.0, respectively, then the value of the TNULLnkeyword mustTNULL*n* keyword *must* equal the actual value in the FITS*FITS* file that is used to represent an undefined element and not the corresponding physical value (computed from Eq. 7). To cite a specific, common example, *unsigned* 16-bit integers are represented in a *signed* integer column (with TFORMn TFORM*n* ='I' 'I') by setting TZEROn TZERO*n* =32768and TSCALn 32768 and TSCAL*n* =1 1. If it is desired to use elements that have an *unsigned* value (i.e., the physical value) equal to 0 to represent undefined elements in the field, then the TNULLnkeyword mustTNULL*n* keyword *must* be set to the value -32768 -32768 because that is the actual value stored in the FITS*FITS* file for those elements in the field.

TDISPnTDISP*n* keywords. The value field of this indexed keyword shall*shall* contain a character string describing the format recommended for displaying an ASCII text ASCII-text representation of the contents of field nField *n*. If the table value has been scaled, the physical value, derived using Eq. 7, shall*shall* be displayed. All elements in a field shall*shall* be displayed with a single, repeated format. For purposes of display, each byte of bit (type XType X) and byte (type BType B) arrays is treated as an unsigned integer. Arrays of type AmayType A *may* be terminated with a zero byte. Only the format codes in Table 20, interpreted as Fortran (ISO 2004) output formats, and discussed in more detail in Sect. 7.3.4, are permitted for encoding. The format codes must*must* be specified in upper case. If the Bw.m, Ow.m, and Zw.mBw.m, Ow.m, and Zw.m formats are not readily available to the reader, the Iw.mdisplay format mayIw.m display format *may* be used instead, and if the ENw.dand ESw.dENw.d and ESw.d formats are not available, Ew.dmayEw.d *may* be used. In the case of fields of type Por Q, the TDISPnType P or Q, the TDISP*n* value applies to the data array pointed to by the array descriptor (Sect. 7.3.5), not the values in the array descriptor itself.

THEAPTHEAP keyword. The value field of this keyword shall*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*shall* be the product of the values of NAXIS1NAXIS1 and NAXIS2NAXIS2. This keyword shall not*shall not* be used if the value of PCOUNTis zeroPCOUNT is 0. The use of this keyword is described in in Sect. 7.3.5.

TDIMnTDIM*n* keywords. The value field of this indexed keyword shall*shall* contain a character string describing how to interpret the contents of field nField *n* as a multi-dimensional array with a format of '(*l*,*m*, *n*...)'where *l*, *m*, *n*'(*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 (*ll*) 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 TDIMnTDIM*n* keyword. The size must*must* be less than or equal to the repeat count on the TFORMnin the TFORM*n* keyword, or, in the case of columns that have a 'P'or 'Q'TFORMn'P' or 'Q' TFORM*n* data type, less than or equal to the array length specified in the variable-length array descriptor (see Sect. 7.3.5). In the special case where the variable-length array descriptor has a size of zero, then the TDIMnTDIM*n* keyword is not applicable. If the number of elements in the array implied by the TDIMnis less TDIM*n* is fewer than the allocated size of the array in the FITS*FITS* file, then the unused trailing elements should*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 data types in Sect. 7.3.3. For example, a Fortran CHARACTER*20CHARACTER*20 variable could be represented in a binary table as a character array declared as TFORMn TFORM*n* = '20A'. Arrays of strings, i.e., multi-dimensional character arrays, may*may* be represented using the TDIMnTDIM*n* notation. For example, if TFORMn TFORM*n* =and TDIMn = '(5,4,3)' '60A' and TDIM*n* = '(5,4,3)', then the entry consists of a 4 × 3 array of strings of five characterseacheach comprising five characters.

The following four keywords may*may* be used to specify minimum and maximum values in numerical columns of a FITS*FITS* ASCII or binary table. These keywords must*must* have the same data type as the physical values in the associated column (either an integer or a floating point floating-point number). Any undefined elements in the column or any other IEEE special values in the case of floating point columns shallfloating-point columns *shall* be excluded when determining the value of these keywords.

DRAFT

Table 20: Valid TDISPnTDISP*n* format values in BINTABLEBINTABLE extensions.

Field Value	Data type
Aw Aw	Character
Lw Lw	Logical
Iw.m Iw.m	Integer
Bw.m Bw.m	Binary, integers only
Ow.m Ow.m	Octal, integers only
Zw.m Zw.m	Hexadecimal, integers only
Fw.d Fw.d	Floating-point, fixed decimal notation
Ew.dEe Ew.dEe	Floating-point, exponential notation
ENw.d ENw.d	Engineering; E format with exponent multiple of three
ESw.d ESw.d	Scientific; same as EN but non-zero leading digit if not zero
Gw.dEe Gw.dEe	General; appears as F if significance not lost, else E.
Dw.dEe Dw.dEe	Floating-point, exponential notation

Notes. 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*.

TDMINnTDMIN*n* keywords. The value field shall*shall* contain a number giving the minimum physical value contained in column nColumn *n* of the table. This keyword is analogous to the DATAMINDATAMIN keyword that is defined for arrays in Sect. 4.4.2.5.

TDMAXnTDMAX*n* keywords. The value field shall*shall* contain a number giving the maximum physical value contained in column nColumn *n* of the table. This keyword is analogous to the DATAMAXDATAMAX keyword that is defined for arrays in Sect. 4.4.2.5.

TLMINnTLMIN*n* keywords. The value field shall*shall* contain a number that specifies the minimum physical value in column nColumn *n* that has a valid meaning or interpretation. The column is not required *required* to actually contain any elements that have this value, and the column may *may* contain elements with physical values less than TLMINnTLMIN*n*, however, the interpretation of any such out-of-range column elements is not defined.

TLMAXnTLMAX*n* keywords. The value field shall*shall* contain a number that specifies the maximum physical value in column nColumn *n* that has a valid meaning or interpretation. The column is not required *required* to actually contain any elements that have this value, and the column may *may* contain elements with physical values greater than TLMAXnTLMAX*n*, however, the interpretation of any such out-of-range column elements is not defined.

The TLMINnand TLMAXnTLMIN*n* and TLMAX*n* keywords are commonly used when constructing histograms of the data values in a column. For example, if a table contains columns that give the X and Y *X* and *Y* pixel location of a list of photons that were detected by a photon counting photon-counting device, then the TLMINnand TLMAXnTLMIN*n* and TLMAX*n* keywords could be used respectively to specify the minimum and maximum values that the detector is capable of assigning to the X and Y *X* and *Y* columns.

7.3.3. Data sequence

The data in a binary table extension shallbinary-table extension *shall* consist of a main data tablewhich may, which *may*, but need not, be followed by additional bytes in the supplemental data area. The positions in the last data block after the last additional byte, or, if there are no additional bytes, the last character of the last row of the main data table, shall*shall* be filled by setting all bits to zero.

7.3.3.1. Main data table

The table is constructed from a two-dimensional byte array. The number of bytes in a row shall*shall* be specified by the value of the NAXIS1NAXIS1 keyword and the number of rows shall*shall* be specified by the NAXIS2NAXIS2 keyword of the associated header. Within a row, fields shall*shall* be stored in order of increasing column number, as determined from the nof the TFORMn*n* of the TFORM*n* keywords. The number of bytes in a row and the number of rows in the table shall*shall* determine the size of the byte array. Every row in the array shall*shall* have the same number of bytes. The first row shall*shall* begin at the start of the data block immediately following the last header block. Subsequent rows shall*shall* begin immediately following the end of the previous row, with no intervening bytes, independent of the FITS*FITS* block structure. Words need not be aligned along word boundaries.

Each row in the array shall*shall* consist of a sequence of from 0 to 999 fields as specified by the TFIELDSTFIELDS keyword. The number of elements in each field and their data type shall*shall* be specified by the TFORMnTFORM*n* keyword in the associated header. A separate format keyword must*must* be provided for each field. The location and format of fields shall*shall* be the same for every row. Fields may*may* be empty, if the repeat count specified in the value of the TFORMnTFORM*n* keyword of the header is 0. 0. Writers of binary tables should*should* select a format appropriate to the form, range of values, and accuracy of the data in the table. The following data types, and no others, are permitted.

Logical. If the value of the TFORMnkeyword specifies data type LTFORM*n* keyword specifies Data Type 'L', the contents of

DRAFT

field nshallField *n shall* consist of ASCII TT indicating true or ASCII FF, indicating false. A 0 byte (hexadecimal 00) indicates a NULL value.

Bit array. If the value of the TFORMnTFORM*n* keyword specifies data type X`'X'`, the contents of field nshallField *n shall* consist of a sequence of bits starting with the most significant most-significant bit; the bits following shall*shall* be in order of decreasing significance, ending with the least significant bit. A bit array shall*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 TFORMnkeyword specifies data type A, field nshallTFORM*n* keyword specifies Data Type `'A'`, Field *n shall* contain a character string of zero or more zero-or-more members, composed of the restricted set of ASCII text ASCII-text characters. This character string may*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 TFORMnkeyword specifies data type BTFORM*n* keyword specifies Data Type `'B'`, the data in field nshallField *n shall* consist of unsigned 8-bit eight-bit integers, withthe most significant the most-significant bit first, and subsequent bits in order of decreasing significance. Null values are given by the value of the associated TNULLnTNULL*n* keyword. Signed integers can be represented using the convention described in Sect. 5.2.5.

16-Bit integer. If the value of the TFORMnkeyword specifies data type ITFORM*n* keyword specifies Data Type `'I'`, the data in field nshallField *n shall* consist of two's complement signed 16-bit integers, contained in two bytes. The most significant byte shallmost-significant byte *shall* be first (big-endian byte order). Within each byte the most significant bit shallmost-significant bit *shall* be first, and subsequent bits shall*shall* be in order of decreasing significance. Null values are given by the value of the associated TNULLnTNULL*n* keyword. Unsigned integers can be represented using the convention described in Sect. 5.2.5.

32-Bit integer. If the value of the TFORMnkeyword specifies data type JTFORM*n* keyword specifies Data Type `'J'`, the data in field nshallField *n shall* consist of two's complement signed 32-bit integers, contained in four bytes. The most significant byte shallmost-significant byte *shall* be first, and subsequent bytes shall*shall* be in order of decreasing significance (big-endian byte order). Within each byte, the most significant bit shallmost-significant bit *shall* be first, and subsequent bits shall*shall* be in order of decreasing significance. Null values are given by the value of the associated TNULLnTNULL*n* keyword. Unsigned integers can be represented using the convention described in Sect. 5.2.5.

64-Bit integer. If the value of the TFORMnkeyword specifies data type KTFORM*n* keyword specifies Data Type `'K'`, the data in field nshallField *n shall* consist of two's complement signed 64-bit integers, contained in eight bytes. The most significant byte shallmost-significant byte *shall* be first, and subsequent bytes shall*shall* be in order of decreasing significance. Within each byte, the most significant bit shallmost-significant bit *shall* be first, and subsequent bits shall*shall* be in order of decreasing significance (big-endian byte order). Null values are given by the value of the associated TNULLnTNULL*n* keyword. Unsigned integers can be represented using the convention described in Sect. 5.2.5.

Single precision Single-precision floating point. If the value of the TFORMnkeyword specifies data type ETFORM*n* keyword specifies Data Type `'E'`, the data in field nshallField *n shall* consist of ANSI/IEEE-754 (IEEE 1985) 32-bit floating-point numbers, in big-endian byte order, as described in Appendix E. All IEEE special values are recognized. The IEEE NaN is used to represent null values.

Double precision Double-precision floating point. If the value of the TFORMnkeyword specifies data type DTFORM*n* keyword specifies Data Type `'D'`, the data in field nshallField *n shall* consist of ANSI/IEEE-754 (IEEE 1985) 64-bit double precision double-precision floating-point numbers, in big-endian byte order, as described in Appendix E. All IEEE special values are recognized. The IEEE NaN is used to represent null values.

Single precision complex. If the value of the TFORMnkeyword specifies data type CTFORM*n* keyword specifies Data Type `'C'`, the data in field nshallField *n shall* consist of a sequence of pairs of 32-bit single precision single-precision floating-point numbers. The first member of each pair shall*shall* represent the real part of a complex number, and the second member shall*shall* represent the imaginary part of that complex number. If either member contains an IEEE NaN, the entire complex value is null.

Double precision Double-precision complex. If the value of the TFORMnkeyword specifies data type MTFORM*n* keyword specifies Data Type `'M'`, the data in field nshallField *n shall* consist of a sequence of pairs of 64-bit double precision double-precision floating-point numbers. The first member of each pair shall*shall* represent the real part of a complex number, and the second member of the pair shall*shall* represent the imaginary part of that complex number. If either member contains an IEEE NaN, the entire complex value is null.

Array descriptor. The repeat count on the Pand Qarray descriptor fields must P and Q array-descriptor fields *must* either have a value of 0 `0` (denoting an empty field of zero bytes) or 1. 1. If the value of the TFORMnkeyword specifies data type 1PTFORM*n* keyword specifies Data Type `'1P'`, the data in field nshallField *n shall* consist of one pair of 32-bit integers. If the value of the TFORMnkeyword specifies data type 1QTFORM*n* keyword specifies Data Type `'1Q'`, the data in field nshallField *n shall* consist of one pair of 64-bit integers. The meaning of these integers is defined in Sect. 7.3.5.

DRAFT

7.3.3.2. Bytes following main table

The main data table may*may* be followed by a supplemental data area called the heap. The size of the supplemental data area, in bytes, is specified by the value of the PCOUNTPCOUNT keyword. The use of this data area is described in Sect. 7.3.5.

7.3.4. Data display

The indexed TDISPnkeyword mayTDISP*n* keyword *may* be used to describe the recommended format for displaying an ASCII text ASCII-text representation of the contents of field nField *n*. The permitted display format codes for each type of data (i.e., character strings, logical, integer, or real) are given in Table 20 and described below.

Character data. If the table column contains a character string (with TFORMn TFORM*n* ='rA' 'rA') then the TDISPnformat code mustbe 'Aw'where w TDISP*n* format code *must* be Aw, where w is the number of characters to display. If the character datum has length less than or equal to ww, it is represented on output right-justified in a string of ww characters. If the character datum has length greater than ww, the first ww characters of the datum are represented on output in a string of ww characters. Character data are not surrounded by single or double quotation single- or double-quotation marks unless those marks are themselves part of the data value.

Logical data. If the table column contains logical data (with TFORMn TFORM*n* ='rL' 'rL') then the TDISPnformat code mustbe 'Lw'where w TDISP*n* format code *must* be Lw, where w is the width in characters of the display field. Logical data are represented on output with the character TT for true or FF for false right-justified in a space-filled string of ww characters. A null value may*may* be represented by a string of ww space characters.

Integer data. If the table column contains integer data (with TFORMn TFORM*n* ='rX', 'rB', 'rI', 'rJ', or 'rK' 'rX', 'rB', 'rI', 'rJ', or 'rK') then the TDISPnformat code mayTDISP*n* format code *may* have any of these forms: Iw.m, Bw.m, Ow.m, or Zw.mIw.m, Bw.m, Ow.m, or Zw.m. The default value of mm is one and the '.m'is optional'.m' is *optional*. The first letter of the code specifies the number base for the encoding with II for decimal (10), BB for binary (2), OO for octal (8), and ZZ 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 wcharacters containing zero or more w characters containing zero-or-more leading spaces followed by a minus sign if the internal datum is negative (only in the case of decimal encoding with the II format code), followed by the magnitude of the internal datum in the form of an unsigned integer unsigned-integer constant in the specified number base, with only as many leading zeros as are needed to have at least mm numeric digits. Note that mm ≤ ww is allowed if all values are positive, but mm < wis required w is *required* if any values are negative. If the number of digits required to represent the integer datum exceeds ww, then the output field consists of a string of wasterisk (*w asterisk (*) characters.

Real data. If the table column contains real data (with TFORMn TFORM*n* ='rE', or 'rD' 'rE', or 'rD') or contains integer data (with any of the TFORMnTFORM*n* format codes listed in the previous paragraph)which are recommended , which are *recommended* to be displayed as real values (i.e., especially in cases where the integer values represent scaled physical values using Eq. 7), then the TDISPnformat code mayTDISP*n* format code *may* have any of these forms: Fw.d, Ew.dEe, Dw.dEe, ENw.d, or ESw.dFw.d, Ew.dEe, Dw.dEe, ENw.d, or ESw.d. In all cases, the output is a string of ww 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 ww, then the output field consists of a string of wasterisk (*w asterisk (*) characters. In all cases, dd specifies the number of digits to appear to the right of the decimal point.

The FF format code output field consists of w − d − 1 characters containing zero or more leading spacesw − d − 1 characters containing zero-or-more leading spaces, followed by a minus sign if the internal datum is negative, followed by the absolute magnitude of the internal datum in the form of an unsigned integer unsigned-integer constant. These characters are followed by a decimal point ('.') and d'.') and d characters giving the fractional part of the internal datum, rounded by the normal rules of arithmetic to dd fractional digits.

For the Eand DE and D format codes, an exponent is taken such that the fraction 0.1 ≤ |datum|/$10^{exponent}$ < 1.00.1 ≤ |datum|/$10^{exponent}$ < 1.0. The fraction (with appropriate sign) is output with an FF format of width w − e − 2 characters with dw − e − 2 characters with d characters after the decimal followed by an Eor DE or D followed by the exponent as a signed e + 1 e + 1 character integer with leading zeros as needed. The default value of eis 2 when the Eee is 2 when the Ee portion of the format code is omitted. If the exponent value will not fit in e+1 e+1 characters but will fit in e+2 then the E(or De+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 e + 2 characters, then the entire ww-character output field is filled with asterisks (**).

The ESES format code is processed in the same manner as the EE format code except that the exponent is taken so that 1.0 ≤ fraction < 101.0 ≤ fraction < 10.

The ENEN format code is processed in the same manner as the EE format code except that the exponent is taken to be an integer multiple of three and so that 1.0 ≤ fraction < 1000.01.0 ≤ fraction < 1000.0. All real format codes have number base 10. There is no difference between Eand DE and D format codes on input other than an implication with the latter of greater precision in the internal datum.

The Gw.dEeformat code mayGw.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 AwIw, Lw, or Aw, respectively. For data of type real, it is equivalent to an FF format (with different numbers of characters after the decimal) when that format will accurately represent the value and is equivalent to an EE format when the number (in absolute value) is either very small or very large. Specifically, for real values outside the range 0.1 − 0.5×10^{-d-1} ≤ value < 10^{d} − 0.50.1 − 0.5×10^{-d-1} ≤ value < 10^{d} − 0.5, it is equivalent to Ew.dEeEw.dEe. For real values within the above range, it is equivalent to Fw′.d′ followed by 2+e Fw′.d′ followed by 2+e spaces, where w′ = w−e−2 and d′ = d−*k*

for $k = 0, 1, \ldots, \mathtt{d}$ $\mathtt{w}' = \mathtt{w} - \mathtt{e} - 2$ and $\mathtt{d}' = \mathtt{d} - k$ for $k = 0, 1, \ldots, \mathtt{d}$ if the real datum value lies in the range $10^{k-1}\left(1 - 0.5\times10^{-\mathtt{d}}\right) \leq$ value $\leq 10^{k}\left(1 - 0.5\times10^{-\mathtt{d}}\right)10^{k-1}\left(1 - 0.5\times10^{-\mathtt{d}}\right) \leq$ value $\leq 10^{k}\left(1 - 0.5\times10^{-\mathtt{d}}\right)$.

Complex data. If the table column contains complex data (with TFORMn `TFORM`*n* ='rC', or 'rM') then the may `'rC'`, or `'rM'`) then they *may* be displayed with any of the real data formats as described above. The same format is used for the real and imaginary parts. It is recommended*recommended* that the two values be separated by a comma and enclosed in parentheses with a total field width of $2\mathtt{w} + 3$2$\mathtt{w} + 3$.

7.3.5. 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 row 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 field, but if the stored array length varies for different rows, 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 binary-table mechanism becomes seriously inefficient. If the limit is set too low then storing certain types of data in the table could become impossible.

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 that does not understand variable-length arrays will still be able to read the main data table (in other words a table containing variable-length arrays conforms to the basic binary table binary-table standard). The implementation chosen is such that the rows in the main data table remain fixed in size even if the table contains a variable-length array field, allowing efficient random access to the main data 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 rows, and 2) the array data are not stored directly in the main data table. Since a field of any data type can be a static array, a field of any data type can also be a variable-length array (excluding the type Pand QType `P` and `Q` variable-length array descriptors themselves, which are not a data type so much as a storage-class specifier). Other established FITS*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 one of the following two special field data type specifiers data-type specifiers

$rPt(e_{\max})$$r\mathtt{P}t(e_{\max})$

$rQt(e_{\max})$$r\mathtt{Q}t(e_{\max})$

where the 'P'or 'Q'`'P'` or `'Q'` indicates the presence of an array descriptor (described below), the element count rshouldbe 0, 1*r should* be `0`, `1`, or absent, t*t* is a character denoting the data type of the array data (L, X, B, I, J, KL, `X`, `B`, `I`, `J`, `K`, etc., but not Por QP or `Q`), and $\mathtt{e}_{\max}$ $e_{\max}$ is a quantity guaranteed to be equal to or greater than the maximum number of elements of type t*t* actually stored in any row of the table. 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); $\mathtt{e}_{\max}$ $e_{\max}$ 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*may* be additional characters in the TFORMn`TFORM`*n* keyword following the $\mathtt{e}_{\max}$$e_{\max}$.

For example,

`TFORM8 = 'PB(1800)' / Variable byte arrayTFORM8 = 'PB(1`

indicates that fieldField 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 main data table; they are stored in a supplemental data area, the heap, following the main data table. What is stored in the main data table field is an *array descriptor*. This consists of two 32-bit signed integer values in the case of 'P'`'P'` array descriptors, or two 64-bit signed integer values in the case of 'Q'`'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 standardStandard. Storage for the array is contiguous. The array descriptor for field Field *N* as it would appear embedded in a table row is illustrated symbolically below: .

. . . [field Field *N*–1] [(nelem,offset)] [field Field *N*+1] . . .

If the stored array length is zero, there is no array data, and the offset value is undefined (it should*should* be set to zero). The storage referenced by an array descriptor must*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] [main data table] (optional gap) [heap area]

The table header consists of one or more 2880-byte header blocks with the last block indicated by the keyword END`END` somewhere in the block. The main data table begins with the first data block following the last header block and is `NAXIS1` × `NAXIS2` `NAXIS1` × `NAXIS2` bytes in length. The zeroindexed -indexed byte offset to the start of the heap, measured from the start of the main data table, may *may* be given by the THEAP`THEAP` keyword in the header. If this keyword is missing then the heap begins with the byte immediately following main data table (i.e., the default value of THEAPis `NAXIS1` × `NAXIS2``THEAP` is `NAXIS1` × `NAXIS2`). This default value is the minimum allowed value for the THEAP`THEAP` keyword, because any smaller value would imply that the heap and the main data table overlap. If the THEAP`THEAP` keyword has a value larger than this default value, then there is a gap between the end of the main data table and the start of the heap. The total length in bytes of the supplemental data area following the main data table (gap plus heap) is given by the PCOUNT`PCOUNT` keyword in the table header.

For example, suppose a table contains five rows which that are each 168 bytes long, with a heap area 3000 bytes long, beginning at an offset of 2880, thereby aligning the main data table

DRAFT

and heap areas on data block boundaries (this alignment is not necessarily recommended but is useful for this example). The data portion of the table consists of three 2880-byte data blocks: the first block contains the 840 bytes from the five rows of the main data table followed by 2040 fill bytes; the heap completely fills the second block; the third block contains the remaining 120 bytes of the heap followed by 2760 fill bytes. PCOUNTPCOUNT gives the total number of bytes from the end of the main data 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 : shown below.

```
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 TSCALnand TZEROnTSCAL*n* and TZERO*n* 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.

7.3.6. Variable-length-array guidelines

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 might also be useful.

Programs that read binary tables should take care to not assume more about the physical layout of the table than is required *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 might 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*may* be stored in the heap in any order, i.e., the data for row N+1 are not necessarily stored at a larger offset than that for row N. There may*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*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*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*may* be zero. This makes it possible to have a column of the table for which, typically, no data are present in each stored row. When data are present, the stored array can be as large as necessary. This can be useful when storing complex objects as rows in a table.

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

Reading a binary table stored on a sequential access sequential-access storage medium requires that a table of array descriptors be built up as the main data table rows are read in. Once all the table rows 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 rows using the back pointers to the row and field from the table of array descriptors. Since array aliasing is permitted, it might be necessary to store a given array in more than one field or row.

Variable-length arrays are more complicated than regular static arrays and might not be supported by some software systems. The producers of FITS*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 might present difficulties for applications that ingest the FITS*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.

8. World coordinate World-coordinate systems

Representations of the mapping between image coordinates and physical (i.e., world) coordinate systems (WCSs) may *may* be represented within FITS*FITS* HDUs. The keywords that are used to express these mappings are now rigorously defined in a series of papers on world coordinate world-coordinate systems (Greisen & Calabretta 2002), celestial coordinate celestial-coordinate systems (Calabretta & Greisen 2002), spectral coordinate spectral-coordinate systems (Greisen et al. 2006), and time coordinate time-coordinate systems (Rots et al. 2015). An additional spherical projection, called HEALPix, is defined in reference (Calabretta & Roukema 2007). These WCS papers have been formally approved by the IAUFWG and therefore are *incorporated by reference* as an official part of this Standard. The reader should refer to these papers for additional details and background information that cannot be included here. Various updates and corrections to the primary WCS papers have been compiled by the authors, and are reflected in this section. Therefore, where conflicts exist, the description in this Standard will prevail.

8.1. Basic concepts

Rather than store world coordinates separately for each datum, the regular lattice structure of a FITS*FITS* image offers the possibility of defining rules for computing world coordinates at each point. As stated in Sect. 3.3.2 and depicted in Fig. 1, image array data are addressed via *integral array indices* that range in value from 1 to NAXIS*j* on axis Axis *j*. Recognizing that image data values may have an extent, for example an angular separation, spectral channel width or time span, and thus that it may make sense to interpolate between them, these integral array

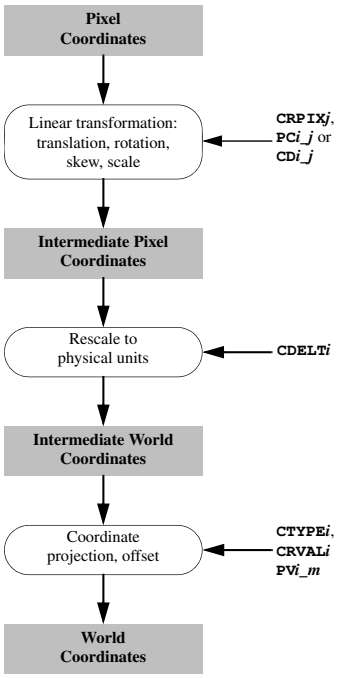


Fig. 2: A schematic view of converting pixel coordinates to world coordinates.

indices may *may* be generalized to floating-point *pixel coordinates*. Integral pixel coordinate pixel-coordinate values coincide with the corresponding array indices, while fractional pixel coordinate pixel-coordinate values lie between array indices and thus imply interpolation. Pixel coordinate Pixel-coordinate values are defined at all points within the image lattice and outside it (except along *conventional* axes, see Sect. 8.5). They form the basis of the world coordinate formalism in FITSworld-coordinate formalism in *FITS* depicted schematically in Fig. 2.

The essence of representing world coordinate systems in FITSworld-coordinate systems in *FITS* is the association of various reserved keywords with elements of a transformation (or a series of transformations), or with parameters of a projection function. The conversion from pixel coordinates in the data array to world coordinates is simply a matter of applying the specified transformations (in order) via the appropriate keyword values; conversely, defining a WCS for an image amounts to solving for the elements of the transformation matrix(es) or coefficients of the function(s) of interest and recording them in the form of WCS keyword values. The description of the WCS systems and their expression in FITS*FITS* HDUs is quite extensive and detailed, but is aided by a careful choice of notation. Key elements of the notation are summarized in Table 21, and are used throughout this section. The formal definitions of the keywords appear in the following subsections.

The conversion of image pixel coordinates to world coordinates is a multi-step process, as illustrated in Fig. 2.

For all coordinate types, the first step is a linear transformation applied via matrix multiplication of the vector of pixel coordinate pixel-coordinate elements, p_j:

$$q_i = \sum_{j=1}^{N} m_{ij}(p_j - r_j) \tag{9}$$

where r_j are the pixel coordinate pixel-coordinate elements of the reference point, j indexes the pixel axis, and i the world axis. The m_{ij} matrix is a non-singular, square matrix of dimension $N \times N$, where N is the number of world coordinate world-coordinate axes. The elements q_i of the resulting *intermediate pixel coordinate* vector are offsets, in dimensionless pixel units, from the reference point along axes coincident with those of the *intermediate world coordinates*. Thus, the conversion of q_i to the corresponding intermediate world coordinate element Intermediate-world-coordinate Element x_i is a simple scale:

$$x_i = s_i q_i. \tag{10}$$

There are three conventions for associating FITS*FITS* keywords with the above transformations. In the first formalism, the matrix elements m_{ij} are encoded in the PC*i_j* keywords and the scale factors s_i are encoded in the CDELT*i* keywords, which must*must* have non-zero values. In the second formalism Eqs. (9) and (10) are combined as

$$x_i = \sum_{j=1}^{N} (s_i m_{ij})(p_j - r_j) \tag{11}$$

and the CD*i_j* keywords encode the product $s_i m_{ij}$. The third convention was widely used before the development of the two previously described conventions and uses the CDELT*i* keywords to define the image scale and the CROTA2keyword to define CROTA2 keyword to specify a bulk rotation of the image plane. Use of the CROTA2CROTA2 keyword is now deprecated, and instead the newer PC*i_j* or CD*i_j* keywords are recommended*recommended* because they allow for skewed axes and fully general rotation of multi-dimensional arrays. The CDELT*i* and CROTA2keywords mayCROTA2 keywords *may* coexist with the CD*i_j* keywords (but the CROTA2must notCROTA2 *must not* occur with the PC*i_j* keywords) as an aid to old FITS*FITS* interpreters, but these keywords must*must* be ignored by software that supports the CD*i_j* keyword convention. In all these formalisms the reference pixel coordinates r_j are encoded in the CRPIX*i* keywords, and the world coordinates at the reference point are encoded in the CRVAL*i* keywords. For additional details, see Greisen & Calabretta (2002).

The third step of the process, computing the final world coordinates, depends on the type of coordinate system, which is indicated with the value of the CTYPE*i* keyword. For some simple, linear cases an appropriate choice of normalization for the scale factors allows the world coordinates to be taken directly (or by applying a constant offset) from the x_i (e.g., some spectra). In other cases it is more complicated, and may require the application of some non-linear algorithm (e.g., a projection, as for celestial coordinates), which may require the specification of additional parameters. Where necessary, numeric parameter values for non-linear algorithms must*must* be specified via PV*i_m* keywords and character-valued parameters will be specified via PS*i_m* keywords, where m is the parameter number.

The application of these formalisms to coordinate systems of interest is discussed in the following sub-sections: Sect. 8.2 describes general WCS representations (see Greisen & Calabretta 2002), Sect. 8.3 describes celestial coordinate 8.3 describes celestial-coordinate systems (see Calabretta & Greisen 2002)), Sect. 8.4 describes spectral coordinate 8.4 describes spectral-coordinate systems (see Greisen et al. 2006), and Sect. 9 describes the representation of time coordinates (see Rots et al. 2015).

DRAFT

Table 21: WCS and celestial coordinates notation.

Variable(s)	Meaning	Related FITS *FITS* keywords
i	Index variable for world coordinates	
j	Index variable for pixel coordinates	
a	Alternative WCS version code	
p_j	Pixel coordinates	
r_j	Reference pixel coordinates	CRPIX*ja*
m_{ij}	Linear transformation Linear-transformation matrix	CD*i_ja* or PC*i_ja*
s_i	Coordinate scales	CDELT*ia*
(x, y)	Projection plane coordinates	
(ϕ, θ)	Native longitude and latitude	
(α, δ)	Celestial longitude and latitude	
(ϕ_0, θ_0)	Native longitude and latitude of the fiducial point	PV*i*_1*a*† , PV*i*_2*a*†
(α_0, δ_0)	Celestial longitude and latitude of the fiducial point	CRVAL*ia*
(α_p, δ_p)	Celestial longitude and latitude of the native pole	
(ϕ_p, θ_p)	Native longitude and latitude of the celestial pole	LONPOLE*a* (=PV*i*_3*a*†), LATPOLE*a* (=PV*i*_4*a*†)

Notes. † Associated with *Longitude* Axis *i*.

8.2. World coordinate system World-coordinate-system representations

A variety of keywords have been reserved for computing the coordinate values that are to be associated with any pixel location within an array. The full set is given in Table 22; those in most common usage are defined in detail below for convenience. Coordinate system specifications may Coordinate-system specifications *may* appear in HDUs that contain simple images in the primary array or in an image IMAGE extension. Images may *may* also be stored in a multi-dimensional vector cell of a binary table, or as a tabulated list of pixel locations (and optionally, the pixel value) in a table. In these last two types of image representations, the WCS keywords have a different naming convention, which reflects the needs of the tabular data structure and the 8-character eight-character limit for keyword lengths, but otherwise follow exactly the same rules for type, usage, and default values. See reference Calabretta & Greisen (2002) for example usage of these keywords. All forms of these reserved keywords must*must* be used only as specified in this Standard.

In the case of the binary table binary-table vector representation, it is possible that the images contained in a given column of the table have different coordinate transformation values. Table 9 of Calabretta & Greisen (2002) illustrates a technique (commonly known as the "Green Bank Convention[10]"), which utilizes additional columns in the table to record the coordinate transformation coordinate-transformation values that apply to the corresponding image in each row of the table. More information is provided in Appendix L.

The keywords given below constitute a complete set of fundamental attributes for a WCS description. Although their inclusion in an HDU is optional, FITSwriters should*FITS* writers *should* include a complete set of keywords when describing a WCS. In the event that some keywords are missing, default values must*must* be assumed, as specified below.

WCSAXES – [integer; default: NAXIS, or larger of WCS indexes indices *i* or *j*]. Number of axes in the WCS description. This keyword, if present, must*must* precede all WCS keywords except NAXIS in the HDU. The value of WCSAXES may*may* exceed the number of pixel axes for the HDU.

CTYPE*i* – [string; indexed; default: '␣' (i.e. a linear, undefined axis)]. Type for the intermediate coordinate axis Intermediate-coordinate Axis *i*. Any coordinate type that is not covered by this standard Standard or an officially recognized FITSconvention shall*FITS* convention *shall* be taken to be linear. All non-linear coordinate system names must*must* be expressed in '4–3' form: the first four characters specify the coordinate type, the fifth character is a hyphen ('-'), and the remaining three characters specify an algorithm code for computing the world coordinate value. Coordinate types with names of less fewer than four characters are padded on the right with hyphens, and algorithm codes with less fewer than three characters are padded on the right with blanks[11]. Algorithm codes should*should* be three characters.

CUNIT*i* – [string; indexed; default: '␣' (i.e., undefined)]. Physical units of CRVAL and CDELT for axis iAxis *i*. Note that units should*should* always be specified (see Sect. 4.3). Units for celestial coordinate systems defined in this Standard must*must* be degrees.

CRPIX*j* – [floating point; indexed; default: 0.0**0.0**]. Location of the reference point in the image for axis Axis *j* corresponding to r_j in Eq. (9). Note that the reference point may*may* lie outside the image and that the first pixel in the image has pixel coordinates (1.0, 1.0, . . .).

CRVAL*i* – [floating point; indexed; default: 0.0**0.0**]. World Coordinate World-coordinate value at the reference point of axis Axis *i*.

CDELT*i* – [floating point; indexed; default: 1.0**1.0**]. Increment of the world coordinate at the reference point for axis Axis *i*. The value must not*must not* be zero.

CROTA*i* – [floating point; indexed; default: 0.0**0.0**]. The amount of rotation from the standard coordinate system to a different coordinate system. Further use of this of this keyword is deprecated, in favor of the newer formalisms that use the CD*i_j* or PC*i_j* keywords to define the rotation.

PC*i_j* – [floating point; defaults: 1.0 **1.0** when $i = j$, 0.0 **0.0** otherwise]. Linear transformation matrix between pixel axes Pixel Axes *j* and intermediate coordinate axes Intermediate-

[10] Named after a meeting held in Green Bank, West Virginia, USA in 1989 to develop standards for the interchange of single dish radio astronomy single-dish radio-astronomy data.

[11] Example: 'RA---UV ''RA---UV '.

DRAFT

Table 22: Reserved WCS keywords (continues on next page)

Keyword Description	Global	Image	BINTABLE vector Primary	BINTABLE vector Alternative	Pixel list Primary	Pixel list Alternative
Coordinate dimensionality		WCSAXES*a*	WCAX*na*		...	
Axis type		CTYPE*ia*	*i*CTYP*n*	*i*CTY*na*	TCTYP*n*	TCTY*na*
Axis units		CUNIT*ia*	*i*CUNI*n*	*i*CUN*na*	TCUNI*n*	TCUN*na*
Reference value		CRVAL*ia*	*i*CRVL*n*	*i*CRV*na*	TCRVL*n*	TCRV*na*
Coordinate increment		CDELT*ia*	*i*CDLT*n*	*i*CDE*na*	TCDLT*n*	TCDE*na*
Reference point		CRPIX*ja*	*j*CRPX*n*	*j*CRP*na*	TCRPX*n*	TCRP*na*
Coordinate rotation[1]		CROTA*i*	*i*CROT*n*		TCROT*n*	
Transformation matrix[2]		PC*i_ja*	*ij*PC*na*		TPC*n_ka* or TP*n_ka*	
Transformation matrix[2]		CD*i_ja*	*ij*CD*na*		TCD*n_ka* or TC*n_ka*	
Coordinate parameter		PV*i_ma*	*i*PV*n_ma* or *i*V*n_ma*		TPV*n_ma* or TV*n_ma*	
Coordinate parameter array		...	*i*V*n_Xa*		...	
Coordinate parameter		PS*i_ma*	*i*PS*n_ma* or *i*S*n_ma*		TPS*n_ma* or TS*n_ma*	
Coordinate name		WCSNAME*a*	WCSN*na*		WCS*na* or TWCS*na*	
Coordinate axis name		CNAME*ia*	*i*CNA*na*		TCNA*na*	
Random error		CRDER*ia*	*i*CRD*na*		TCRD*na*	
Systematic error		CSYER*ia*	*i*CSY*na*		TCSY*na*	
WCS cross-reference target		...	WCST*na*		...	
WCS cross reference		...	WCSX*na*		...	
Coordinate rotation		LONPOLE*a*	LONP*na*		LONP*na*	
Coordinate rotation		LATPOLE*a*	LATP*na*		LATP*na*	
Coordinate epoch		EQUINOX*a*	EQUI*na*		EQUI*na*	
Coordinate epoch[3]	EPOCH		EPOCH		EPOCH	
Reference frame	RADECSYSRADECSYS[4]	RADESYS*a*	RADE*na*		RADE*na*	
Line rest frequency (Hz)	RESTFREQ[4]	RESTFRQ*a*	RFRQ*na*		RFRQ*na*	
Line rest vacuum wavelength (m)		RESTWAV*a*	RWAV*na*		RWAV*na*	
Spectral reference frame		SPECSYS*a*	SPEC*na*		SPEC*na*	
Spectral reference frame		SSYSOBS*a*	SOBS*na*		SOBS*na*	
Spectral reference frame		SSYSSRC*a*	SSRC*na*		SSRC*na*	
Observation X (m)	OBSGEO-X[5]		OBSGX*n*		OBSGX*n*	
Observation Y (m)	OBSGEO-Y[5]		OBSGY*n*		OBSGY*n*	
Observation Z (m)	OBSGEO-Z[5]		OBSGZ*n*		OBSGZ*n*	
Radial velocity (m s^{-1})		VELOSYS*a*	VSYS*na*		VSYS*na*	
Redshift of source		ZSOURCE*a*	ZSOU*na*		ZSOU*na*	
Angle of true velocity		VELANGL*a*	VANG*na*		VANG*na*	
Date-time related keywords (see Sect.9)						
Date of HDU creation	DATE					
Date/time of observation	DATE-OBS		DOBS*n*		DOBS*n*	
	MJD-OBS		MJDOB*n*		MJDOB*n*	
	BEPOCH					
	JEPOCH					
Average date/time of observation	DATE-AVG		DAVG*n*		DAVG*n*	
	MJD-AVG		MJDA*n*		MJDA*n*	
Start date/time of observation	DATE-BEG					
	MJD-BEG					
	TSTART					
End date/time of observation	DATE-END					
	MJD-END					
	TSTOP					
Net exposure duration	XPOSURE					Wall clock
Wall-clock exposure duration	TELAPSE					
Time scale	TIMESYS	CTYPE*ia*	*i*CTYP*n*	*i*CTY*na*	TCTYP*n*	TCTY*na*
Time zero-point zero point (MJD)	MJDREF[6]					
Time zero-point zero point (JD)	JDREF[6]					
Time zero-point zero point (ISO)	DATEREF					
Reference position	TREFPOS		TRPOS*n*		TRPOS*n*	
Reference direction	TREFDIR		TRDIR*n*		TRDIR*n*	
Solar system System ephemeris	PLEPHEM					
Time unit	TIMEUNIT	CUNIT*ia*	*i*CUNI*n*	*i*CUN*na*	TCUNI*n*	TCUN*na*
Time offset	TIMEOFFS					
Time absolute error	TIMSYER	CSYER*ia*	*i*CSYE*n*	*i*CSY*na*	TCSY*n*	TCSY*na*
Time relative error	TIMRDER	CRDER*ia*	*i*CRDE*n*	*i*CRD*na*	TCRD*n*	TCRD*na*
Time resolution	TIMEDEL					
Time location in pixel	TIMEPIXR					
Phase axis Phase-axis zero point		CZPHS*ia*	*i*CZPH*n*	*i*CZP*na*	TCZPH*n*	TCZP*na*
Phase axis Phase-axis period		CPERI*ia*	*i*CPER*n*	*i*CPR*na*	TCPER*n*	TCPR*na*

DRAFT

Table 22 (continued)

Notes. The indices *j* and *i* are pixel and intermediate-world-coordinate axis numbers, respectively. Within a table, the index *n* refers to a column number, and *m* refers to a coordinate parameter number. The index *k* also refers to a column number. The indicator *a* is either blank (for the primary coordinate description) or a character A through Z that specifies the coordinate version. See the text.
(1) CROTA*i* form is deprecated but still in use. It *must not* be used with PC*i*_*j*, PV*i*_*m*, and PS*i*_*m*. (2) PC*i*_*j* and CD*i*_*j* forms of the transformation matrix are mutually exclusive, and *must not* appear together in the same HDU. (3) EPOCH is deprecated. Use EQUINOX instead. (4) These eight-character keywords are deprecated; the seven-character forms, which can include an alternate version code letter at the end, *should* be used instead. (5) For the purpose of time reference position, geodetic latitude/longitude/elevation OBSGEO-B, OBSGEO-L, OBSGEO-H or an orbital-ephemeris keyword OBSORBIT can be also used (see Sect. 9.2.3). (6) [M]JDREF can be split in integer and fractional values [M]JDREFI and [M]JDREFF as explained in Sect. 9.2.2.

coordinate Axes *i*. The PC*i*_*j* matrix must not*must not* be singular.

CD*i*_*j* – [floating point; defaults: 0.0**0.0**, but see below]. Linear transformation matrix (with scale) between pixel axes Pixel Axes *j* and intermediate coordinate axes Intermediate-coordinate Axes *i*. This nomenclature is equivalent to PC*i*_*j* when CDELT*i* is unity. The CD*i*_*j* matrix must not*must not* be singular. Note that the CD*i*_*j* formalism is an exclusive alternative to PC*i*_*j*, and the CD*i*_*j* and PC*i*_*j* keywords must not*must not* appear together within an HDU.

In addition to the restrictions noted above, if any CD*i*_*j* keywords are present in the HDU, all other unspecified CD*i*_*j* keywords shall*shall* default to zero. If no CD*i*_*j* keywords are present then the header shall*shall* be interpreted as being in PC*i*_*j* form whether or not any PC*i*_*j* keywords are actually present in the HDU.

Some non-linear algorithms that describe the transformation between pixel and intermediate coordinate intermediate-coordinate axes require parameter values. A few non-linear algorithms also require character-valued parameters, e.g., table lookups require the names of the table extension and the columns to be used. Where necessary parameter values must*must* be specified via the following keywords: .

PV*i*_*m* – [floating point]. Numeric parameter values for intermediate world coordinate axis Intermediate-world-coordinate Axis *i*, where *m* is the parameter number. Leading zeros must not*must not* be used, and *m* may *may* have only values in the range 0 through 99, and that are defined for the particular non-linear algorithm.

PS*i*_*m* – [string]. Character-valued parameters for intermediate world coordinate axis Intermediate-world-coordinate Axis *i*, where *m* is the parameter number. Leading zeros must not*must not* be used, and *m* may *may* have only values in the range 0 through 99, and that are defined for the particular non-linear algorithm.

The following keywords, while not essential for a complete specification of an image WCS, can be extremely useful for readers to interpret the accuracy of the WCS representation of the image.

CRDER*i* – [floating point; default: 0.0**0.0**]. Random error in coordinate Coordinate *i*, which must*must* be non-negative.

CSYER*i* – [floating point; default: 0.0**0.0**]. Systematic error in coordinate Coordinate *i*, which must*must* be non-negative.

These values should*should* give a representative average value of the error over the range of the coordinate in the HDU. The total error in the coordinates would be given by summing the individual errors in quadrature.

8.2.1. Alternative WCS axis descriptions

In some cases it is useful to describe an image with more than one coordinate type[12]. Alternative WCS descriptions may*may* be added to the header by adding the appropriate sets of WCS keywords, and appending to all keywords in each set an alphabetic code in the range Athrough ZA through Z. Keywords that may be used in this way to specify a coordinate system version are indicated in Table 22 with the suffix *a*. All implied keywords with this encoding are *reserved keywords*, and mustonly *must only* be used in FITS*FITS* HDUs as specified in this Standard. The axis numbers must*must* lie in the range 1 through 99, and the coordinate parameter *m* must*must* lie in the range 0 through 99, both with no leading zeros.

The *primary* version of the WCS description is that specified with a*a* as the blank character[13]. Alternative axis descriptions are optional, but must not*must not* be specified unless the primary WCS description is also specified. If an alternative WCS description is specified, all coordinate keywords for that version must*must* be given even if the values do not differ from those of the primary version. Rules for the default values of alternative coordinate descriptions are the same as those for the primary description. The alternative descriptions are computed in the same fashion as the primary coordinates. The type of coordinate depends on the value of CTYPE*ia*, and may be linear in one of the alternative descriptions and non-linear in another.

The alternative version codes are selected by the FITS*FITS* writer; there is no requirement that the codes be used in alphabetic sequence, nor that one coordinate version differ in its parameter values from another. An optional keyword WCSNAME*a* is also defined to name, and otherwise document, the various versions of WCS descriptions: .

WCSNAME*a* – [string; default for a*a*: '␣' (i.e., blank, for the primary WCS, else a character Athrough ZA through Z that specifies the coordinate version]. Name of the world coordinate world-coordinate system represented by the WCS keywords with the suffix a*a*. Its primary function is to provide a means by which to specify a particular WCS if multiple versions are defined in the HDU.

[12] Examples include the frequency, velocity, and wavelength along a spectral axis (only one of which, of course, could be linear), or the position along an imaging detector in both meters and degrees on the sky.

[13] There are a number of keywords (e.g. *ij*PC*na*) where the a*a* could be pushed off the 8-char eight-character keyword name for plausible values of i*i*, j*j*, k*k*, n*n*, and m*m*. In such cases a*a* is still said to be 'blank' although it is not the blank character.

DRAFT

Table 23: Reserved celestial coordinate algorithm celestial-coordinate-algorithm codes.

Code	Default ϕ_0	Default θ_0	Properties[1]	Projection name
			Zenithal (azimuthal) projections	
AZP	0°	90°	Sect. 5.1.1	Zenithal perspective
SZP	0°	90°	Sect. 5.1.2	Slant zenithal perspective
TAN	0°	90°	Sect. 5.1.3	Gnomonic
STG	0°	90°	Sect. 5.1.4	Stereographic
SIN	0°	90°	Sect. 5.1.5	Slant orthographic
ARC	0°	90°	Sect. 5.1.6	Zenithal equidistant
ZPN	0°	90°	Sect. 5.1.7	Zenithal polynomial
ZEA	0°	90°	Sect. 5.1.8	Zenithal equal-area
AIR	0°	90°	Sect. 5.1.9	Airy
			Cylindrical projections	
CYP	0°	0°	Sect. 5.2.1 .	Cylindrical perspective
CEA	0°	0°	Sect. 5.2.2	Cylindrical equal area
CAR	0°	0°	Sect. 5.2.3	Plate carrée
MER	0°	0°	Sect. 5.2.4	Mercator
			Pseudo-cylindrical and related projections	
SFL	0°	0°	Sect. 5.3.1	Samson-Flamsteed
PAR	0°	0°	Sect. 5.3.2	Parabolic
MOL	0°	0°	Sect. 5.3.3	Mollweide
AIT	0°	0°	Sect. 5.3.4	Hammer-Aitoff
			Conic projections	
COP	0°	θ_a	Sect. 5.4.1	Conic perspective
COE	0°	θ_a	Sect. 5.4.2	Conic equal-area
COD	0°	θ_a	Sect. 5.4.3	Conic equidistant
COO	0°	θ_a	Sect. 5.4.4	Conic orthomorphic
			Polyconic and pseudoconic projections	
BON	0°	0°	Sect. 5.5.1	Bonne's equal area
PCO	0°	0°	Sect. 5.5.2	Polyconic
			Quad-cube projections	
TSC	0°	0°	Sect. 5.6.1	Tangential spherical cube
CSC	0°	0°	Sect. 5.6.2	COBE quadrilateralized spherical cube
QSC	0°	0°	Sect. 5.6.3	Quadrilateralized spherical cube
			HEALPix grid projection	
HPX	0°	0°	Sect. 6[2]	HEALPix grid

[1] Refer to the indicated section in Calabretta & Greisen (2002) for a detailed description. [2] This projection is defined in Calabretta & Roukema (2007).

8.3. Celestial coordinate system Celestial-coordinate-system representations

The conversion from intermediate world coordinates (x, y) in the plane of projection to celestial coordinates involves two steps: a spherical projection to native longitude and latitude (ϕ, θ), defined in terms of a convenient coordinate system (i.e., *native spherical coordinates*), followed by a spherical rotation of these native coordinates to the required celestial coordinate system (α, δ). The algorithm to be used to define the spherical projection must*must* be encoded in the CTYPE*i* keyword as the three-letter algorithm code, the allowed values for which are specified in Table 23 and defined in references Calabretta & Greisen (2002) and Calabretta & Roukema (2007). The target celestial coordinate celestial-coordinate system is also encoded into the left-most portion of the CTYPE*i* keyword as the coordinate type.

For the final step, the parameter LONPOLE*a* must *must* be specified, which is the native longitude of the celestial pole, ϕ_p. For certain projections (such as cylindricals and conics, which are less commonly used in astronomy), the additional keyword LATPOLE*a* must *must* be used to specify the native latitude of the celestial pole. See Calabretta & Greisen (2002) for the transformation equations and other details.

The accepted celestial coordinate celestial-coordinate systems are: the standard equatorial (RA-- and DEC-RA-- and DEC-), and others of the form *x*LONLON and *x*LATLAT for longitude-latitude pairs, where *x* is GG for Galactic, EE for ecliptic, HH for helioecliptic and SS for supergalactic coordinates. Since the representation of planetary, lunar, and solar coordinate planetary-, lunar-, and solar-coordinate systems could exceed the 26 possibilities afforded by the single character *x*, pairs of the form *yz*LNLN and *yz*LTmayLT *may* be used as well.

RADESYS*a* – [string; default: 'FK4', 'FK5', or 'ICRS''FK4', 'FK5', or 'ICRS': see below]. Name of the reference frame of equatorial or ecliptic coordinates, whose value must*must* be one of those specified in Table 24. The default value is

FK4 'FK4' if the value of EQUINOX*a* < 1984.0, FK5 if 'FK5' if 'EQUINOX'*a* ≥ 1984.0, or ICRS if 'ICRS' if 'EQUINOX'*a* is not given.

EQUINOX*a* – [floating point; default: see below]. Epoch of the mean equator and equinox in years, whose value must*must* be non-negative. The interpretation of epoch depends upon the value of RADESYS*a* if present: *Besselian* if the value is FK4 or FK4-NO-E'FK4' or 'FK4-NO-E', *Julian* if the value is FK5; 'FK5'; and *not applicable* if the value is ICRS or GAPPT'ICRS' or 'GAPPT'.

EPOCH – [floating point]. This keyword is deprecated and should not*should not* be used in new FITS*FITS* files. It is reserved primarily to prevent its use with other meanings. The EQUINOXEQUINOX keyword shall*shall* be used instead. The value field of this keyword was previously defined to contain a floating-point number giving the equinox in years for the celestial coordinate celestial-coordinate system in which positions are expressed.

DATE-OBS – [floating point]. This reserved keyword is defined in Sect. 4.4.2.

MJD-OBS – [floating point; default: DATE-OBS if given, otherwise no default]. Modified Julian Date (JD – – 2,400,000.5) of the observation, whose value corresponds (by default) to the *start* of the observation, unless another interpretation is explained in the comment field. No specific time system (e.g. UTC, TAI, etc.) is defined for this or any of the other time-related keywords. It is recommended*recommended* that the TIMESYS keyword, as defined in Sect. 9.2.1 be used to specify the time system. See also Sect. 9.5.

LONPOLE*a* – [floating point; default: ϕ_0 if $\delta_0 \geq \theta_0$, $\phi_0 + 180°$ otherwise]. Longitude in the native coordinate system of the celestial system's north pole. Normally, ϕ_0 is zero unless a non-zero value has been set for PVi_1aPV*i*_1*a*, which is associated with the *longitude* axis. This default applies for all values of θ_0, including $\theta_0 = 90°$, although the use of non-zero values of θ_0 are discouraged in that case.

LATPOLE*a* – [floating point; default: 90°, or no default if $(\theta_0, \delta_0, \phi_p - \phi_0) = (0, 0, \pm 90°)$]. Latitude in the native coordinate system of the celestial system's north pole, or equivalently, the latitude in the celestial coordinate celestial-coordinate system of the native system's north pole. May This keyword *may* be ignored or omitted in cases where LONPOLE*a* completely specifies the rotation to the target celestial system.

8.4. Spectral coordinate system Spectral-coordinate-system representations

This section discusses the conversion of intermediate world coordinates to spectral coordinates with common axes such as frequency, wavelength, and apparent radial velocity (represented here with the coordinate variables ν, λ, or v). The key point for constructing spectral WCS in FITS*FITS* is that one of these coordinates must*must* be sampled linearly in the dispersion axis; the others are derived from prescribed, usually non-linear transformations. Frequency and wavelength axes may*may* also be sampled linearly in their logarithm.

Following the convention for the CTYPE*ia* keyword, when *i* is the spectral axis the first four characters must*must* specify a code for the coordinate type; for non-linear algorithms the fifth character must*must* be a hyphen, and the next three characters must*must* specify a predefined algorithm for computing the world coordinates from the intermediate physical coordinates. The coordinate type must*must* be one of those specified in Table 25. When the algorithm is linear, the remainder of the CTYPE*ia* keyword must*must* be blank. When the algorithm is non-linear, the 3-letter algorithm code mustthree-letter algorithm code *must* be one of those specified in Table 26. The relationships between the basic physical quantities ν, λ, and v, as well as the relationships between various derived quantities are given in reference Greisen et al. (2006).

Table 24: Allowed values of RADESYS*a*.

Value	Definition
'ICRS'	International Celestial Reference System
'FK5'	Mean place, new (IAU 1984) system
'FK4'[1]	Mean place, old (Bessel-Newcomb) system
'FK4-NO-E'[1]	Mean place: but without eccentricity terms
'GAPPT'	Geocentric apparent place, IAU 1984 system

[(1)] New *FITS* files *should* avoid using these older reference systems.

The generality of the algorithm for specifying the spectral coordinate spectral-coordinate system and its representation suggests that some additional description of the coordinate may be helpful beyond what can be encoded in the first four characters of the CTYPE*ia* keyword; CNAME*ia* is reserved for this purpose. Note that this keyword provides a name for an axis in a particular WCS, while the WCSNAME*a* keyword names the particular WCS as a whole. In order to convert between some form of radial velocity and either frequency or wavelength, the keywords RESTFRQ*a* and RESTWAV*a*, respectively, are reserved.

CNAME*ia* – [string; default: default: '␣' (i.e. a linear, undefined axis)]. Spectral coordinate description which must notSpectral-coordinate description that *must not* exceed 68 characters in length.

RESTFRQ*a* – [floating point; default: none]. Rest frequency of the of the spectral feature of interest. The physical unit must*must* be Hz.

RESTWAV*a* – [floating point; default: none]. Vacuum rest wavelength of the of the spectral feature of interest. The physical unit must*must* be m.

One or the other of RESTFRQ*a* or RESTWAV*a* should*should* be given when it is meaningful to do so.

8.4.1. Spectral coordinate Spectral-coordinate reference frames

Frequencies, wavelengths, and apparent radial velocities are always referred to some selected standard of rest (i.e., reference frame). While the spectra are obtained they are, of necessity, in the observer's rest frame. The velocity correction from topocentric (the frame in which the measurements are usually made) to standard reference frames (which must*must* be one of those given in Table 27) are dependent on the dot product with time-variable velocity vectors. That is, the velocity with respect to a standard reference frame depends upon direction, and the velocity (and frequency and wavelength) with respect to the local standard of rest is a function of the celestial coordinate within the image. The keywords SPECSYS*a* and SSYSOBS*a* are reserved and, if used, must*must* describe the reference frame in use for the spectral axis spectral-axis coordinate(s) and the spec-

DRAFT

Table 25: Reserved spectral coordinate spectral-coordinate type codes.

Code[1]	Type	Symbol	Assoc. variableAssociated variable	Default units
FREQ	Frequency	ν	ν	Hz
ENER	Energy	E	ν	J
WAVN	Wavenumber	κ	ν	m^{-1}
VRAD	Radio velocity[2]	V	ν	m s^{-1}
WAVE	Vacuum wavelength	λ	λ	m
VOPT	Optical velocity[2]	Z	λ	m s^{-1}
ZOPT	Redshift	z	λ	...
AWAV	Air wavelength	λ_a	λ_a	m
VELO	Apparent radial velocity	v	v	m s^{-1}
BETA	Beta factor (v/c)	β	v	...

(1) Characters 1 through 4 of the value of the keyword CTYPE*ia*. (2) By convention, the 'radio' velocity is given by $c(\nu_0 - \nu)/\nu_0$ and the 'optical' velocity is given by $c(\lambda - \lambda_0)/\lambda_0$.

tral reference frame that was held constant during the observation, respectively. In order to compute the velocities it is necessary to have the date and time of the observation; the keywords DATE-AVG and MJD-AVG are reserved for this purpose. See also Sect. 9.5.

DATE-AVG – [string; default: none]. Calendar date of the mid-point of the observation, expressed in the same way as the DATE-OBS keyword.

MJD-AVG – [floating point; default: none]. Modified Julian Date (JD – – 2,400,000.5) of the mid-point of the observation.

SPECSYS*a* – [string; default: none]. The reference frame in use for the spectral axis spectral-axis coordinate(s). Valid values are given in Table 27.

SSYSOBS*a* – [string; default: 'TOPOCENT''TOPOCENT']. The spectral reference frame that is constant over the range of the non-spectral world coordinates. Valid values are given in Table 27.

The transformation from the rest frame of the observer to a standard reference frame requires a specification of the location on Earth[14] of the instrument used for the observation in order to calculate the diurnal Doppler correction due to the Earth's rotation. The location, if specified, shall*shall* be represented as a geocentric Cartesian triple with respect to a standard ellipsoidal geoid at the time of the observation. While the position can often be specified with an accuracy of a meter or better, for most purposes positional errors of several kilometers will have negligible impact on the computed velocity correction. For details, see reference Greisen et al. (2006).

OBSGEO-X – [floating point; default: none]. $X-$coordinate (in meters) of a Cartesian triplet that specifies the location, with respect to a standard, geocentric terrestrial reference frame, where the observation took place. The coordinate must*must* be valid at the epoch MJD-AVG or DATE-AVG.

OBSGEO-Y – [floating point; default: none]. $Y-$coordinate (in meters) of a Cartesian triplet that specifies the location, with respect to a standard, geocentric terrestrial reference frame, where the observation took place. The coordinate must*must* be valid at the epoch MJD-AVG or DATE-AVG.

[14] The specification of location for an instrument on a spacecraft in flight requires an ephemeris; keywords that might be required in this circumstance are not defined here.

Table 26: Non-linear spectral algorithm codes.

Code[1]	Regularly sampled in	Expressed as
F2W	Frequency	Wavelength
F2V		Apparent radial velocity
F2A		Air wavelength
W2F	Wavelength	Frequency
W2V		Apparent radial velocity
W2A		Air wavelength
V2F	Apparent radial vel.	Frequency
V2W		Wavelength
V2A		Air wavelength
A2F	Air wavelength	Frequency
A2W		Wavelength
A2V		Apparent radial velocity
LOG	Logarithm	Any four-letter type code
GRI	Detector	Any type code from Table 25
GRA	Detector	Any type code from Table 25
TAB	Not regular	Any four-letter type code

(1) Characters 6 through 8 of the value of the keyword CTYPE*ia*.

OBSGEO-Z – [floating point; default: none]. $Z-$coordinate (in meters) of a Cartesian triplet that specifies the location, with respect to a standard, geocentric terrestrial reference frame, where the observation took place. The coordinate must*must* be valid at the epoch MJD-AVG or DATE-AVG.

Information on the relative radial velocity between the observer and the selected standard of rest in the direction of the celestial reference coordinate may*may* be provided, and if so shall*shall* be given by the VELOSYS*a* keyword. The frame of rest defined with respect to the emitting source may be represented in FITS*FITS*; for this reference frame it is necessary to define the velocity with respect to some other frame of rest. The keywords SPECSYS*a* and ZSOURCE*a* are used to document the choice of reference frame and the value of the systemic velocity of the source, respectively.

SSYSSRC*a* – [string; default: none]. Reference frame for the value expressed in the ZSOURCE*a* keyword to document the systemic velocity of the observed source. Value must*must* be one of those given in Table 27 *except* for SOURCE'SOURCE'.

VELOSYS*a* – [floating point; default: none]. Relative radial velocity between the observer and the selected standard of rest in the direction of the celestial reference coordinate. Units

DRAFT

Table 27: Spectral reference systems.

Value	Definition
'TOPOCENT'	Topocentric
'GEOCENTR'	Geocentric
'BARYCENT'	Barycentric
'HELIOCEN'	Heliocentric
'LSRK'	Local standard of rest (kinematic)
'LSRD'	Local standard of rest (dynamic)
'GALACTOC'	Galactocentric
'LOCALGRP'	Local Group
'CMBDIPOL'	Cosmic microwave background Cosmic-microwave-background dipole
'SOURCE'	Source rest frame

Notes. These are the allowed values of the SPECSYS*a*, SSYSOBS*a*, and SSYSSRC*a* keywords.

Table 28: Example keywords keyword records for a 100 element 100-element array of complex values.

Keyword records		
SIMPLE	=	T
BITPIX	=	-32
NAXIS	=	2
NAXIS1	=	2
NAXIS2	=	100
CTYPE1	=	'COMPLEX'
CRVAL1	=	0.
CRPIX1	=	0.
CDELT1	=	1.
END		

Table 29: Conventional Stokes values.

Value	Symbol	Polarization
1	I 'I'	Standard Stokes unpolarized
2	Q 'Q'	Standard Stokes linear
3	U 'U'	Standard Stokes linear
4	V 'V'	Standard Stokes circular
−1	RR 'RR'	Right-right circular
−2	LL 'LL'	Left-left circular
−3	RL 'RL'	Right-left cross-circular
−4	LR 'LR'	Left-right cross-circular
−5	XX 'XX'	X *X* parallel linear
−6	YY 'YY'	Y *Y* parallel linear
−7	XY 'XY'	XY *XY* cross linear
−8	YX 'YX'	YX *YX* cross linear

mustmust be m s^{-1}. The CUNIT*ia* keyword is not used for this purpose since the WCS version Version *a* might not be expressed in velocity units.

ZSOURCE*a* – [floating point; default: none]. Radial velocity with respect to an alternative frame of rest, expressed as a unitless redshift (i.e., velocity as a fraction of the speed of light in vacuum). Used in conjunction with SSYSSRC*a* to document the systemic velocity of the observed source.

VELANGL*a* – [floating point; default:+90.+90.]. In the case of relativistic velocities (e.g., a beamed astrophysical jet) the transverse velocity component is important. This keyword may *may* be used to express the orientation of the space velocity vector with respect to the plane of the sky. See Appendix A of reference Greisen et al. (2006) for further details.

8.5. Conventional coordinate Conventional-coordinate types

The first FITS*FITS* paper (Wells et al. 1981) listed a number of 'suggested values' for the CTYPE*i* keyword. Two of these have the attribute the associated world coordinates can assume only integer values and that the meaning of these integers is only defined by convention. The first 'conventional' coordinate is CTYPE*ia* ='COMPLEX' 'COMPLEX' to specify that complex values (i.e., pairs of real and imaginary components) are stored in the data array (along with an optional weight factor). Thus, the complex axis of the data array will contain two values (or three if the weight is specified). By convention, the real component has a coordinate value of 1, the imaginary component has a coordinate value of 2, and the weight, if any, has a coordinate value of 3. Table 28 illustrates the required keywords for an array of 100 complex values (without weights).

The second conventional coordinate is CTYPE*ia* ='STOKES' 'STOKES' to specify the polarization of the data. Conventional values, their symbols, and polarizations are given in Table 29.

9. Representations of time coordinates

Time as a dimension in astronomical data presents challenges for its representation in FITS *FITS* files. This section formulates the representation of the time axis, or possibly multiple time axes, into the World Coordinate System world-coordinate system (WCS) described in Sect. 8. Much of the basic structure is employed, while extensions are developed to cope with the differences between time and spatial dimensions; notable amongst these differences is the huge dynamic range, covering the highest resolution timing relative to the age of the Universeuniverse.

The precision with which any time stamp conforms to any conventional time scale is highly dependent on the characteristics of the acquiring system. The definitions of many conventional time scales vary over their history along with the precision that can be attributed to any time stamp. The meaning of any time stamp may be ambiguous if a time scale is used for dates prior to its definition by a recognized authority, or for dates after that definition is abandoned. However, common sense should prevail: the precision in the description of the time coordinate should *should* be appropriate to the accuracy of the temporal information in the data.

9.1. Time values

The three most common ways to specify time are: ISO-8601 (ISO 2004b), Julian Date (JD), or Modified Julian Date (MJD = JD − 2,400,000.5; see IAU 1997). Julian Dates are counted from Julian proleptic calendar date 1 January 4713 BCE at noon, or Gregorian proleptic calendar date 24 November 4714 BCE, also at noon. For an explanation of the calendars, see Rots et al. (2015). Even though it is common to think of certain representations of time as absolute, time values in FITS *FITS* files *shall* all be considered relative: elapsed time since a particular reference point in time. It may help to view the "absolute" values as merely relative to a globally accepted zero point. For a discussion of the precision required to represent time values in floating-point numbers, see Rots et al. (2015).

DRAFT

9.1.1. ISO-8601 datetime*datetime* strings

FITS *FITS* datetime strings conform to a subset of ISO-8601 (which in itself does not imply a particular time scale) for several time-related keywords (Bunclark & Rots 1997), such as . Here datetime`DATE-xxxx`. Here *datetime* will be used as a pseudo data type to indicate its use, although its values must*must* be written as a character string in `'A'` format. The full specification for the format of the datetime*datetime* string has been:

`CCYY-MM-DD[Thh:mm:ss[.s...]]`

All in which all of the time part *may* be omitted (just leaving the date) or the decimal seconds *may* be omitted. Leading zeroes must notzeros *must not* be omitted and timezone designators are *not allowed*. This definition is extended to allow five-digit years with a *mandatory* sign, in accordance with ISO-8601. That is, one *shall* use either the *unsigned* four-digit year format, or the *signed* five-digit year format : shown below.

`[±C]CCYY-MM-DD[Thh:mm:ss[.s...]]`

Note the following: .

- In counting years, ISO-8601 follows the convention of including year zeroYear Zero. Consequently, for negative year numbers there is an offset of one from BCE dates, which do not recognize a year zero. Thus year Year Zero. Thus Year 1 corresponds to 1 CE, year Year 0 to 1 BCE, year Year −1 to 2 BCE, and so on.
- The earliest date that may be represented in the four-digit year format is `0000-01-01T00:00:00` `'0000-01-01T00:00:00'` (in the year 1 BCE); the latest date is `9999-12-31T23:59:59`'9999-12-31T23:59:59'. This representation of time is tied to the Gregorian calendar. In conformance with the present ISO-8601:2004(E) standard (ISO 2004b) dates prior to 1582 must*must* be interpreted according to the proleptic application of the rules of Gregorius XIII. For dates not covered by that range the use of Modified Julian Date (MJD) or Julian Date (JD) numbers or the use of the signed five-digit year format is recommended*recommended*.
- In the five-digit year format the earliest and latest dates are `-99999-01-01T00:00:00` `'-99999-01-01T00:00:00'` (i.e., −100 000 BCE) and `+99999-12-31T23:59:59`'+99999-12-31T23:59:59'.
- The origin of JD can be written as: `-04713-11-24T12:00:00`'-04713-11-24T12:00:00'.
- In time scale UTC the UTC time scale the integer part of the seconds field runs from 00 to 60 (in order to accommodate leap seconds); in all other time scales the range is 00 to 59.
- The ISO-8601 datetime*datetime* data type is *not allowed* in image axis descriptions since is required image-axis descriptions since `CRVAL` is *required* to be a floating point floating-point value.
- ISO-8601 datetime*datetime* does not imply the use of any particular time scale (see SectionSect. 9.2.1).
- As specified by Bunclark & Rots (1997), time zones are explicitly not supported in FITS *FITS* and, consequently, appending the letter `'Z'` to a FITS `'Z'` to a *FITS* ISO-8601 string is *not allowed*. The rationale for this rule is that its role in the ISO standard is that of a time zone time-zone indicator, not a time scale time-scale indicator. As the concept of a time zone is not supported in FITS*FITS*, the use of time zone time-zone indicator is inappropriate.

9.1.2. Julian and Besselian epochs

In a variety of contexts *epochs* are provided with astronomical data. Until 1976 these were commonly based on the Besselian year (see Sect. 9.3), with standard epochs B1900.0 and B1950.0. After 1976 the transition was made to Julian epochs based on the Julian year of 365.25 days, with the standard epoch J2000.0. They are tied to time scales the ET and TDB time scales, respectively. Note that the Besselian epochs are scaled by the variable length of the Besselian year (see Sect. 9.3 and its cautionary note, which also applies to this context). The Julian epochs are easier to calculate, as long as one keeps track of leap days.

9.2. Time coordinate frame

9.2.1. Time scale

The *time scale* defines the temporal reference frame, and is specified in the header in one of a few ways, depending upon the context. When recorded as a global keyword, the time scale *shall* be specified by : the following keyword.

`TIMESYS` – [string; default: 'UTC'`'UTC'`]. The value field of this keyword *shall*contain a character string contain a character-string code for the time scale of the time-related keywords. The recommended*recommended* values for this keyword in Table30 have well defined 30 have well-defined meanings, but other values *may* be used. If this keyword is absent, `'UTC'` must*must* be assumed.

In relevant contexts (e.g., time axes in image arrays, table columns, or random groups) `TIMESYS` *may* be overridden by a time scale recorded in `CTYPE`*ia*, its binary table binary-table equivalents, or `PTYPE`*i* (see Table 22).

The keywords `TIMESYS`, `CTYPE`*ia*, `TCTYP`*n*, and `TCTY`*na* or binary table binary-table equivalent *may* assume the values listed in Table 30. In addition, for backward compatibility, all except `TIMESYS` and `PTYPE`*i* *may* also assume the value `'TIME'` (case-insensitive), whereupon the time scale *shall* be that recorded in `TIMESYS` or, in its absence, its default value, `'UTC'`. As noted above, local time scales other than those listed in Table 30 *may* be used, but their use *should* be restricted to alternate coordinates in order that the primary coordinates will always refer to a properly recognized time scale.

See Rots et al. (2015), Appendix A, for a detailed discussion of the various time scales. In cases where high-precision timing is important one may *may* append a specific realization, in parentheses, to the values in the table; e.g., , , `'TT(TAI)'`, `'TT(BIPM08)'`, `'UTC(NIST)'`. Note that linearity is not preserved across all time scales. Specifically, if the reference position remains unchanged (see Section Sect. 9.2.3), the first ten, with the exception of `'UT1'`, are linear transformations of each other (excepting leap seconds), as are and `'TDB'` and `'TCB'`. On average `'TCB'` runs faster than `'TCG'` by approximately 1.6×10^{-8}, but the transformation from or `'TT'` or `'TCG'` (which are linearly related) is to be achieved through a time ephemeris as provided by Irwin & Fukushima (1999).

The relations between coordinate time scales and their dynamical equivalents have been defined as:

DRAFT

$$T(\mathrm{TCG}) = T(\mathrm{TT}) + L_G \times 86400 \times (JD(\mathrm{TT}) - JD_0)$$
$$T(\mathrm{TDB}) = T(\mathrm{TCB}) - L_B \times 86400 \times (JD(\mathrm{TCB}) - JD_0) + TDB_0,$$

where:

T is in seconds
$L_G = 6.969290134 \times 10^{-10}$
$L_B = 1.550519768 \times 10^{-8}$
$JD_0 = 2443144.5003725$
$TDB_0 = -6.55 \times 10^{-5}$ s.

Linearity is virtually guaranteed since images and individual table columns do not allow more than one reference position to be associated with them, and since there is no overlap between reference positions that are meaningful for the first nine time scales on the one hand, and for the barycentric ones on the other. All use of the time scale GMT in FITS GMT time scale in *FITS* files *shall* be taken to have its zero point at midnight, conformant with UT, including dates prior to 1925. For high-precision timing prior to 1972, see Rots et al. (2015), Appendix A.

Some time scales in use are not listed in Table 30 because they are intrinsically unreliable or ill-defined. When used, they *should* be tied to one of the existing scales with appropriate specification of the uncertainties; the same is true for free-running clocks. However, a local time scale such as MET (Mission Elapsed Time) or OET (Observation Elapsed Time) *may* be defined for practical reasons. In those cases the time reference value (see Section 9.2.2) shall notSect. 9.2.2) *shall not* be applied to the values, and it is strongly recommendedthat such timescales *recommended* that such time scales be provided as alternate time scales, with a defined conversion to a recognized time scale.

It is useful to note that while UT1 is, in essence, an angle (of the Earth's rotation – *i.e.,* a *clock*), the others are SI-second counters (*chronometers*); UTC, by employing leap seconds, serves as a bridge between the two types of time scales.

9.2.2. Time reference value

The time reference value is not required*not required* to be present in an HDU. However, if the time reference point is specified explicitly it must*must* be expressed in one of ISO-8601, JD, or MJD. These reference values *must*only *only* be applied to time values associated with one of the recognized time scales listed in Table 30, and that time scale must*must* be specified explicitly or implicitly as explained in Sect. 9.2.1.

The reference point in time, to which all times in the HDU are relative, *shall* be specified through one of three keywords : specified below.

`MJDREF` – [floating-point]; default: 0.0`0.0`] The value field of this keyword *shall* contain the value of the reference time in MJD.

`JDREF` – [floating-point; default: none] The value field of this keyword *shall* contain the value of the reference time in JD.

`DATEREF` – [datetime; default: none] The value field of this keyword *shall*contain a character string contain a character-string representation of the reference time in ISO-8601 format.

`MJDREF` and `JDREF` may*may*, for clarity or precision reasons, be split into two keywords holding the integer and fractional parts separately: .

Table 30: Recognized Time Scale Values

Value	Meaning
`'TAI'`	(International Atomic Time): atomic time atomic-time standard maintained on the rotating geoid
`'TT'`	(Terrestrial Time; IAU standard): defined on the rotating geoid, usually derived as TAI + 32.184 s
`'TDT'`	(Terrestrial Dynamical Time): synonym for TT (deprecated)
`'ET'`	(Ephemeris Time): continuous with TT; should not *should not* be used for data taken after 1984-01-01
`'IAT'`	synonym for TAI (deprecated)
`'UT1'`	(Universal Time): Earth rotation time
`'UTC'`	(Universal Time, Coordinated; default): runs synchronously with TAI, except for the occasional insertion of leap seconds intended to keep UTC within 0.9 s of UT1; as of 2015 -07-01 2015-07-01 UTC = TAI – 36 s
`'GMT'`	(Greenwich Mean Time): continuous with UTC; its use is deprecated for dates after 1972-01-01
UT()[1] `UT()`[1]	(Universal Time, with qualifier): for high-precision use of radio signal radio-signal distributions between 1955 and 1972; see Rots et al. (2015), Appendix A
`'GPS'`	(Global Positioning System): runs (approximately) synchronously with TAI; GPS ≈ TAI – 19 s
`'TCG'`	(Geocentric Coordinate Time): TT reduced to the geocenter, corrected for the relativistic effects of the Earth's rotation and gravitational potential; TCG runs faster than TT at a constant rate
`'TCB'`	(Barycentric Coordinate Time): derived from TCG by a 4-dimensional four-dimensional transformation, taking into account the relativistic effects of the gravitational potential at the barycenter (relative to that on the rotating geoid) as well as velocity time dilation time-dilation variations due to the eccentricity of the Earth's orbit, thus ensuring consistency with fundamental physical constants; Irwin & Fukushima (1999) provide a time ephemeris
`'TDB'`	(Barycentric Dynamical Time): runs slower than TCB at a constant rate so as to remain approximately in step with TT; runs therefore quasi-synchronously with TT, except for the relativistic effects introduced by variations in the Earth's velocity relative to the barycenter. When referring to celestial observations, a pathlength correction to the barycenter may be needed, which requires the Time Reference Direction used in calculating the pathlength correction.
`'LOCAL'`	for simulation data and for free-running clocks.

[1] Specific realization codes *may* be appended to these values, in parentheses; see the text. For a more-detailed discussion of time scales, see Rots et al. (2015), Appendix A.

`MJDREFI` – [integer; default: 0`0`] The value field of this keyword *shall* contain the integer part of reference time in MJD.

`MJDREFF` – [floating-point; default: 0.0`0.0`] The value field of this keyword *shall* contain the fractional part of reference time in MJD.

`JDREFI` – [integer; default: none] The value field of this keyword *shall* contain the integer part of reference time in JD.

`JDREFF` – [floating-point; default: none] The value field of this keyword *shall* contain the fractional part of reference time in JD.

DRAFT

If [M]JDREF and both [M]JDREFI and [M]JDREFF are present, the integer and fractional values shall *shall* have precedence over the single value. If the single value is present with one of the two parts, the single value shall *shall* have precedence. In the following, MJDREF and JDREF refer to their literal meaning or the combination of their integer and fractional parts. If a header contains more than one of these keywords, JDREF *shall* have precedence over DATEREF and MJDREF *shall* have precedence over both the others. If none of the three keywords is present, there is no problem as long as all times in the HDU are expressed in ISO-8601; otherwise MJDREF =0.0 must 0.0 *must* be assumed. If TREFPOS ='' (Section 9.2.3) 'CUSTOM' (Sect. 9.2.3), it is legitimate for none of the reference time reference-time keywords to be present, as one may assume the data are from a simulation. Note that the *value* of the reference time has global validity for all time values, but it does not have a particular time scale associated with it.

9.2.3. Time reference position

An observation is an event in space-time. The reference position specifies the spatial location at which the time is valid, either where the observation was made or the point in space for which light-time corrections have been applied. When recorded as a global keyword, the time reference position *shall* be specified by : the following keyword.

TREFPOS – [string; default: 'TOPOCENTER''TOPOCENTER']. The value field of this keyword *shall*contain a character string contain a character-string code for the spatial location at which the observation time is valid. The value *should* be one of those given in Table 31. This keyword *shall*apply to time coordinate apply to time-coordinate axes in images as well.

In binary tables, different columns *may* represent completely different Time Coordinate Frames. However, each column can have only one time reference position, thus guaranteeing linearity (see Section Sect. 9.2.1).

TRPOS*n* – [string; default: 'TOPOCENTER''TOPOCENTER'] The value field of this keyword *shall*contain a character string contain a character-string code for the spatial location at which the observation time is valid. This table keyword *shall* override TREFPOS.

The reference position value *may* be a standard location (such as or 'GEOCENTER' or 'TOPOCENTER') or a point in space defined by specific coordinates. In the latter case one should be aware that a (3-D) spatial coordinate three-dimensional) spatial-coordinate frame needs to be defined that is likely to be different from the frame(s) that with which the data are associatedwith. Note that 'TOPOCENTER' is only moderately informative if no observatory location is provided or indicated. The commonly allowed standard values are shown in Table 31. Note that for the gaseous planets the barycenters of their planetary systems, including satellites, are used for obvious reasons. While it is preferable to spell the location names out in full, in order to be consistent with the practice of Greisen et al. (2006) the values are allowed to be truncated to eight characters. Furthermore, in order to allow for alternative spellings, only the first three characters of all these values *shall* be considered significant. The value of the keyword *shall* be case-sensitive.

Table 31: Standard Time Reference Position Values

Value[1]	Meaning
'TOPOCENTER'	Topocenter: the location from where the observation was made (default)
'GEOCENTER'	Geocenter
'BARYCENTER'	Barycenter of the Solar System
'RELOCATABLE'	Relocatable: to be used for simulation data only
'CUSTOM'	A position specified by coordinates that is not the observatory location
Less-common, but allowed standard values	
'HELIOCENTER'	Heliocenter
'GALACTIC'	Galactic center
'EMBARYCENTER'	Earth-Moon barycenter
'MERCURY'	Center of Mercury
'VENUS'	Center of Venus
'MARS'	Center of Mars
'JUPITER'	Barycenter of the Jupiter system
'SATURN'	Barycenter of the Saturn system
'URANUS'	Barycenter of the Uranus system
'NEPTUNE'	Barycenter of the Neptune system

Notes. [(1)]Recognized values for TREFPOS, TRPOS*n*; only the first three characters of the values are significant and Solar System locations are as specified in the ephemerides.

The reader is cautioned that time scales and reference positions cannot be combined arbitrarily if one wants a clock that runs linearly at TREFPOS. Table 32 provides a summary of compatible combinations. 'BARYCENTER' *should* only be used in conjunction with time scales and 'TDB' and 'TCB', and *should* be the only reference position used with these time scales. With proper care , , and 'GEOCENTER', 'TOPOCENTER', and 'EMBARYCENTER' are appropriate for the first ten time scales in Table 30. However, relativistic effects introduce a (generally linear) scaling in certain combinations; highly eccentric spacecraft orbits are the exceptions. Problems will arise when using a reference position on another solar system Solar System body (including 'HELIOCENTER'). Thereforeit is recommended, it is *recommended* to synchronize the local clock with one of the time scales defined on the Earth's surface, , , or 'TT', 'TAI', 'GPS', or 'UTC' (in the last case: beware of leap seconds). This is common practice for spacecraft clocks. Locally, such a clock will not appear to run at a constant rate, because of variations in the gravitational potential and in motions with respect to Earth, but the effects can be calculated and are probably small compared with errors introduced by the alternative: establishing a local time standard.

In order to provide a complete description, 'TOPOCENTER' requires the observatory's coordinates to be specified. There are three options: (a)*(a)* the ITRS Cartesian coordinates defined in Sect. 8.4.1 (OBSGEO-X, OBSGEO-Y, OBSGEO-Z), which are *strongly preferred*; (b)*(b)* a geodetic latitude/longitude/elevation triplet (defined below); or (c)*(c)* a reference to an orbit ephemeris orbit-ephemeris file. A set of geodetic coordinates is recognized : by the following keywords.

OBSGEO-B – [floating-point] The value field of this keyword *shall* contain the latitude of the observation in deg, with North positive.

Table 32: Compatibility of Time Scales and Reference Positions

Reference Position	Time scale[1] TT, TDT TAI, IAT GPS UTC, GMT	TCG	TDB	TCB	LOCAL
'TOPOCENTER'	t	ls			
'GEOCENTER'	ls	c			
'BARYCENTER'			ls	c	
'RELOCATABLE'					c
Other[2]	re	re			

Notes. [(1)]Legend (combination is *not recommended* if there is no entry); **c**: correct match; reference position coincides with the spatial origin of the space-time coordinates; **t**: correct match on Earth's surface, otherwise usually linear scaling; **ls**: linear relativistic scaling; **re**: non-linear relativistic scaling. [(2)]All other locations in the Solar System.

OBSGEO-L – [floating-point] The value field of this keyword *shall* contain the longitude of the observation in deg, with East positive.

OBSGEO-H – [floating-point] The value field of this keyword *shall* contain the altitude of the observation in meters.

An orbital ephemeris orbital-ephemeris file can instead be specified: .

OBSORBIT – [string] The value field of this keyword *shall* contain the character-string URI, URL, or the name of an orbit ephemeris orbit-ephemeris file.

Beware that only one set of coordinates is allowed in a given HDU. Cartesian ITRS coordinates are the preferred coordinate system; however, when using these in an environment requiring nanosecond accuracy, one should take care to distinguish between meters consistent with TCG or with TT. If one uses geodetic coordinates, the geodetic altitude OBSGEO-H is measured with respect to the IAU 1976 ellipsoid, which is defined as having a semi-major axis of 6 378 140 m and an inverse flattening of 298.2577.

A non-standard location indicated by must'CUSTOM' *must* be specified in a manner similar to the specification of the observatory location (indicated by 'TOPOCENTER'). One should be careful with the use of the 'CUSTOM' value and not confuse it with 'TOPOCENTER', as use of the latter imparts additional information on the provenance of the data.

ITRS coordinates (X,Y,ZX,Y,Z) may be derived from geodetic coordinates (L,B,HL,B,H) through:

$$X = (N(B) + H)\cos(L)\cos(B)$$

$$Y = (N(B) + H)\sin(L)\cos(B)$$

$$Z = (N(B)(1 - e^2) + H)\sin(B)$$

where:

$$N(B) = \frac{a}{\sqrt{1 - e^2\sin^2(B)}}$$

$$e^2 = 2f - f^2$$

a is the semi-major axis, and f is the inverse of the inverse flattening. Nanosecond precision in timing requires that OBSGEO-[BLH] be expressed in a geodetic reference frame defined after 1984 in order to be sufficiently accurate.

9.2.4. Time reference direction

If any pathlength corrections have been applied to the time stamps (i.e., if the reference position is not 'TOPOCENTER' for observational data), the reference direction that is used in calculating the pathlength delay *should* be provided in order to maintain a proper analysis trail of the data. However, this is useful only if there is also information available on the location from where the observation was made (the observatory location). The direction will usually be provided in a spatial coordinate spatial-coordinate frame that is already being used for the spatial metadata, although it is conceivable that multiple spatial frames are involved, e.g., spherical ICRS coordinates for celestial positions, and Cartesian FK5 for spacecraft ephemeris. The time reference direction does not by itself provide sufficient information to perform a fully correct transformation; however, within the context of a specific analysis environment it should suffice.

The uncertainty in the reference direction affects the errors in the time stamps. A typical example is provided by barycentric corrections where the time error is related to the position error:

$$t_{err\,\mathrm{err}}(\mathrm{ms}) \leq 2.4\; pos_{err\,\mathrm{err}}(\mathrm{arcsec}).$$

The reference direction is indicated through a reference to specific keywords. These keywords *may* hold the reference direction explicitly or (for data in BINTABLEsBINTABLE extensions) indicate columns holding the coordinates. In event lists where the individual photons are tagged with a spatial position, those coordinates *may* have been used for the reference direction and the reference will point to the columns containing these coordinate values. The time reference direction *shall* be specified by the keyword: following keyword.

TREFDIR – [string] The value field of this keyword *shall* contain a character string composed of: the name of the keyword containing the longitudinal coordinate, followed by a comma, followed by the name of the keyword containing the latitudinal coordinate. This reference direction *shall*apply to time coordinate apply to time-coordinate axes in images as well.

In binary tables, different columns *may* represent completely different Time Coordinate Frames. However, also in that situation the condition holds that each column can have only one Time Reference Direction. Hence, the following keyword may *may* override TREFDIR: .

TRDIRn – [string] The value field of this keyword *shall* contain a character string consisting of the name of the keyword or column containing the longitudinal coordinate, followed by a comma, followed by the name of the keyword or column containing the latitudinal coordinate. This reference direction *shall*apply to time coordinate apply to time-coordinate axes in images as well.

9.2.5. Solar System Ephemerisephemeris

If applicable, the Solar System ephemeris used for calculating pathlength delays *should* be identified. This is particularly pertinent when the time scale is or 'TCB' or 'TDB'. The ephemerides that are currently most often used are those from JPL (2014a,b).

The Solar System ephemeris used for the data (if required) *shall* be indicated by the following keyword: .

DRAFT

PLEPHEM – [string; default: 'DE405'] The value field of this keyword *shall* contain a character string that *should* represent a recognized designation for the Solar System ephemeris. Recognized designations for JPL Solar System ephemerides that are often used are listed in Table 33.

Table 33: Valid solar system Solar System ephemerides

Value	Reference
'DE200'	Standish (1990); considered obsolete, but still in use
'DE405'	Standish (1998); default
'DE421'	Folkner, et al. (2009)
'DE430'	Folkner, et al. (2014)
'DE431'	Folkner, et al. (2014)
'DE432'	Folkner, et al. (2014)

Future ephemerides in this series *shall* be accepted and recognized as they are released. Additional ephemerides designations may *may* be recognized by the IAUFWG upon request.

9.3. Time unit

When recorded as a global keyword, the unit used to express time *shall* be specified by : the following keyword.

TIMEUNIT – [string; default: 's''s'] The value field of this keyword *shall* contain a character string that specifies the time unit; the value *should* be one of those given in Table 34. This time unit *shall* apply to all time instances and durations that do not have an implied time unit (such as is the case for JD, MJD, ISO-8601, J and B epochs). If this keyword is absent, 's' *shall* be assumed.

In an appropriate context, e.g., when an image has a time axis, TIMEUNIT *may* be overridden by the CUNIT*ia* keywords and their binary table binary-table equivalents (see Table 22).

The specification of the time unit allows the values defined in Greisen & Calabretta (2002), shown in Table 34, with the addition of the century. See also Sect. 4.3 for generalities about units.

Table 34: Recommended time units

Value	Definition
's'	second (default)
'd'	day (= 86,400 s)
'a'	(Julian) year (= 365.25 d)
'cy'	(Julian) century (= 100 a)
The following values are also acceptable.	
'min'	minute (= 60 s)
'h'	day (= 86,400 s)
'yr'	(Julian) year (= 'a' = 365.25 d)
'ta'	tropical year
'Ba'	Besselian year

The use of and 'ta' and 'Ba' is not encouraged, but there are data and applications that require the use of tropical years or Besselian epochs (see Section Sect. 9.1.2). The length of the tropical year, 'ta', in days is:

$$1\ \mathrm{ta} = 365.24219040211236 - 0.00000615251349\,T - 6.0921\times10^{-10}\,T^2 + 2.6525\times10^{-10}\,T^3\ \mathrm{(d)}$$

where T is in Julian centuries since J2000, using time scale TDB. The length of the Besselian year in days is:

$$1\mathrm{Ba} = 365.2421987817 - 0.00000785423\,T\ \mathrm{(d)}$$

where T is in Julian centuries since J1900, using time scale ET, although for these purposes the difference with TDB is negligible.

Readers are cautioned that the subject of tropical and Besselian years presents a particular quandary for the specification of standards. The expressions presented here are the most accurate available, but are applicable for use when creating data files (which is strongly discouraged), rather than for interpreting existing data that are based upon these units. But However, there is no guarantee that the authors of the data applied these particular definitions. Users are therefore advised to pay close attention and attempt to ascertain what the authors of the data really used.

9.4. Time offset, binning, and errors

9.4.1. Time offset

A uniform clock correction may *may* be applied in bulk with the following single keyword.

TIMEOFFS – [floating-point; default: 0.00.0] The value field of this keyword *shall* contain the value of the offset in time that *shall* be added to the reference time, given by one of: MJDREF, JDREF, or DATEREF.

The time offset may serve to set a zero-point offset to a relative time series, allowing zero-relative times, or just higher precision, in the time stamps. Its default value is zero. The value of this keyword affects the values of TSTART, and TSTOP, as well as any time pixel values in a binary table. However, this construct *may* only be used in tables and must not*must not* be used in images.

9.4.2. Time resolution and binning

The resolution of the time stamps (the width of the time sampling function) *shall* be specified by : the following keyword.

TIMEDEL – [floating-point] The value field of this keyword *shall* contain the value of the time resolution in the units of TIMEUNIT. This construct, when present, shall only*shall only* be used in tables and must not*must not* be used in images.

In tables this may, for instance, be the size of the bins for time series time-series data or the bit precision of the time stamp time-stamp values.

When data are binned in time bins (or, as a special case, events are tagged with a time stamp of finite precision) it is important to know to the position within the bin (or pixel) to which the time stamp refers. Coordinate values normally correspond to the center of all pixels (see Sect. 8.2); yet clock readings are effectively truncations, not rounded values, and therefore correspond to the lower bound of the pixel.

DRAFT

TIMEPIXR – [floating-point; default: 0.50.5] The value field of this keyword *shall* contain the value of the position within the pixel, from 0.0 to 1.0, to which the time-stamp refers. This construct, when present, shall only*shall only* be used in tables and must not*must not* be used in images.

A value of 0.0 0.0 may be more common in certain contexts, e.g. when truncated clock readings are recorded, as is the case for almost all event lists.

9.4.3. Time errors

The absolute time error is the equivalent of a systematic error, *shall* be given by the following keyword: .

TIMSYER – [floating-point; default: 0.0.] The value field of this keyword *shall* contain the value of the absolute time error, in units of TIMESYS.

This keyword *may* be overridden, in appropriate context (e.g., time axes in image arrays or table columns; by the CSYER*ia* keywords and their binary table binary-table equivalents (see Table 22).

The relative time error specifies accuracy of the time stamps relative to each other. This error will usually be much smaller than the absolute time error. This error is equivalent to a random error, and *shall* be given by the following keyword: .

TIMRDER – [floating-point; default: 0.0.] The value field of this keyword *shall* contain the value of the relative time error, i.e. the random error between time stamps, in units of TIMESYS.

This keyword *may* be overridden, in appropriate context (e.g., time axes in image arrays or table columns; by the CRDER*ia* keywords and their binary table binary-table equivalents (see Table 22).

9.5. Global time keywords

The time keywords in Table 35 are likely to occur in headers even when there are no time axes in the data. Except for DATE, they provide the top-level temporal bounds of the data in the HDU. As noted before, they may also be implemented as table columns. Keywords not previously described are defined below; all are included in the summary Table 22.

DATE-BEG – [datetime] The value field of this keyword *shall* contain a character string in ISO-8601 format that specifies the start time of data acquisition in the time system specified by the TIMESYS keyword.

DATE-END – [datetime] The value field of this keyword *shall* contain a character string in ISO-8601 format that specifies the stop time of data acquisition in the time system specified by the TIMESYS keyword.

MJD-BEG – [floating-point] The value field of this keyword *shall* contain the value of the MJD start time of data acquisition in the time system specified by the TIMESYS keyword.

MJD-END – [floating-point] The value field of this keyword *shall* contain the value of the MJD stop time of data acquisition in the time system specified by the TIMESYS keyword.

TSTART – [floating-point] The value field of this keyword *shall* contain the value of the start time of data acquisition in units of TIMEUNIT, relative to MJDREF, JDREF, or DATEREF and TIMEOFFS, in the time system specified by the TIMESYS keyword.

Table 35: Keywords for global time values

Keyword	Notes
DATE	Defined in Sect. 4.4.2.
DATE-OBS	Defined in Sect. 4.4.2. Keyword value was not restricted to mean the start time of an observation, and has historically also been used to indicate some form of mean observing date and time. To avoid ambiguity use DATE-BEG instead.
DATE-BEG	Defined in this section.
DATE-AVG	Defined in Sect. 8.4.1. The method by which average times should be calculated is not defined by this Standard.
DATE-END	Defined in this section.
MJD-OBS	Defined in Sect. 8.3.
MJD-BEG	Defined in this section.
MJD-AVG	Defined in Sect. 8.4.1. The method by which average times should be calculated is not defined by this Standard.
MJD-END	Defined in this section.
TSTART	Defined in this section.
TSTOP	Defined in this section.

TSTOP – [floating-point] The value field of this keyword *shall* contain the value of the stop time of data acquisition in units of TIMEUNIT, relative to MJDREF, JDREF, or DATEREF and TIMEOFFS, in the time system specified by the TIMESYS keyword.

The alternate-axis equivalent keywords for BINTABLESBINTABLE extensions, DOBS*n*, MJDOB*n*, DAVG*n*, and MJDA*n*, as defined in Table 22, are also allowed. Note that of the above only TSTART and TSTOP are relative to the time reference value. As in the case of the time reference value (see Section Sect. 9.2.2), the JD values supersede DATE values, and MJD values supersede both, in cases where conflicting values are present.

It should be noted that, although they do not represent global time values within an HDU, the CRVAL*ia* and CDELT*ia* keywords, and their binary table binary-table equivalents (see Table 22), also represent (binary) time values. They should be handled with the same care regarding precision when combining them with the time reference value, as any other time value.

Finally, Julian and Besselian epochs (see Sections 9.1.2and Sects. 9.1.2 and 9.3) *may* be expressed by these two keywords – to be used with great caution, as their definitions are more complicated and hence their use more prone to confusion: .

JEPOCH – [floating-point] The value field of this keyword *shall* contain the value of the Julian epoch, with an implied time scale of 'TDB'.

BEPOCH – [floating-point] The value field of this keyword *shall* contain the value of the Besselian epoch, with an implied time scale of 'ET'.

When these epochs are used as time stamps in a table column, their interpretation will be clear from the context. When the keywords appear in the header without obvious context, they must*must* be regarded as equivalents of and DATE-OBS and MJD-OBS, i.e., with no fixed definition as to what part of the dataset they referto.

DRAFT

9.6. Other time coordinate time-coordinate axes

There are a few coordinate axes that are related to time and that are accommodated in this standardStandard: (temporal) *phase*, *timelag*, and *frequency*. Phase results from folding a time series on a given period, and can appear in parallel with *time* as an alternate description of the same axis. Timelag is the coordinate of cross- and auto-correlation spectra. The temporal *frequency* is the Fourier transform equivalent of time and, particularly, the coordinate axis of power spectra; spectra where the dependent variable is the electromagnetic field are excluded here, but see Greisen et al. (2006). These coordinate axes *shall* be specified by giving `CTYPE`*i* and its binary table binary-table equivalents one of the values: , , or `'PHASE'`, `'TIMELAG'`, or `'FREQUENCY'`.

Timelag units are the regular time units, and the basic unit for frequency is `'Hz'`. Neither of these two coordinates is a linear or scaled transformation of time, and therefore cannot appear in parallel with time as an alternate description. That is, a given vector of values for an observable can be paired with a coordinate vector of time, or timelag, or frequency, but not with more than one of these; the three coordinates are orthogonal.

Phase can appear in parallel with time as an alternate description of the same axis. Phase *shall* be recorded in the following keywords: .

`CZPHS`*ia* – [floating-point] The value field of this keyword *shall* contain the value of the time at the zero point of a phase axis. Its units *may*be , , or be `'deg'`, `'rad'`, or `'turn'`.

`CPERI`*ia* – [floating-point] The value field of this keyword, if present *shall* contain the value of the period of a phase axis. This keyword can be used only if the period is a constant; if that is not the case, this keyword *should* either be absent or set to zero.

`CZPHS`*ia* may *may* instead appear in binary table binary-table forms `TCZPH`*n*, `TCZP`*na*, *i*`CZPH`*n*, and *i*`CZP`*na*. `CPERI`*ia* may *may* instead appear in binary table binary-table forms `TCPER`*n*, `TCPR`*na*, *i*`CPER`*n*, and *i*`CPR`*na*. The Phase, periodphase, period, and zero point *shall* be expressed in the globally valid time reference frame and unit as defined by the global keywords (or their defaults) in the header.

9.7. Durations

There is an extensive collection of header keywords that indicate time durations, such as exposure times, but there are many pitfalls and subtleties that make this seemingly simple concept treacherous. Because of their crucial role and common use, keywords are defined below to record exposure and elapsed time.

`XPOSURE` – [floating-point] The value field of this keyword *shall* contain the value for the effective exposure duration for the data, corrected for dead time and lost time in the units of `TIMEUNIT`. If the HDU contains multiple time slices, this value *shall* be the total accumulated exposure time over all slices.

`TELAPSE` – [floating-point] The value field of this keyword *shall* contain the value for the amount of time elapsed, in the units of `TIMEUNIT`, between the start and the end of the observation or data stream.

Durations must notmust not be expressed in ISO-8601 format, but only as actual durations (i.e., numerical values) in the units of the specified time unit.

Good-Time-Interval (GTI) tables are common for exposures with gaps in them, particularly photon-event files, as they make it possible to distinguish time intervals with "no signal detected" from "no data taken." GTI tables in BINTABLE extensions mustBINTABLE extensions *must* contain two mandatory columns, and `START` and `STOP`, and *may*contain one optional column, contain one *optional* column, `WEIGHT`. The first two define the interval, the third, with a value between 0 and 1, the quality of the interval; *i.e.,* a weight of 0 indicates a *Bad*-Time-Interval. `WEIGHT` has a default value of 1. Any time interval not covered in the table shall *shall* be considered to have a weight of zero.

9.8. Recommended best practices

The following guidelines should be helpful in creating data products with a complete and correct time representation.

- The presence of the informational `DATE` keyword is *strongly recommended* in all HDUs.
- One or more of the informational keywords `DATE-xxxx` and/or `MJD-xxxx` *should* be present in all HDUs whenever a meaningful value can be determined. This also applies, e.g., to catalogs derived from data collected over a well-defined time range.
- The global keyword `TIMESYS` is *strongly recommended*.
- The global keywords `MJDREF` or `JDREF` or `DATEREF` are recommended*recommended.*
- The remaining informational and global keywords *should* be present whenever applicable.
- All context-specific keywords *shall* be present as needed and required *required* by the context of the data.

9.8.1. Global keywords and overrides

For reference to the keywords that are discussed here, see Table 22. The globally applicable keywords listed in section Sect. B of the table serve as default values for the corresponding `C*` and `TC*` keywords in that same section, but only when axis and column specifications (including alternate coordinate definitions) use a time scale listed in Table30 30, or when the corresponding `CTYPE` or `TTYPE` keywords are set to the value `'TIME'`. Any alternate coordinate specified in a non-recognized time scale assumes the value of the axis pixels or the column cells, optionally modified by applicable scaling and/or reference value keywords; see also Section Sect. 9.2.1.

9.8.2. Restrictions on alternate descriptions

An image will have at most one time axis as identified by having the `CTYPE`*i* value of `'TIME'` or one of the values listed in Table 30. Consequently, as long as the axis is identified through `CTYPE`*i*, there is no need to have axis number axis-number identification on the global time-related keywords. It is expressly prohibited to specify more than one time reference position on this axis for alternate time coordinate time-coordinate frames, since this would give rise to complicated model-dependent nonlinear relations between these frames. Hence, time scales and (or `'TDB'` and `'TCB'` (or `'ET'`, to its precision) may *may* be specified in the same image, but cannot be combined with any of the first nine time scales in Table 30; those first nine can be expressed as linear transformations of each other, too, provided the reference position remains unchanged. Time scale `'LOCAL'`

DRAFT

is by itself, intended for simulations, and should not *should not* be mixed with any of the others.

9.8.3. Image time axes

SectionSect. 8.2 requires keywords CRVAL*ia* to be numeric and they cannot be expressed in ISO-8601 format. Therefore it is required*required* that CRVAL*ia* contain the elapsed time in units of TIMEUNIT or CUNIT*ia*, even if the zero point of time is specified by DATEREF. If the image does not use a matrix for scaling, rotation, and shear (Greisen & Calabretta 2002), CDELT*ia* provides the numeric value for the time interval. If the PC form of scaling, rotation, and shear (Greisen & Calabretta 2002) is used, CDELT*ia* provides the numeric value for the time interval, and PC*i_j*, where $i = j$ = the index of the time axis (in the typical case of an image cube with axis Axis 3 being time, $i = j = 3$) would take the exact value 1, the default (Greisen & Calabretta 2002). When the CD*i_j* form of mapping is used, CD*i_j* provides the numeric value for the time interval. If one of the axes is time and the matrix form is used, then the treatment of the PC*i_ja* (or CD*i_ja*) matrices involves at least a Minkowsky metric and Lorentz transformations (as contrasted with Euclidean and Galilean).

10. Representations of compressed data

Minimizing data volume is important in many contexts, particularly for publishers of large astronomical data collections. The following sections describe compressed representations of data in FITS images and BINTABLES *FITS* images and BINTABLE extensions that preserve metadata and allow for full or partial extraction of the original data as necessary. The resulting FITS *FITS* file structure is independent of the specific data compression data-compression algorithm employed. The implementation details for some compression algorithms that are widely used in astronomy are defined in Sect. 10.4, but other compression techniques could also be supported. See the FITS *FITS* convention by White et al. (2013) for details of the compression techniques, but beware that the specifications in this Standard *shall* supersede those in the registered convention.

Compression of FITS Compression of *FITS* files can be beneficial for sites that store or distribute large quantities of data; the present section provides a standard framework that addresses such needs. As implementation of compression/decompression codes can be quite complex, not all FITS software for reading and writing software *FITS* is necessarily expected to support these capabilities. External utilities are available to compress and uncompress FITS decompress *FITS* files[15].

10.1. Tiled Image Compressionimage compression

The following describes the process for compressing n–dimensional FITS *FITS* images and storing the resulting byte stream in a variable-length column in a FITS *FITS* binary table, and for preserving the image header keywords in the table header. The general principle is to first divide the n–dimensional image into a rectangular grid of subimages or "tiles." Each tile is then compressed as a block of data, and the resulting compressed byte stream is stored in a row of a variable length variable-length column in a FITS *FITS* binary table (see SectionSect. 7.3). By dividing the image into tiles it is possible to extract and uncompress decompress subsections of the image without having to uncompress decompress the whole image. The default tiling pattern treats each row of a 2-dimensional two-dimensional image (or higher dimensional higher-dimensional cube) as a tile, such that each tile contains NAXIS1 pixels. This default may not be optimal for some applications or compression algorithms, so any other rectangular tiling pattern may *may* be defined using keywords that are defined below. In the case of relatively small images it may suffice to compress the entire image as a single tile, resulting in an output binary table with containing a single row. In the case of 3-dimensional three-dimensional data cubes, it may be advantageous to treat each plane of the cube as a separate tile if application software typically needs to access the cube on a plane-by-plane basis.

10.1.1. Required Keywordskeywords

In addition to the mandatory keywords for BINTABLE BINTABLE extensions (see Sect. 7.3.1) the following keywords are reserved for use in the header of a FITS binary table *FITS* binary-table extension to describe the structure of a valid compressed FITS *FITS* image. All are mandatory.

ZIMAGE – [logical; value 'T'T] The value field of this keyword *shall* contain the logical value 'T' T to indicate that the FITS binary table *FITS* binary-table extension contains a compressed image, and that logically this extension *should* be interpreted as an image rather than a table.

ZCMPTYPE – [string; default: none] The value field of this keyword *shall* contain a character string giving the name of the algorithm that was used to compress the image. Only the values given in Table 36 are permitted; the corresponding algorithms are described in Sect. 10.4. Other algorithms may be added in the future.

ZBITPIX – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the BITPIX keyword in the uncompressed FITS *FITS* image.

ZNAXIS – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the NAXIS keyword (i.e., the number of axes) in the uncompressed FITS *FITS* image.

ZNAXISn – [integer; indexed; default: none) The value field of these keywords *shall* contain a positive integer that gives the value of the corresponding NAXISn keywords (i.e., the size of axis nAxis n) in the uncompressed FITS *FITS* image.

The comment fields for the BITPIX, NAXIS, and NAXISn keywords in the uncompressed image *should* be copied to the corresponding fields in the ZBITPIX, ZNAXIS, and ZNAXISn keywords.

10.1.2. Other Reserved Keywordsreserved keywords

The compressed image tiles must*must* be stored in the binary table in the same order that the first pixel in each tile appears in the FITS *FITS* image; the tile containing the first pixel in the image

[15] e.g. fpack/funpack, see https://heasarc.gsfc.nasa.gov/fitsio/fpack/

must*must* appear in the first row of the table, and the tile containing the last pixel in the image must*must* appear in the last row of the binary table. The following keywords are reserved for use in describing compressed images stored in BINTABLE `BINTABLE` extensions; they *may* be present in the header, and their values depend upon the type of image compression employed.

`ZTILE`*n* – [integer; indexed; default: 1 1 for $n > 1$] The value field of these keywords (where *n* is a positive integer index that ranges from 1 to `ZNAXIS`) *shall* contain a positive integer representing the number of pixels along axis nAxis *n* of the compressed tiles. Each tile of pixels must*must* be compressed separately and stored in a row of a variable-length vector column in the binary table. The size of each image dimension (given by `ZNAXIS`*n*) need not be an integer multiple of `ZTILE`*n*, and if it is not, then the last tile along that dimension of the image will contain fewer image pixels than the other tiles. If the `ZTILE`*n* keywords are not present then the default "row-by-row" tiling will be assumed, i.e., ZTILE1 `ZTILE1` =ZNAXIS1 `ZNAXIS1`, and the value of all the other `ZTILE`*n* keywords mustequal 1. *must* equal 1.

`ZNAME`*i* – [string; indexed; default: none] The value field of these keywords (where *i* is a positive integer index starting with 1) *shall* supply the names of up to 999 algorithm-specific parameters that are needed to compress or uncompress decompress the image. The order of the compression parameters *may* be significant, and *may* be defined as part of the description of the specific decompression algorithm.

`ZVAL`*i* – [string; indexed; default: none] The value field of these keywords (where *i* is a positive integer index starting with 1) *shall* contain the values of up to 999 algorithm-specific parameters with the same index *i*. The value of `ZVAL`*i* may *may* have any valid FITS *FITS* data type.

`ZMASKCMP` – [string; default: none] The value field of this keyword *shall* contain the name of the image compression algorithm that was used to compress the optional null-pixel data mask. This keyword may *may* be omitted if no null-pixel data masks appear in the table. See Sect. 10.2.2 for details.

`ZQUANTIZ` – [string; default: `'NO_DITHER'`] The value field of this keyword *shall* contain the name of the algorithm that was used to quantize floating-point image pixels into integer values, which were then passed to the compression algorithm as discussed further in Sect. 10.2. If this keyword is not present, the default is to assume that no dithering was applied during quantization.

`ZDITHER0` – [integer; default: none] The value field of this keyword *shall* contain a positive integer (that may range from 1 to 10000 inclusive) that gives the seed value for the random dithering pattern that was used when quantizing the floating-point pixel values. This keyword *may* be absent if no dithering was applied. See Sect. 10.2 for further discussion.

The following keywords are reserved to preserve a verbatim copy of the *value and comment fields* for keywords in the original uncompressed FITS *FITS* image that were used to describe its strructurestructure. These optional keywords, when present, shall*shall* be used when reconstructing an identical copy of the original FITS *FITS* HDU of the uncompressed image. They should not*should not* appear in the compressed image header unless the corresponding keywords were present in the uncompressed image.

`ZSIMPLE` – [logical; value `'T'`T] The value field of this keyword must*must* contain the value of the original `SIMPLE` keyword in the uncompressed image.

`ZEXTEND` – [string] The value field of this keyword must*must* contain the value of the original `EXTEND` keyword in the uncompressed image.

`ZBLOCKED` – [logical] The value field of this keyword must*must* contain the value of the original `BLOCKED` keyword in the uncompressed image.

`ZTENSION` – [string] The value field of this keyword must*must* contain the original `XTENSION` keyword in the uncompressed image.

`ZPCOUNT` – [integer] The value field of this keyword must*must* contain the original `PCOUNT` keyword in the uncompressed image.

`ZGCOUNT` – [integer] The value field of this keyword must*must* contain the original `GCOUNT` keyword in the uncompressed image.

`ZHECKSUM` – [string] The value field of this keyword must*must* contain the original `CHECKSUM` keyword (see Sect. 4.4.2.7) in the uncompressed image.

`ZDATASUM` – [string] The value field of this keyword must*must* contain the original `DATASUM` keyword (see Sect. 4.4.2.7) in the uncompressed image.

The `ZSIMPLE`, `ZEXTEND`, and `ZBLOCKED` keywords must not*must not* be used unless the original uncompressed image was contained in the primary array of a FITS *FITS* file. The `ZTENSION`, `ZPCOUNT`, and `ZGCOUNT` keywords must not*must not* be used unless the original uncompressed image was contained in an IMAGE `IMAGE` extension.

The FITS *FITS* header of the compressed image *may* contain other keywords. If a FITS *FITS* primary array or IMAGE `IMAGE` extension is compressed using the procedure described here, it is *strongly recommended* that all the keywords (including comment fields) in the header of the original image, except for the mandatory keywords mentioned above, be copied verbatim and in the same order into the header of the binary table binary-table extension that contains the compressed image. All these keywords will have the same meaning and interpretation as they did in the original image, even in cases where the keyword is not normally expected to occur in the header of a binary table binary-table extension (e.g., the `BSCALE` and `BZERO` keywords, or the World Coordinate System world-coordinate-system keywords such as `CTYPE`*n*, `CRPIX`*n*, and `CRVAL`*n*).

10.1.3. Table Columnscolumns

Two columns in the FITS *FITS* binary table are defined below to contain the compressed image tiles; the order of the columns in the table is not significant. One of the table columns describes optional content; but when this column appears it must*must* be used as defined in this section. The column names (given by the `TTYPE`*n* keyword) are reserved; they are shown here in upper case upper-case letters, but case is not significant.

DRAFT

COMPRESSED_DATACOMPRESSED_DATA – [required; variable-length; *required*] Each row of this column mustmust contain the byte stream that is generated as a result of compressing the corresponding image tile. The data type of the column (as given by the TFORM*n* keyword) mustmust be one of '1PB', '1PI', or '1PJ''1PB', '1PI', or '1PJ' (or the equivalent '1QB', '1QI', or '1QJ''1QB', '1QI', or '1QJ'), depending on whether the compression algorithm generates an output stream of 8-bit bytes, or integers of 16-, or 32-bits16, or 32 bits respectively.

When using the quantization method to compress floating-point images that is described in Sect. 10.2, it sometimes may not be possible to quantize some of the tiles (e.g., if the range of pixels values is too large or if most of the pixels have the same value and hence the calculated RMS noise level in the tile is close to zero). There also may be other rare cases where the nominal compression algorithm cannot be applied to certain tiles. In these cases, an alternate technique *may* be used in which the raw pixel values are losslessly compressed with the GZIP Gzip algorithm.

GZIP_COMPRESSED_DATAGZIP_COMPRESSED_DATA [optional; variable-length; *optional*] If the raw pixel values in an image tile are losslessly compressed with the GZIP Gzip algorithm, the resulting byte stream mustmust be stored in this column (with a '1PB'or '1QB''1PB' or '1QB' variable-length array column array-column format). The corresponding COMPRESSED_DATACOMPRESSED_DATA column for these tiles mustmust contain a null pointer (i.e., the pair of integers that constitute the descriptor for the column *must* both have the value zero: see Sect. 7.3.5).

The compressed data columns described above *may* use either the '1P' or '1Q' variable-length array FITS *FITS* column format if the size of the heap in the compressed FITS *FITS* file is < 2.1 GB. If the the heap is larger than 2.1 GB, then the '1Q' format (which uses 64-bit pointers) mustmust be used.

When using the optional *optional* quantization method described in Sect. 10.2 to compress floating-point images, the following columns are required*required*.

ZSCALE – [floating-point; optional*optional*] This column *shall* be used to contain linear scale factors that, along with ZZERO, transform the floating-point pixel values in each tile to integers via,

$$Formula - ok - gives - latexdiff - errors \quad (12)$$

where I_i and F_i are the integer and (original) floating-point values of the image pixels, respectivelyand the round, and the round function rounds the result to the nearest integer value.

ZZERO – [floating-point; optional] This column *shall* be used to contain zero point zero-point offsets that are used to scale the floating-point pixel values in each tile to integers via Eq. **??**.

Do not confuse the ZSCALE and ZZEROZSCALE and ZZERO columns with the BSCALE and BZERO keywords (defined in Sect. 4.4.2) which that may be present in integer FITS *FITS* images. Any such integer images should *should* normally be compressed without any further scaling, and the BSCALEand BZEROBSCALE and BZERO keywords *should* be copied verbatim into the header of the binary table containing the compressed image.

Some images contain undefined pixel values; in uncompressed floating-point images these pixels have an IEEE NaN value. However, these pixel values will be altered when using the quantization method described in Sect. 10.2 to compress floating-point images. The value of the undefined pixels *may* be preserved in the following way.

ZBLANKZBLANK – [integer; optional*optional*] When present, this column *shall* be used to store the integer value that represents undefined pixels in the scaled integer array. The recommendedvalue for ZBLANKis −2147483648*recommended* value for ZBLANK is −2147483648, the largest negative 32-bit integer. If the same null value is used in every tile of the image, then ZBLANKZBLANK *may* be given in a header keyword instead of a table column; if both a keyword and a table column named ZBLANKZBLANK are present, the values in the table column mustmust be used. If there are no undefined pixels in the image then is not requiredZBLANK is *not required* to be present either as a table column or a keyword.

If the uncompressed image has an integer data type (ZBITPIX > 0) then the value of undefined pixels is given by the BLANK keyword (see Sect. 5.3), which *should* be used instead of ZBLANK. ZBLANK.

When using some compression techniques that do not exactly preserve integer pixel values, it may be necessary to store the location of the undefined pixels prior to compressing the image. The locations *may* be stored in an image mask, which *must* itself be compressed and stored in a table column with the following definition. See Sect. 10.2.2 for more details.

NULL_PIXEL_MASKNULL_PIXEL_MASK – [integer array; optional*optional*] When present, this column *shall* be used to store, in compressed form, an image mask with the same original dimensions as the uncompressed image, that records the location of the undefined pixels. The process defined in Sect. 10.2.2 *shall* be used to construct the compressed pixel mask.

Additional columns *may* be present in the table to supply other parameters that relate to each image tile. However, these parameters should notshould *not* be recorded in the image HDU when the uncompressed image is restored.

10.2. Quantization of Floating-Point Datafloating-point data

While floating-point format images may be losslessly compressed, noisy images often do not compress very well. Higher compression can only be achieved by removing some of this noise without losing the useful information content. One commonly used technique for reducing the noise is to scale the floating-point values into quantized integers using Eq. **??**, and using the ZSCALEand ZZEROZSCALE and ZZERO columns to record the two scaling coefficients that are used for each tile. Note that the absence of these two columns in a tile-compressed floating-point image is an indication that the image was not scaled, and was instead losslessly compressed.

DRAFT

An effective scaling algorithm for preserving a specified amount of noise in each pixel value is described by White & Greenfield (1999) and by Pence et al. (2009). With this method, the ZSCALE ZSCALE value (which is numerically equal to the spacing between adjacent quantization levels) is calculated to be some fraction, QQ, of the RMS noise as measured in background regions of the image. Pence et al. (2009) shows that the number of binary bits of noise that are preserved in each pixel value is given by $log_2(Q) + 1.792$. Q results directly related to The Q value directly affects the compressed file size: decreasing Q Q by a factor of 2 two will decrease the file size by about 1 bit /one bit per pixel. In order to achieve the greatest amount of compression, one should use the smallest value of Q Q that still preserves the required amount of photometric and astrometric precision in the image. Image quality will remain comparable regardless of the noise level.

A potential problem when applying this scaling method to astronomical images, in particular, is that it can lead to a systematic bias in the measured intensities in faint parts of the image: . As the image is quantized more coarsely, the measured intensity of the background regions of the sky will tend to be biased towards the nearest quantize level. One very effective technique for minimizing this potential bias is to *dither* the quantized pixel values by introducing random noise during the quantization process. So instead of simply scaling every pixel value in the same way using Eq. **??**, the quantized levels are randomized by using this slightly modified equation:

$$Formula - ok - gives - latexdiff - errors \quad (13)$$

where R_i is a random number between 0.0 and 1.0, and 0.5 is subtracted so that the mean quantity equals 0. Then restoring the floating-point value, the same R_i is used with the inverse formula:

$$F_i = ((I_i - R_i + 0.5) * \text{ZSCALE}\text{ZSCALE}) + \text{ZZERO}\text{ZZERO}. \quad (14)$$

This "subtractive dithering" technique has the effect of dithering the zero-point zero point of the quantization grid on a pixel by pixel pixel-by-pixel basis without adding any actual noise to the image. The net effect of this is that the mean (and median) pixel value in faint regions of the image more closely approximate the value in the original unquantized image than if all the pixels are scaled without dithering.

The key requirement when using this subtractive dithering subtractive-dithering technique is that *the exact same random number random-number sequence* must *must* be used when quantizing the pixel values to integers, and when restoring them to floating point floating-point values. While most computer languages supply a function for generating random numbers, these functions are not guaranteed to generate the same sequence of numbers every time. An algorithm for generating a repeatable sequence of pseudo random pseudo-random numbers is given in Appendix I; this algorithm *must* be used when applying a subtractive dither.

10.2.1. Dithering Algorithmsalgorithms

The ZQUANTIZ keyword, if present, mustmust have one of the following values to indicate the type of quantization, if any, that was applied to the floating-point image for compression: .

NO_DITHER'NO_DITHER' – No dithering was performed; the floating-point pixels were simply quantized using Eq. **??**. This option *shall* be assumed if the ZQUANTIZ keyword is not present in the header of the compressed floating-point image.

SUBTRACTIVE_DITHER_1'SUBTRACTIVE_DITHER_1' – The basic subtractive dithering was performed, the algorithm for which is described below. Note that an image quantized using this technique can still be unquantized using the simple linear scaling function given by Eq. **??**, at the cost of introducing slightly more noise in the image than if the full subtractive dithering subtractive-dithering algorithm were applied.

SUBTRACTIVE_DITHER_2'SUBTRACTIVE_DITHER_2' – This dithering algorithm is identical to that for 'SUBTRACTIVE_DITHER_1', except that any pixels in the floating-point image that are exactly equal to 0.0 are represented by the reserved value −2147483647 in the quantized integer array. When the image is subsequently uncompressed decompressed and unscaled, these pixels must be *must* be restored to their original value of 0.0. This dithering option is useful if the zero-valued pixels have special significance to the data analysis software, so that the value of these pixels must not*must not* be dithered.

The process for generating a subtractive dither for a floating-point image is the following: .

1. Generate a sequence of 10000 single-precision floating-point random numbers, RN, with a value between 0.0 and 1.0. Since it could be computationally expensive to generate a unique random number for every pixel of large images, simply cycle through this look-up table of random numbers.
2. Choose an integer in the range 1 to 10000 to serve as an initial seed value for creating a unique sequence of random numbers from the array that was calculated in the previous step. The purpose of this is to reduce the chances of applying the same dithering pattern to two images that are subsequently subtracted from each other (or co-added), because the benefits of randomized dithering are lost if all the pixels are dithered in phase with each other. The exact method for computing this seed integer is not important as long as the value is chosen more or less randomly.
3. Write the integer seed value that was selected in the previous step as the value of the ZDITHER0ZDITHER0 keyword in the header of the compressed image. This value is required to recompute the same dithering pattern when uncompressing decompressing the image.
4. Before quantizing each tile of the floating point floating-point image, calculate an initial value for two offset parameters, I_0 and I_1, with the following formulae:

$$I_0 = mod\text{mod}(N_{tile}\text{tile} - 1 + \text{ZDITHER0}\text{ZDITHER0}, 10000)(15)$$

$$I_1 = \text{INT}(\text{RN}(I_0) * 500.)(16)$$

where N_{tile} N_{tile} is the row number in the binary table that is used to store the compressed bytes for that tile, ZDITHER0ZDITHER0 is that value of that keyword, and $\text{RN}(I_0)$ is the value of the I_0^{th} I_0^{th} random number in the sequence that was computed in the first step. Note that I_0 has a

DRAFT

value in the range 0 to 9999 and I_1 has a value in the range 0 to 499. This method for computing I_0 and I_1 was chosen so that a different sequence of random numbers is used to compress successive tiles in the image, and so that the sequence of I_1 values has a length of order 100 million 100-million elements before repeating.

5. Now quantize each floating-point pixel in the tile using Eq. **??** and using random number RN(I_1) for the first pixel. Increment the value of I_1 for each subsequent pixel in the tile. If I_1 reaches the upper limit of 500, then increment the value of I_0 and recompute I_1 from Eq. 16. If I_0 also reaches the upper limit of 10000, then reset I_0 to 0.
   If the floating-point pixel has an IEEE NaN value, then it is not quantized or dithered but instead is set to the reserved integer value specified by the ZBLANKZBLANK keyword. For consistency, the value of I_1 should *should* also be incremented in this case even though it is not used.
6. Compress the array of quantized integers using the lossless algorithm that is specified by the ZCMPTYPEZCMPTYPE keyword (use RICE_1 'RICE_1' by default).
7. Write the compressed bytestream into the COMPRESSED_DATAbyte stream into the COMPRESSED_DATA column in the appropriate row of the binary table corresponding to that tile.
8. Write the linear scaling and zero point zero-point values that were used in Eq. **??** for that tile into the ZSCALEand ZZEROZSCALE and ZZERO columns, respectively, in the same row of the binary table.
9. Repeat Steps 4 through 8 for each tile of the image.

10.2.2. Preserving undefined pixels with lossy compression

The undefined pixels in integer images are flagged by a reserved BLANK value and will be preserved if a lossless compression algorithm is used. (ZBLANK is used for undefined pixels in floating-point images.) If the image is compressed with a lossy algorithm, then some other technique must *must* be used to identify the undefined pixels in the image. In this case it is recommended*recommended* that the undefined pixels be recorded with the following procedure: .

1. Create an integer data mask with the same dimensions as the image tile.
2. For each undefined pixel in the image, set the corresponding mask pixels to 1 and all the other pixels to 0.
3. Compress the mask array using a lossless algorithm such as PLIO or GZIPGzip, and record the name of that algorithm with the keyword ZMASKCMP.
4. Store the compressed byte stream in a variable-length array column called 'NULL_PIXEL_MASK' variable-length-array column called NULL_PIXEL_MASK in the table row corresponding to that image tile.

The data mask array pixels *should* have the shortest integer data type that is supported by the compression algorithm (i.e., usually 8-bit eight-bit bytes). When uncompressing decompressing the image tile, the software must*must* check if the corresponding compressed data mask exists with a length greater than 0, and if so, uncompress decompress the mask and set the corresponding undefined pixels in the image array to the value given by the BLANK keyword.

10.3. Tiled Table Compressiontable compression

The following section describes the process for compressing the content of BINTABLE BINTABLE columns. Some additional details of BINTABLE BINTABLE compression may be found in Pence et al. (2013), but the specifications in this Standard *shall* supersede those in the registered convention. The uncompressed table may *may* be subdivided into tiles, each containing a subset of rows, then each column of data within each tile is extracted, compressed, and stored as a variable-length array of bytes in the output compressed table. The header keywords from the uncompressed table, with only a few limited exceptions, *shall* be copied verbatim to the header of the compressed table. The compressed table *must* itself be a valid FITS *FITS* binary table (albeit one where the contents cannot be interpreted without uncompressing decompressing the contents) that contains the same number and order of columns as in the uncompressed table, and that contains one row for each tile of rows in the uncompressed table. Only the compression algorithms specified in Sect. 10.3.5 are permitted.

10.3.1. Required Keywordskeywords

With only a few exceptions noted below, all the keywords and corresponding comment fields from the uncompressed table must*must* be copied verbatim, in order, into the header of the compressed table. Note in particular that the values of the reserved column descriptor keywords TTYPEn, TUNIT*n*, TSCAL*n*, TZERO*n*, TNULL*n*, TDISP*n*, and TDIM*n*, as well as all the column-specific WCS keywords defined in the FITS standard, must*FITS* Standard, *must* have the same values and data types in both the original and in the compressed table, with the understanding that these keywords apply to the uncompressed data values.

The only keywords that must not*must not* be copied verbatim from the uncompressed table header to the compressed table header are the mandatory NAXIS1, NAXIS2, PCOUNT, and TFORM*n* keywords, and the optional CHECKSUM, DATASUM (see Sect. 4.4.2.7), and THEAP keywords. These keywords must necessarily describe the contents and structure of the compressed table itself. The original values of these keywords in the uncompressed table must*must* be stored in a new set of reserved keywords in the compressed table header. Note that there is no need to preserve a copy of the GCOUNT keyword because the value is always equal to 1 for BINTABLES1 for BINTABLE extensions. The complete set of keywords that have a reserved meaning within a tile-compressed binary table are given below: .

ZTABLE – [logical; value: 'T'T] The value field of this keyword *shall*be 'T' be T to indicate that the FITS binary table *FITS* binary-table extension contains a compressed BINTABLEBINTABLE, and that logically this extension *should* be interpreted as a tile-compressed binary table.

ZNAXIS1 – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the NAXIS1 keyword in the original uncompressed FITS *FITS* table header. This represents the width in bytes of each row in the uncompressed table.

ZNAXIS2 – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the NAXIS2 keyword in the original uncompressed FITS *FITS* table header. This represents the number of rows in the uncompressed table.

DRAFT

ZPCOUNT – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the PCOUNT keyword in the original uncompressed FITS *FITS* table header.

ZFORM*n* – [string; indexed; default: none] The value field of these keywords *shall*contain the character string contain the character-string values of the corresponding TFORM*n* keywords that defines the data type of column Column *n* in the original uncompressed FITS *FITS* table.

ZCTYP*n* – [string; indexed; default: none] The value field of these keywords *shall*contain the character string contain the character-string value mnemonic name of the algorithm that was used to compress column nColumn *n* of the table. The only permitted values are given in Sect. 10.3.5, and the corresponding algorithms are described in Sect. 10.4.

ZTILELEN – [integer; default: none] The value field of this keyword *shall* contain an integer representing the number of rows of data from the original binary table that are contained in each tile of the compressed table. The number of rows in the last tile may be less fewer than in the previous tiles. Note that if the entire table is compressed as a single tile, then the compressed table will only contains a single row, and the ZTILELEN and ZNAXIS2 keywords will have the same value.

10.3.2. Procedure for Table Compressiontable compression

The procedure for compressing a FITS *FITS* binary table consists of the following sequence of steps: .

1. Divide table into tiles (*optional*)
   In order to limit the amount of data that must be managed at one time, large FITS *FITS* tables *may* be divided into tiles, each containing the same number of rows (except for the last tile, which *may* contain fewer rows). Each tile of the table is compressed in order, and each is stored in a single row in the output compressed table. There is no fixed upper limit on the allowed tile size, but for practical purposes it is recommended*recommended* that it not exceed 100 MB.
2. Decompose each tile into the component columns

   FITS *FITS* binary tables are physically stored in row-by-row sequential order, such that the data values for the first row in each column are followed by the values in the second row, and so on (see Sect. 7.3.3). Because adjacent columns in binary tables can contain very non-homogeneous types of data, it can be challenging to efficiently compress the native stream of bytes in the FITS *FITS* tables. For this reason, the table is first decomposed into its component columns, and then each column of data is compressed separately. This also allows one to choose the most efficient most-efficient compression algorithm for each column.
3. Compress each column of data
   Each column of data *must* be compressed with one of the lossless compression algorithms described in Sect. 10.4. If the table is divided into tiles, then the same compression algorithm *must* be applied to a given column in every tile. In the case of variable-length array columns (where the data are stored in the table heap: see Sect. 7.3.5), each individual variable length vector mustvariable-length vector *must* be compressed separately.
4. Store the compressed bytes
   The compressed stream of bytes for each column must*must* be written into the corresponding column in the output table. The compressed table must*must* have exactly the same number and order of columns as the input table, however, the data type of the columns in the output table will all have a variable-length byte data type, with TFORMn = '1QB'TFORM*n* = '1QB'. Each row in the compressed table corresponds to a tile of rows in the uncompressed table.
   In the case of variable-length array columns, the array of descriptors that point to each compressed variable-length array, as well as the array of descriptors from the input uncompressed table, must*must* also be compressed and written into the corresponding column in the compressed table. See Sect. 10.3.6 for more details.

10.3.3. Compression Directive Keywordsdirective keywords

The following compression-directive keywords, if present in the header of the table to be compressed, are reserved to provide guidance to the compression software on how the table should be compressed. The compression software *should* attempt to obey these directives, but if that is not possible the software may *may* disregard them and use an appropriate alternative. These keywords are optional, but must *must* be used as specified below.

- FZTILELN – [integer] The value field of this keyword *shall* contain an integer that specifies the requested number of table rows in each tile which that are to be compressed as a group.
- FZALGOR – [string] The value field of this keyword *shall* contain a character string giving the mnemonic name of the algorithm that is requested to be used by default to compress every column in the table. The permitted values are given in Sect. 10.3.5.
- FZALG*n* – [string; indexed] The value fields of these keywords *shall* contain a character string giving the mnemonic name of the algorithm that is requested to compress column nColumn *n* of the table. The current allowed values are the same as for the FZALGORFZALGOR keyword. The FZALGnFZALGn keyword takes precedence over FZALGORFZALGOR in determining which algorithm to use for a particular column if both keywords are present.

10.3.4. Other Reserved Keywordsreserved keywords

The following keywords are reserved to store a verbatim copy of the value and comment fields for specific keywords in the original uncompressed BINTABLEBINTABLE. These keywords, if present, *should* be used to reconstruct an identical copy of the uncompressed BINTABLE, and should notBINTABLE, and *should not* appear in the compressed table header unless the corresponding keywords were present in the uncompressed BINTABLEBINTABLE.

ZTHEAP – [integer; default: none] The value field of this keyword *shall* contain an integer that gives the value of the THEAP keyword if present in the original uncompressed FITS *FITS* table header.

ZHECKSUM – [string; default: none] The value field of this keyword *shall* contain a character string that gives the value of

DRAFT

the CHECKSUM keyword (see Sect. 4.4.2.7) in the original uncompressed FITS *FITS* HDU.

ZDATASUM – [string; default: none] The value field of this keyword *shall* contain a character string that gives the value of the DATASUM keyword (see Sect. 4.4.2.7) in the original uncompressed FITS *FITS* HDU.

10.3.5. Supported Compression Algorithms compression algorithms for Tablestables

The permitted algorithms for compressing BINTABLE columns are , , and (plus BINTABLE columns are 'RICE_1', 'GZIP_1', and 'GZIP_2' (plus 'NOCOMPRESS'), which are lossless and are described in Sect. 10.4. Lossy compression could be allowed in the future once a process is defined to preserve the details of the compression.

10.3.6. Compressing Variable-Length Array Columnsvariable-length array columns

Compression of BINTABLE BINTABLE tiles that contain variable-length array (VLA) columns requires special consideration because the array values in these columns are not stored directly in the table, but are instead stored in a data heap, which follows the main table (see Sect. 7.3.5). The VLA column in the original, uncompressed table only contains descriptors, which are composed of comprise two integers that give the size and location of the arrays in the heap. When uncompressingdecompressing, these descriptor values will be needed to write the uncompressed decompressed VLAs back into the same location in the heap as in the original uncompressed table. Thus, the following process *must* be followed, in order, when compressing a VLA column within a tile: . Refer to Pence et al. (2013) for additional details.

1. For each VLA in the column:
   - Read read the array from the input table, and compress it using the algorithm specified by ZCTYP for this VLA column. ;
   - Write the resulting bytestream write the resulting byte stream to the heap of the compressed table. ; and
   - Store store (or append) the descriptors to the compressed bytestream byte stream (which *must* be 64-bit Q-type) in a temporary array.
2. Append the VLA descriptors from the uncompressed table (which *may* be either Q-type or P-type) to the temporary array of VLA descriptors for the compressed table.
3. Compress the combined array of descriptors using 'GZIP_1', and write that byte stream into the corresponding VLA column in the output table, so that the compressed array is appended to the heap.

When uncompressing decompressing a VLA column, two stages of uncompression decompression *must* be performed in order: .

1. Uncompress Decompress the combined array of descriptors using the Gzip algorithm.
2. For each descriptor to a compressed array:
   - Read read the compressed VLA from the compressed tableand uncompress , and decompress it using the algorithm specified by ZCTYP for this VLA column. ; and
   - Write write it to the correct location in the uncompressed decompressed table.

10.4. Compression Algorithmsalgorithms

Table 36: Valid mnemonic values for the ZCMPTYPE and ZCTYP*n* keywords

Value	Sect.	Compression Type
'RICE_1'	10.4.1	Rice algorithm for integer data
'GZIP_1'	10.4.2	Combination of the LZ77 algorithm and Huffman coding, used in Gnu GZIP GNU Gzip
'GZIP_2'	10.4.2	Like 'GZIP_1', but with reshuffled pixel byte values
'PLIO_1'	10.4.3	IRAF PLIO algorithm for integer data
'HCOMPRESS_1'	10.4.4	H-compress algorithm for 2-D two-dimensional images
'NOCOMPRESS'		The HDU remains uncompressed

The name of the permitted algorithms for compressing FITS *FITS* HDUs, as recorded in the ZCMPTYPE keyword, are listed in Table 36; if other types are later supported, they must*must* be registered with the IAUFWG to reserve the keyword values. Keywords for the parameters of supported compression algorithms have also been reserved, and are described with each algorithm in the subsections below. If alternative compression algorithms require keywords beyond those defined below, they must*must* also be registered with the IAUFWG to reserve the associated keyword names.

10.4.1. Rice compression

When ZCMPTYPE = 'RICE_1' ZCMPTYPE = 'RICE_1', the Rice algorithm (Rice et al. 1993) *shall* be used for data (de)compression. When selected, the keywords in Table 37 *should* also appear in the header with one of the values indicated. If these keywords are absent, then their default values must*must* be used. The Rice algorithm is lossless, but can only be applied to integer-valued arrays. It offers a significant performance advantage over the other compression techniques (see White et al. 2013).

Table 37: Keyword parameters for Rice compression

Keyword	Values: Permitted	Values: Default	Meaning
ZNAME1	'BLOCKSIZE'	–	Size of block in pixels
ZVAL1	16, 32 16, 32	32 32	No. of pixels in a block
ZNAME2	'BYTEPIX'	–	Size of pixel value in bytes
ZVAL2	1, 2, 4, 8 1, 2, 4, 8	4 4	No. 8-bit of eight-bit bytes per original pixel value

DRAFT

10.4.2. GZIP Gzip compression

When ZCMPTYPE = 'GZIP_1' the gzip ZCMPTYPE = 'GZIP_1', the Gzip algorithm *shall* be used for data (de)compression. There are no algorithm parameters, so the keywords ZNAME*n* and ZVAL*n* should notshould not appear in the header. The gzip Gzip algorithm is used in the free GNU software compression utility of the same name. It was created by J.-L. Gailly and M. Adler, based on the DEFLATE algorithm (Deutsch 1996), which is a combination of LZ77 (Ziv & Lempel 1977) and Huffman coding. The unix gzipUnix `gzip` program accepts an integer parameter that provides a trade between optimization for speed (1) and compression ratio (9), which does not affect the format of the resultant data stream. The selection of this parameter is an implementation detail that is not covered by this Standard.

When ZCMPTYPE = 'GZIP_2' ZCMPTYPE = 'GZIP_2', the gzip2 algorithm *shall* be used for data (de)compression. The gzip2 algorithm is a variation on 'GZIP_1'. There are no algorithm parameters, so the keywords ZNAME*n* and ZVAL*n* should notshould not appear in the header. In this case the bytes in the array of data values are shuffled so that they are arranged in order of decreasing significance before being compressed. For example, a 5-element five-element contiguous array of 2-byte two-byte (16-bit) integer values, with an original big-endian byte order of:

$$A_1A_2B_1B_2C_1C_2D_1D_2E_1E_2$$

will have the following byte order after shuffling:

$$A_1B_1C_1D_1E_1A_2B_2C_2D_2E_2,$$

where A_1, B_1, C_1, D_1, and E_1 are the most significant most-significant bytes from each of the integer values. Byte shuffling *shall*only *only* be performed for integer or floating-point numeric data types; logical, bit, and character types must notmust not be shuffled.

10.4.3. IRAF/PLIO compression

When ZCMPTYPE = 'PLIO_1' ZCMPTYPE = 'PLIO_1', the IRAF PLIO algorithm *shall* be used for data (de)compression. There are no algorithm parameters, so the keywords ZNAME*n* and ZVAL*n* should notshould not appear in the header. The PLIO algorithm was developed to store integer-valued image masks in a compressed form. The compression algorithm used is based on run-length encoding, with the ability to dynamically follow level changes in the image, in principle allowing a 16-bit encoding to be used regardless of the image depth. However, this algorithm has only been implemented in a way that supports image depths of no more than 12 bits; therefore 'PLIO_1''PLIO_1' *must*only *only* be used for integer image types with values between 0 and 2^{24}.

The compressed line lists are stored as variable length variable-length arrays of type short integer (16 bits per list element), regardless of the mask depth. A line list consists of a series of simple instructions, which are executed in sequence to reconstruct a line of the mask. Each 16 bit 16-bit instruction consists of the sign bit (not used), a three bit three-bit opcode, and twelve bits of data, i.e. : as depicted below.

```
+--+--------+-------------------+
|16|15    13|12                1|
+--+--------+-------------------+
|  | opcode |       data        |
+--+--------+-------------------+
```

The significance of the data depends upon the instruction. In order to reconstruct a mask line, the application executing these instructions is required *required* to keep track of two values, the current high value and the current position in the output line. The detailed operation of each instruction is given in Table 38.

Table 38: PLIO Instructions

Instr.	Opcode	Meaning
'ZN'	'00'	Zero the next N *N* output pixels.
'HN'	'04'	Set the next N *N* output pixels to the current high value.
'PN'	'05'	Zero the next N-1 $N-1$ output pixels, and set pixel N Pixel *N* to the current high value.
'SH'	'05'	Set the high value (absolute rather than incremental), taking the high 15 bits from the next word in the instruction stream, and the low 12 bits from the current data value.
'IH,DH'	'02,03'	Increment ('IH'IH') or decrement ('DH'DH') the current high value by the data value. The current position is not affected.
'IS,DS'	'06,07'	Increment ('IS'IS') or decrement ('DS'DS') the current high value by the data value, and step, i.e., output one high value.

The high value must*must* be set to 1 at the beginning of a line, hence the IH,DH and IS,DS 'IH,DH' and 'IS,DS' instructions are not normally needed for Boolean masks.

10.4.4. H-Compress algorithm

When ZCMPTYPE = 'HCOMPRESS_1' ZCMPTYPE = 'HCOMPRESS_1', the H-compress algorithm *shall* be used for data (de)compression. The algorithm was described by White (1992), and can be applied only to images with two dimensions. Briefly, the compression method is to apply, in order:

1. a wavelet transform called the H-transform (a Haar transform generalized to two dimensions), followed by
2. a quantization that discards noise in the image while retaining the signal on all scales, followed by and finally
3. a quadtree coding of the quantized coefficients.

The H-transform is a two-dimensional generalization of the Haar transform. The H-transform is calculated for an image of size $2^N \times 2^N$ as follows: .

1. Divide the image up into blocks of 2×2 pixels. Call the four pixel values in a block a_{00}, a_{10}, a_{01}, and a_{11}.
2. For each block compute four coefficients:
   $h_0 = (a_{11} + a_{10} + a_{01} + a_{00})/(\mathrm{SCALE} * \sigma)$
   $h_x = (a_{11} + a_{10} - a_{01} - a_{00})/(\mathrm{SCALE} * \sigma)$
   $h_y = (a_{11} - a_{10} + a_{01} - a_{00})/(\mathrm{SCALE} * \sigma)$
   $h_c = (a_{11} - a_{10} - a_{01} + a_{00})/(\mathrm{SCALE} * \sigma)$
   where SCALE is an algorithm parameter defined below, and σ characterizes the RMS noise in the uncompressed image.

DRAFT

3. Construct a $2^{N-1} \times 2^{N-1}$ image from the h_0 values for each 2×2 block. Divide that image up into 2×2 blocks and repeat the above calculation. Repeat this process N times, reducing the image in size by a factor of 2 two at each step, until only one h_0 value remains.

This calculation can be easily inverted to recover the original image from its transform. The transform is exactly reversible using integer arithmetic. Consequently, the program can be used for either lossy or lossless compression, with no special approach needed for the lossless case.

Noise in the original image is still present in the H-transform, however. To compress noisy images, each coefficient can be divided by SCALE $* \sigma$, where SCALE ~ 1 is chosen according to how much loss is acceptable. This reduces the noise in the transform to 0.5/SCALE, so that large portions of the transform are zero (or nearly zero) and the transform is highly compressible.

There is one user-defined parameter associated with the H-Compress algorithm: a scale factor to the RMS noise in the image that determines the amount of compression that can be achieved. It is not necessary to know what scale factor was used when compressing the image in order to uncompress decompress it, but it is still useful to record it. The keywords in Table 39 *should* be recorded in the header for this purpose.

Table 39: Keyword parameters for H-compression

	Values		
Keyword	Permitted	Default	Meaning
ZNAME1	'SCALE'	– '-'	Scale factor
ZVAL1	0.0 0.0 or larger	0.0 0.0	Scaling of the RMS noise; 0.0 yields lossless compression

Scale Factor – The floating-point scale parameter (whose value is stored in Keyword ZVAL1) determines the amount of compression; higher values result in higher compression, but with greater loss of information. SCALE =0.0 0.0 is a special case that yields lossless compression, i.e. the decompressed image has exactly the same pixel values as the original image. SCALE > 0.0 leads to lossy compression, where SCALE determines how much of the noise is discarded.

DRAFT

Appendix A: Syntax of keyword records

This Appendix appendix is not part of the FITSstandard FITS Standard but is included for convenient reference.

:=	means 'is defined to be'
X | Y	means one of X or Y *X or Y* (no ordering relation is implied)
[X]	means that X *X* is *optional*
X...	means X *X* is repeated one or more times
'B'	means the ASCII character B
'A'–'Z'	means one of the ASCII characters A through Z in the ASCII collating sequence, as shown in Appendix D
\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*FITS* free-format keyword records, as well as for long-string keywords spanning more than one keyword record).

FITS_keyword :=
    single_record_keyword |
    long_string_keyword

single_record_keyword :=
    FITS_keyword_record

FITS_keyword_record :=
    FITS_commentary_keyword_record |
    FITS_value_keyword_record

FITS_commentary_keyword_record :=
    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_keyword_record must*must* be exactly equal to 80.}

FITS_value_keyword_record :=
    keyword_field value_indicator [space...] [value]
        [space...] [comment]
{Constraint: The total number of characters in a FITS_value_keyword_record must*must* be exactly equal to 80.}
{Comment: If the value field is not present, the value of the FITS*FITS* keyword is not defined.}

long_string_keyword :=
    initial_kwd_record [continuation_kwd_record...]
    last_continuation_record
{Comment: the value of a long_string_keyword is reconstructed by concatenating the partial_string_values of the initial_kwd_record and of any continuation_kwd_records in the order they occur, and the character_string_value of the last_continuation_record.}

initial_kwd_record :=
    keyword_field value_indicator [space...]
    [partial_string_value] [space...] [comment]
{Constraint: The total number of characters in an initial_kwd_record *must* be exactly equal to 80.}

continuation_kwd_record :=
    CONTINUE_keyword [space...]
    [partial_string_value] [space...] [comment]
{Constraint: The total number of characters in a continuation_kwd_record *must* be exactly equal to 80.}

last_continuation_record :=
    CONTINUE_keyword [space...]
    [character_string_value] [space...] [comment]
{Constraint: The total number of characters in a last_continuation_record *must* be exactly equal to 80.}

keyword_field :=
    [keyword_char...] [space...]
{Constraint: The total number of characters in the keyword_field must*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

CONTINUE_keyword :=
    'C' 'O' 'N' 'T' 'I' 'N' 'U' 'E'

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 |

DRAFT

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 character-string value but only serve as delimiters. Leading spaces are significant; trailing spaces are not.}

partial_string_value :=
begin_quote [string_text_char...] ampersand end_quote
{Constraint: The begin_quote, end_quote, and ampersand are not part of the character-string value but only serve respectively as delimiters or continuation indicator. }

begin_quote :=
quote

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

quote :=
\0x27

ampersand :=
'&'

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*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*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: Suggested time scale time-scale specification

The content of this Appendix appendix has been superseded by SectionSect. 9 of the formal Standard, which derives from Rots et al. (2015).

DRAFT

Appendix C: Summary of keywords

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for convenient reference.

All of the mandatory and reserved keywords that are defined in the standardStandard, except for the reserved WCS keywords that are discussed separately in Sect. 8, are listed in Tables C.1, C.2, and C.3. An alphabetized alphabetical list of these keywords and their definitions is available online: `http://heasarc.gsfc.nasa.gov/docs/fcg/standard_dict.html`.

Table C.1:

Mandatory FITS*FITS* keywords for the structures described in this document.

Primary HDU	Conforming extension	Image extension	ASCII-table extension	Binary-table extension	Compressed images[6]
SIMPLE SIMPLE	XTENSIONXTENSION	XTENSIONXTENSION[1]	XTENSIONXTENSION[2]	XTENSIONXTENSION[3]	ZIMAGE =T
BITPIXBITPIX	BITPIXBITPIX	BITPIXBITPIX	BITPIX = 8	BITPIX = 8	ZBITPIXZBITPIX
NAXISNAXIS	NAXISNAXIS	NAXISNAXIS	NAXIS = 2	NAXIS = 2	ZNAXISZNAXIS
NAXISnNAXIS*n*[4]	NAXISnNAXIS*n*[4]	NAXISnNAXIS*n*[4]	NAXIS1NAXIS1	NAXIS1NAXIS1	ZNAXISnZNAXIS*n*
END END	PCOUNTPCOUNT	PCOUNT = 0	NAXIS2NAXIS2	NAXIS2NAXIS2	ZCMPTYPEZCMPTYPE
	GCOUNTGCOUNT	GCOUNT = 1	PCOUNT = 0	PCOUNTPCOUNT	
	ENDEND	ENDEND	GCOUNT = 1	GCOUNT = 1	
			TFIELDSTFIELDS	TFIELDSTFIELDS	
			TFORMnTFORM*n*[5]	TFORMnTFORM*n*[5]	
			TBCOLnTBCOL*n*[5]	ENDEND	
			ENDEND		

[1]XTENSION=␣'IMAGE␣␣␣' XTENSION=␣'IMAGE␣␣␣' for the image extension. [2]XTENSION=␣'TABLE␣␣␣' for the ASCII table XTENSION=␣'TABLE␣␣␣' for the ASCII-table extension. [3]XTENSION=␣'BINTABLE' for the binary table XTENSION=␣'BINTABLE' for the binary-table extension. [4]Runs from 1 through the value of NAXISNAXIS. [5]Runs from 1 through the value of TFIELDSTFIELDS. [6]required Required in addition to the mandatory keywords for binary tables.

Table C.2:

Reserved FITS*FITS* keywords for the structures described in this document.

All[1] HDUs		Array[2] HDUs	ASCII-table extension	Binary-table extension	Compressed images	
DATE DATE	EXTNAMEEXTNAME	BSCALEBSCALE	TSCALnTSCAL*n*	TSCALnTSCAL*n*	ZTILEnZTILE*n*	FZ
DATE-OBSDATE-OBS	EXTVEREXTVER	BZEROBZERO	TZEROnTZERO*n*	TZEROnTZERO*n*	ZNAMEiZNAME*i*	FZ
ORIGINORIGIN	EXTLEVELEXTLEVEL	BUNITBUNIT	TNULLnTNULL*n*	TNULLnTNULL*n*	ZVALiZVAL*i*	FZ
AUTHORAUTHOR	EQUINOXEQUINOX	BLANKBLANK	TTYPEnTTYPE*n*	TTYPEnTTYPE*n*	ZMASKCMPZMASKCMP	
REFERENCREFERENC	EPOCHEPOCH[3]	DATAMAXDATAMAX	TUNITnTUNIT*n*	TUNITnTUNIT*n*	ZQUANTIZZQUANTIZ	
COMMENTCOMMENT	BLOCKEDBLOCKED[3]	DATAMINDATAMIN	TDISPnTDISP*n*	TDISPnTDISP*n*	ZDITHER0ZDITHER0	
HISTORYHISTORY	EXTENDEXTEND[4]		TDMAXnTDMAX*n*	TDIMnTDIM*n*	ZSIMPLEZSIMPLE	ZT
␣␣␣␣␣␣␣␣	TELESCOPTELESCOP		TDMINnTDMIN*n*	THEAPTHEAP	ZEXTENDZEXTEND	
OBJECTOBJECT	INSTRUMEINSTRUME		TLMAXnTLMAX*n*	TDMAXnTDMAX*n*	ZBLOCKEDZBLOCKED	
OBSERVEROBSERVER			TLMINnTLMIN*n*	TDMINnTDMIN*n*	ZTENSIONZTENSION	
CONTINUECONTINUE				TLMAXnTLMAX*n*	ZPCOUNTZPCOUNT	
INHERITINHERIT [5]				TLMINnTLMIN*n*	ZGCOUNTZGCOUNT	
CHECKSUMCHECKSUM					ZHECKSUMZHECKSUM	ZI
DATASUMDATASUM					ZDATASUMZDATASUM	ZI

[1]These keywords are further categorized in Table C.3. [2]Primary HDU, image IMAGE extension, user-defined HDUs with same array structure. [3]Deprecated. [4]Only permitted in the primary HDU. [5]Only permitted in extension HDUs, immediately following the mandatory keywords.

DRAFT

Table C.3:

General reserved FITS*FITS* keywords described in this document.

Production	Bibliographic	Commentary	Observation
DATE DATE	AUTHORAUTHOR	COMMENTCOMMENT	DATE-OBSDATE-OBS
ORIGINORIGIN	REFERENCREFERENC	HISTORYHISTORY	TELESCOPTELESCOP
BLOCKEDBLOCKED[1]		␣␣␣␣␣␣␣␣	INSTRUMEINSTRUME
			OBSERVEROBSERVER
			OBJECTOBJECT
			EQUINOXEQUINOX
			EPOCHEPOCH[1]

(1)Deprecated.

DRAFT

Table D.1: ASCII character set.

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

Appendix D: ASCII text

This appendix is not part of the FITSstandardFITS Standard; the material in it is based on the ANSI standard for ASCII (ANSI 1977) and is included here for informational purposes.)

In Table D.1, 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 the restricted set of ASCII text ASCII-text characters.

Appendix E: IEEE floating-point formats

The material in this Appendix appendix is not part of this standardStandard; it is adapted from the IEEE-754 floating-point standard (IEEE 1985) 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*FITS* recognizes all IEEE basic formats, including the special values.

E.1. Basic formats

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

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

Table E.1: Summary of format parameters.

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 biasbias	+127	unspecified	+1023	unspecified
Exponent width in bits	8	≥ 11	11	≥ 15
Format width in bits	32	≥ 43	64	≥ 79

Fig. E.1: Single Format. msbmsb means most significant bitmost-significant bit, lsblsb means least significant bitleast-significant bit

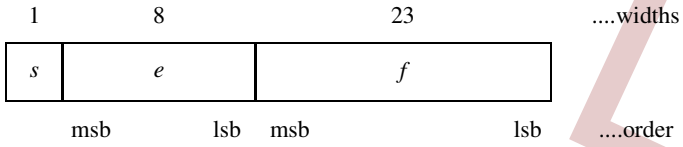
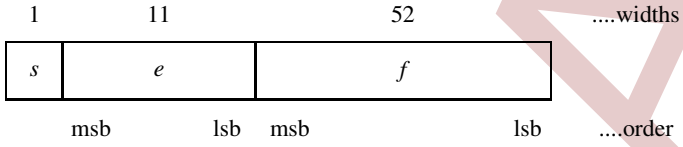


Fig. E.2: Double Format. msbmsb means most significant bitmost-significant bit, lsblsb means least significant bitleast-significant bit



The range of the unbiased exponent E shallshall 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 E.1. Each nonzero numerical value has just one encoding. The fields are interpreted as follows: .

E.1.1. Single

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

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

E.1.2. Double

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

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).

E.2. Byte patterns

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

DRAFT

Table E.2: IEEE floating-point formats.

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

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

DRAFT

Appendix F: Reserved extension type names

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes. It describes the extension type names registered as of the date this standard Standard was issued.) A current list is available from the FITSSupport Office web site *FITS* Support Office website at `http://fits.gsfc.nasa.gov`.

F.1. Standard extensions

These three extension types have been approved by the IAUFWG and are defined in Sect. 7 of this standard Standard document as well as in the indicated *Astronomy and Astrophysics* journal articles.

- `'IMAGE␣␣␣'` – This extension type provides a means of storing a multi-dimensional array similar to that of the FITS*FITS* primary header and data unit. Approved as a standard extension in 1994 (Ponz et al. 1994).
- `'TABLE␣␣␣'` – This ASCII table ASCII-table extension type contains rows and columns of data entries expressed as ASCII characters. Approved as a standard extension in 1988 (Harten et al. 1988).
- `'BINTABLE'` – This binary table binary-table extension type provides a more flexible more-flexible and efficient means of storing data structures than is provided by the TABLE`TABLE` extension type. The table rows can contain a mixture of numerical, logical, and character data entries. In addition, each entry is allowed to be a single dimensioned single-dimensioned array. Numeric data are kept in binary formats. Approved as a standard extension in 1994 (Cotton et al. 1995).

F.2. Conforming extensions

These conventions meet the requirements for a conforming extension as defined in in Sect. 3.4.1 of this standardStandard, however they have not been formally approved or endorsed by the IAUFWG.

- `'IUEIMAGE'` – This name was given to the prototype of the IMAGE`IMAGE` extension type and was primarily used in the IUE project data archive from approximately 1992 to 1994. Except for the name, the format is identical to the IMAGE`IMAGE` extension.
- `'A3DTABLE'` – This name was given to the prototype of the BINTABLE`BINTABLE` extension type and was primarily used in the AIPS data processing system developed at NRAO from about 1987 until it was replaced by BINTABLE`BINTABLE` in the early 1990s. The format is defined in the 'Going AIPS' manual (Cotton et al. 1990), Chapter 14. It is very similar to the BINTABLE`BINTABLE` type except that it does not support the variable-lengtharray-array convention.
- `'FOREIGN␣'` – This extension type is used to put a FITS*FITS* wrapper about an arbitrary file, allowing a file or tree of files to be wrapped up in FITS*FITS* and later restored to disk. A full description of this extension type is given in the Registry of FITS*FITS* conventions on the FITSSupport Office web site*FITS* Support Office website.
- `'DUMP␣␣␣␣'` – This extension type can be used to store a stream of binary data binary-data values. The only known use of this extension type is to record telemetry header packets for data from the Hinode mission. The more general FOREIGNmore-general `FOREIGN` extension type could also be used to store this type of data.

F.3. Other suggested extension names

There have been occasional suggestions for other extension names that might be used for other specific purposes. These include a COMPRESS`COMPRESS` extension for storing compressed images, a FITS*FITS* extension for hierarchically embedding entire FITS*FITS* files within other FITS*FITS* files, and a FILEMARK`FILEMARK` extension for representing the equivalent of an end-of-file mark on magnetic tape magnetic-tape media. None of these extension types have been implemented or used in practice, therefore these names are not reserved. These extension names (or any other extension name not specifically mentioned in the previous sections of this appendix) should not*should not* be used in any FITS*FITS* file without first registering the name with the IAU FITS *FITS* Working Group.

Appendix G: MIME types

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

RFC 4047 (Allen & Wells 2005) describes the registration of the Multipurpose Internet Mail Extensions (MIME) sub-types 'application/fits`application/fits`' and 'image/fits`image/fits`' to be used by the international astronomical community for the interchange of FITS*FITS* files. The MIME type serves as a electronic tag or label that is transmitted along with the FITS*FITS* file that tells the receiving application what type of file is being transmitted. The remainder of this appendix has been extracted verbatim from the RFC 4047 document.

The general nature of the full FITSstandard *FITS* Standard requires the use of the media type 'application/fits`application/fits`'. Nevertheless, the principal intent for a great many FITS*FITS* files is to convey a single data array in the primary HDU, and such arrays are very often 2-dimensional two-dimensional images. Several common image viewing applications already display single-HDU FITS*FITS* files, and the prototypes for virtual observatory virtual-observatory projects specify that data provided by web services be conveyed by the data array in the primary HDU. These uses justify the registration of a second media type, namely 'image/fits`image/fits`', for files which that use the subset of the standard Standard described by the original FITSstandard *FITS* Standard paper. The MIME type 'image/fits' may`image/fits`' *may* be used to describe FITS*FITS* primary HDUs that have other than two dimensions, however it is expected that most files described as 'image/fits`image/fits`' will have two-dimensional (NAXIS `NAXIS` =2 `2`) primary HDUs.

G.1. MIME type 'application/fits`application/fits`'

A FITS*FITS* file described with the media type 'application/fits' should`application/fits`' *should* conform to the published standards for FITS*FITS* files as determined by convention and agreement within the international FITS*FITS* community. No

other constraints are placed on the content of a file described as 'application/fitsapplication/fits'.

A FITS*FITS* file described with the media type 'application/fits' may application/fits' *may* have an arbitrary number of conforming extension HDUs that follow its mandatory primary header and data unit. The extension HDUs may *may* be one of the standard types (IMAGE, TABLE, and BINTABLEIMAGE, TABLE, and BINTABLE) or any other type that satisfies the 'Requirements for Conforming Extensionsconforming extensions' (Sect. 3.4.1). The primary HDU or any IMAGEextension may IMAGE extension *may* contain zero to 999 dimensions with zero or more zero-or-more pixels along each dimension.

The primary HDU may use the random groups *may* use the random-groups convention, in which the dimension of the first axis is zero and the keywords GROUPS, PCOUNTand GCOUNTGROUPS. PCOUNT and GCOUNT appear in the header. NAXIS1 NAXIS1 =0and GROUPS 0 and GROUPS =T T is the signature of random groups; see Sect. 6.

G.1.1. Recommendations for application writers

An application intended to handle 'application/fits' shouldapplication/fits' *should* be able to provide a user with a manifest of all of the HDUs that are present in the file and with all of the keyword/value pairs from each of the HDUs.

An application intended to handle 'application/fits' shouldapplication/fits' *should* be prepared to encounter extension HDUs that contain either ASCII or binary tables, and to provide a user with access to their elements.

An application which can modify FITSthat can modify *FITS* files or retrieve FITS*FITS* files from an external service should*should* be capable of writing such files to a local storage medium.

Complete interpretation of the meaning and intended use of the data in each of the HDUs typically requires the use of heuristics that attempt to ascertain which local conventions were used by the author of the FITS*FITS* file.

As examples, files with media type 'application/fitsapplication/fits' might contain any of the following contents: .

- An empty primary HDU (containing zero data elements) followed by a table HDU that contains a catalog of celestial objects.
- An empty primary HDU followed by a table TABLE HDU that encodes a series of time-tagged photon events from an exposure using an X-ray detector.
- An empty primary HDU followed by a series of IMAGEIMAGE HDUs containing data from an exposure taken by a mosaic of CCD detectors.
- An empty primary HDU followed by a series of table TABLE HDUs that contain a snapshot of the state of a relational database.
- A primary HDU containing a single image along with keyword/value pairs of metadata.
- A primary HDU with NAXIS1 NAXIS1 =0and GROUPS 0 and GROUPS =Tfollowed by random groups T followed by random-groups data records of complex fringe visibilities.

G.2. MIME type 'image/fitsimage/fits'

A FITS*FITS* file described with the media type 'image/fits' shouldimage/fits' *should* have a primary HDU with positive integer values for the NAXISand NAXISnNAXIS and NAXIS*n* keywords, and hence should*should* contain at least one pixel. Files with 4 four or more non-degenerate axes (NAXISnNAXIS*n* > 1) should*should* be described as 'application/fitsapplication/fits', not as 'image/fitsimage/fits'. (In rare cases it may be appropriate to describe a NULL image – a dataless container for FITS*FITS* keywords, with NAXIS NAXIS =0or NAXISn 0 or NAXIS*n* =0 0 – or an image with 4+ four or more non-degenerate axes as 'image/fitsimage/fits' but this usage is discouraged because such files may confuse simple image viewer image-viewer applications.)

FITS*FITS* files declared as 'image/fits' mayimage/fits' *may* also have one or more conforming extension HDUs following their primary HDUs. These extension HDUs may*may* contain standard, non-linear, world coordinate world-coordinate system (WCS) information in the form of tables or images. The extension HDUs may*may* also contain other, non-standard metadata pertaining to the image in the primary HDU in the forms of keywords and tables.

A FITS*FITS* file described with the media type 'image/fits' shouldimage/fits' *should* be principally intended to communicate the single data array in the primary HDU. This means that 'image/fits' should notimage/fits' *should not* be applied to FITS*FITS* files containing multi-exposure-frame mosaic images. Also, random groups files mustrandom-groups files *must* be described as 'application/fitsapplication/fits' and not as 'image/fitsimage/fits'.

A FITS*FITS* file described with the media type 'image/fitsimage/fits' is also valid as a file of media type 'application/fitsapplication/fits'. The choice of classification depends on the context and intended usage.

G.2.1. Recommendations for application writers

An application that is intended to handle 'image/fits' shouldimage/fits' *should* be able to provide a user with a manifest of all of the HDUs that are present in the file and with all of the keyword/value pairs from each of the HDUs. An application writer may*may* choose to ignore HDUs beyond the primary HDU, but even in this case the application should*should* be able to present the user with the keyword/value pairs from the primary HDU.

Note that an application intended to render 'image/fitsimage/fits' for viewing by a user has significantly more responsibility than an application intended to handle, e.g., 'image/tiff'or 'image/gif'. FITS'image/tiff' or 'image/gif'. *FITS* data arrays contain elements which that typically represent the values of a physical quantity at some coordinate location. Consequently they need not contain any pixel rendering information in the form of transfer functions, and there is no mechanism for color look-up tables. An application should*should* provide this functionality, either statically using a more or less sophisticated more- or less-sophisticated algorithm, or interactively allowing a user various degrees of choice.

Furthermore, the elements in a FITSdata array may *FITS* data array *may* be integers or floating-point numbers. The dy-

DRAFT

namic range of the data array data-array values may exceed that of the display medium and the eye, and their distribution may be highly nonuniformnon-uniform. Logarithmic, square-root, and quadratic transfer functions along with histogram equalization histogram-equalization techniques have proved helpful for rendering FITS*FITS* data arrays. Some elements of the array may have values which that indicate that their data are undefined or invalid; these should *should* be rendered distinctly. Via WCS Paper I (Greisen & Calabretta 2002) the standard permits CTYPEn Standard permits `CTYPE`*n* ='COMPLEX' `'COMPLEX'` to assert that a data array contains complex numbers (future revisions might admit other elements such as quaternions or general tensors).

Three-dimensional data arrays (NAXIS `NAXIS` =3with NAXIS1, NAXIS2and NAXIS3 `3` with `NAXIS1`, `NAXIS2`, and `NAXIS3` all greater than 1) are of special interest. Applications intended to handle 'image/fits' may`image/fits`' *may* default to displaying the first 2D two-dimensional plane of such an image cube, or they may*may* default to presenting such an image in a fashion akin to that used for an animated GIF, or they may*may* present the data cube as a mosaic of 'thumbnail' images. The time-lapse movie-looping display technique can be effective in many instances, and application writers should*should* consider offering it for all three-dimensional arrays.

An 'image/fits`image/fits`' primary HDU with NAXIS `NAXIS` =1 `1` is describing a one-dimensional entity such as a spectrum or a time series. Applications intended to handle 'image/fits' may`image/fits`' *may* default to displaying such an image as a graphical plot rather than as a two-dimensional picture with a single row.

An application that cannot handle an image with dimensionality other than two should*should* gracefully indicate its limitations to its users when it encounters NAXIS `NAXIS` =1or NAXIS `1` or `NAXIS` =3 `3` cases, while still providing access to the keyword/value pairs.

FITS*FITS* files with degenerate axes (i.e., one or more NAXISn `NAXIS`*n* =1) may `1`) *may* be described as 'image/fits`image/fits`', but the first axes should*should* be non-degenerate (i.e., the degenerate axes should*should* be the highest dimensions). An algorithm designed to render only two-dimensional images will be capable of displaying such an NAXIS `NAXIS` =3or NAXIS `3` or `NAXIS` =4FITS `4` *FITS* array that has one or two of the axes consisting of a single pixel, and an application writer should*should* consider coding this capability into the application. Writers of new applications that generate FITS*FITS* files intended to be described as 'image/fits' should`image/fits`' *should* consider using the WCSAXES`WCSAXES` keyword (Greisen et al. 2006) to declare the dimensionality of such degenerate axes, so that NAXIS`NAXIS` can be used to convey the number of non-degenerate axes.

G.3. File extensions

The FITSstandard *FITS* Standard originated in the era when files were stored and exchanged via magnetic tape; it does not prescribe any nomenclature for files on disk. Various sites within the FITS*FITS* community have long-established practices where files are presumed to be FITS*FITS* by context. File extensions used at such sites commonly indicate content of the file instead of the data format.

In the absence of other information it is reasonably safe to presume that a file name ending in '.fits`.fits`' is intended to be a FITS*FITS* file. Nevertheless, there are other commonly used extensions; e.g., '.fit`.fit`', '.fts`.fts`', and many others not suitable for listing in a media type registration.

Appendix H: Past changes or clarifications to the formal definition of FITS*FITS*

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

H.1. Differences between the requirements in this standard Standard and the requirements in the original FITSFITS papers.

1. Sect. 4.1.2: The original FITS*FITS* definition paper (Wells et al. 1981) disallows lower case lower-case letters in the keyword name, but does not specify what other characters may or may not appear in the name.
2. Sect. 4.1.2: The slash between the value and comment is 'recommended' in the original paper (Wells et al. 1981) whereas the standard Standard requires that it be present, which is consistent with the prescription of Fortran list-directed input.
3. Sect. 4.2: The original paper (Wells et al. 1981) speculated that FITS*FITS* would eventually support the full range of flexibility that is allowed by Fortran list-directed input, including dimensioned parameters. The standard Standard restricts the value field to a single value, not an array.
4. Sect. 4.2.5 and Sect. 4.2.6: The original paper (Wells et al. 1981) defined a fixed format for complex keyword values, with the real part right justified in bytesBytes 11 through 30 and the imaginary part right justified in bytesBytes 31 through 50. There are no known FITS*FITS* files that use this fixed format.
   The standard Standard does not define a fixed format for complex keyword values. Instead, complex values are represented in conformance with the rules for Fortran list-directed input, namely, with the real and imaginary parts separated by a comma and enclosed in parentheses.
5. Sect. 4.4.1.1 and Sect. 4.4.1.2: The paper that defines generalized extensions (Grosbøl et al. 1988) does not prohibit the appearance of the SIMPLE`SIMPLE` keyword in extensions nor the XTENSION`XTENSION` keyword in the primary header.

H.2. List of modification to the FITSstandardFITS Standard, version 3Version 3.0

After the IAUFWG officially approved version 3 of the FITSstandard Version 3.0 of the *FITS* Standard in 2008, the following additional corrections, clarifications, or format modifications have been made to the document.

1. Two typographical errors in Table 21 (previously Table 8.1) were corrected. The last 2 two lines of the third column should read 'LONPOLEa `LONPOLE`*a* (= PVi_3a)`PV`*i*`_3`*a*)' and 'LATPOLEa `LATPOLE`*a* (= PVi_4a)'`PV`*i*`_4`*a*)'), instead

DRAFT

of PVi_1aand PVi_2aPV*i*_1*a* and PV*i*_2*a*, respectively. (October 2008)

2. The latex LaTeX text source document was reformatted to conform to the Astronomy & Astrophysics journal page style (June 2010). The visible changes include the following: .
   - The tables, figures, equations, and footnotes are numbered sequentially throughout the entire the document, instead of sequentially within each chapter.
   - The citations use the standard 'Author (year)' format instead of being referenced by a sequential number. Also, the 'Bibliography' section at the end of the document has been replaced by a 'References' section in which the citations are listed alphabetically by author.
3. The following minor corrections or clarifications were made during the refereeing process after submitting version 3 of the FITSstandard Version 3.0 of the *FITS* Standard for publication in the Astronomy & Astrophysics journal (July 2010): .
   - A sentence was added to the end of Sect. 1.2: 'This web site website also contains the contact information for the Chairman of the IAUFWG, to whom any questions or comments regarding this standard Standard should be addressed.'
   - A 'Section' column was added to Table 1 to reference the relevant section of the document.
   - The wording of the second sentence in Sect. 4.1.1 was revised from 'Except where specifically stated otherwise in this standard, keywords may appear in any order.' to 'Keywords may appear in any order except where specifically stated otherwise in this standardStandard.'
   - A sentence was added to the end of the 'Keyword name' subsection in Sect. 4.1.2: 'Note that keyword names that begin with (or consist solely of) any combination of hyphens, underscores, and digits are legal.'
   - A footnote to the description of the REFERENC `REFERENC` keyword in Sect. 4.4.2 was added: 'This bibliographic convention (Schmitz 1995) was initially developed for use within NED (NASA/IPAC Extragalactic Database) and SIMBAD (operated at CDS, Strasbourg, France).'
   - In Sect. 7.3.4, the phrase 'TFORMn `TFORM`*n* format code' was corrected to read 'TDISPn `TDISP`*n* format code' (in four places).
   - The wording in the 'Expressed as' column in Table 26 for the 'LOG', 'GRI', 'GRA', and 'TAB' `LOG`, `GRI`, `GRA`, and `TAB` spectral algorithm codes was clarified.
   - In TableC.2 the EXTNAME, EXTVER, and EXTLEVEL C.2 the `EXTNAME`, `EXTVER`, and `EXTLEVEL` keywords were moved under the 'All HDUs' column because they are now allowed in the primary array header.
   - The last paragraph of Sect. 4.1.2.3 was corrected to state that the ASCII text ASCII-text characters have hexadecimal values 20 through 7E, not 41 through 7E.

H.3. List of modifications to the latest FITS standardFITS Standard

1. The representation of time coordinates has been incorporated by reference from Rots et al. (2015) and is summarized in Sect. 9. Cross-references have been inserted in pre-existing sections of the Standard (namely in SectSects. 4.2.7, 4.3, 4.4.2.1, 4.4.2.2 and 5.4, as well as in various places of Sect. 8, like 8.3 and such as Sect. 8.3 and Sect. 8.4.1). New keywords are listed in a rearranged Table 22. Contextually an erratum was applied in Sect. 8.4.1: keywords `OBSGEO-[XYZ]` were incorrectly marked as `OBSGEO-[XYZ]`*a*; the TAI-UTC difference in Table 30 was updated with respect to Rots et al. (2015) taking into account the latest leap second; the possibility of introducing more sources for the solar system Solar System ephemerides was re-worded (at the end of Sect.9.2.5 and in Table 31).
2. The continued string keywords described in Sect. 4.2.1.2 were originally introduced as a FITSconvention since *FITS* convention during 1994, and registered in 2007. The text of the original convention is reported at `http://fits.gsfc.nasa.gov/registry/continue_keyword.html`. The differences with this standard concern : Standard concern the following.
   - In the convention, the LONGSTRN`LONGSTRN` keyword was used to signal the possible presence of long strings in the HDU. The use of this keyword is no longer required or recommended*required* or *recommended*.
   - Usage of the convention was not recommended*not recommended* for reserved or mandatory keywords. Now it is *explicitly forbidden* unless keywords are explicitly declared long-string.
   - To avoid ambiguities in the application of the previous clause, the declaration of string keywords in sections Sects. 8, 9 and 10 has been reset from the generic 'character' to 'string'.
   - It is also explicitly clarified there is no limit to the number of continuation records.
   - The description of continued comment field is new.
3. The blank header space convention described in Sect. 4.4.2.4 was used since from 1996, and registered in 2014. The text of the original convention is reported at `http://fits.gsfc.nasa.gov/registry/headerspace.html`. It included a *recommendation* about using the convention in a controlled environment, which does not appear in this standardStandard.
4. The INHERITINHERIT keyword described in Sect. 4.4.2.6 was originally introduced as a FITS*FITS* convention in 1995, and registered in 2007. The text of the original convention is reported at `http://fits.gsfc.nasa.gov/registry/inherit.html`. See also references and practical considerations therein. The differences with the present document concern a more precise RFC-2219 more-precise RFC 2119 compliant wording in a couple of sentences in Appendix K.
5. The checksum keywords described in Sect. 4.4.2.7 were originally introduced as a FITSconvention since *FITS* convention during 1994, and registered in 2007. The text of the original convention is reported at `http://fits.gsfc.nasa.gov/registry/checksum.html`. The differences with this standard Standard concern:
   - The the omission of some additional implementation guidelines. , and
   - The the omission of a discussion on alternate algorithms and relevant additional references.

DRAFT

6. The table keywords described in Sect. 7.2.2 and 7.3.2 were originally introduced as a FITSconvention since *FITS* convention during 1993, and registered in 2006. The text of the original convention is reported at `http://fits.gsfc.nasa.gov/registry/colminmax.html`. The differences with this standard concern: Standard are as follows.
   - The exclusion of undefined or IEEE special values when computing maximum and minimum is now *mandatory* while it was *optional*.
   - The original text included the possibility of using the fact TDMINn`TDMIN`*n* were greater than TDMAXn(or TLMINngreater than TLMAXn`TDMAX`*n* (or `TLMIN`*n* greater than `TLMAX`*n*) as an indication the values were undefined. This clause has been removed
   - The original text contained usage examples and additional minor explanatory details.
7. The Green Bank convention, mentioned in Sect. 8.2 and described in Appendix L, has been in use since 1989, and was registered in 2010. The text of the registered convention is reported at `http://fits.gsfc.nasa.gov/registry/greenbank/greenbank.pdf` and contains some additional details about the history of the convention.
8. The conventions for compressed data described in Sect. 10. were originally introduced as a couple of FITS*FITS* conventions registered in 2007 and 2013. The text of the original conventions is reported at `http://fits.gsfc.nasa.gov/registry/tilecompression.html` for compressed images and at `http://fits.gsfc.nasa.gov/registry/tiletablecompression.html` for compressed binary tables. The differences with this standard concern: Standard are listed below.
   - In Sect. 10.3.3 the original text for `FZALG`*n* mentioned the possibility that, 'If the column cannot be compressed with the requested algorithm (e.g., if it has an inappropriate data type), then a default compression algorithm will be used instead.' But there is no default algorithm. This is irrelevant for the Standard.
   - In Sect. 10.4 the alias `'RICE_ONE'` is *not* adopted in the Standard as a synonym for `'RICE_1'`.
   - In Sect. 10.4.3 a sentence was left out about requiring additional instructions in PLIO to make it work for more then 2^{12} bits, since we aren't allowing this possibility in the Standard.
   - In Sect. 10.4.4 the reference to a 'smoothing flag' was dropped.
   - Also in Sect. 10.4.4 the *scale factor* is now floating point, while it was originally integer.
   - In Table 36 (and Sect. 10.3.5) the `'NOCOMPRESS'` algorithm is explicitly mentioned.

H.4. List of modifications for language editing

1. Apply systematically LaTeX macros for keyword names and values, and for RFC 2119 expressions, according to instructions reported in the LaTeX source preamble (for future editors of the Standard).
2. The acronym *FITS* is always indicated in italics.
3. Use italics systematically for RFC 2119 obligations and recommendations.
4. Apply consistent use of italic and typewriter fonts, and ' quotation marks around literal keyword values. Correct other minor LaTeX issues.
5. Apply systematic capitalization of the names of specific entities, where appropriate. These include Standard (when referring to the *FITS* Standard document), Version (where numbered), Byte, Column, Parameter, Field, and Axis. Start some words with a lower-case letter that previously began with a capital letter.
6. Address other typographical issues, such as the insertion of commas in several places, adding a few non-breaking spaces, and better handling of references to sections, etc.
7. Several cases of minor rewording.
8. Express small numbers in letter form (one to nine), not in numerals (1 to 9), wherever sensible. However, there is the customary exception for normalization in sentences and headings that also contain numbers greater than nine.
9. Compound nouns are systematically hyphenated to highlight the correct grouping (and hence meaning) of the components. This includes the attributive references to ASCII-table, binary-table, and random-groups.
10. Improve the aesthetics of some tables.

Appendix I: Random Number GeneratorRandom-number generator

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

The portable random number random-number generator algorithm below is from Park & Miller (1988). This algorithm repeatedly evaluates the function

$$seed = (a * seed) \bmod m$$

where the values of a and m are shown below, but it is implemented in a way to avoid integer overflow problems.

```
int random_generator(void) {

/* initialize an array of random numbers */

   int ii;
   double a = 16807.0;
   double m = 2147483647.0;
   double temp, seed;
   float rand_value[10000];

   /* initialize the random numbers */
   seed = 1;
   for (ii = 0; ii < N_RANDOM; ii++) {
      temp = a * seed;
      seed = temp -m * ((int) (temp / m) );
      /* divide by m for value between 0 and 1 */
      rand_value[ii] = seed / m;
   }
}
```

If implemented correctly, the 10 000th 10 000th value of seed will equal 1 043 618 065.

Appendix J: CHECKSUMImplementation GuidelinesCHECKSUM implementation guidelines

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

J.1. Recommended CHECKSUMKeyword ImplementationCHECKSUM keyword implementation

The recommendedCHECKSUM*recommended* CHECKSUM keyword algorithm described here generates a 16-character ASCII string that forces the 32-bit 1' s ones' complement checksum accumulated over the entire FITS*FITS* HDU to equal negative 0 (all 32 bits equal to 1). In addition, this string will only contain alphanumeric characters within the ranges 0–9, A–Z, and a–z to promote human readability and transcription. If the present algorithm is used, the CHECKSUMkeyword value mustCHECKSUM keyword value *must* be expressed in fixed format, with the starting single quote character in column single-quote character in Column 11 and the ending single quote character in column single-quote character in Column 28 of the FITS*FITS* keyword record, because the relative placement of the value string within the keyword record affects the computed HDU checksum. The steps in the algorithm are as follows: .

1. Write the CHECKSUMCHECKSUM keyword into the HDU header with an initial value consisting of 16 ASCII zeros ('0000000000000000''0000000000000000') where the first single quote single-quote character is in column Column 11 of the FITS*FITS* keyword record. This specific initialization string is required *required* by the encoding algorithm described in Sect. J.2 J.2. The final comment field of the keyword, if any, must *must* also be written at this time. It is recommended *recommended* that the current date and time be recorded in the comment field to document when the checksum was computed.
2. Accumulate the 32-bit 1' s ones' complement checksum over the FITS*FITS* logical records that make up the HDU header in the same manner as was done for the data records by interpreting each 2880-byte logical record as 720 32-bit unsigned integers.
3. Calculate the checksum for the entire HDU by adding (using 1' s ones' complement arithmetic) the checksum accumulated over the header records to the checksum accumulated over the data records (i.e., the previously computed DATASUMDATASUM keyword value).
4. Compute the bit-wise complement of the 32-bit total HDU checksum value by replacing all 0 bits with 1 and all 1 bits with 0.
5. Encode the complement of the HDU checksum into a 16-character ASCII string using the algorithm described in Sect. J.2
6. Replace the initial CHECKSUMCHECKSUM keyword value with this 16-character encoded string. The checksum for the entire HDU will now be equal to negative 0.

J.2. Recommended ASCII Encoding Algorithmencoding algorithm

The algorithm described here is used to generate an ASCII string, which, when substituted for the value of the CHECKSUMCHECKSUM keyword, will force the checksum for the entire HDU to equal negative 0. It is based on a fundamental property of 1' s ones' complement arithmetic that the sum of an integer and the negation of that integer (i.e, the bitwise complement formed by replacing all 0 bits with 1s and all 1 bits with 0s) will equal negative 0 (all bits set to 1). This principle is applied here by constructing a 16-character string, which, when interpreted as a byte stream of four 32-bit integers, has a sum that is equal to the complement of the sum accumulated over the rest of the HDU. This algorithm also ensures that the 16 bytes that make up the four integers all have values that correspond to ASCII alpha-numeric characters in the range 0–9, A–Z, and a–z.

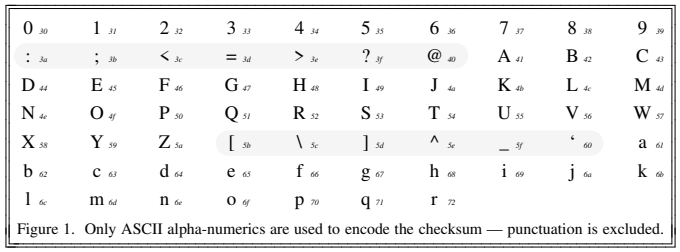
1. Begin with the 1' s ones' complement (replace 0s with 1s and 1s with 0s) of the 32-bit checksum accumulated over all the FITS *FITS* records in the HDU after first initializing the CHECKSUMCHECKSUM keyword with a fixed-format string consisting of 16 ASCII zeros ('0000000000000000''0000000000000000').
2. Interpret this complemented 32-bit value as a sequence of four unsigned 8-bit integers, A, B, C and D, where A is the most significant byte and D is the least significanteight-bit integers, *A*, *B*, *C*, and *D*, where *A* is the most-significant byte and *D* is the least-significant byte. Generate a sequence of four integers, A1, A2, A3, A4*A*1, *A*2, *A*3, *A*4, that are all equal to A *A* divided by 4 (truncated to an integer if necessary). If A *A* is not evenly divisible by 4, add the remainder to A1*A*1. The key property to note here is that the sum of the four new integers is equal to the original byte value (e.g., A = A1 + A2 + A3 + A4*A* = *A*1 + *A*2 + *A*3 + *A*4). Perform a similar operation on B, C, and D*B*, *C*, and *D*, resulting in a total of 16 integer values, 4 four from each of the original bytes, which should *should* be rearranged in the following order:

```
A1 B1 C1 D1 A2 B2 C2 D2 A3 B3 C3 D3 A4 B4 C4 D4.
```

   Each of these integers represents one of the 16 characters in the final CHECKSUMCHECKSUM keyword value. Note that if this byte stream is interpreted as 4 four 32-bit integers, the sum of the integers is equal to the original complemented checksum value.
3. Add 48 (hex 30), which is the value of an ASCII zero character, to each of the 16 integers generated in the previous step. This places the values in the range of ASCII alphanumeric characters '0' (ASCII zero) to 'r'. This offset is effectively subtracted back out of the checksum when the initial CHECKSUMCHECKSUM keyword value string of 16 ASCII 0s is replaced with the final encoded checksum value.
4. To improve human readability and transcription of the string, eliminate any non-alphanumeric characters by considering the bytes a pair at a time (e.g., A1 + A2, A3 + A4, B1 + B2*A*1 + *A*2, *A*3 + *A*4, *B*1 + *B*2, etc.) and repeatedly increment the first byte in the pair by 1 and decrement the 2nd second byte by 1 as necessary until they both correspond to the ASCII value of the allowed alphanumeric characters 0–9, A–Z, and a–z shown in Figure1. J.1. Note that this operation conserves the value of the sum of the 4 four equivalent 32-bit integers, which is required for use in this checksum application.
5. Cyclically shift all 16 characters in the string one place to the right, rotating the last character (D4*D*4) to the beginning of the string. This rotation compensates for the fact that the fixed format FITScharacter string *FITS* character-string

DRAFT

Fig. J.1: Only ASCII alpha-numeric characters are used to encode the checksum – punctuation is excluded



values are not aligned on 4-byte four-byte word boundaries in the FITS*FITS* file. (The first character of the string starts in column Column 12 of the header card image, rather than column Column 13).

6. Write this string of 16 characters to the value of the CHECKSUM`CHECKSUM` keyword, replacing the initial string of 16 ASCII zeros.

To invert the ASCII encoding, cyclically shift the 16 characters in the encoded string one place to the left, subtract the hex 30 offset from each character, and calculate the checksum by interpreting the string as four 32-bit unsigned integers. This can be used, for instance, to read the value of CHECKSUM`CHECKSUM` into the software when verifying or updating a HDU.

J.3. Encoding Exampleexample

This example illustrates the encoding algorithm given in Sect. J.2 Consider a FITSHDU whose 1' s *FITS* HDU whose ones' complement checksum is 868229149, which is equivalent to hex 33C0201D`33C0201D`. This number was obtained by accumulating the 32-bit checksum over the header and data records using 1' s ones' complement arithmetic after first initializing the CHECKSUM`CHECKSUM` keyword value to '0000000000000000'`'0000000000000000'`. The complement of the accumulated checksum is 3426738146, which is equivalent to hex CC3FDFE2`CC3FDFE2`. The steps needed to encode this hex value into ASCII are shown schematically below: .

```
    Byte                          Preserve byte alignment
 A  B  C  D    A1 B1 C1 D1   A2 B2 C2 D2   A3 B3 C3 D3   A4 B4 C4 D4

CC 3F DF E2 -> 33 0F 37 38   33 0F 37 38   33 0F 37 38   33 0F 37 38
+ remainder     0  3  3  2

      = hex    33 12 3A 3A   33 0F 37 38   33 0F 37 38   33 0F 37 38
 + 0 offset    30 30 30 30   30 30 30 30   30 30 30 30   30 30 30 30

      = hex    63 42 6A 6A   63 3F 67 68   63 3F 67 68   63 3F 67 68
      ASCII     c  B  j  j    c  ?  g  h    c  ?  g  h    c  ?  g  h

                              Eliminate punctuation characters
initial values  c  B  j  j    c  ?  g  h    c  ?  g  h    c  ?  g  h
    .           c  C  j  j    c  >  g  h    c  @  g  h    c  >  g  h
    .           c  D  j  j    c  =  g  h    c  A  g  h    c  =  g  h
    .           c  E  j  j    c  <  g  h    c  B  g  h    c  <  g  h
    .           c  F  j  j    c  ;  g  h    c  C  g  h    c  ;  g  h
    .           c  G  j  j    c  :  g  h    c  D  g  h    c  :  g  h
final values    c  H  j  j    c  9  g  h    c  E  g  h    c  9  g  h

final string   "hcHjjc9ghcEghc9g"   (rotate 1 place to the right)
```

In this example byte B1 Byte *B*1 (originally ASCII BB) is shifted higher (to ASCII HH) to balance byte B2 Byte *B*2 (originally ASCII ??) being shifted lower (to ASCII 99). Similarly, bytes B3 and B4 Bytes *B*3 and *B*4 are shifted by opposing amounts. This is possible because the two sequences of ASCII punctuation characters that can occur in encoded checksums are both preceded and followed by longer sequences of ASCII alphanumeric characters. This operation is purely for cosmetic reasons to improve readability of the final string.

This is how these CHECKSUMand DATASUM`CHECKSUM` and `DATASUM` keywords would appear in a FITS*FITS* header (with the recommended time stamp in the comment field). :

```
DATASUM = '2503531142'          /  2015-06-28T18:30:45
CHECKSUM= 'hcHjjc9ghcEghc9g'    /  2015-06-28T18:30:45
```

J.4. Incremental Updating updating of the Checksumchecksum

The symmetry of 1' s ones' complement arithmetic also means that after modifying a FITS*FITS* HDU, the checksum may *may* be incrementally updated using simple arithmetic without accumulating the checksum for portions of the HDU that have not changed. The new checksum is equal to the old total checksum plus the checksum accumulated over the modified records, minus the original checksum for the modified records.

An incremental update provides the mechanism for end-to-end checksum verification through any number of intermediate processing steps. By *calculating* rather than *accumulating* the intermediate checksums, the original checksum test is propagated through to the final data file. On the other hand, if a new checksum is accumulated with each change to the HDU, no information is preserved about the HDU's original state.

The recipe for updating the CHECKSUM`CHECKSUM` keyword following some change to the HDU is: $C' = C - m + m'$, where C and C' represent the HDU's checksum (that is, the complement of the CHECKSUM`CHECKSUM` keyword) before and after the modification and m and m' are the corresponding checksums for the modified FITS*FITS* records or keywords only. Since the CHECKSUM`CHECKSUM` keyword contains the complement of the checksum, the correspondingly complemented form of the recipe is more directly useful: $\tilde{C}' = \tilde{(}C + \tilde{m} + m'\tilde{)}$, where $\tilde{}$ (tilde) denotes the (1'sones') complement operation. See Braden et al. (1988); Mallory & Kullberg (1990); Rijsinghani (1994). Note that the tilde on the right hand side of the equation cannot be distributed over the contents of the parentheses due to the dual nature of zero in 1' s ones' complement arithmetic (Rijsinghani 1994).

J.5. Example C Code code for Accumulating accumulating the Checksumchecksum

The 1' s ones' complement checksum is simple and fast to compute. This routine assumes that the input records are a multiple of 4 four bytes long (as is the case for *FITS FITS logical records*), but it is not difficult to allow for odd length records if necessary. To use this routine, first initialize the CHECKSUMkeyword to '0000000000000000'and initialize sum32 = 0`CHECKSUM` keyword to `'0000000000000000'` and initialize sum32 = `0`, then step through all the FITS*FITS* logical records in the FITS *FITS* HDU.

```
void checksum (
   unsigned char *buf,  /* Input array of bytes to be checksummed */
                        /*  (interpret as 4-byte unsigned ints)   */
```

DRAFT

```
   int length,          /* Length of buf array, in bytes          */
                        /*  (must be multiple of 4)               */
   unsigned int *sum32) /* 32-bit checksum                        */
{
/*
   Increment the input value of sum32 with the 1's complement sum
   accumulated over the input buf array.
*/
   unsigned int hi, lo, hicarry, locarry, i;

   /*  Accumulate the sum of the high-order 16 bits and the */
   /*  low-order 16 bits of each 32-bit word, separately.  */
   /*  The first byte in each pair is the most significant. */
   /*  This algorithm works on both big and little endian machines.*/
   hi = (*sum32 >> 16);
   lo = *sum32 & 0xFFFF;
   for (i=0; i < length; i+=4) {
       hi += ((buf[i]   << 8) + buf[i+1]);
       lo += ((buf[i+2] << 8) + buf[i+3]);
   }

   /* fold carry bits from each 16 bit sum into the other sum */
   hicarry = hi >> 16;
   locarry = lo >> 16;
   while (hicarry || locarry) {
       hi = (hi & 0xFFFF) + locarry;
       lo = (lo & 0xFFFF) + hicarry;
       hicarry = hi >> 16;
       locarry = lo >> 16;
   }

   /* concatenate the full 32-bit value from the 2 halves */
   *sum32 = (hi << 16) + lo;
}
```

J.6. Example C Code code for ASCII Encodingencoding

This routine encodes the complement of the 32-bit HDU checksum value into a 16-character string. The byte alignment of the string is permuted one place to the right for FITS*FITS* to left justify the string value starting in column Column 12.

```
unsigned int exclude[13] = {0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40,
                            0x5b, 0x5c, 0x5d, 0x5e, 0x5f, 0x60 };

int offset = 0x30;                          /* ASCII 0 (zero) */
unsigned long mask[4] = { 0xff000000, 0xff0000, 0xff00, 0xff };

void char_encode (
  unsigned int value, /* 1's complement of the checksum     */
                      /*     value to be encoded            */
  char *ascii)        /* Output 16-character encoded string */
{
   int byte, quotient, remainder, ch[4], check, i, j, k;
   char asc[32];

   for (i=0; i < 4; i++) {
       /* each byte becomes four */
       byte = (value & mask[i]) >> ((3 - i) * 8);
       quotient = byte / 4 + offset;
       remainder = byte % 4;
       for (j=0; j < 4; j++)
           ch[j] = quotient;

       ch[0] += remainder;

       for (check=1; check;)     /* avoid ASCII punctuation */
          for (check=0, k=0; k < 13; k++)
             for (j=0; j < 4; j+=2)
                 if (ch[j]==exclude[k] || ch[j+1]==exclude[k]) {
                     ch[j]++;
                     ch[j+1]--;
                     check++;
                  }

        for (j=0; j < 4; j++)     /* assign the bytes */
            asc[4*j+i] = ch[j];
    }

    for (i=0; i < 16; i++)       /* permute the bytes for FITS */
       ascii[i] = asc[(i+15)%16];
    ascii[16] = 0;               /* terminate the string */
}
```

Appendix K: Header inheritance convention

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

The reserved boolean INHERITBoolean `INHERIT` keyword described in Sect. 4.4.2.6 is optionaloptional, but if present it shallshall appear in the extension header immediately after the mandatory keywords. The INHERITkeyword must not`INHERIT` keyword *must not* appear in the primary header. Keyword inheritance provides a mechanism to store keywords in the primary HDU, and have them be shared by one or more extensions in the file. This mechanism minimizes duplication (and maintenance) of metadata in multi-extension FITS *FITS* files.

It should*should* only be used in FITS*FITS* files that have a null primary array (e.g., with NAXIS `NAXIS` =0 `0`). to avoid possible confusion if array-specific keywords (e.g., BSCALEand BZEROBSCALE and `BZERO`) were to be inherited.

When an application reads an extension header with INHERIT `INHERIT` =T, it should `T`, it *should* merge the keywords in the current extension with the primary header keywords. The exact merging mechanism is left up to the application. The mandatory primary array keywords (e.g., BITPIX, NAXIS, and NAXISn`BITPIX`, `NAXIS`, and `NAXIS`*n*) and any COMMENT, HISTORY`COMMENT`, `HISTORY`, and blank keywords in the primary header must not*must not* be inherited. It is assumed also that the table-specific keywords described in Sect. 7.2 and 7.3, and the table-specific WCS keywords described in Sect. 8, cannot be inherited since they will never appear in the primary header. If the same keyword is present in both the primary header and the extension header, the value in the extension header shall*shall* take precedence. If INHERIT `INHERIT` =F `F` in an extension header, the keywords from the primary header shouldnot *should not* be inherited.

An application which that merely reads a FITS *FITS* file is authorized by INHERIT `INHERIT` =T `T` to look in the primary HDU for an expected keyword not found in the current HDU. However if the application writes out a modified file, it has to be very careful to avoid unwanted duplication of keywords, and preserve the separation of primary and extension headers. If an application modifies the value of an inherited keyword while processing an extension HDU, then it is recommended *recommended* to write the modified value of that keyword into the extension header, leaving the value of the keyword in the primary header unchanged. The primary array keywords should *should* only be modified when the intent is to explicitly change the value that will subsequently be inherited in the extensions.

Also if the FITS *FITS* file is read in sequentially (e.g., from tape or Internet downloads), the reader would need to cache the primary header in case it turns out that a later extension in the file uses the INHERIT `INHERIT` keyword.

Appendix L: Green Bank convention

This Appendix appendix is not part of the FITSstandardFITS Standard, but is included for informational purposes.

The Green Bank convention was developed at a meeting in October 1989 at the US National Radio Astrononomy Astronomy Observatory in Green Bank, West Virginia, to discuss the use of FITS for single dish radio astronomy *FITS* for

single-dish radio-astronomy data, and has since been widely used in conjunction with the SDFITS convention[16]. It was devised primarily to record WCS keywords independently for each row of a table containing an image array column, but subsequently it has found more general more-general application.

The basic idea is that of expanding header keywords into binary table binary-table columns, and vice versa, of collapsing unvarying binary table binary-table columns into header keywords.

For example, the standard header keyword DATE-OBS, which records the date and time of observation, could be expanded into a column with TTYPEnTTYPE*n* = 'DATE-OBS''DATE-OBS' to record the date and time independently for each row of a binary table. Conversely, a binary table column with TTYPEn binary-table column with TTYPE*n*= 'HUMIDITY''HUMIDITY' containing the same value in each row, could be collapsed into a keyword, HUMIDITY, that recorded the constant value.

When the Green Bank convention is used (and arguably otherwise)a keyword should not , a keyword *should not* coexist with a column of the same name within a single binary table. Should this situation occur, the column value takes precedence over the keyword.

When expanding keywords into columns, the Green Bank convention applies to all FITS *FITS* keywords that may appear in a binary table except for the following, most of which describe the structure or identity of a binary table HDU: XTENSION, BITPIX, NAXIS, NAXIS*n*, PCOUNT, GCOUNT, TFIELDS, EXTNAME, EXTVER, EXTLEVEL, , , , , , TTYPE*n*, TFORM*n*, TUNIT*n*, TSCAL*n*, TZERO*n*, TNULL*n*, TDISP*n*, THEAP, TDIM*n*, DATE, ORIGIN, COMMENT, HISTORY, CONTINUE, and END.

In order to collapse a column into a keyword, the name of the column (given by) must TTYPE*n*) *must* be a valid keyword name, and the column's constant value must *must* be amenable to representation as a valid keyvalue.

Software that implements the Green Bank convention must *must* take into account the possibility that any "keyword" (apart from those on the proscribed list), such as DATE-OBS, may change value from row to row of a table. Moreover, when searching the header for a particular keyword, it must first consider the values of the TTYPE*n* keywords in case the desired keyword has been expanded as a column. Likewise, it must consider each header keyword potentially as a collapsed column, so that a request for the value in a particular row or rows of the non-existent 'HUMIDITY'HUMIDITY column would be satisfied by the value of the HUMIDITY keyword.

References

Note: Many of these FITS*FITS* references are available electronically from the NASA Astrophysics Data System (ADS) and/or the FITSSupport Office web sites *FITS* Support Office websites at http://adswww.harvard.edu and http://fits.gsfc.nasa.gov/fits_documentation.html.

Allen, S. & Wells, D. 2005, IETF RFC 4047, http://www.ietf.org/rfc/rfc4047.txt

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

Braden, R. T., Borman, D.A., and Partridge, C. 1988 ACM Computer Communication Review, 19, no. 2, 86, IETF RFC 1071, https://tools.ietf.org/html/rfc1071

Bradner, S. 1997, IETF RFC 2119, http://www.ietf.org/rfc/rfc2119.txt

Bunclark, P. & Rots, A. 1997, Precise re-definition ofDATE-OBSKeyword encompassing the millenium*Precise re-definition of* DATE-OBS *Keyword encompassing the millennium*, http://fits.gsfc.nasa.gov/year2000.html

Calabretta, M. R. & Greisen, E. W. 2002, A&A, 395, 1077

Calabretta, M. R. & Roukema, B. F. 2007, MNRAS, 381, 865

Cotton, W. D., Tody, D. B., & Pence, W. D. 1995, A&AS, 113, 159

Cotton, W. D., et al. 1990, Going AIPS: A Programmer's Guide to the NRAO Astronomical Image Processing System*Going AIPS: A Programmer's Guide to the NRAO Astronomical Image Processing System*, Charlottesville: NRAO

Deutsch P. 1996, RFC 1951, Network Working Group; available online: http://tools.ietf.org/html/rfc1951

Folkner, W. M., Williams, J. G., & Boggs, D. H. 2009, Interplanetary Network Progress Report 42-178, available online: http://tmo.jpl.nasa.gov/progress_report/42-178/178C.pdf

Folkner, W. M. et al. 2014, Interplanetary Network Progress Report 42-196, available online: http://ipnpr.jpl.nasa.gov/progress_report/42-196/196C.pdf

Greisen, E. W. & Calabretta, M. R. 2002, A&A, 395, 1061

Greisen, E. W., Calabretta, M. R., Valdes, F. G., & Allen, S. L. 2006, A&A, 446, 747

Greisen, E. W. & Harten, R. H. 1981, A&AS, 44, 371

Grosbøl, P., Harten, R. H., Greisen, E. W., & Wells, D. C. 1988, A&AS, 73, 359

Grosbøl, P. & Wells, D. C. 1994, Blocking of Fixed-block Sequential Media and Bitstream Devices*Blocking of Fixed-block Sequential Media and Bitstream Devices*, http://fits.gsfc.nasa.gov/blocking94.html

Hanisch, R., et al. 2001, A&A, 376, 359

Harten, R. H., Grosbøl, P., Greisen, E. W., & Wells, D. C. 1988, A&AS, 73, 365

IAU 1983, Transactions of the IAU*Transactions of the IAU*, XVIIIB, 45

IAU 1988, Transactions of the IAU*Transactions of the IAU*, XXB, 51

IAU 1997, Resolution B1 of the XXIIIrd General Assembly – Transactions of the IAU Vol. XXIII B, Ed. J. Andersen, (Dordrecht: Kluwer). Available online: http://www.iau.org/static/resolutions/IAU1997_French.pdf

IEEE 1985, American National Standard – IEEE Standard for Binary Floating Point Arithmetic*American National Standard – IEEE Standard for Binary Floating Point Arithmetic*, ANSI/IEEE 754–1985, New York: American National Standards Institute, Inc.

Irwin, A. W. & Fukushima, T. A. 1999, A&A348, 642

ISO 2004, Information technology – Programming languages – Fortran*Information technology – Programming languages – Fortran*, ISO/IEC 1539-1:2004, Geneva: International Organization for Standardization

ISO 2004b, International Standard ISO 8601:2004(E), *Data elements and interchange formats – Information interchange – Representation of dates and times*

NASA/JPL Planetary Ephemerides 2014a, available online: http://ssd.jpl.nasa.gov/?ephemerides

NASA/JPL Solar and Planetary Ephemerides 2014b, available online: http://ssd.jpl.nasa.gov/?planet_eph_export

Mallory, T. & Kullberg, A. 1990, IETF RFC 1141, https://tools.ietf.org/html/rfc1141

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

Park, X. & Miller, X. 1988, Comm. ACM, 31, Issue 10, 1192; available online: http://dl.acm.org/citation.cfm?id=63042

Pence, W. D., Seaman, R., & White, R. L. 2009, PASP, 121, 414

Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., & Stobie, E. 2010, A&A, 524, A42

Pence, W. D., Seaman, R., & White, R. L. 2013, *Tiled Table Convention for Compressing FITS Binary Tables*, FITS *FITS* Support Office; available online: http://fits.gsfc.nasa.gov/registry/tiletablecompression.html

Ponz, J. D., Thompson, R. W., & Muñoz, J. R. 1994, A&AS, 105, 53

Rice, R. F., Yeh, P.-S., & Miller, W. H. 1993, in Proc. 9th AIAA Computing in Aerospace Conf., AIAA-93-4541-CP, American Institute of Aeronautics and Astronautics

Rijsinghani, A. (ed.) 1994, IETF RFC 1624, https://tools.ietf.org/html/rfc1624

Rots, A. H., Bunclark, P. S., Calabretta, M. R., Allen, S. L., Manchester, R. N. & Thompson, W. T. 2015, A&A, 574, A36

[16] http://fits.gsfc.nasa.gov/registry/sdfits.html

DRAFT

Schmitz, M., et al. 1995, Information & On-line data in Astronomy*Information & On-line data in Astronomy*, eds. D. Egret & M. A. Albrecht (Kluwer Academic Pub.), 259
Standish, E. M. 1990, A&A, 233, 252
Standish, E. M. 1998, JPL Memo IOM 312.F-98-048
Wells, D. C., Greisen, E. W., & Harten, R. H. 1981, A&AS, 44, 363
Wells, D. C. & Grosbøl, P. 1990, Floating Point Agreement forFITS*Floating Point Agreement for FITS*, `http://fits.gsfc.nasa.gov/fp89.txt`
White, R. L. 1992, in Proceedings of the NASA Space and Earth Science Data Compression Workshop, ed. J. C. Tilton, Snowbird, UT; available online: `https://archive.org/details/nasa_techdoc_19930016742`
White, R. L., & Greenfield, P. 1999, in ADASS VIII, ASP Conf. Ser. 172, eds. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 125
White, R. L., Greenfield, P., Pence, W., Tody, D. & Seaman, R. 2013, *Tiled Image Convention for Storing Compressed Images in FITS FITS Binary Tables*, FITS *FITS* Support Office; available online: `http://fits.gsfc.nasa.gov/registry/tilecompression.html`
Ziv, J., & Lempel, A. 1977, IEEE Transactions on Information Theory, 23 (3), 337

DRAFT

Index

N_{bits}, 11, 12, 17

angular units, 10, 31
ANSI, 3, 56
ANSI, IEEE, 16, 26
array descriptor, 22–24, 26–29
array size, 11, 12, 17
array value, 3, 4, 14, 20
array, multi-dimensional, 5, 24, 30, 59
array, variable-length, 1, 24, 27–29, 59
ASCII character, 3, 16, 19, 21, 56
ASCII table, 19
ASCII text, 3–5, 7, 13, 21, 25, 56
ASCII, ANSI, 66
AUTHOR, 13

binary table, 4, 16, 22, 54
BITPIX, 11, 12, 14, 16–19, 22
BLANK, 14, 16
blocking, 6
BSCALE, 14, 16
BUNIT, 14
byte order, 5, 16, 26
BZERO, 14, 16

case sensitivity, 6, 7, 10, 20, 23
character string, 3, 7, 21, 24, 25
checksum, 15, 45, 48, 49, 62–65
COMMENT, 6, 13
complex data, 9, 23, 26, 27
compressed binary tables, 48, 54, 62
compressed images, 44, 54, 62
compression algorithms, 44, 49, 50
compression, lossy, 48
conforming extension, 3–5
CONTINUE, 6, 7, 62
coordinate systems, 29

DATAMAX, 14
DATAMIN, 14
DATE, 12, 42
DATE-OBS, 13, 42
DATExxxx, 13, 42
deprecate, 3, 4, 6, 9, 13, 16, 22, 30, 31, 34
dithering, 47
durations, 43

END, 5, 11, 17, 18, 20, 22, 28
EPOCH, 34
EQUINOX, 34
EXTEND, 12
extension, 3–5, 15, 16, 59, 61
extension registration, 5, 11
extension type name, 3, 5, 11, 15
extension, conforming, 3–5
extension, standard, 4, 5
EXTLEVEL, 15
EXTNAME, 15
EXTVER, 15

field, empty, 22, 25, 26
file size, 4
fill, 5, 6, 11, 18, 20, 21, 24, 25, 28
FITS structure, 3, 4, 6, 12
floating-point, 8, 26, 56
floating-point FITS agreement, 67
floating-point, complex, 9, 26
format, data, 16
format, fixed, 7
format, free, 7, 8, 52
format, keywords, 7
Fortran, 5, 19–21, 24, 66

GCOUNT, 12, 17–19, 22
Green Bank convention, 31, 62, 66
group parameter value, 3, 17, 18
GROUPS, 17
GTI tables, 43
GZIP compression, 50

H-compress algorithm, 51
HDU, 4, 11
HDU, extension, 3, 4
HDU, primary, 3–5
header space, preallocation, 14, 62
heap, 4, 12, 22–24, 26, 28
HISTORY, 6, 14
hyphen, 6, 20, 23, 31, 34

IAU, 1, 4, 66
IAU Style Manual, 10, 67
IAUFWG, 1, 4–6, 11, 29, 41, 59
IEEE floating-point, 16
IEEE special values, 4, 14, 16, 56
image extension, 18
INHERIT, 15, 62, 65
INSTRUME, 13
integer, 16-bit, 16, 26
integer, 32-bit, 16, 26
integer, 64-bit, 16, 26
integer, complex, 9
integer, eight-bit, 16, 26
integer, unsigned, 14, 16, 23, 26
ISO-8601 date, 37

JD, 37

keyword record, 5, 6
keyword, commentary, 6, 13
keyword, indexed, 4, 6, 11
keyword, mandatory, 7, 10, 17–19, 22, 54
keyword, new, 16
keyword, order, 11, 17, 19
keyword, required, 4, 10, 11, 17, 18, 22, 44, 48
keyword, reserved, 4, 12, 17, 18, 20, 22, 30, 31, 42, 45, 49, 54
keyword, valid characters, 6

logical value, 8, 25, 26

DRAFT

magnetic tape, 6
min and max in columns, keywords, 20, 24, 62
MJD, 37

NaN, IEEE, 16, 26, 56, 57
NAXIS, 4, 5, 11, 12, 17–19, 22
NAXIS1, 17, 19, 21, 22, 24, 25, 28
NAXIS2, 19, 21, 22, 24, 25, 28
NAXISn, 5, 11, 12, 17, 18
NULL, ASCII, 3, 25

OBJECT, 13
OBSERVER, 13
order, byte, 5, 16, 26
order, extensions, 5
order, FITS structures, 4
order, keyword, 6, 11, 17, 19
ORIGIN, 12

PCOUNT, 12, 17–19, 22, 28
phase, 43
physical value, 3, 4, 14, 17, 18, 20, 23
PLIO compression, 50
primary data array, 3–5, 17, 18
primary header, 3, 4, 11, 17
PSCALn, 17, 18
PTYPEn, 17, 18
PZEROn, 17, 18

quantization of data, 46, 63

random groups, 3, 16, 17
random-groups, 14
random-groups array, 17
REFERENC, 13
repeat count, 4, 22, 25
Rice compression, 50

scaling, data, 17, 18, 20, 23
sign bit, 16
sign character, 8, 21
SIMPLE, 5, 11, 17
slash, 7, 10
solar system ephemeris, 41
special records, 3–5
special values, IEEE, 26
standard extension, 4, 5

TABLE, 19
TBCOLn, 19
TDIMn, 24
TDISPn, 20, 24
TELESCOP, 13
TFIELDS, 19, 22
TFORMn, 19, 22, 25, 26, 28
THEAP, 24, 28
time, 12, 13, 34, 36, 37
time keywords, 42
time reference, 38
time reference direction, 40
time reference position, 39
time resolution, 42
time scale, 37, 38
time units, 41
time, universal, 12, 38
timelag, 43
TIMESYS, 37
TNULLn, 20, 21, 24, 26
TSCALn, 20, 23, 28
TTYPEn, 20, 23
TUNITn, 20, 23
two's complement, 16, 26
TZEROn, 20, 23, 28

underscore, 6, 20, 23
units, 4, 10, 14, 20, 23, 31, 36

value, 6, 7, 12
value, undefined, 7, 14, 16, 20, 21, 24, 60
variable-length array, 1, 24, 27–29, 59
variable-length arrays, compression, 49

WCS, 29
WCS, celestial, 34
WCS, spectral, 34
WCS, timing, 36, 62

XTENSION, 3, 5, 11, 15, 18, 19, 22

DRAFT