FITS Embedded Function Specification
ASC-FITS-FUNCTION-1.2
SDS-11.2*

Arnold Rots and Jonathan McDowell

2000-08-30

Contents

*$Id: func3.tex,v 1.2 2000/08/30 18:21:09 arots Exp arots $

1 Introduction

This document defines the FITS FUNCTION binary table syntax. The objective is to allow specification of an n-dimensional image in the form of an analytical function of n variables, in a FITS binary table HDU. The salient points are that it specifies all external coordinates (independent variables) through the FTYPEs, allowing them to be free-running or enumerated; internal mappings and transformations are allowed through internal functions and arithmetic expressions, cast as virtual columns; constants may be set; and the "return value" of the table is an arithmetic combination of all these. The design is explained through two simple examples.

In the following sections, we first provide basic definitions; then a specification of the independent (external) function variables and of the function definition itself; after that a specification of the two internal virtual columns types; and finally, a more sophisticated example.

The specification will have the status of an ASC internal convention, although it is intended to be general enough that it may gain wider acceptance.

2 Definitions

Function tables are identified by:

```
HDUCLASS= 'ASC     '
HDUCLAS1= 'FUNCTION'
```

The function is defined by:

```
FUNCTION= '<expression>'
FUNCNAME= '<name>'
```

The function expression may contain the arithmetic operators $+\ -\ *\ /\ **$ and five types of operands or "parameters"; except for the first, these operands can be thought of as columns in the table (real columns, constant columns, and virtual columns), and the function would thus be an aritmetic combination of these "columns". One should be warned, though, that operands may be defined in terms of other operands; hence reality is more complicated than this simplified view.

- FTYPEi: axes of the function evaluation space; these may just be specified as a range or may also appear as a table column (TTYPEj) if other parameters are to be enumerated along such an axis.
- DTYPEi: constants; the name is provided by DTYPEi, the value by DVALi, and the units by DUNITi. This is the "constant column".

- TTYPEi: parameters enumerated in table column i.
- VTYPEi: function components; the name is provided by VTYPEi, the function expression by VFUNCi. This is the "virtual function column".
- WTYPEi: arithmetic components; the name is provided by WTYPEi, the arithmetic expression by WFUNCi. This is the "virtual arithmetic column".

The assumption is that implementation of a FEF reader will include the following functionality.

```
typedef struct {
  char name[32] ;
  double min ;
  double max ;
  int num ;
} FefParam ;

FefParam parms[n] ;
fits_file fef ;

double* image = readFef (fef, n, parms) ;
```

`readFef` returns an n-dimensional image where the lengths of the axes are set by `parms[i].num`, the names by `parms[i].name`, and the minimum and maximum by `parms[i].min` and `parms[i].max`, respectively. Each `parms[i].name` must correspond with an FTYPE value and the minimum and maximum values must not exceed the corresponding FLMIN and FLMAX values. n should be equal to FAXIS.

Extraneous columns are allowed. However, one should take care when filtering on such a column (or any other, for that matter), that such filtering can only be guaranteed to yield a valid FEF extension if all but one of the FTYPE variables contains more than one point. Even then, it will still require that all FAXISi keywords are corrected.

3 FTYPE

Define all independent variables (the evaluation space), whether they will be enumerated or not, as FTYPEi. FAXISi specifies the number of samples provided for enumerated variables and FLMINi/FLMAXi set the legal range for free running ones (though it is a good idea to specify FLMIN/FLMAX also for enumerated variables in order to indicate over which range interpolation is allowed). Obviously, the number of rows in the table should equal the product of the FAXISi keyword values.

Example:

```
FAXIS   =          3
FTYPE1  = 'X       '
FUNIT1  = 'mm      '
FLMIN1  =      -70.0
FLMAX1  =       70.0
FTYPE2  = 'Y       '
FUNIT2  = 'mm      '
FLMIN2  =      -70.0
FLMAX2  =       70.0
FTYPE3  = 'Energy  '
FUNIT3  = 'keV     '
FAXIS3  =          3
```

Enumerated independent variables also need a column specification.

```
TTYPE1  = 'Energy  '
TUNIT1  = 'keV     '
TFORM1  = '1D      '
```

At least for the time being, we require a full matrix; *i.e.* , the columns with enumerated variables should form a regular grid, such as:

```
1  1
2  1
4  1
1  2
2  2
4  2
1  3
2  3
4  3
```

We shall assume that if function values are requested for values of enumerated variables that fall between the enumerated grip points, the values of all enumerated columns will be interpolated linearly. It is important to note that *the parameter values are interpolated, not the function values.* If the values of an entire row are to be held constant for a range of values of an enumerated variable, one may specify bins by using a _LO and _HI column and adding the appropriate DM keywords:

```
TTYPE1  = 'Energy_LO'
TUNIT1  = 'keV     '
TFORM1  = '1D      '
TTYPE2  = 'Energy_HI'
```

```
TUNIT2  = 'keV     '
TFORM2  = '1D      '
MTYPE1  = 'Energy  '
MFORM1  = 'Energy_LO,Energy_HI'
METYP1  = 'R       '
```

The values in the row will be constant for all energies between Energy_LO and Energy_HI.

In some cases (e.g., response matrices) it is desirable to limit the domain of a free running variable depending on the values of the enumerated variables. This may be done by incorporating two columns for FDMIN*i* and FDMAX*i*. If *PHA* is free running in the following example:

```
FAXIS   =          3
FTYPE1  = 'X       '
FUNIT1  = 'mm      '
FLMIN1  =      -70.0
FLMAX1  =       70.0
FTYPE2  = 'Y       '
FUNIT2  = 'mm      '
FLMIN2  =      -70.0
FLMAX2  =       70.0
FTYPE3  = 'Energy  '
FUNIT3  = 'keV     '
FAXIS3  =          3
FTYPE4  = 'PHA     '
FUNIT4  = 'chan    '
TTYPE1  = 'X_LO    '
TUNIT1  = 'mm      '
TFORM1  = '1E      '
TTYPE2  = 'X_HI    '
TUNIT2  = 'mm      '
TFORM2  = '1E      '
TTYPE3  = 'Y_LO    '
TUNIT3  = 'mm      '
TFORM3  = '1E      '
TTYPE4  = 'Y_HI    '
TUNIT4  = 'mm      '
TFORM4  = '1E      '
TTYPE5  = 'Energy  '
TUNIT5  = 'keV     '
TFORM5  = '1E      '
TTYPE6  = 'FDMIN4  '
TUNIT6  = 'chan    '
TFORM6  = '1E      '
TTYPE7  = 'FDMAX4  '
```

```
TUNIT7  = 'chan    '
TFORM7  = '1E      '
MTYPE1  = 'X       '
MFORM1  = 'X_LO,X_HI'
METYP1  = 'R       '
MTYPE2  = 'Y       '
MFORM2  = 'Y_LO,Y_HI'
METYP2  = 'R       '
```

4 FUNCTION Specification

Define the table's function:

```
FUNCTION= 'Norm - Scale * (X2 + Y2)'
FUNCNAME= 'HRMA_EffArea'
BUNIT   = 'mm**2   '
```

The FUNCTION definition is an arithmetic expression in which the operands may be the *values* of any of the following:

- FTYPEn
- DTYPEn
- TTYPEn
- VTYPEn
- WTYPEn

with operators $+\ -\ *\ /\ **$; parentheses are allowed.

```
TTYPE2  = 'Norm    '
TUNIT2  = 'mm**2   '
TFORM2  = '1D      '
TTYPE3  = 'Scale   '
TFORM3  = '1D      '
VFUNC1  = 'Square (X)'
VTYPE1  = 'X2      '
VFUNC2  = 'Square (Y)'
VTYPE2  = 'Y2      '
```

An alternative would be:

```
FUNCTION= 'Norm - Scale * (X * X + Y * Y)'
```

To put the whole example from this and the previous section together:

```
FUNCTION= 'Norm - Scale * (X2 + Y2)'
FUNCNAME= 'HRMA_EffArea'
BUNIT   = 'mm**2   '
FAXIS   =          3
TFIELDS =          3
VFIELDS =          2
FTYPE1  = 'X       '
FUNIT1  = 'mm      '
FLMIN1  =      -70.0
FLMAX1  =       70.0
FTYPE2  = 'Y       '
FUNIT2  = 'mm      '
FLMIN2  =      -70.0
FLMAX2  =       70.0
FTYPE3  = 'Energy  '
FUNIT3  = 'keV     '
FAXIS3  =          3
FLMIN3  =        0.0
FLMAX3  =        6.0
TTYPE1  = 'Energy  '
TUNIT1  = 'keV     '
TFORM1  = '1D      '
TTYPE2  = 'Norm    '
TUNIT2  = 'mm**2   '
TFORM2  = '1D      '
TTYPE3  = 'Scale   '
TFORM3  = '1D      '
VFUNC1  = 'Square (X)'
VTYPE1  = 'X2      '
VFUNC2  = 'Square (Y)'
VTYPE2  = 'Y2      '
```

```
Data:
Energy Norm Scale
   0.5 100.0   1.0
   1.5  90.0   0.9
   4.5   80.0   0.8
```

5 VTYPE

Virtual function columns (VTYPEi) are defined through functions:

```
VFUNCi = 'G(P1, P2, P3, ...; C)'
```

Where G is chosen from a defined set of function names and where P_i may be the value of any valid operand – one of the following:

- FTYPEn
- DTYPEn
- a constant number
- TTYPEn
- VTYPEn
- WTYPEn

C is the name of a parameter object or a coefficient object that is specific to the function G and which attributes need to be specified as $C\_ < name >$; not all functions require a parameter object. For each parameter object attribute there has to be one TTYPE, DTYPE, VTYPE, or WTYPE that carries its name as value.

VFIELDS specifies the number of VTYPEs that are defined.

Note that allowing VTYPEs and WTYPEs to be used in the definition of VTYPEs provides for the specification of nested functions. This is a powerful capability that requires particular care to prevent recursion.

Examples from the predefined set of functions (names are case-insensitive):

```
sin (x)             / x in radians
cos (x)             / x in radians
tan (x)             / x in radians
exp (x)
log (x)             / Natural log
square (x)
sqrt (x)
Gauss1D (x; g)      / g consists of g_ampl, g_pos, g_fwhm
Gauss2D (x, y; g)   / g consists of g_ampl, g_posx, g_posy,
                    /             g_fwhm1, g_fwhm2, g_pa
```

```
powerseries (x; p)   / p consists of p_n, p_a
                     / order is n-1; a is a column that contains
                     / vectors of length >=n with the coefficients
fourierseries (x; p) / same convention
etc.
```

A later version of this document will have a definitive list of allowed functions.

6 WTYPE

In general, arithmetic virtual columns (WTYPEi) are defined as:

```
WFUNCi = 'P1 ^ P2 ^ ...'
```

Where "^" denotes an arithmetic operator $+ \ - \ * \ / \ **$; parentheses are allowed.

P_i may be the value of any of the following:

- FTYPEn
- DTYPEn
- a constant number
- TTYPEn
- VTYPEn
- WTYPEn

WFIELDS specifies the number of WTYPEs that are defined.

7 Example

A double Gaussian function:

```
XTENSION= 'BINTABLE'    / Preliminaries, yada yada
BITPIX  =           8
NAXIS   =           2
```

```
NAXIS1  =          52
NAXIS2  =           6
PCOUNT  =           0
GCOUNT  =           1
TFIELDS =           9
EXTNAME = 'FUNCTION'
EXTVER  =           1

COMMENT                                                  +------------------+
COMMENT                                                  | AXAF FITS   File |
COMMENT                                                  +------------------+
COMMENT              ********************************************************
COMMENT              >    This file is written following certain AXAF-ASC   <
COMMENT              >    conventions which are documented in ASC-FITS-1.1   <
COMMENT              ********************************************************
COMMENT                        / Configuration control-------------------------
ORIGIN  = 'ASC     '
CREATOR = 'xxx - Version 0.0'
DATE    = '1998-01-01T00:00:00'  / Date and time of file creation (UTC)
REVISION=                    0 / Processing system revision number
COMMENT $Id: func3.tex,v 1.2 2000/08/30 18:21:09 arots Exp arots $
CHECKSUM= '0000000000000000'   / ASCII encoded HDU checksum
DATASUM = '         0'         / Data unit checksum written in ASCII
CONTENT = '        '           / What data product
HDUNAME = '        '           / If no EXTNAME
HDUSPEC = 'ASC-FITS-FUNCTION-1.2: McDowell, Rots: FITS Embedded Function Specification'
HDUDOC  = 'ASC-FITS-FUNCTION-1.2: McDowell, Rots: FITS Embedded Function Specification'
HDUVERS = '1.0.0   '
HDUCLASS= 'ASC     '
HDUCLAS1= 'FUNCTION'
LONGSTRN= 'OGIP 1.0'           / The OGIP long string convention may be used.
COMMENT   This FITS file may contain long string keyword values that are
COMMENT   continued over multiple keywords.  This convention uses the  '&'
COMMENT   character at the end of a string which is then continued
COMMENT   on subsequent keywords whose name = 'CONTINUE'.

COMMENT                 / This is the output function definition
FUNCTION= 'Campl * CX * CY + Hampl * HX * XY'
FUNCNAME= 'ACIS_PSF'
BUNIT   = '        '

FAXIS   =           4    / Here are the independent variables
FTYPE1  = 'X       '     / X: free running
FUNIT1  = 'mm      '
FLMIN1  =         0.0
FLMAX1  =       100.0
```

```
FTYPE2  = 'Y       '    / Y: free running
FUNIT2  = 'mm      '
FLMIN2  =        0.0
FLMAX2  =      100.0
FTYPE3  = 'Energy  '    / Energy: enumerated
FUNIT3  = 'keV     '
FAXIS3  =          2
FTYPE4  = 'Theta   '    / Off-axis angle: enumerated
FUNIT4  = 'arcmin  '
FAXIS4  =          3

TTYPE1  = 'Energy  '    / Column with Energy enumerated values
TUNIT1  = 'keV     '
TFORM1  = '1E      '
TTYPE2  = 'Theta   '    / Column with enumerated off-axis angle values
TUNIT2  = 'arcmin  '
TFORM2  = '1E      '
TTYPE3  = 'Csigma  '    / Core sigma
TUNIT3  = 'mm      '
TFORM3  = '1E      '
TTYPE4  = 'Campl   '    / Core amplitude
TUNIT4  = 'count   '
TFORM4  = '1E      '
TTYPE5  = 'HaloX_pos'  / Halo X center
TUNIT5  = 'mm      '
TFORM5  = '1E      '
TTYPE6  = 'HaloY_pos'  / Halo Y center
TUNIT6  = 'mm      '
TFORM6  = '1E      '
TTYPE7  = 'Hsigma  '    / Halo sigma
TUNIT7  = 'mm      '
TFORM7  = '1E      '
TTYPE8  = 'Hampl   '    / Halo amplitude
TUNIT8  = 'count   '
TFORM8  = '1E      '
TTYPE9  = 'TRW_ID  '    / Extraneous column
TFORM9  = '20A     '

DTYPE1  = 'CoreX_pos'  / Constant CoreX.pos
DUNIT1  = 'mm      '
DVAL1   =        0.0
DTYPE2  = 'CoreY_pos'  / Constant CoreY.pos
DUNIT2  = 'mm      '
DVAL2   =        0.0
DTYPE3  = 'CoreX_ampl' / Unit CoreX.ampl
DVAL3   =        1.0
```

```
DTYPE4  = 'CoreY_ampl' / Unit CoreY.ampl
DVAL4   =        1.0
DTYPE5  = 'HaloX_ampl' / Unit HaloX.ampl
DVAL5   =        1.0
DTYPE6  = 'HaloY_ampl' / Unit HaloY.ampl
DVAL6   =        1.0

COMMENT                  / Here are the internal function definitions
VFIELDS =          4     / The number of function virtual columns
VTYPE1  = 'CX      '
VFUNC1  = 'Gauss1D (X; CoreX)'
VTYPE2  = 'CY      '
VFUNC2  = 'Gauss1D (Y; CoreY)'
VTYPE3  = 'HX      '
VFUNC3  = 'Gauss1D (X; HaloX)'
VTYPE4  = 'HY      '
VFUNC4  = 'Gauss1D (Y; HaloY)'
COMMENT                  / Here are the internal function definitions
WFIELDS =          4     / The number of arithmetic virtual columns
WTYPE1  = 'CoreX_fwhm'
WFUNC1  = 'Csigma  '
WTYPE2  = 'CoreY_fwhm'
WFUNC2  = 'Csigma  '
WTYPE3  = 'HaloX_fwhm'
WFUNC3  = 'Hsigma  '
WTYPE4  = 'HaloY_fwhm'
WFUNC4  = 'Hsigma  '

END
```