

[Next](#) [Up](#) [Previous](#) [Contents](#) [FITSIO Home](#)

Next: [5.4.3 Good Time Interval](#) **Up:** [5.4 Table Filtering](#) **Previous:** [5.4.1 Column and Keyword Contents](#)

5.4.2 Row Filtering

The row filter is used to select a subset of the rows from a table based on a boolean expression. A temporary new FITS file is created on the fly (usually in memory) which contains only those rows for which the row filter expression evaluates to true (i.e., not equal to zero). The primary array and any other extensions in the input file are also copied to the temporary file. The original FITS file is closed and the new temporary file is then opened by the application program.

The row filter expression is enclosed in square brackets following the file name and extension name. For example, `'file.fits[events][GRADE==50]'` selects only those rows in the EVENTS table where the GRADE column value is equal to 50).

The row filtering expression can be an arbitrarily complex series of operations performed on constants, keyword values, and column data taken from the specified FITS TABLE extension. The expression also can be written into a text file and then used by giving the filename preceded by a '@' character, as in `'[@rowfilt.txt]'`.

(+)

Keyword and column data are referenced by name. Any string of characters not surrounded by quotes (ie, a constant string) or followed by an open parentheses (ie, a function name) will be initially interpreted as a column name and its contents for the current row inserted into the expression. If no such column exists, a keyword of that name will be searched for and its value used, if found. To force the name to be interpreted as a keyword (in case there is both a column and keyword with the same name), precede the keyword name with a single pound sign, '#', as in #NAXIS2. Due to the generalities of FITS column and keyword names, if the column or keyword name contains a space or a character which might appear as an arithmetic term then inclose the name in '\$' characters as in \$MAX PHA\$ or #\$MAX-PHA\$. The names are case insensitive.

To access a table entry in a row other than the current one, follow the column's name with a row offset within curly braces. For example, 'PHA{-3}' will evaluate to the value of column PHA, 3 rows above the row currently being processed. One cannot specify an absolute row number, only a relative offset. Rows that fall outside the table will be treated as undefined, or NULLs.

Boolean operators can be used in the expression in either their Fortran or C forms. The following boolean operators are available:

"equal"	.eq. .EQ. ==	"not equal"	.ne. .NE. !=
"less than"	.lt. .LT. <	"less than/equal"	.le. .LE.
<= =<			
"greater than"	.gt. .GT. >	"greater than/equal"	.ge.
.GE. >= =>			

(+)

"or" .or. .OR. || "and" .and. .AND. &&
 "negation" .not. .NOT. ! "approx. equal(1e-7)" ~

Note that the exclamation point, '!', is a special UNIX character, so if it is used on the command line rather than entered at a task prompt, it must be preceded by a backslash to force the UNIX shell to ignore it.

The expression may also include arithmetic operators and functions. Trigonometric functions use radians, not degrees. The following arithmetic operators and functions can be used in the expression (function names are case insensitive):

"addition"	+	"subtraction"	-
"multiplication"	*	"division"	/
"negation"	-	"exponentiation"	** ^
"absolute value"	abs(x)	"cosine"	cos(x)
"sine"	sin(x)	"tangent"	tan(x)
"arc cosine"	arccos(x)	"arc sine"	arcsin(x)
"arc tangent"	arctan(x)	"arc tangent"	
arctan2(x,y)			
"exponential"	exp(x)	"square root"	sqrt(x)
"natural log"	log(x)	"common log"	log10(x)
"modulus"	i % j	"random # [0.0,1.0)"	
random()			
"minimum"	min(x,y)	"maximum"	
max(x,y)			
"if-then-else"	b?x:y		

The following type casting operators are available, where the inclosing parentheses are required and taken from the C language usage. Also, the integer to real casts values to double precision:

"real to integer"	(int) x	(INT) x	(+)
-------------------	---------	---------	-----

"integer to real" (float) i (FLOAT) i

Several constants are built in for use in numerical expressions:

#pi	3.1415...	#e	2.7182...
#deg	#pi/180	#row	current row
number			
#null	undefined value	#snull	undefined
string			

A string constant must be enclosed in quotes as in 'Crab'. The "null" constants are useful for conditionally setting table values to a NULL, or undefined, value (For example, "col1== -99 ? #NULL : col1").

There is also a function for testing if two values are close to each other, i.e., if they are "near" each other to within a user specified tolerance. The arguments, **value_1** and **value_2** can be integer or real and represent the two values whose proximity is being tested to be within the specified tolerance, also an integer or real:

near(value_1, value_2, tolerance)

When a NULL, or undefined, value is encountered in the FITS table, the expression will evaluate to NULL unless the undefined value is not actually required for evaluation, e.g. "TRUE .or. NULL" evaluates to TRUE. The following two functions allow some NULL detection and handling:

ISNULL(x)

DEFNULL(x,y)

(+)

The former returns a boolean value of TRUE if the argument `x` is NULL. The later "defines" a value to be substituted for NULL values; it returns the value of `x` if `x` is not NULL, otherwise it returns the value of `y`.

Bit masks can be used to select out rows from bit columns (**TFORMn = #x**) in FITS files. To represent the mask, binary, octal, and hex formats are allowed:

```
binary: b0110xx1010000101xxxx0001
octal:  o720x1 -> (b111010000xxx001)
hex:    h0FxD -> (b00001111xxxx1101)
```

In all the representations, an `x` or `X` is allowed in the mask as a wild card. Note that the `x` represents a different number of wild card bits in each representation. All representations are case insensitive.

To construct the boolean expression using the mask as the boolean equal operator described above on a bit table column. For example, if you had a 7 bit column named `flags` in a FITS table and wanted all rows having the bit pattern `0010011`, the selection expression would be:

```
flags == b0010011
or
flags .eq. b10011
```


It is also possible to test if a range of bits is less than, less than equal, greater than and greater than equal to a particular boolean value:

```
flags <= bxxx010xx
flags .gt. bxxx100xx
flags .le. b1xxxxxxx
```

(+)

Notice the use of the x bit value to limit the range of bits being compared.

It is not necessary to specify the leading (most significant) zero (0) bits in the mask, as shown in the second expression above.

Bit wise AND, OR and NOT operations are also possible on two or more bit fields using the '&'(AND), ''(OR), and the '!'(NOT) operators. All of these operators result in a bit field which can then be used with the equal operator. For example:

```
(!flags) == b1101100  
(flags & b1000001) == bx000001
```

Bit fields can be appended as well using the '+' operator. Strings can be concatenated this way, too.

[Next](#) [Up](#) [Previous](#) [Contents](#) [FITSIO Home](#)

Next: [5.4.3 Good Time Interval](#) **Up:** [5.4 Table Filtering](#) **Previous:** [5.4.1 Column and Keyword Contents](#)

(+)