MATLAB Function Reference

Provide feedback about this page

fread

Read binary data from file

Syntax

```
A = fread(fid)
A = fread(fid, count)
A = fread(fid, count, precision)
A = fread(fid, count, precision, skip)
A = fread(fid, count, precision, skip, machineformat)
[A, count] = fread(...)
```

Description

`A = fread(fid)` reads data in binary format from the file specified by `fid` into matrix A. Open the file using `fopen` before calling `fread`. The `fid` argument is the integer file identifier obtained from the `fopen` operation. The MATLAB® software reads the file from beginning to end, and then positions the file pointer at the end of the file (see `feof` for details).

> **Note** `fread` is intended primarily for binary data. When reading text files, use the `fgetl` function.

`A = fread(fid, count)` reads the number of elements specified by count. At the end of the `fread`, MATLAB sets the file pointer to the next byte to be read. A subsequent fread will begin at the location of the file pointer. See Specifying the Number of Elements, below.

> **Note** In the following syntaxes, the `count` and `skip` arguments are optional. For example, `fread(fid, precision)` is a valid syntax.

`A = fread(fid, count, precision)` reads the file according to the data format specified by the string `precision`. This argument commonly contains a data type specifier such as `int` or `float`, followed by an integer giving the size in bits. See Specifying precision and Specifying Output Format, below.

`A = fread(fid, count, precision, skip)` includes an optional `skip` argument that specifies the number of bytes to skip after each `precision` value is read. If `precision` specifies a bit format like `'bitN'` or `'ubitN'`, the `skip` argument is interpreted as the number of bits to skip. See Specifying a Skip Value, below.

`A = fread(fid, count, precision, skip, machineformat)` treats the data read as having a format given by `machineformat`. You can obtain the `machineformat` argument from the output of the `fopen` function. See `fopen` for possible values for `machineformat`.

`[A, count] = fread(...)` returns the data read from the file in A, and the number of elements successfully read in `count`.

Specifying the Number of Elements

Valid options for count are

`n` Reads n elements into a column vector.

`inf` Reads to the end of the file, resulting in a column vector containing the same number of elements as are in the file. If using `inf` results in an "out of memory" error, specify a numeric count value.

`[m,n]` Reads enough elements to fill an `m`-by-n matrix, filling in elements in column order, padding with zeros if the file is too small to fill the matrix. n can be specified as `inf`, but `m` cannot.

Specifying precision

Any of the strings in the following table, either the MATLAB version or their C or Fortran equivalent, can be used for `precision`. If `precision` is not specified, MATLAB uses the default, which is `'uint8'`.

MATLAB	C or Fortran	Interpretation
'schar'	'signed char'	Signed integer; 8 bits
'uchar'	'unsigned char'	Unsigned integer; 8 bits
'int8'	'integer*1'	Integer; 8 bits
'int16'	'integer*2'	Integer; 16 bits
'int32'	'integer*4'	Integer; 32 bits
'int64'	'integer*8'	Integer; 64 bits
'uint8'	'integer*1'	Unsigned integer; 8 bits
'uint16'	'integer*2'	Unsigned integer; 16 bits
'uint32'	'integer*4'	Unsigned integer; 32 bits
'uint64'	'integer*8'	Unsigned integer; 64 bits
'float32'	'real*4'	Floating-point; 32 bits

MATLAB	C or Fortran	Interpretation
`'float64'`	`'real*8'`	Floating-point; 64 bits
`'double'`	`'real*8'`	Floating-point; 64 bits

The following platform-dependent formats are also supported, but they are not guaranteed to be the same size on all platforms.

MATLAB	C or Fortran	Interpretation
`'char'`	`'char*1'`	Character
`'short'`	`'short'`	Integer; 16 bits
`'int'`	`'int'`	Integer; 32 bits
`'long'`	`'long'`	Integer; 32 or 64 bits
`'ushort'`	`'unsigned short'`	Unsigned integer; 16 bits
`'uint'`	`'unsigned int'`	Unsigned integer; 32 bits
`'ulong'`	`'unsigned long'`	Unsigned integer; 32 or 64 bits
`'float'`	`'float'`	Floating-point; 32 bits

> **Note** If the format is `'char'` or `'char*1'`, MATLAB reads characters using the encoding scheme associated with the file. See `fopen` for more information.

The following formats map to an input stream of bits rather than bytes.

MATLAB	C or Fortran	Interpretation
`'bitN'`	-	Signed integer; `N` bits (1 ≤ `N` ≤ 64)

MATLAB	C or Fortran	Interpretation
'ubitN'	-	Unsigned integer; N bits (1 ≤ N ≤ 64)

Specifying Output Format

By default, numeric and character values are returned in class `double` arrays. To return these values stored in classes other than `double`, create your `format` argument by first specifying your source format, then following it with the characters "=>," and finally specifying your destination format. You are not required to use the exact name of a MATLAB class type for destination. (See `class` for details). `fread` translates the name to the most appropriate MATLAB class type. If the source and destination formats are the same, the following shorthand notation can be used.

```
*source
```

which means

```
source=>source
```

For example, `'*uint16'` is the same as `'uint16=>uint16'`.

> **Note** You can also use the `*source` notation with an input stream that is specified as a number of bits (e.g., `bit4` or `ubit18`). MATLAB translates this into an output type that is a signed or unsigned integer (depending on the input type), and that is large enough to hold all of the bits in the source format. For example, `*ubit18` does not translate to `ubit18=>ubit18`, but instead to `ubit18=>uint32`.

This table shows some example precision format strings.

'uint8=>uint8'	Read in unsigned 8-bit integers and save them in an unsigned 8-bit integer array.
'*uint8'	Shorthand version of the above.
'bit4=>int8'	Read in signed 4-bit integers packed in bytes and save them in a signed 8-bit array. Each 4-bit integer becomes an 8-bit integer.
'double=>real*4'	Read in doubles, convert, and save as a 32-bit floating-point array.

Specifying a Skip Value

When `skip` is used, the `precision` string can contain a positive integer repetition factor of the form `'N*'`, which prefixes the source format specification, such as `'40*uchar'`.

Note Do not confuse the asterisk (*) used in the repetition factor with the asterisk used as precision format shorthand. The format string `'40*uchar'` is equivalent to `'40*uchar=>double'`, not `'40*uchar=>uchar'`.

When `skip` is specified, `fread` reads in, at most, a repetition factor number of values (default is 1), skips the amount of input specified by the `skip` argument, reads in another block of values, again skips input, and so on, until `count` number of values have been read. If a `skip` argument is not specified, the repetition factor is ignored. Use the repetition factor with the `skip` argument to extract data in noncontiguous fields from fixed-length records.

Remarks

If the input stream is bytes and `fread` reaches the end of file (see `feof`) in the middle of reading the number of bytes required for an element, the partial result is ignored. However, if the input stream is bits, then the partial result is returned as the last value. If an error occurs before reaching the end of file, only full elements read up to that point are used.

Examples

Example 1

The file `alphabet.txt` contains the 26 letters of the English alphabet, all capitalized. Open the file for read access with `fopen`, and read the first five elements into output `c`. Because a precision has not been specified, MATLAB uses the default precision of `uint8`, and the output is numeric:

```
fid = fopen('alphabet.txt', 'r');
c = fread(fid, 5)'
c =
    65    66    67    68    69
fclose(fid);
```

This time, specify that you want each element read as an unsigned 8-bit integer and output as a character. (Using a precision of `'char=>char'` or `'*char'` will produce the same result):

```
fid = fopen('alphabet.txt', 'r');
c = fread(fid, 5, 'uint8=>char')'
c =
    ABCDE
fclose(fid);
```

When you leave out the optional count argument, MATLAB reads the file to the end, A through Z:

```
fid = fopen('alphabet.txt', 'r');
c = fread(fid, '*char')'
c =
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
fclose(fid);
```

The `fopen` function positions the file pointer at the start of the file. So the first `fread` in this example reads the first five elements in the file, and then repositions the file pointer at the beginning of the next element. For this reason, the next `fread` picks up where the previous `fread` left off, at the character `F`.

```
fid = fopen('alphabet.txt', 'r');
c1 = fread(fid, 5, '*char');
c2 = fread(fid, 8, '*char');
c3 = fread(fid, 5, '*char');
fclose(fid);

sprintf('%c', c1, ' * ', c2, ' * ', c3)
ans =
   ABCDE * FGHIJKLM * NOPQR
```

Skip two elements between each read by specifying a `skip` argument of 2:

```
fid = fopen('alphabet.txt', 'r');
c = fread(fid, '*char', 2);    % Skip 2 bytes per read
fclose(fid);

sprintf('%c', c)
ans =
   ADGJMPSVY
```

Example 2

This command displays the complete M-file containing this `fread` help entry:

```
type fread.m
```

To simulate this command using `fread`, enter the following:

```
fid = fopen('fread.m', 'r');
F = fread(fid, '*char')';
fclose(fid);
```

In the example, the `fread` command assumes the default size, `'inf'`, and precision `'*char'` (the same as `'char=>char'`). `fread` reads the entire file. To display the result as readable text, the column vector is transposed to a row vector.

Example 3

As another example,

```
s = fread(fid, 120, '40*uchar=>uchar', 8);
```

reads in 120 bytes in blocks of 40, each separated by 8 bytes. Note that the class type of s is `'uint8'` since it is the appropriate class corresponding to the destination format `'uchar'`. Also, since 40 evenly divides 120, the last block read is a full block, which means that a final skip is done before the command is finished. If the last block read is not a full block, then `fread` does not finish with a skip.

See `fopen` for information about reading big and little-endian files.

Example 4

Invoke the `fopen` function with just an `fid` input argument to obtain the machine format for the file. You can see that this file was written in IEEE® floating point with little-endian byte ordering (`'ieee-le'`) format:

```
fid = fopen('A1.dat', 'r');

[fname, mode, mformat] = fopen(fid);
mformat
mformat =
   ieee-le
```

Use the MATLAB `format` function (not related to the machine format type) to have MATLAB display output using hexadecimal:

```
format hex
```

Now use the `machineformat` input with `fread` to read the data from the file using the same format:

```
x = fread(fid, 6, 'uint64', 'ieee-le')
x =
   4260800000002000
   0000000000000000
   4282000000180000
   0000000000000000
   42ca5e0000258000
   42f0000464d45200
fclose(fid);
```

Change the machine format to IEEE floating point with big-endian byte ordering (`'ieee-be'`) and verify that you get different results:

```
fid = fopen('A1.dat', 'r');
x = fread(fid, 6, 'uint64', 'ieee-be')
x =
   4370000008400000
   0000000000000000
   4308000200100000
   0000000000000000
   4352c0002f0d0000
   43c022a6a3000000
fclose(fid);
```

Example 5

This example reads some Japanese text from a file that uses the Shift-JIS character encoding scheme. It creates a string of Unicode® characters, `str`, and displays the string. Note that the computer must be configured to display Japanese (e.g., a Japanese machine running the Windows® operating system) for the output of `disp(str)` to be correct.

```
fid = fopen('japanese.txt', 'r', 'n', 'Shift_JIS');
str = fread(fid, '*char')';
fclose(fid);
disp(str);
```

See Also

`fgetl`, `fscanf`, `fwrite`, `fprintf`, `fopen`, `fclose`, `fseek`, `ftell`, `feof`

Provide feedback about this page

◀ frameedit fread (serial) ▶