

MATLAB[®]

The Language of Technical Computing

Computation

Visualization

Programming

Application Program Interface Reference

Version 5

How to Contact The MathWorks:



508-647-7000 Phone



508-647-7001 Fax



The MathWorks, Inc. Mail
24 Prime Park Way
Natick, MA 01760-1500



<http://www.mathworks.com> Web
<ftp.mathworks.com> Anonymous FTP server
<comp.soft-sys.matlab> Newsgroup



support@mathworks.com Technical support
suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
subscribe@mathworks.com Subscribing user registration
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information

Application Program Interface Reference

© COPYRIGHT 1984 - 1998 by The MathWorks, Inc. All Rights Reserved.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

U.S. GOVERNMENT: If Licensee is acquiring the Programs on behalf of any unit or agency of the U.S. Government, the following shall apply: (a) For units of the Department of Defense: the Government shall have only the rights specified in the license under which the commercial computer software or commercial software documentation was obtained, as set forth in subparagraph (a) of the Rights in Commercial Computer Software or Commercial Software Documentation Clause at DFARS 227.7202-3, therefore the rights set forth herein shall apply; and (b) For any other unit or agency: NOTICE: Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction, and disclosure are as set forth in Clause 52.227-19 (c)(2) of the FAR.

MATLAB, Simulink, Handle Graphics, and Real-Time Workshop are registered trademarks and Stateflow and Target Language Compiler are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: December 1996 First printing
May 1997 Revised for 5.1 (online version)
January 1998 Revised for 5.2 (online version)

API Notes

The mex Script	1
The MATLAB Array.	5
Passing Pointers in Fortran	8

DDE Routines

ddeadv.	9
ddeexec	11
ddeinit.	12
ddepoke.	13
ddereq	14
ddeterm.	15
ddeunadv	16

C Engine Routines

engClose	17
engEvalString.	19
engGetArray.	23
engGetFull (Obsolete)	24
engGetMatrix (Obsolete)	25
engOpen	26
engOutputBuffer	29
engPutArray.	30
engPutFull (Obsolete)	31
engPutMatrix (Obsolete)	32
engSetEvalCallback (Obsolete)	33
engSetEvalTimeout (Obsolete)	34
engWinInit (Obsolete)	35

C MAT-File Routines

matClose	36
matDeleteArray	37
matDeleteMatrix (Obsolete)	38
matGetArray	39
matGetArrayHeader	40
matGetDir	41
matGetFp	43
matGetFull (Obsolete)	44
matGetMatrix (Obsolete)	45
matGetNextArray	46
matGetNextArrayHeader	47
matGetNextMatrix (Obsolete)	48
matGetString (Obsolete)	49
matOpen	50
matPutArray	51
matPutArrayAsGlobal	52
matPutFull (Obsolete)	53
matPutMatrix (Obsolete)	54
matPutString (Obsolete)	55

C MEX-Functions

mexAddFlops	56
mexAtExit	57
mexCallMATLAB	60
mexErrMsgTxt	63
mexEvalString	64
mexFunction	67
mexFunctionName	70
mexGet	71
mexGetArray	74
mexGetArrayPtr	77
mexGetEps (Obsolete)	80
mexGetFull (Obsolete)	81
mexGetGlobal (Obsolete)	82
mexGetInf (Obsolete)	83

mexGetMatrix (Obsolete)	84
mexGetMatrixPtr (Obsolete)	85
mexGetNaN (Obsolete)	86
mexIsFinite (Obsolete)	87
mexIsGlobal	88
mexIsInf (Obsolete)	89
mexIsLocked	90
mexIsNaN (Obsolete)	91
mexLock	92
mexMakeArrayPersistent	93
mexMakeMemoryPersistent	94
mexPrintf	95
mexPutArray	97
mexPutFull (Obsolete)	100
mexPutMatrix (Obsolete)	101
mexSet	102
mexSetTrapFlag	103
mexUnlock	104
mexWarnMsgTxt	105

C MX-Functions

mxArrayToString	108
mxAssert	110
mxAssertS	111
mxCalcSingleSubscript	112
mxCalloc	117
mxChar	120
mxClassID	121
mxClearLogical	123
mxComplexity	124
mxCreateCellArray	125
mxCreateCellMatrix	127
mxCreateCharArray	129
mxCreateCharMatrixFromStrings	132
mxCreateDoubleMatrix	135
mxCreateFull (Obsolete)	137

mxCreateNumericArray	138
mxCreateSparse	141
mxCreateString	144
mxCreateStructArray	146
mxCreateStructMatrix	149
mxDestroyArray	151
mxDuplicateArray	152
mxFree	153
mxFreeMatrix (Obsolete)	155
mxGetCell	156
mxGetClassID	159
mxGetClassName	162
mxGetData	163
mxGetDimensions	164
mxGetElementSize	166
mxGetEps	168
mxGetField	169
mxGetFieldByNumber	172
mxGetFieldNameByNumber	175
mxGetFieldNumber	178
mxGetImagData	179
mxGetInf	180
mxGetIr	181
mxGetJc	184
mxGetM	186
mxGetN	187
mxGetName	189
mxGetNaN	191
mxGetNumberOfDimensions	192
mxGetNumberOfElements	195
mxGetNumberOfFields	196
mxGetNzmax	197
mxGetPi	199
mxGetPr	201
mxGetScalar	204
mxGetString	206
mxIsCell	208
mxIsChar	209
mxIsClass	210
mxIsComplex	212

mxIsDouble	213
mxIsEmpty	214
mxIsFinite	215
mxIsFromGlobalWS	216
mxIsFull (Obsolete)	217
mxIsInf	218
mxIsInt8	219
mxIsInt16	220
mxIsInt32	221
mxIsLogical	222
mxIsNaN	223
mxIsNumeric	224
mxIsSingle	225
mxIsSparse	226
mxIsString (Obsolete)	227
mxIsStruct	228
mxIsUint8	229
mxIsUint16	230
mxIsUint32	231
mxMalloc	232
mxRealloc	234
mxSetAllocFcns	235
mxSetCell	237
mxSetClassName	239
mxSetData	240
mxSetDimensions	241
mxSetField	244
mxSetFieldByNumber	247
mxSetImagData	248
mxSetIr	249
mxSetJc	251
mxSetLogical	254
mxSetM	255
mxSetN	258
mxSetName	262
mxSetNzmax	263
mxSetPi	266
mxSetPr	269

Fortran Engine Routines

engClose	272
engEvalString	273
engGetFull	274
engGetMatrix	276
engOpen	277
engOutputBuffer	278
engPutFull	279
engPutMatrix	280

Fortran MAT-File Routines

matClose	281
matDeleteMatrix	282
matGetDir	283
matGetFull	285
matGetMatrix	287
matGetNextMatrix	289
matGetString	291
matOpen	293
matPutFull	294
matPutMatrix	296
matPutString	298

Fortran MEX-Functions

mexAtExit	299
mexCallMATLAB	300
mexErrMsgTxt	302
mexEvalString	303
mexFunction	304
mexGetEps	305
mexGetFull	306
mexGetGlobal	307

mexGetInf	308
mexGetMatrix	309
mexGetMatrixPtr	310
mexGetNaN	311
mexIsFinite	312
mexIsInf	313
mexIsNaN	314
mexPrintf	315
mexPutFull	316
mexPutMatrix	317
mexSetTrapFlag	318

Fortran MX-Functions

mxCalloc	319
mxCopyCharacterToPtr	320
mxCopyComplex16ToPtr	321
mxCopyInteger4ToPtr	322
mxCopyPtrToCharacter	323
mxCopyPtrToComplex16	324
mxCopyPtrToInteger4	325
mxCopyPtrToPtrArray	326
mxCopyPtrToReal8	327
mxCopyReal8ToPtr	328
mxCreateFull	329
mxCreateSparse	330
mxCreateString	331
mxFree	332
mxFreeMatrix	333
mxGetIr	334
mxGetJc	335
mxGetM	336
mxGetN	337
mxGetName	338
mxGetNzmax	339
mxGetPi	340
mxGetPr	341

mxGetScalar	342
mxGetString	343
mxIsComplex.	344
mxIsDouble	345
mxIsFull	346
mxIsNumeric.	347
mxIsSparse	348
mxIsString	349
mxSetIr	350
mxSetJc.	351
mxSetM.	352
mxSetN	353
mxSetName.	354
mxSetNzmax.	355
mxSetPi.	356
mxSetPr.	357

Purpose Compiles a MEX-function from C or Fortran source code

Syntax MEX <options> <files>

Arguments All non-source code file names passed as arguments are passed to the linker without being compiled.

These options are available on all platforms except where noted:

Option	Function
-argcheck	Perform argument checking on MATLAB API functions (C functions only).
-c	Compile only; do not link.
-D<name>[=<def>]	(UNIX and Macintosh) Define C preprocessor macro <name> [as having value <def>.]
-D<name>	(Windows) Define C preprocessor macro <name>.
-f <file>	(UNIX and Windows) Use <file> as the options file; <file> is a full path name if it is not in current directory. . (On Windows, not necessary if you use the -setup option.)
-f <file>	(Macintosh) Use <file> as the options file. (Not necessary if you use the -setup option.) If <file> is specified, it is used as the options file. If <file> is not specified and there is a file called mexopts in the current directory, it is used as the options file. If <file> is not specified and mexopts is not in the current directory and there is a file called mexopts in the directory <matlab>:extern:scripts:, it is used as the options file. Otherwise, an error occurs.

The mex Script

Option	Function
-F <file>	(UNIX) Use <file> as the options file. <file> is searched for in the following manner: The file that occurs first in this list is used: <ul style="list-style-type: none">• ./<filename>• \$HOME/matlab/<filename>• \$TMW_ROOT/bin/<filename>
-F <file>	(Windows) Use <file> as the options file. (Not necessary if you use the -setup option.) <file> is searched for in the current directory first and then in the same directory as mex.bat.
-g	Build an executable with debugging symbols included.
-h[elp]	Help; lists the switches and their functions.
-I<pathname>	Include <pathname> in the compiler include search path.
-l<file>	(UNIX) Link against library lib<file>.
-L<pathname>	(UNIX) Include <pathname> in the list of directories to search for libraries.
<name>=<def>	(UNIX and Macintosh) Override options file setting for variable <name>.
-n	No execute flag. Using this option causes the commands that would be used to compile and link the target to be displayed without executing them.
-output <name>	Create an executable named <name>. (An appropriate executable extension is automatically appended.)
-O	Build an optimized executable.

Option	Function
-setup	(Windows and Macintosh Only) Set up default options file. This switch should be the only argument passed.
-U<name>	(UNIX and Windows) Undefine C preprocessor macro <name>.
-V4	Compile MATLAB 4-compatible MEX-files.
-v	Verbose; print all compiler and linker settings.

Description

MEX <options> <files> compiles a MEX-function from C or Fortran source code. All non-source code file names passed as arguments are passed to the linker without being compiled.

MEX's execution is affected by both command-line arguments and an options file. The options file contains all compiler-specific information necessary to create a MEX-function. The default name for this options file, if none is specified with the -f option, is mexopts.bat (Windows), mexopts.sh (UNIX), and mexopts (Macintosh).

Note: The MathWorks provides an option (setup) for the mex script that lets you set up a default options file on Windows and Macintosh systems. See the *Application Program Interface Guide* for additional information.

On UNIX, the options file is written in the Bourne shell script language. The options file that occurs first in the following list is used:

```
./mexopts.sh, $HOME/matlab/mexopts.sh, $MATLAB/bin/mexopts.sh.
```

Any variable specified in the options file can be overridden at the command line by use of the <name>=<def> command line argument. If <def> has spaces in it, then it should be wrapped in single quotes (e.g., OPTFLAGS='opt1 opt2'). The definition can rely on other variables defined in the options file; in this case the variable referenced should have a prepended \$ (e.g., OPTFLAGS='\$OPTFLAGS opt2').

The mex Script

On Windows, the options file is written in the Perl script language. The options file, `mexopts.bat`, is searched for in the following directories: the current directory first, then the same directory as `mex.bat`. No arguments can have an embedded equal sign (=); thus, `-DF00` is valid, but `-DF00=BAR` is not.

On the Macintosh, the options file is written in the MPW scripting language for MPW C and Language Systems Fortran, and in M-code language for Metrowerks Codewarrior C. The default location for the options file is the `<MATLABROOT>:extern:scripts` folder. Any variable specified in the options file can be overridden at the command line by use of the `<name>=<def>` command line argument. If `<def>` has spaces in it, then it should be wrapped in single quotes (e.g., `OPTIMFLAGS='opt1 opt2'`). The definition can rely on other variables defined in the options file; in this case the variable referenced should have a prepended `$` (e.g., `OPTIMFLAGS='$OPTIMFLAGS opt2'`).

Description

The MATLAB language works with only a single object type: the MATLAB array. All MATLAB variables, including scalars, vectors, matrices, strings, cell arrays, structures, and objects are stored as MATLAB arrays. In C, the MATLAB array is declared to be of type `mxArray`. The `mxArray` structure contains, among other things:

- Its type
- Its dimensions
- The data associated with this array
- If numeric, whether the variable is real or complex
- If sparse, its indices and nonzero maximum elements
- If a structure or object, the number of fields and fieldnames

Data Storage

All MATLAB data is stored columnwise. This is how Fortran stores matrices; MATLAB uses this convention because it was originally written in Fortran. For example, given the matrix:

```
a=['house'; 'floor'; 'porch']
```

```
a =  
house  
floor  
porch
```

its dimensions are:

```
size(a)
```

```
ans =  
3     5
```

and its data is stored as:

h	f	p	o	l	o	u	o	r	s	o	c	e	r	h
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Data Types in MATLAB

Complex Double-Precision Matrices

The most common data type in MATLAB is the complex double-precision, nonsparse matrix. These matrices are of type `double` and have dimensions `m-by-n`, where `m` is the number of rows and `n` is the number of columns. The data is stored as two vectors of double-precision numbers — one contains the real data and one contains the imaginary data. The pointers to this data are referred to as `pr` (pointer to real data) and `pi` (pointer to imaginary data), respectively. A real-only, double-precision matrix is one whose `pi` is `NULL`.

Numeric Matrices

MATLAB also supports other types of numeric matrices. These are single-precision floating-point and 8-, 16-, and 32-bit integers, both signed and unsigned. The data is stored in two vectors in the same manner as double-precision matrices.

MATLAB Strings

MATLAB strings are of type `char` and are stored the same way as unsigned 16-bit integers except there is no imaginary data component. Each character in the string is stored as 16-bit ASCII Unicode. Unlike C, MATLAB strings are not null terminated.

Sparse Matrices

Sparse matrices have a different storage convention in MATLAB. The parameters `pr` and `pi` are still arrays of double-precision numbers, but there are three additional parameters, `nzmax`, `ir`, and `jc`:

- `nzmax` is an integer that contains the length of `ir`, `pr`, and, if it exists, `pi`. It is the maximum possible number of nonzero elements in the sparse matrix.
- `ir` points to an integer array of length `nzmax` containing the row indices of the corresponding elements in `pr` and `pi`.
- `jc` points to an integer array of length `N+1` that contains column index information. For `j`, in the range $0 \leq j \leq N-1$, `jc[j]` is the index in `ir` and `pr` (and `pi` if it exists) of the first nonzero entry in the `j`th column and `jc[j+1] - 1` index of the last nonzero entry. As a result, `jc[N]` is also equal to `nnz`, the number of nonzero entries in the matrix. If `nnz` is less than `nzmax`,

then more nonzero entries can be inserted in the array without allocating additional storage.

Cell Arrays

Cell arrays are a collection of MATLAB arrays where each `mxArray` is referred to as a cell. This allows MATLAB arrays of different types to be stored together. Cell arrays are stored in a similar manner to numeric matrices, except the data portion contains a single vector of pointers to `mxArrays`. Members of this vector are called cells. Each cell can be of any supported data type, even another cell array.

Structures

A 1-by-1 structure is stored in the same manner as a 1-by-n cell array where n is the number of fields in the structure. Members of the data vector are called fields. Each field is associated with a name stored in the `mxArray`.

Objects

Objects are stored and accessed the same way as structures. In MATLAB, objects are named structures with registered methods. Outside MATLAB, an object is a structure that contains storage for an additional classname that identifies the name of the object.

Multidimensional Arrays

MATLAB arrays of any type can be multidimensional. A vector of integers is stored where each element is the size of the corresponding dimension. The storage of the data is the same as matrices.

Logical Arrays

Any noncomplex numeric or sparse array can be flagged as logical. The storage for a logical array is the same as the storage for a nonlogical array.

Empty Arrays

MATLAB arrays of any type can be empty. An empty `mxArray` is one with at least one dimension equal to zero. For example, a double-precision `mxArray` of type `double`, where m and n equal 0 and pr is `NULL`, is an empty array.

Passing Pointers in Fortran

Description

The MATLAB API works with a unique data type, the `mxArray`. Because there is no way to create a new data type in Fortran, MATLAB passes a special identifier, called a pointer, to a Fortran program. You can get information about an `mxArray` by passing this pointer to various API functions called “Access Routines”. These access routines allow you to get a native Fortran data type containing exactly the information you want, i.e., the size of the `mxArray`, whether or not it is a string, or its data contents.

There are several implications when using pointers in Fortran:

1 The %VAL construct.

If your Fortran compiler supports the `%VAL` construct, then there is one type of pointer you can use without requiring an access routine, namely a pointer to data (i.e., the pointer returned by `mxGetPr` or `mxGetPi`). You can use `%VAL` to pass this pointer’s contents to a subroutine, where it is declared as a Fortran double-precision array.

If your Fortran compiler does not support the `%VAL` construct, you must use the `mxCopy__` routines (e.g., `mxCopyPtrToReal8`) to access the contents of the pointer.

2 Variable declarations.

To use pointers properly, you must declare them to be the correct size. On DEC Alpha and 64-bit SGI machines, all pointers should be declared as `integer*8`. On all other platforms, pointers should be declared as `integer*4`.

If your Fortran compiler supports preprocessing with the `C` preprocessor, you can use the preprocessing stage to map pointers to the appropriate declaration. In UNIX, see the examples ending with `.F` in the `examples` directory for a possible approach.

Note: Declaring a pointer to be the incorrect size can cause your program to crash.

Purpose	Set up advisory link between MATLAB and DDE server application
Syntax	<code>rc = ddeadv(channel, item, callback, upmtx, format, timeout)</code>
Arguments	<p><code>rc</code> The return code: 0 indicates the function call failed, 1 indicates it succeeded.</p> <p><code>channel</code> The channel assigned to the conversation, returned by <code>ddeinit</code>.</p> <p><code>item</code> A string that specifies the DDE item name for the advisory link. Changing the data identified by <code>item</code> at the server triggers the advisory link.</p> <p><code>callback</code> A string that specifies the callback that is evaluated on update notification. Changing <code>item</code> at the server causes <code>callback</code> to get passed to the <code>eval</code> function to be evaluated.</p> <p><code>upmtx</code> (optional) A string that specifies the name of a matrix that holds data sent with update notification. If <code>upmtx</code> is included, changing <code>item</code> at the server causes <code>upmtx</code> to be updated with the revised data.</p> <p>Specifying an update matrix creates a <i>hot link</i>. Omitting <code>upmtx</code> or specifying it as an empty string, creates a <i>warm link</i>. If <code>upmtx</code> exists in the workspace, its contents get overwritten. If <code>upmtx</code> does not exist, it is created.</p> <p><code>format</code> (optional) A two-element array that specifies the format of the data to be sent on update.</p> <p>The first element specifies the Windows clipboard format to use for the data. MATLAB supports only Text format, which corresponds to a value of 1. The second element specifies the type of the resultant matrix. Valid types are <code>NUMERIC</code> (the default, which corresponds to a value of 0) and <code>STRING</code> (which corresponds to a value of 1).</p> <p>The default format array is [1 0].</p> <p><code>timeout</code> (optional) A scalar that specifies the time-out limit for this operation. <code>timeout</code> is specified in milliseconds (1000 milliseconds = 1 second).</p>

ddeadv

If advisory link is not established within `timeout` milliseconds, the function fails. The default value of `timeout` is three seconds.

Description

`ddeadv` sets up an advisory link (described in the “DDE Advisory Links” section of the *Application Program Interface Guide*) between MATLAB and a server application.

When the data identified by the `item` argument changes, the string specified by the `callback` argument is passed to the `eval` function and evaluated. If the advisory link is a hot link, DDE modifies `upmtx`, the update matrix, to reflect the data in `item`.

If `item` corresponds to a range of data values, a change to any value in the range causes `callback` to be evaluated.

Example

```
% Set up a hot link between a range of cells in Excel
% and the matrix 'x'.
% If successful, display the matrix.
rc = ddeadv(channel, 'r1c1:r5c5', 'disp(x)', 'x');
```

Purpose	Send execution string to DDE server application
Syntax	<code>rc = ddeexec(channel, command, item, timeout)</code>
Arguments	<p><code>rc</code> The return code: 0 indicates the function call failed, 1 indicates it succeeded.</p> <p><code>channel</code> The channel assigned to the conversation, returned by <code>ddeinit</code>.</p> <p><code>command</code> A string that specifies the command to be executed.</p> <p><code>item</code> (optional) A string that specifies the DDE item name for execution. This argument is not used for many applications. If your application requires this argument, it provides additional information for <code>command</code>. Consult your server documentation for more information.</p> <p><code>timeout</code> (optional) A scalar that specifies the time-out limit for this operation. <code>timeout</code> is specified in milliseconds (1000 milliseconds = 1 second). The default value of <code>timeout</code> is three seconds.</p>
Description	<code>ddeexec</code> sends a string for execution to another application via an established DDE conversation. Specify the string as the <code>command</code> argument.
Example	<pre>% Given the channel assigned to a conversation, % send a command to Excel. rc = ddeexec(channel, '[formula.goto("r1c1")]');</pre>

ddeinit

Purpose Initiate DDE conversation between MATLAB and another application

Syntax `channel = ddeinit(service, topic)`

Arguments
`channel`
The channel assigned to the conversation.

`service`
A string that specifies the service or application name for the conversation.

`topic`
A string that specifies the topic for the conversation.

Description `ddeinit` requires two arguments: a service or application name and a topic for that service. The function returns a channel handle, which is used with other MATLAB DDE functions.

For more information about services and topics, see the “DDE Concepts and Terminology” section in the *Application Program Interface Guide*.

Example

```
% Initiate a conversation with Microsoft Excel
% for the spreadsheet 'forecast.xls'.
channel = ddeinit('excel', 'forecast.xls');
```

Purpose	Send data from MATLAB to DDE server application
Syntax	<code>rc = ddepoke(channel, item, data, format, timeout)</code>
Arguments	<p><code>rc</code> The return code: 0 indicates the function call failed, 1 indicates it succeeded.</p> <p><code>channel</code> The channel assigned to the conversation, returned by <code>ddeinit</code>.</p> <p><code>item</code> A string that specifies the DDE item for the data sent. <code>item</code> is the server data entity that is to contain the data sent in the <code>data</code> argument.</p> <p><code>data</code> A matrix that contains the data to be sent.</p> <p><code>format</code> (optional) A scalar that specifies the Windows clipboard format of the data. MATLAB supports only Text format, which corresponds to a value of 1.</p> <p><code>timeout</code> (optional) A scalar that specifies the time-out limit for this operation. <code>timeout</code> is specified in milliseconds (1000 milliseconds = 1 second). The default <code>timeout</code> is three seconds.</p>
Description	<p><code>ddepoke</code> sends data to an application via an established DDE conversation. <code>ddepoke</code> formats the data matrix as follows before sending it to the server application:</p> <ul style="list-style-type: none">• String matrices are converted, element by element, to characters and the resulting character buffer is sent.• Numeric matrices are sent as tab-delimited columns and carriage-return, line-feed delimited rows of numbers. Only the real part of non-sparse matrices are sent.
Example	<pre>% Send a 5-by-5 identity matrix to Excel. rc = ddepoke(channel, 'r1c1:r5c5', eye(5));</pre>

ddereq

Purpose	Request data from DDE server application
Syntax	<code>data = ddereq(channel, item, format, timeout)</code>
Arguments	<p><code>data</code> A matrix that contains the requested data, empty if the function call failed.</p> <p><code>channel</code> The channel assigned to the conversation, returned by <code>ddeinit</code>.</p> <p><code>item</code> A string that specifies the server application's DDE item name for the data requested.</p> <p><code>format</code> (optional) A two-element array that specifies the format of the data requested. The first element indicates a Windows clipboard format to use for the request. MATLAB supports only Text format, which corresponds to a value of 1. The second element of the format array specifies the type of the resultant matrix. The valid types are NUMERIC (the default, corresponding to a value of 0) and STRING (corresponding to a value of 1). The default format array is [1 0].</p> <p><code>timeout</code> (optional) A scalar that specifies the time-out limit for this operation. <code>timeout</code> is specified in milliseconds (1000 milliseconds = 1 second). The default <code>timeout</code> is three seconds.</p>
Description	<code>ddereq</code> requests data from a server application via an established DDE conversation. <code>ddereq</code> returns a matrix containing the requested data or an empty matrix if the function is unsuccessful.
Example	<pre>% Request a matrix of cells from Excel. mymtx = ddereq(channel, 'r1c1:r10c10');</pre>

Purpose	Terminate DDE conversation between MATLAB and server application
Syntax	<code>rc = ddeterm(channel)</code>
Arguments	<code>rc</code> The return code: 0 indicates the function call failed, 1 indicates it succeeded. <code>channel</code> The channel assigned to the conversation, returned by <code>ddeinit</code> .
Description	<code>ddeterm</code> takes one argument, the channel handle returned by the previous call to <code>ddeinit</code> that established the DDE conversation.
Example	<pre>% Terminate the DDE conversation. rc = ddeterm(channel);</pre>

ddeunadv

Purpose Release an advisory link between MATLAB and DDE server application

Syntax `rc = ddeunadv(channel, item, format, timeout)`

Arguments

`rc`
The return code: 0 indicates the function call failed, 1 indicates it succeeded.

`channel`
The channel assigned to the conversation, returned by `ddeinit`.

`item`
A string that specifies the DDE item name associated with the advisory link.

`format`
(optional) A two-element array that specifies the format of the data for the advisory link. If you specified a `format` argument on the `ddeadv` function call that defined the advisory link, you must specify the same value on the `ddeunadv` function call. See `ddeadv` for a description of the `format` array.

`timeout`
(optional) A scalar that specifies the time-out limit for this operation. `timeout` is specified in milliseconds (1000 milliseconds = 1 second). The default value of `timeout` is three seconds.

Description `ddeunadv` releases the advisory link between MATLAB and the server application, established by an earlier `ddeadv` call. The `channel`, `item`, and `format` must be the same as those specified in the call to `ddeadv` that initiated the link. If you include the `timeout` argument but accept the default `format`, you must specify `format` as an empty matrix.

Example

```
% Release the hot link established in the ddeadv example.
rc = ddeunadv(channel, 'r1c1:r5c5');

% Release a hot link with default format and a timeout value.
rc = ddeunadv(chan, 'r1c1:r5c5', [], 6000);
```

Purpose	Quit a MATLAB engine session
C Syntax	<pre>#include "engine.h" int engClose(Engine *ep);</pre>
Arguments	ep Engine pointer.
Description	<p>This routine allows you to quit a MATLAB engine session.</p> <p>engClose sends a quit command to the MATLAB engine session and closes the connection. It returns 0 on success, and 1 otherwise. Possible failure includes attempting to terminate a MATLAB engine session that was already terminated.</p>
Examples	<p>For UNIX:</p> <pre>/* engtest2.c */ #include <stdio.h> #include <stdlib.h> #include "engine.h" void main() { ep = engOpen(""); if (ep == NULL) { fprintf(stderr, "Unable to start MATLAB session\n"); exit(EXIT_FAILURE); } void engClose(ep); exit(EXIT_SUCCESS); }</pre> <p>For Windows:</p> <pre>/*ENGTEST2*/ #include <stdlib.h> #include <stdio.h> #include "engine.h" int WINAPI WinMain(HANDLE hInstance,</pre>

engClose

```
        HANDLE hPrevInstance,  
        LPSTR lpszCmdLine,  
        int nCmdShow)  
{  
    Engine *ep;  
  
    if (!(ep = engOpen())) {  
        MessageBox ((HWND) NULL,  
            (LPSTR) "Can't start MATLAB engine",  
            (LPSTR) "Engtest2. c", MB_OK);  
    }  
  
    engClose(ep);  
    return (TRUE);  
}
```

See `engdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a C program.

Purpose	Evaluate expression in string
C Syntax	<pre>#include "engine.h" int engEvalString(Engine *ep, const char *string);</pre>
Arguments	<p>ep Engine pointer.</p> <p>string String to execute.</p>
Description	<p>engEvalString evaluates the expression contained in string for the MATLAB engine session, ep, previously started by engOpen. It returns a nonzero value if the MATLAB session is no longer running, and zero otherwise.</p> <p>On UNIX systems, engEvalString sends commands to MATLAB by writing down a pipe connected to MATLAB's <i>stdin</i>. Any output resulting from the command that ordinarily appears on the screen is read back from <i>stdout</i> into the buffer defined by engOutputBuffer.</p> <p>Under Windows on a PC, engEvalString communicates with MATLAB via ActiveX.</p>
Examples	<p>Send a simple mxArray to the engine, compute its eigenvalues, get back the vector containing the eigenvalues, and print the second one:</p> <p>For UNIX:</p> <pre>/* engtest1.c */ #include <stdio.h> #include "engine.h" static double Areal[6] = {1, 2, 3, 4, 5, 6}; void main() { Engine *ep; Matrix *a, *d; double *Dreal, *Dimag; a = mxCreateDoubleMatrix(3, 2, mxREAL); memcpy(mxGetPr(a), Areal, 6*sizeof(double)); mxSetName(a, "A"); if (!(ep = engOpen(""))) {</pre>

engEvalString

```
fprintf(stderr, "\nCan't start MATLAB engine");
exit(EXIT_FAILURE);
}

engPutArray(ep, a);
engEvalString(ep, "d = eig(A*A')");
d = engGetArray(ep, "d");
engClose(ep);

Dreal = mxGetPr(d);
Dimag = mxGetPi(d);

if (Dimag) {
    printf("Eigenval 2: %g+%gi", Dreal[1], Dimag[1]);
} else {
    printf("Eigenval 2: %g\n", Dreal[1]);
}

mxFree(a);
mxFree(d);
}
```

For Windows:

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "engine.h"

static double Areal[6] = { 1, 2, 3, 4, 5, 6 };

int WINAPI WinMain (HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow)
{
    Engine *ep;
    Matrix *a, *d;
    double *Dreal, *Di mag;
    char buff[256];

    a = mxCreateDoubleMatrix(3, 2, mxREAL);
    memcpy((char *) mxGetPr(a), (char *) Areal, 6*sizeof(double));
    mxSetName(a, "A");

    if (!(ep = engOpen(NULL))) {
        MessageBox ((HWND) NULL,
                   (LPSTR) "Can't start MATLAB engine",
                   (LPSTR) "Engtest1.c", MB_OK);
    } else {
        mxfree(a);
        return(TRUE);
        engPutArray(ep, a);
        engEvalString(ep, "d = eig(A*A')");
        d = engGetArray(ep, "d");
        engClose(ep);

        Dreal = mxGetPr(d);
        Di mag = mxGetPi(d);

        if (Di mag){
            sprintf(buff, "Eigenval 2: %g+%gi",

```

engEvalString

```
Dreal [1], Di mag[1]);
} else {
    sprintf(buff, "Ei genval 2: %g", Dreal [1]);
    MessageBox((HWND) NULL, (LPSTR) buff,
        (LPSTR) "Engtest1. c", MB_OK);
}
mxFree(d);
}
mxFree(a);
return (TRUE);
}
```

See `engdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a C program.

Purpose	Copy a variable from a MATLAB engine's workspace
C Syntax	<pre>#include "engine.h" mxArray *engGetArray(Engine *ep, const char *name);</pre>
Arguments	<p>ep Engine pointer.</p> <p>name Name of mxArray to get from engine.</p>
Description	<p>This routine allows you to copy a variable out of the workspace.</p> <p>engGetArray reads the named mxArray from the engine pointed to by ep and returns a pointer to a newly allocated mxArray structure, or NULL if the attempt fails. engGetArray will fail if:</p> <ul style="list-style-type: none">• The named variable does not exist.• <i>(V4 compatible mode under UNIX, or any mode under Windows)</i> The named variable is not a MATLAB 4 data type. <p>Be careful in your code to free the mxArray created by this routine when you are finished with it.</p>
Example	See engdemo.c in the eng_mat subdirectory of the examples directory for a sample program that illustrates how to call the MATLAB engine functions from a C program.
See Also	engPutArray

engGetFull (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

`engGetArray` followed by appropriate `mxGet` routines (`mxGetM`, `mxGetN`,
`mxGetPr`, `mxGetPi`)

See Also `engGetArray` and examples in the `eng_mat` subdirectory of the `examples` directory

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
engGetArray
```

See Also `engGetArray`, `engPutArray`, and examples in the `eng_mat` subdirectory of the `examples` directory

engOpen

Purpose Start a MATLAB engine session

C Syntax

```
#include "engine.h"
Engine *engOpen(const char *startcmd);
```

Arguments `ep`
Engine pointer.

`startcmd`
String to start MATLAB process.
Note: On Windows, this string must be NULL.

Description This routine allows you to start a MATLAB process for the purpose of using MATLAB as a computational engine.

`engOpen(startcmd)` starts a MATLAB process using the command specified in the string `startcmd`, establishes a connection, and returns a unique engine identifier, or NULL if the open fails.

On UNIX systems, if `startcmd` is NULL or the empty string, `engOpen` starts MATLAB on the current host using the command `matlab`. If `startcmd` is a hostname, `engOpen` starts MATLAB on the designated host by embedding the specified hostname string into the larger string:

```
"rsh hostname \"/bin/csh -c 'setenv DISPLAY\
hostname:0; matlab' \"
```

If `startcmd` is any other string (has white space in it, or nonalphanumeric characters), the string is executed literally to start MATLAB.

On UNIX systems, `engOpen` performs the following steps:

- 1 Creates two pipes.
- 2 Forks a new process and sets up the pipes to pass *stdin* and *stdout* from MATLAB (parent) to two file descriptors in the engine program (child).
- 3 Executes a command to run MATLAB (`rsh` for remote execution).

Under Windows on a PC, `engOpen` opens an ActiveX channel to MATLAB. This starts the MATLAB that was registered during installation. If you did not register during installation, on the command line you can enter the command:

```
matlab /regserver
```

See “ActiveX Automation for Windows” in the *Application Program Interface Guide* for additional details.

Examples

Start a MATLAB engine on the UNIX machine that you are currently logged into:

```
/* engtest2.c */
#include <stdio.h>
#include "engine.h"

void main()
{
    Engine ep;

    if (!(ep = engOpen(NULL))) {
        fprintf(stderr, "\nCan't start MATLAB engine");
        exit(EXIT_FAILURE);
    }
}
```

Call `engOpen` to start a process on the UNIX machine called `labrea`:

```
engOpen("labrea");
```

Call `engOpen` to run MATLAB on Fred’s account on the UNIX machine called `labrea`, and set the X display to the machine called `wilkinson`:

```
engOpen("rsh -l fred labrea \"/bin/csh -c \
'setenv DISPLAY wilkinson:0; matlab' \"");
```

See `engdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a C program on a UNIX machine.

Start a MATLAB engine in Windows on a PC:

```
/*ENGTEST2*/
#include <stdlib.h>
#include <stdio.h>
#include "engine.h"

int WINAPI WinMain (HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow)
{
    Engine *ep;

    if (!(ep = engOpen(NULL))) {
        MessageBox ((HWND) NULL, (LPSTR) "Can't start MATLAB
engine",
                   (LPSTR) "Engtest2.c", MB_OK);
    }

    engClose(ep);
    return TRUE;
}
```

See `engwindemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a C program in Windows.

Purpose	Specify buffer for MATLAB output
C Syntax	<pre>#include "engine.h" int engOutputBuffer(Engine *ep, char *p, int n);</pre>
Arguments	<p><code>ep</code> Engine pointer.</p> <p><code>n</code> Length of buffer <code>p</code>.</p> <p><code>p</code> Pointer to character buffer of length <code>n</code>.</p>
Description	<p><code>engOutputBuffer</code> defines a character buffer for <code>engEvalString</code> to return any output that ordinarily appears on the screen.</p> <p>The default behavior of <code>engEvalString</code> is to discard any standard output caused by the command it is executing. <code>engOutputBuffer(ep, p, n)</code> tells any subsequent calls to <code>engEvalString</code> to save the first <code>n</code> characters of output in the character buffer pointed to by <code>p</code>.</p>
Example	See <code>engdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to call the MATLAB engine functions from a C program.

engPutArray

Purpose Put variables into a MATLAB engine's workspace

C Syntax

```
#include "engine.h"
int engPutArray(Engine *ep, const mxArray *mp);
```

Arguments

ep
Engine pointer.

mp
mxArray pointer.

Description This routine allows you to put variables into a MATLAB engine's workspace. engPutArray writes mxArray mp to the engine ep. If the mxArray does not exist in the workspace, it is created. If an mxArray with the same name already exists in the workspace, the existing mxArray is replaced with the new mxArray. engPutArray returns 0 if successful and 1 if an error occurs. In V4 compatibility mode, engPutArray will fail if the mxArray mp is not a MATLAB 4 data type.

Example See engdemo.c in the eng_mat subdirectory of the examples directory for a sample program that illustrates how to call the MATLAB engine functions from a C program.

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mxCreateDoubleMatrix  
engPutArray
```

See Also `engGetArray`, `mxCreateDoubleMatrix`

engPutMatrix (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

`engPutArray`

See Also `engPutArray`

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program.

engSetEvalTimeout (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program.

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. This function is not necessary in MATLAB 5 engine programs.

matClose

Purpose	Closes a MAT-file
C Syntax	<pre>#include "mat.h" int matClose(MATFile *mfp);</pre>
Arguments	<p><code>mfp</code> Pointer to MAT-file information.</p>
Description	<code>matClose</code> closes the MAT-file associated with <code>mfp</code> . It returns EOF for a write error, and zero if successful.
Example	See <code>matdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

Purpose	Delete named <code>mxArray</code> from MAT-file
C Syntax	<pre>#include "mat.h" int matDeleteArray(MATFile *mfp, const char *name);</pre>
Arguments	<p><code>mfp</code> Pointer to MAT-file information.</p> <p><code>name</code> Name of <code>mxArray</code> to delete.</p>
Description	<code>matDeleteArray</code> deletes the named <code>mxArray</code> from the MAT-file pointed to by <code>mfp</code> . <code>matDeleteArray</code> returns 0 if successful, nonzero otherwise.
Example	See <code>matdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

matDeleteMatrix (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

```
matDeleteArray
```

See Also `matDeleteArray`

Purpose	Read mxArray from MAT-files
C Syntax	<pre>#include "mat.h" mxArray *matGetArray(MATFile *mfp, const char *name);</pre>
Arguments	<p>mfp Pointer to MAT-file information.</p> <p>name Name of mxArray to get from MAT-file.</p>
Description	<p>This routine allows you to copy an mxArray out of a MAT-file.</p> <p>matGetArray reads the named mxArray from the MAT-file pointed to by mfp and returns a pointer to a newly allocated mxArray structure, or NULL if the attempt fails.</p> <p>Be careful in your code to free the mxArray created by this routine when you are finished with it.</p>
Example	See matdemo.c in the eng_mat subdirectory of the examples directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

matGetArrayHeader

Purpose Load array header information only

C Syntax

```
#include "mat.h"
mxArray *matGetArrayHeader(MATFile *mfp, const char *name);
```

Arguments

`mfp`
Pointer to MAT-file information.

`name`
Name of `mxArray`.

Description `matGetArrayHeader` loads only the array header information, including everything except `pr`, `pi`, `ir`, and `jc`. It recursively creates the cells/structures through their leaf elements, but does not include `pr`, `pi`, `ir`, and `jc`. If `pr`, `pi`, `ir`, and `jc` are set to non-NULL when loaded with `matGetArray`, `matGetArrayHeader` sets them to -1 instead. These headers are for informational use only and should NEVER be passed back to MATLAB or saved to MAT-files.

Example See `matdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

Purpose	Get directory of <code>mxArrays</code> in a MAT-file
C Syntax	<pre>#include "mat.h" char **matGetDir(MATFile *mfp, int *num);</pre>
Arguments	<p><code>mfp</code> Pointer to MAT-file information.</p> <p><code>num</code> Address of the variable to contain the number of <code>mxArrays</code> in the MAT-file.</p>
Description	<p>This routine allows you to get a list of the names of the <code>mxArrays</code> contained within a MAT-file.</p> <p><code>matGetDir</code> returns a pointer to an internal array containing pointers to the NULL-terminated names of the <code>mxArrays</code> in the MAT-file pointed to by <code>mfp</code>. The length of the internal array (number of <code>mxArrays</code> in the MAT-file) is placed into <code>num</code>. The internal array is allocated using a single <code>mxCallLoc</code> and must be freed using <code>mxFree</code> when you are finished with it.</p> <p><code>matGetDir</code> returns NULL and sets <code>num</code> to a negative number if it fails. If <code>num</code> is zero, <code>mfp</code> contains no arrays.</p> <p>MATLAB variable names can be up to length <code>mxMAXNAM</code>, where <code>mxMAXNAM</code> is defined in the file <code>matrix.h</code>.</p>
Examples	<p>Print out a directory of the <code>mxArray</code> names contained within a MAT-file:</p> <pre>/* mattest7.c */ #include <stdio.h> #include "mat.h" void main() { MATFile *mfp; char **dir; int ndir, i; mfp = matOpen("foo.mat", "r"); dir = matGetDir(mfp, &ndir); matClose(mfp); if (dir == NULL && ndir < 0) { printf("Can't read directory.\n"); } }</pre>

matGetDir

```
        exit(0);
    } else {
        printf("Directory of MAT-file: \n");
        for (i = 0; i < ndir; i++) {
            printf("%s\n", dir[i]);
        }
    }
    mxFree(dir);
}
```

See `matdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

Purpose	Get file pointer to a MAT-file
C Syntax	<pre>#include "mat.h" FILE *matGetFp(MATFile *mfp);</pre>
Arguments	<p><code>mfp</code> Pointer to MAT-file information.</p>
Description	<code>matGetFp</code> returns the C file handle to the MAT-file with handle <code>mfp</code> . This can be useful for using standard C library routines like <code>ferror()</code> and <code>feof()</code> to investigate error situations.
Example	See <code>matdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

matGetFull (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

`matGetArray` followed by the appropriate `mxGet` routines

See Also `matGetArray`

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

```
matGetArray
```

See Also `matGetArray`

matGetNextArray

Purpose	Read next <code>mxArray</code> from MAT-file
C Syntax	<pre>#include "mat.h" mxArray *matGetNextArray(MATFile *mfp);</pre>
Arguments	<code>mfp</code> Pointer to MAT-file information.
Description	<p><code>matGetNextArray</code> allows you to step sequentially through a MAT-file and read all the <code>mxArrays</code> in a single pass.</p> <p><code>matGetNextArray</code> reads the next <code>mxArray</code> from the MAT-file pointed to by <code>mfp</code> and returns a pointer to a newly allocated <code>mxArray</code> structure. Use it immediately after opening the MAT-file with <code>matOpen</code> and not in conjunction with other MAT-file routines; otherwise, the concept of the <i>next</i> <code>mxArray</code> is undefined.</p> <p><code>matGetNextArray</code> returns <code>NULL</code> when the end-of-file is reached or if there is an error condition. Use <code>feof</code> and <code>ferror</code> from the Standard C Library to determine status.</p> <p>Be careful in your code to free the <code>mxArray</code> created by this routine when you are finished with it.</p>
Example	See <code>matdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

Purpose	Load array header information only
C Syntax	<pre>#include "mat.h" mxArray *matGetNextArrayHeader(MATFile *mfp);</pre>
Arguments	<p><code>mfp</code> Pointer to MAT-file information.</p>
Description	<code>matGetNextArrayHeader</code> loads only the array header information, including everything except <code>pr</code> , <code>pi</code> , <code>ir</code> , and <code>jc</code> , from the file's current file offset.
Example	See <code>matdemo.c</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.
See Also	<code>matGetNextArray</code> , <code>matGetArrayHeader</code>

matGetNextMatrix (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

```
matGetNextArray
```

See Also `matGetNextArray`

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

```
#include "mat.h"
#include "matrix.h"
mxArray *matGetArray(MATFile *mfp, const char *name);
int mxGetString(const mxArray *array_ptr, char *buf, int buflen)
```

See Also `matGetArray`, `mxGetString`

matOpen

Purpose Opens a MAT-file

C Syntax

```
#include "mat.h"
MATFile *matOpen(const char *filename, const char *mode);
```

Arguments

`filename`
Name of file to open.

`mfp`
Pointer to MAT-file information.

`mode`
File opening mode.

Description This routine allows you to open MAT-files for reading and writing. `matOpen` opens the named file and returns a file handle, or NULL if the open fails. Legal values for `mode` are

<code>r</code>	Opens file for reading only; determines the current version of the MAT-file by inspecting the files and preserves the current version.
<code>u</code>	Opens file for update, both reading and writing, but does not create the file if the file does not exist (equivalent to the <code>r+</code> mode of <code>fopen</code>); determines the current version of the MAT-file by inspecting the files and preserves the current version.
<code>w</code>	Opens file for writing only; deletes previous contents, if any.
<code>w4</code>	Creates a MATLAB 4 MAT-file, rather than the default MATLAB 5 MAT-file.

Example See `matdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

Purpose	Write mxArray into MAT-files
C Syntax	<pre>#include "mat.h" int matPutArray(MATFile *mfp, const mxArray *mp);</pre>
Arguments	<p>mfp Pointer to MAT-file information.</p> <p>mp mxArray pointer.</p>
Description	<p>This routine allows you to put an mxArray into a MAT-file.</p> <p>matPutArray writes mxArray mp to the MAT-file mfp. If the mxArray does not exist in the MAT-file, it is appended to the end. If an mxArray with the same name already exists in the file, the existing mxArray is replaced with the new mxArray by rewriting the file. The size of the new mxArray can be different than the existing mxArray.</p> <p>matPutArray returns 0 if successful and nonzero if an error occurs. Use feof and ferror from the Standard C Library along with matGetFp to determine status.</p>
Example	See matdemo.c in the eng_mat subdirectory of the examples directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

matPutArrayAsGlobal

Purpose Put mxArray into MAT-files

C Syntax

```
#include "mat.h"
int matPutArrayAsGlobal(MATFile *mfp, const mxArray *mp);
```

Arguments

`mfp`
Pointer to MAT-file information.

`mp`
mxArray pointer.

Description This routine allows you to put an mxArray into a MAT-file. `matPutArrayAsGlobal` is similar to `matPutArray`, except the array is loaded by MATLAB into the global workspace and a reference to it is set in the local workspace. If you write to a MATLAB 4 format file, `matPutArrayAsGlobal` will not load it as global, and will act the same as `matPutArray`.

`matPutArrayAsGlobal` writes mxArray `mp` to the MAT-file `mfp`. If the mxArray does not exist in the MAT-file, it is appended to the end. If an mxArray with the same name already exists in the file, the existing mxArray is replaced with the new mxArray by rewriting the file. The size of the new mxArray can be different than the existing mxArray.

`matPutArrayAsGlobal` returns 0 if successful and nonzero if an error occurs. Use `feof` and `ferror` from the Standard C Library with `matGetFp` to determine status.

Example See `matdemo.c` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to use the MATLAB MAT-file routines in a C program.

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

`mxCreat eDoubl eMatrix` and `matPutArray`

See Also `mxCreat eDoubl eMatrix`, `matPutArray`

matPutMatrix (Obsolete)

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
matPutArray
```

See Also `matPutArray`

V4 Compatible This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 you should call

```
#include "matrix.h"
#include "mat.h"
mxArray *mxCreateString(char *str)
int matPutArray(MATFile *mfp, const mxArray *mp);
void mxDestroyArray(mxArray *array_ptr)
```

See Also `matPutArray`

mexAddFlops

Purpose	Update MATLAB's internal floating-point operations (flops) counter
C Syntax	<pre>#include "mex.h" void mexAddFlops(int count);</pre>
Arguments	<code>count</code> Specified value.
Description	The <code>mexAddFlops</code> function adds the number specified to MATLAB's internal floating-point operations (flops) counter. Use <code>mexAddFlops</code> when you want MATLAB's internal counter to accurately reflect the number of flops executed by your MEX-file.

Purpose	Register a function to be called when the MEX-file is cleared or when MATLAB terminates
C Syntax	<pre>#include "mex.h" int mexAtExit(void (*ExitFcn)(void));</pre>
Arguments	<code>ExitFcn</code> Pointer to function you wish to run on exit.
Returns	Always returns 0.
Description	<p>Use <code>mexAtExit</code> to register a C function to be called just before the MEX-file is cleared or MATLAB is terminated. <code>mexAtExit</code> gives your MEX-file a chance to perform tasks such as freeing persistent memory and closing files. Typically, the named <code>ExitFcn</code> performs tasks like closing streams or sockets.</p> <p>Each MEX-file can register only one active exit function at a time. If you call <code>mexAtExit</code> more than once, MATLAB uses the <code>ExitFcn</code> from the more recent <code>mexAtExit</code> call as the exit function.</p> <p>If a MEX-file is locked, all attempts to clear the MEX-file will fail. Consequently, if a user attempts to clear a locked MEX-file, MATLAB does not call the <code>ExitFcn</code>.</p>
Examples	<p>Consider a MEX-file named <code>WrtFile</code> that calls <code>mexAtExit</code> to register an exit function. The first time <code>WrtFile</code> is called, <code>WrtFile</code> invokes the ANSI <code>fopen</code> routine to open a file named <code>MyData</code> for writing. Each time <code>WrtFile</code> is called, <code>WrtFile</code> writes the first five values of the input vector to <code>MyData</code>. When <code>WrtFile</code> is cleared, MATLAB automatically invokes <code>ExitFcn</code>, which calls <code>fclose</code> to close the stream to <code>MyData</code>.</p>

mexAtExit

```
FILE *fp;
int first_time=1;
/* Here is the exit function. */
void CloseStream(void)
{
    mexPrintf("Closing MyData. \n");
    fclose(fp);
}
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    double *start;
    int c;
/* Register an exit function. */
mexAtExit(CloseStream);
/* If this is the first time that this MEX-file was called,
    open a write-only stream to file "MyData" */
    if (first_time) {
        fp = fopen("MyData", "w");
        if (fp == NULL)
            mexErrMsgTxt("Could not open MyData. ");
        else
            mexPrintf("Opening MyData. \n");
            mexAtExit(CloseStream);
            first_time = 0;
    }

/* The user passes a vector in prhs[0]; write the first five
    elements of this vector to the data file. */
    start = mxGetPr(prhs[0]);
    for (c=0; c<5; c++)
        fprintf(fp, "%.2g\t", *start++);
        fprintf(fp, "\n");
    }
}
```

Calling `WrtFile` three times writes three lines of data to `MyData`.

```
>> WrtFile(1: 5)
Opening MyData.
>> WrtFile([2 3 5 7 11])
>> WrtFile([3 5 7 9 11])
```

Clearing `WrtFile` causes the exit function to be called.

```
>> clear WrtFile
Closing MyData.
```

The contents of `MyData` are

```
>> type MyData
1      2      3      4      5
2      3      5      7      11
3      5      7      9      11
```

For an additional example, see `mexatexit.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexLock`, `mexUnlock`

mexCallMATLAB

Purpose	Call a MATLAB function, or a user-defined M-file or MEX-file
C Syntax	<pre>#include "mex.h" int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[], const char *command_name);</pre>
Arguments	<p>nlhs Number of desired output arguments. This value must be less than or equal to 50.</p> <p>plhs Pointer to an array of <code>mxArrays</code>. The called command puts pointers to the resultant <code>mxArrays</code> into <code>plhs</code>. Note that the called command allocates dynamic memory to store the resultant <code>mxArrays</code>. By default, MATLAB automatically deallocates this dynamic memory when you clear the MEX-file. However, if heap space is at a premium, you may want to call <code>mxDestroyArray</code> as soon as you are finished with the <code>mxArrays</code> that <code>plhs</code> points to.</p> <p>nrhs Number of input arguments. This value must be less than or equal to 50.</p> <p>prhs Pointer to an array of input arguments.</p> <p>command_name Character string containing the name of the MATLAB built-in, operator, M-file, or MEX-file that you are calling. If <code>command_name</code> is an operator, just place the operator inside a pair of single quotes; for example, '+'.</p>
Returns	0 if successful, and a nonzero value if unsuccessful.
Description	<p>Call <code>mexCallMATLAB</code> to invoke internal MATLAB numeric functions, MATLAB operators, M-files, or other MEX-files. See <code>mexFunction</code> for a complete description of the arguments.</p> <p>By default, if <code>command_name</code> detects an error, MATLAB terminates the MEX-file and returns control to the MATLAB prompt. If you want a different error behavior, turn on the trap flag by calling <code>mexSetTrapFlag</code>.</p>

Note that it is possible to generate an object of type `mxUNKNOWN_CLASS` using `mexCallMATLAB`. For example, if you create an M-file that returns two variables but only assigns one of them a value,

```
function [a, b]=foo(c)
a=2*c;
```

you'll get this warning message in MATLAB:

```
Warning: One or more output arguments not assigned during
call to 'foo'.
```

MATLAB assigns output `b` to an empty matrix. If you then call `foo` using `mexCallMATLAB`, the unassigned output variable will now be of type `mxUNKNOWN_CLASS`.

Examples

Create a populated `mxArray` and call `mexCallMATLAB` to display its contents. Then, call `mexCallMATLAB` a second time to calculate the eigenvalues and eigenvectors of the newly created `mxArray`. Finally, call `mexCallMATLAB` a third time to display the eigenvalues.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    double pr[] = {5.2, 7.9, 1.3, 4.2};
    double pi[] = {3.4, 6.5, 2.2, 9.1};
    mxArray *array_ptr;
    int num_out, num_in;
    mxArray *output_array[2], *input_array[2];
    /* Create a 2-by-2 populated matrix. */
    array_ptr = mxCreateDoubleMatrix(2, 2, mxREAL);
    memcpy(mxGetPr(array_ptr), pr, sizeof(pr));
    memcpy(mxGetpi(array_ptr), pi, sizeof(pi));
    /* Equivalent to disp(array) */
    num_out = 0;
    num_in = 1;
    input_array[0] = array_ptr;
    mexCallMATLAB(num_out, output_array, num_in, input_array,
"disp");
    /* Equivalent to [v, d] = eig(array) */
```

mexCallMATLAB

```
    num_out = 2;
    num_in = 1;
    input_array[0] = array_ptr;
    mexCallMATLAB(num_out, output_array, num_in, input_array,
"ei g");
/* Equivalent to disp(v) */
    num_out = 0;
    num_in = 1;
    input_array[0] = output_array[0];
    mexCallMATLAB(num_out, output_array, num_in, input_array,
"di sp");
}
```

For an additional example, see `mexcallmatlab.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexFunction`, `mexSetTrapFlag`

Purpose	Issue error message and return to the MATLAB prompt
C Syntax	<pre>#include "mex.h" void mexErrMsgTxt(const char *error_msg);</pre>
Arguments	<p><code>error_msg</code> String containing the error message to be displayed.</p>
Description	<p>Call <code>mexErrMsgTxt</code> to write an error message to the MATLAB window. After the error message prints, MATLAB terminates the MEX-file and returns control to the MATLAB prompt.</p> <p>Calling <code>mexErrMsgTxt</code> does not clear the MEX-file from memory. Consequently, <code>mexErrMsgTxt</code> does not invoke the exit function.</p> <p>If your application called <code>mxCallLoc</code> or one of the <code>mxCreate</code> routines to allocate memory, <code>mexErrMsgTxt</code> automatically frees the allocated memory.</p>
Examples	<p>Determine the number of input arguments passed to the MEX-file. If that number is not 2, display an error message and return to the MATLAB prompt</p> <pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) { if (nrhs != 2) mexErrMsgTxt("You must pass 2 rhs args."); else { ... } }</pre> <p>For an additional example, see <code>mexerrmsgtxt.c</code> in the <code>mex</code> subdirectory of the <code>examples</code> directory.</p>
See Also	<code>mexWarnMsgTxt</code>

mexEvalString

Purpose	Execute a MATLAB command in the workspace of the caller
C Syntax	<pre>#include "mex.h" int mexEvalString(const char *command);</pre>
Arguments	<pre>command</pre> A string containing the MATLAB command to execute.
Returns	0 if successful, and a nonzero value if unsuccessful.
Description	<p>Call <code>mexEvalString</code> to invoke a MATLAB command in the workspace of the caller.</p> <p><code>mexEvalString</code> and <code>mexCallMATLAB</code> both execute MATLAB commands. However, <code>mexCallMATLAB</code> provides a mechanism for returning results (left-hand side arguments) back to the MEX-file; <code>mexEvalString</code> provides no way for return values to be passed back to the MEX-file.</p> <p>All arguments that appear to the right of an equals sign in the command string must already be current variables of the caller's workspace.</p>
Examples	<p>Consider an M-file named <code>FibSqr.m</code> that invokes a MEX-file named <code>CalcFib</code>:</p> <pre>function r = FibSqr(n) CalcFib(n); % CalcFib loads the nth Fibonacci number into variable fib. r = fib.^2;</pre> <p>Suppose the <code>CalcFib</code> calculates the <i>n</i>th Fibonacci number. (MEX-files tend to calculate Fibonacci numbers significantly faster than M-files.)</p>

CalcFib calls mexEvalString to write the result into a variable named fib.

```

void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    int    n, c, status;
    double answer, most_recent, next_most_recent;
    char   command_string[20];

    /* Find the nth Fibonacci number. */
    n = (int)mxGetScalar(prhs[0]);
    if (n < 1)
        mexErrMsgTxt("First argument must be positive");
    else if (n<3)
        mexEvalString("fib = 1;");
    else {
        for (most_recent=1, next_most_recent=1, c=3; c<=n; c++)
        {
            answer = most_recent + next_most_recent;
            next_most_recent = most_recent;
            most_recent = answer;
        }
        sprintf(command_string, "fib = %g", answer);
        status = mexEvalString(command_string);
        if (status)
            mexErrMsgTxt("Could not Calculate this Squi bonacci
number.");
    }
}

```

Invoking FibSqr yields

```

>> FibSqr(7);
fib =
    13

ans =
    169

```

mexEvalString

For an additional example, see `mexevalstring.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexCallMATLAB`

Purpose	Entry point to a C MEX-file
C Syntax	<pre>#include "mex.h" void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]);</pre>
Arguments	<p><code>nlhs</code> MATLAB sets <code>nlhs</code> with the number of expected <code>mxArrays</code>.</p> <p><code>plhs</code> MATLAB sets <code>plhs</code> to a pointer to an array of NULL pointers.</p> <p><code>nrhs</code> MATLAB sets <code>nrhs</code> to the number of input <code>mxArrays</code>.</p> <p><code>prhs</code> MATLAB sets <code>prhs</code> to a pointer to an array of input <code>mxArrays</code>. These <code>mxArrays</code> are declared as <code>constant</code>; they are read only and should not be modified by your MEX-file. Changing the data in these <code>mxArrays</code> may produce undesired side effects.</p>
Description	<p><code>mexFunction</code> is not a routine you call. Rather, <code>mexFunction</code> is the generic name of the function entry point that must exist in every C source MEX-file. When you invoke a MEX-function, MATLAB finds and loads the corresponding MEX-file of the same name. MATLAB then searches for a symbol named <code>mexFunction</code> within the MEX-file. If it finds one, it calls the MEX-function using the address of the <code>mexFunction</code> symbol. If MATLAB cannot find a routine named <code>mexFunction</code> inside the MEX-file, it issues an error message.</p> <p>When you invoke a MEX-file, MATLAB automatically seeds <code>nlhs</code>, <code>plhs</code>, <code>nrhs</code>, and <code>prhs</code> with the caller's information. In the syntax of the MATLAB language, functions have the general form</p> $[a, b, c, \dots] = \text{fun}(d, e, f, \dots)$ <p>where the <code>...</code> denotes more items of the same format. The <code>a, b, c, ...</code> are left-hand side arguments and the <code>d, e, f, ...</code> are right-hand side arguments. The arguments <code>nlhs</code> and <code>nrhs</code> contain the number of left-hand side and right-hand side arguments, respectively, with which the MEX-function is called. <code>prhs</code> is a pointer to a length <code>nrhs</code> array of</p>

pointers to the right-hand side `mxArrays`. `plhs` is a pointer to a length `nlhs` array where your C function must put pointers for the returned left-hand side `mxArrays`.

Examples

Consider a MEX-file named `FirstMex` that figures out the class (category) of each input argument. `FirstMex` then places a random scalar in each left-hand side argument:

```
mxClassID determine_class(mxArray *array_ptr)
{
    mxClassID category;
    /* Display the class name of each input mxArray. */
    category = mxGetClassID(array_ptr);
    switch (category) {
        case mxCHAR_CLASS:    mexPrintf("String ");    break;
        case mxSTRUCT_CLASS:  mexPrintf("Structure "); break;
        case mxSPARSE_CLASS:  mexPrintf("Sparse ");    break;
        case mxCELL_CLASS:    mexPrintf("Cell ");      break;
        case mxUNKNOWN_CLASS: mexWarnMsgTxt("Unknown Class.");
                             break;
        default:              mexPrintf("Full Numeric "); break;
    }
}

void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    mxArray *array_ptr, *new_array_ptr;
    int c;
    double pr[1];

    /* Examine input (right-hand-side) arguments. */
    mexPrintf("\nThere are %d right-hand-side arguments.",
nrhs);
    for (c=0; c<nrhs; c++) {
        mexPrintf("\n\tInput Arg %d: ", c+1);
        array_ptr = (mxArray *)prhs[c];
        determine_class(array_ptr);
    }
}
```

```
/* Examine output (left-hand-side) arguments. */
mexPrintf("\n\nThere are %d left-hand-side arguments. \n",
nlhs);
for (c=0; c<nlhs; c++) {
    plhs(c) = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxGetPr(plhs(c))(0) = rand();
}
}
```

For an additional example, see `mexfunction.c` in the `mex` subdirectory of the `examples` directory.

mexFunctionName

Purpose	Gives the name of the current MEX-function
C Syntax	<pre>#include "mex.h" const char *mexFunctionName;</pre>
Arguments	none
Returns	The name of the current MEX-function.
Description	<code>mexFunctionName</code> returns the name of the current MEX-function.

Purpose	Get the value of the specified Handle Graphics property
C Syntax	<pre>#include "mex.h" const mxArray *mexGet(double handle, const char *property);</pre>
Arguments	<p><code>handle</code> Handle to a particular graphics object.</p> <p><code>property</code> A Handle Graphics property.</p>
Returns	The value of the specified property in the specified graphics object on success. Returns NULL on failure. The return argument from <code>mexGet</code> is declared as <code>constant</code> , meaning that it is read only and should not be modified. Changing the data in these <code>mxArrays</code> may produce undesired side effects.
Description	Call <code>mexGet</code> to get the value of the property of a certain graphics object. <code>mexGet</code> is the API equivalent of MATLAB's <code>get</code> function. To set a graphics property value, call <code>mexSet</code> .

Examples

Consider a MEX-file that expects a graphics handle as its first input argument. The MEX-file asks for the Color property associated with the handle, and then modifies this color:

```
#define red 0
#define green 1
#define blue 2

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[])
{
    double         handle;
    const mxArray  *color_array_ptr;
    mxArray        *new_color_array_ptr;
    double         *color, *new_color;

    /* Assume that the first input argument is a graphics
       handle. */
    if(nrhs != 1 || !mxIsDouble(prhs[0]))
        mexErrMsgTxt("Must be called with a valid handle");
    handle = mxGetScalar(prhs[0]);

    /* Get the "Color" property associated with this handle. */
    color_array_ptr = mexGet(handle, "Color");
    if (color_array_ptr == NULL)
        mexErrMsgTxt("Could not get this handle property");

    /* The returned "Color" property is a 1-by-3 matrix of
       primary colors. */
    color = mxGetPr(color_array_ptr);
    /* Create a new mxArray for color */
    new_color_array_ptr = mxCreateDoubleMatrix(1, 3, mxREAL);
    new_color = mxGetPr(plhs[0]);

    new_color[red] = (1 + color[red]) /2;
    new_color[green] = color[green]/2;
    new_color[blue] = color[blue]/2;
}
```

```
/* Reset the "Color" property to use the new color. */
if(mexSet(handle, "Color", new_color_array_ptr))
    mexErrMsgTxt("Could not set a new 'Color' property.");
}
```

For an additional example, see `mexget.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexSet`

mexGetArray

Purpose	Get a copy of a variable from another workspace						
C Syntax	<pre>#include "mex.h" mxAArray *mexGetArray(const char *name, const char *workspace);</pre>						
Arguments	<p>name Name of the variable to copy into the MEX-file workspace.</p> <p>workspace Specifies where <code>mexGetArray</code> should search in order to find variable <code>name</code>. The possible values are</p> <table><tr><td>"base"</td><td>Search for variable <code>name</code> in the current MATLAB workspace.</td></tr><tr><td>"caller"</td><td>Search for variable <code>name</code> in the workspace of whatever entity (M-file, another MEX-file, MATLAB) called this MEX-file.</td></tr><tr><td>"global"</td><td>Search for variable <code>name</code> in the list of global variables. If variable <code>name</code> exists but is not tagged as a global variable, then <code>mexGetArray</code> returns NULL.</td></tr></table>	"base"	Search for variable <code>name</code> in the current MATLAB workspace.	"caller"	Search for variable <code>name</code> in the workspace of whatever entity (M-file, another MEX-file, MATLAB) called this MEX-file.	"global"	Search for variable <code>name</code> in the list of global variables. If variable <code>name</code> exists but is not tagged as a global variable, then <code>mexGetArray</code> returns NULL.
"base"	Search for variable <code>name</code> in the current MATLAB workspace.						
"caller"	Search for variable <code>name</code> in the workspace of whatever entity (M-file, another MEX-file, MATLAB) called this MEX-file.						
"global"	Search for variable <code>name</code> in the list of global variables. If variable <code>name</code> exists but is not tagged as a global variable, then <code>mexGetArray</code> returns NULL.						
Returns	A copy of the <code>mxAArray</code> on success. Returns NULL on failure. A common cause of failure is specifying a name not currently in the workspace. Perhaps the variable was in the workspace at one time but has since been cleared.						
Description	<p>Call <code>mexGetArray</code> to copy the specified variable name into your MEX-file's workspace. Once inside your MEX-file's workspace, your MEX-file may examine or modify the variable's data and characteristics.</p> <p>The returned <code>mxAArray</code> contains a copy of all the data and characteristics that variable <code>name</code> had in the other workspace. <code>mexGetArray</code> initializes the <code>name</code> field of the returned <code>mxAArray</code> to the variable name.</p>						
Examples	Consider a MEX-file named <code>FrwdBack</code> that uses <code>mexGetArray</code> to grab a variable named <code>Tangerines</code> from the MATLAB workspace. The MEX-file						

then manipulates the data in `Tangerines`, and finally calls `mexPutArray` to place `Tangerines` back into the MATLAB workspace.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    mxArray *array_ptr;
    double *pr;
    int num_elements, c;
    const char *n;

    /* Get variable "Tangerines" from the MATLAB workspace. */
    array_ptr = mexGetArray("Tangerines", "base");
    if (array_ptr == NULL)
        mexErrMsgTxt("Could not get Tangerines from MATLAB
workspace. ");

    /* Manipulate Tangerine's real data. */
    num_elements = mxGetM(array_ptr) * mxGetN(array_ptr);
    pr = mxGetPr(array_ptr);
    pr++;
    for (c=2; c<num_elements; c++) {
        *pr = *(pr-1) + *(pr+1);
        pr++;
    }

    /* Copy the modified "Tangerines" back to the MATLAB
workspace. */
    mexPutArray(array_ptr, "base");
}
```

In MATLAB, create a variable named `Tangerines`. Then, call `FrwdBack`:

```
>> Tangerines = 1:10;
>> FrwdBack;
```

mexGetArray

The call to `mexPutArray` puts `Tangeri nes` back into MATLAB's workspace. The resulting value of `Tangeri nes` is:

```
>> Tangeri nes
      1      4      8     13     19     26     34     43     53     10
```

For an additional example, see `mexgetarray.c` in the `mex` subdirectory of the `exampl es` directory.

See Also

`mexGetArrayPtr`, `mexPutArray`

Purpose	Get a read-only pointer to a variable from another workspace						
C Syntax	<pre>#include "mex.h" const mxArray *mexGetArrayPtr(const char *name, const char *workspace);</pre>						
Arguments	<p>name Name of a variable in another workspace. (Note that this is a variable name, not an mxArray pointer.)</p> <p>workspace Specifies which workspace you want <code>mexGetArrayPtr</code> to search. The three possible values are</p> <table><tr><td>"base"</td><td>Search the current variables of MATLAB.</td></tr><tr><td>"caller"</td><td>Search the current variables of whatever entity (M-file, another MEX-file, MATLAB workspace) called this MEX-file.</td></tr><tr><td>"global"</td><td>Search the current global variables of MATLAB only.</td></tr></table>	"base"	Search the current variables of MATLAB.	"caller"	Search the current variables of whatever entity (M-file, another MEX-file, MATLAB workspace) called this MEX-file.	"global"	Search the current global variables of MATLAB only.
"base"	Search the current variables of MATLAB.						
"caller"	Search the current variables of whatever entity (M-file, another MEX-file, MATLAB workspace) called this MEX-file.						
"global"	Search the current global variables of MATLAB only.						
Returns	A read-only pointer to <code>name</code> on success. Returns <code>NULL</code> on failure.						
Description	Call <code>mexGetArrayPtr</code> to get a read-only copy of the specified variable name into your MEX-file's workspace. This command is useful for examining an <code>mxArray</code> 's data and characteristics, but useless for changing them. If you need to change data or characteristics, call <code>mexGetArray</code> instead of <code>mexGetArrayPtr</code> . If you simply need to examine data or characteristics, <code>mexGetArrayPtr</code> offers superior performance as the caller need pass only a pointer to the array. By contrast, <code>mexGetArray</code> passes back the entire array.						

mexGetArrayPtr

Examples

Consider a MEX-file named `FrstLst` that displays the value of the first and last elements of the `Limes` variable

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    const mxArray *array_ptr;
    double *pr, value_of_first_element,
value_of_last_element;
    int dims[5];
    int num_elements;

    /* Get the array "Limes" from the MATLAB workspace. */
    array_ptr = mexGetArrayPtr("Limes", "base");
    if (array_ptr == NULL)
        mexErrMsgTxt("Could not get Limes from MATLAB
workspace.");

    /* Display the value of first and last element of Limes. */
    pr = (double *)mexGetPr(array_ptr);
    value_of_first_element = *pr;
    num_elements = mxGetM(array_ptr) * mxGetN(array_ptr);
    value_of_last_element = *(pr + (num_elements - 1));
    mexPrintf("First: %g\n", value_of_first_element);
    mexPrintf("Last: %g\n", value_of_last_element);

}
```

Create a `Limes` variable containing a lot of data. Then, call `FrstLst`

```
>> Limes=magic(499);
>> FrstLst
First: 124752
Last: 124250
```

If `FrstLst` calls `mexGetArray` instead of `mexGetArrayPtr`, then all 249,001 elements of `Limes` are copied. By calling `mexGetArrayPtr`, the only thing that gets copied is the address of the start of the `Limes` array.

For an additional example, see `mexgetarrayptr.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexGetArray`

mexGetEps (Obsolete)

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
eps = mxGetEps();
```

instead of

```
eps = mexGetEps();
```

See Also

`mxGetEps`

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mexGetArray(array_ptr, "caller");  
name = mxGetName(array_ptr);  
m = mxGetM(array_ptr);  
n = mxGetN(array_ptr);  
pr = mxGetPr(array_ptr);  
pi = mxGetPi(array_ptr);
```

instead of

```
mexGetFull(name, m, n, pr, pi);
```

See Also

`mexGetArray`, `mxGetName`, `mxGetPr`, `mxGetPi`

mexGetGlobal (Obsolete)

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mexGetArrayPtr(name, "global");
```

instead of

```
mexGetGlobal(name);
```

See Also

`mexGetArray`, `mxGetName`, `mxGetPr`, `mxGetPi`

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
eps = mxGetEps();
```

instead of

```
eps = mexGetEps();
```

See Also

`mxGetEps`

mexGetMatrix (Obsolete)

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mexGetArray(name, "caller");
```

instead of

```
mexGetMatrix(name);
```

See Also

`mexGetArray`

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mexGetArrayPtr(name, "caller");
```

instead of

```
mexGetMatrixPtr(name);
```

See Also

`mexGetArrayPtr`

mexGetNaN (Obsolete)

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
NaN = mxGetNaN();
```

instead of

```
NaN = mexGetNaN();
```

See Also

`mxGetNaN`

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
answer = mxIsFinite(value);
```

instead of

```
answer = mexIsFinite(value);
```

See Also

`mxIsFinite`

mexIsGlobal

Purpose	True if mxArray has global scope
C Syntax	<pre>#include "matrix.h" bool mexIsGlobal (const mxArray *array_ptr);</pre>
Arguments	array_ptr Pointer to an mxArray.
Returns	true if the mxArray has global scope; otherwise, returns false.
Description	<p>Use <code>mexIsGlobal</code> to determine if the specified mxArray has global scope. By default, mxArrays have local scope, meaning that changes made to the mxArray inside a MEX-file or stand-alone application have no effect on a variable of the same name in another workspace. However, if an mxArray has global scope, then changes made to the mxArray inside a MEX-file or stand-alone application can affect other workspaces.</p> <p>The MATLAB <code>global</code> command gives global scope to a MATLAB variable. For example, to make variable <code>x</code> global, just type</p> <pre>>> global x</pre> <p>The most common use of <code>mexIsGlobal</code> is to determine if an mxArray stored inside a MAT-files is global.</p>
Example	See <code>mexIsGlobal.c</code> in the <code>mex</code> subdirectory of the <code>examples</code> directory.
See Also	<code>mexGetArray</code> , <code>mexGetArrayPtr</code> , <code>mexPutArray</code>

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
answer = mxIsInf(value);
```

instead of

```
answer = mexIsInf(value);
```

See Also

`mxIsInf`

mexIsLocked

Purpose	True if this MEX-file is locked
C Syntax	<pre>#include "mex.h" bool mexIsLocked(void);</pre>
Returns	True if the MEX-file is locked; False if the file is unlocked.
Description	Call <code>mexIsLocked</code> to determine if the MEX-file is locked. By default, MEX-files are unlocked, meaning that users can clear a MEX-file at any time. Calling <code>mexLock</code> locks a MEX-file, which makes it impossible for a user to clear a MEX-file.
See Also	<code>mexLock</code> , <code>mexMakeArrayPersistent</code> , <code>mexMakeMemoryPersistent</code> , <code>mexUnlock</code>

V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
answer = mexIsNaN(value);
```

instead of

```
answer = mexIsNaN(value);
```

See Also

`mexIsInf`

mexLock

Purpose	Lock a MEX-file so that it cannot be cleared from memory
C Syntax	<pre>#include "mex.h" void mexLock(void);</pre>
Description	<p>By default, MEX-files are unlocked, meaning that a user can clear them at any time. Call <code>mexLock</code> to prohibit a MEX-file from being cleared.</p> <p>To unlock a MEX-file, call <code>mexUnlock</code>.</p> <p><code>mexLock</code> increments a lock count. If you call <code>mexLock</code> <code>n</code> times, you must call <code>mexUnlock</code> <code>n</code> times to unlock your MEX-file.</p>
Example	See <code>mexlock.c</code> in the <code>mex</code> subdirectory of the <code>examples</code> directory.
See Also	<code>mexIsLocked</code> , <code>mexMakeArrayPersistent</code> , <code>mexMakeMemoryPersistent</code> , <code>mexUnlock</code>

Purpose	Make an <code>mxArray</code> persist after the MEX-file completes
C Syntax	<pre>#include "mex.h" void mexMakeArrayPersistent(mxArray *array_ptr);</pre>
Arguments	<code>array_ptr</code> Pointer to an <code>mxArray</code> created by an <code>mxCreate</code> routine.
Description	<p>By default, <code>mxArrays</code> allocated by <code>mxCreate</code> routines are not persistent. MATLAB's memory management facility automatically frees nonpersistent <code>mxArrays</code> when the MEX-file finishes.</p> <p>Calling <code>mexMakeArrayPersistent</code> marks an <code>mxArray</code> as persistent. MATLAB does not automatically free persistent <code>mxArrays</code> when the MEX-file finishes. In other words, the <code>mxArray</code> persists through multiple invocations of the MEX-file.</p> <p>If you do tag an <code>mxArray</code> as persistent, make sure that the MEX-file does not accidentally recreate the same array each time the MEX-file gets called. That is, make sure that the <code>mxArray</code> gets created the first time the MEX-file is called. Then, make sure that the <code>mxArray</code> does not get created again on subsequent calls to the MEX-file.</p> <ul style="list-style-type: none">• When a MEX-file is cleared.• When a MEX-file finishes using a persistent <code>mxArray</code>; the MEX-file is responsible for calling <code>mxDestroyArray</code> to deallocate the heap space.
Example	See <code>mexmakearraypersistent.c</code> in the <code>mex</code> subdirectory of the <code>examples</code> directory.
See Also	<code>mexAtExit</code> , <code>mexLock</code> , <code>mexMakeMemoryPersistent</code> , and the <code>mxCreate</code> functions.

mexMakeMemoryPersistent

Purpose	Make a section of dynamic memory allocated by <code>mxCall oc</code> persist after the MEX-file completes
C Syntax	<pre>#include "mex.h" void mexMakeMemoryPersistent(void *ptr);</pre>
Arguments	<code>ptr</code> Pointer to the beginning of any memory parcel allocated by <code>mxCall oc</code> .
Description	By default, memory allocated by <code>mxCall oc</code> is nonpersistent. That is, MATLAB's memory management facility automatically frees such memory when the MEX-file finishes. Calling <code>mexMakeMemoryPersistent</code> tells MATLAB not to free such memory when the MEX-file finishes. In other words, the memory allocated by <code>mxCall oc</code> becomes persistent.
See Also	<code>mexLock</code> , <code>mexMakeArrayPersistent</code> , <code>mxCall oc</code>

Purpose	ANSI C printf-style output routine
C Syntax	<pre>#include "mex.h" int mexPrintf(const char *format, ...);</pre>
Arguments	<pre>format, ...</pre> <p>ANSI C printf-style format string and optional arguments.</p>
Description	<p>This routine prints a string on the screen and in the diary (if the diary is in use). It provides a callback to the standard C printf routine already linked inside MATLAB, and avoids linking the entire stdio library into your MEX-file.</p> <p>In a MEX-file, you must call <code>mexPrintf</code> instead of <code>printf</code>.</p>
Examples	<p>Consider a MEX-file named <code>DispComp</code> that expects two input arguments and determines which of them has the larger first element.</p>

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
    mxArray *array_ptr;
    double    value_in_first_array, value_in_second_array;

    /*Get the value of the first element in both input arrays.*/
    value_in_first_array = mxGetScalar(prhs[0]);
    value_in_second_array = mxGetScalar(prhs[1]);
    if (value_in_first_array > value_in_second_array)
        mexPrintf("%g is greater than %g.\n",
            value_in_first_array, value_in_second_array);
    else
        mexPrintf("%g is not greater than %g.\n",
            value_in_first_array, value_in_second_array);
}
```

In MATLAB, create two vectors:

```
>> a=[53 2 17];
>> b=[65 14 23 99 57];
```

mexPrintf

Pass `v1` and `v2` as arguments to `DispComp`:

```
>> DispComp(v1, v2)
53 is not greater than 65.
```

For an additional example, see `mexprintf.c` in the `mex` subdirectory of the `examples` directory.

See Also

`mexErrMsgTxt`, `mexWarnMsgTxt`

- Purpose** Copy an `mxArray` from your MEX-file into another workspace
- C Syntax**
- ```
#include "mex.h"
int mexPutArray(mxArray *array_ptr, const char *workspace);
```
- Arguments**
- `array_ptr`  
Pointer to an `mxArray`.
- `workspace`  
Specifies the scope of the array that you are copying. Possible values are.
- |          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| "base"   | Copy name to the current MATLAB workspace.                                                                                |
| "caller" | Copy name to the workspace of whatever entity (M-file, another MEX-file, MATLAB workspace) actually called this MEX-file. |
| "global" | Copy name to the list of global variables.                                                                                |
- Returns** 0 on success; 1 on failure. A possible cause of failure is that `array_ptr` is NULL. Another possibility is that `array_ptr` points to an `mxArray` that does not have an associated name. (Call `mxSetName` to associate a name with `array_ptr`.)
- Description**
- Call `mexPutArray` to copy the specified `mxArray` from your MEX-file into another workspace. `mexPutArray` makes the specified array accessible to other entities, such as MATLAB, M-files or other MEX-files.
- It is easy to confuse `array_ptr` with a variable name. You manipulate variable names in the MATLAB workspace; you manipulate `array_ptr`s in a MEX-file. When you call `mexPutArray`, you specify an `array_ptr`; however, the recipient workspace appears to receive a variable name. MATLAB determines the variable name by looking at the name field of the received `mxArray`.
- If a variable of the same name already exists in the specified workspace, `mexPutArray` overwrites the previous contents of the variable with the

contents of the new mxArray. For example, suppose the MATLAB workspace defines variable Peaches as

```
>> Peaches
 1 2 3 4
```

and you call mexPutArray to copy Peaches into the MATLAB workspace:

```
mxSetName(array_ptr, "Peaches")
mexPutArray(array_ptr, "base")
```

Then, the old value of Peaches disappears and is replaced by the value passed in by mexPutArray.

## Examples

Consider a MEX-file named Cre8Fibonacci that creates a vector named Fibonacci containing the first 10 Fibonacci numbers. Cre8Fibonacci calls mexPutArray to place Fibonacci into the MATLAB workspace.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double pr[10];
 mxArray *array_ptr;
 int c;
 int status;
 /* Generate the first 10 Fibonacci numbers. */
 pr[0]=1; pr[1]=1;
 for (c=2; c<10; c++)
 pr[c] = pr[c-1] + pr[c-2];
 /* Create an mxArray named Fibonacci seeded to pr. */
 array_ptr = mxCreateDoubleMatrix(0, 0, pr, NULL);
 mxSetM(array_ptr, 1);
 mxSetN(array_ptr, 10);
 mxSetPr(array_ptr, pr);
 mxSetPi(array_ptr, NULL);
 /* mxSetName(array_ptr, "Fibonacci"); */
 /* Put "Fibonacci" into the MATLAB workspace. */
 status = mexPutArray(array_ptr, "base");
 printf("status = %d\n", status);
}
```

Running `Cre8Fib`s from MATLAB places `Fib`s into the MATLAB workspace, for example:

```
>> Cre8Fib
>> Fib
1 1 2 3 5 8 13 21 34 55
```

For an additional example, see `mexputarray.c` in the `mex` subdirectory of the `examples` directory.

## See Also

`mexGetArray`

## mexPutFull (Obsolete)

---

### V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
array_ptr = mxCreateDoubleMatrix(0, 0, mxREAL/mxCOMPLEX);
mxSetName(array_ptr, name);
mexPutArray(array_ptr, "caller");
```

instead of

```
mexPutFull(name, m, n, pr, pi)
```

### See Also

`mxSetM`, `mxSetN`, `mxSetPr`, `mxSetPi`, `mxSetName`, `mexPutArray`

### V4 Compatible

This function is obsolete; it should not appear in a MATLAB 5 program. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files you should call

```
mexPutArray(array_ptr, "caller");
```

instead of

```
mexPutMatrix(matrix_ptr);
```

### See Also

`mexPutArray`

# mexSet

---

|                    |                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the value of the specified Handle Graphics property                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "mex.h" int mexSet(double handle, const char *property,            mxArray *value);</pre>                                                                                                                                      |
| <b>Arguments</b>   | <p><code>handle</code><br/>Handle to a particular graphics object.</p> <p><code>property</code><br/>A Handle Graphics property.</p> <p><code>value</code><br/>The new value to assign to the property.</p>                                   |
| <b>Returns</b>     | 0 on success; 1 on failure. Possible causes of failure include                                                                                                                                                                               |
|                    | <ul style="list-style-type: none"><li>• Specifying a nonexistent property.</li><li>• Specifying an illegal value for that property. For example, specifying a string value for a numerical property.</li></ul>                               |
| <b>Description</b> | Call <code>mexSet</code> to set the value of the property of a certain graphics object. <code>mexSet</code> is the API equivalent of MATLAB's <code>set</code> function. To get the value of a graphics property, call <code>mexGet</code> . |
| <b>Example</b>     | See the example on the <code>mexGet</code> reference page.                                                                                                                                                                                   |
| <b>See Also</b>    | <code>mexGet</code>                                                                                                                                                                                                                          |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Control response of <code>mexCallMATLAB</code> to errors                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>C Syntax</b>    | <pre>#include "mex.h" void mexSetTrapFlag(int trap_flag);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>   | <p><code>trap_flag</code><br/>Control flag. Currently, the only legal values are</p> <ul style="list-style-type: none"><li>0     On error, control returns to the MATLAB prompt.</li><li>1     On error, control returns to your MEX-file.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>Call <code>mexSetTrapFlag</code> to control MATLAB's response to errors in <code>mexCallMATLAB</code>.</p> <p>If you do not call <code>mexSetTrapFlag</code>, then whenever MATLAB detects an error in a call to <code>mexCallMATLAB</code>, MATLAB automatically terminates the MEX-file and returns control to the MATLAB prompt. Calling <code>mexSetTrapFlag</code> with <code>trap_flag</code> set to 0 is equivalent to not calling <code>mexSetTrapFlag</code> at all.</p> <p>If you call <code>mexSetTrapFlag</code> and set the <code>trap_flag</code> to 1, then whenever MATLAB detects an error in a call to <code>mexCallMATLAB</code>, MATLAB does not automatically terminate the MEX-file. Rather, MATLAB returns control to the line in the MEX-file immediately following the call to <code>mexCallMATLAB</code>. The MEX-file is then responsible for taking an appropriate response to the error.</p> |
| <b>Example</b>     | See <code>mexsettrapflag.c</code> in the <code>mex</code> subdirectory of the <code>examples</code> directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>See Also</b>    | <code>mexAtExit</code> , <code>mexErrMsgTxt</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

# mexUnlock

---

## Purpose

Unlock this MEX-file so that it can be cleared from memory

## C Syntax

```
#include "mex.h"
void mexUnlock(void);
```

## Description

By default, MEX-files are unlocked, meaning that a user can clear them at any time. Calling `mexLock` locks a MEX-file so that it cannot be cleared. Calling `mexUnlock` removes the lock so that a MEX-file can be cleared.

`mexLock` decrements a lock count. If you called `mexLock` `n` times, you must call `mexUnlock` `n` times to unlock your MEX-file.

## See Also

`mexIsLocked`, `mexLock`, `mexMakeArrayPersistent`,  
`mexMakeMemoryPersistent`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Issue warning message                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "mex.h" void mexWarnMsgTxt(const char *warning_msg);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>   | <pre>warning_msg</pre> String containing the warning message to be displayed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b> | <p>mexWarnMsgTxt causes MATLAB to display the contents of error_msg.</p> <p>Unlike mexErrMsgTxt, mexWarnMsgTxt does not cause the MEX-file to terminate.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Examples</b>    | <p>Consider a MEX-file that expects two input arguments. If the user enters no arguments, the MEX-file simply terminates. If the user enters one or more arguments, a warning is given but the MEX-file proceeds.</p> <pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray     *prhs[]) {     if (nrhs == 0)         mexErrMsgTxt("You must pass 2 input args.");     else if (nrhs == 1) {         mexWarnMsgTxt("Expected 2 input args; you passed 1.");         mexWarnMsgTxt("Assuming that 'noplot' is the second arg.");          ...     }     else if (nrhs == 2) {          ...     }     else {         mexWarnMsgTxt("Ignoring all arguments past the second.");          ...     } }</pre> |

## mexWarnMsgTxt

---

For an additional example, see `mexwarnmsgtxt.c` in the `mex` subdirectory of the `examples` directory.

### See Also

`mexErrMsgTxt`



# mxArrayToString

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Convert arrays to strings                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" char *mxArrayToString(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a string mxArray; that is, a pointer to an mxArray having the <code>mxCHAR_CLASS</code> class.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Returns</b>     | A C-style string. Returns NULL on out of memory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | <p>Call <code>mxArrayToString</code> to copy the character data of a string mxArray into a C-style string. The copied C-style string starts at <code>buf</code> and contains no more than <code>buflen - 1</code> characters. The C-style string is always terminated with a NULL character.</p> <p>If the string array contains several rows, they are copied, one column at a time, into one long string array. This function is similar to <code>mxGetString</code>, except that:</p> <ul style="list-style-type: none"><li>• it does not require the length of the string as an input</li><li>• it supports multibyte character sets</li></ul> <p><code>mxArrayToString</code> does not free the dynamic memory that the char pointer points to. Consequently, you should typically free the string (using <code>mxFree</code>) immediately after you have finished using it.</p> |

## Example

Use `mxArrayToString` to convert the data from a string array into a C string named `buf`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 char *buf;

 /* Copy the string data from prhs[0] and place it into buf. */
 buf= mxGetString(prhs[0]);
 if (buf != NULL)
 mexPrintf("The converted string is %s.\n", buf);
 else
 mexErrMsgTxt("Could not convert string data.");

 /* Manipulate buf as you would manipulate any C string. */
 ...
 /* When finished, free buf memory. */
 mxFree(buf);
}
```

For an additional example, see `mxarraytostring.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateCharArray`, `mxCreateCharMatrixFromStrings`, `mxCreateString`,  
`mxGetString`

# mxAssert

---

**Purpose** Check assertion value for debugging purposes

**C Syntax**

```
#include "matrix.h"
void mxAssert(int expr, char *error_message);
```

**Arguments**

`expr`  
Value of assertion.

`error_message`  
Description of why assertion failed.

**Description**

Similar to the ANSI C `assert()` macro, `mxAssert` checks the value of an assertion, and continues execution only if the assertion holds. If `expr` evaluates to true, `mxAssert` does nothing. If `expr` is false, `mxAssert` prints an error to the MATLAB Command Window consisting of the failed assertion's expression, the file name and line number where the failed assertion occurred, and the `error_message` string. The `error_message` string allows you to specify a better description of why the assertion failed. Use an empty string if you don't want a description to follow the failed assertion message.

After a failed assertion, control returns to the MATLAB command line.

Note that the MEX script turns off these assertions when building optimized MEX-functions, so you should use this for debugging purposes only.

Assertions are a way of maintaining internal consistency of logic. Use them to keep yourself from misusing your own code and to prevent logical errors from propagating before they are caught; you do not use assertions to prevent users of your code from misusing it.

Assertions can be taken out of your code by the C preprocessor. So, you can use these checks during development and then remove them when the code works properly, giving you the ability to use them for troubleshooting during development without slowing down the final product.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Check assertion value for debugging purposes; doesn't print assertion's text                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxAssertS(int expr, char *error_message);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>   | <p><code>expr</code><br/>Value of assertion.</p> <p><code>error_message</code><br/>Description of why assertion failed.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | <p>Similar to <code>mxAssert</code>, except <code>mxAssertS</code> does not print the text of the failed assertion. <code>mxAssertS</code> checks the value of an assertion, and continues execution only if the assertion holds. If <code>expr</code> evaluates to true, <code>mxAssertS</code> does nothing. If <code>expr</code> is false, <code>mxAssertS</code> prints an error to the MATLAB Command Window consisting of the file name and line number where the assertion failed and the <code>error_message</code> string. The <code>error_message</code> string allows you to specify a better description of why the assertion failed. Use an empty string if you don't want a description to follow the failed assertion message.</p> <p>After a failed assertion, control returns to the MATLAB command line.</p> <p>Note that the <code>mex</code> script turns off these assertions when building optimized MEX-functions, so you should use this for debugging purposes only.</p> |

# mxCalcSingleSubscript

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Return the offset (index) from the first element to the desired element                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>C Syntax</b>    | <pre>#include &lt;matrix.h&gt; int mxCalcSingleSubscript(const mxArray *array_ptr, int nsubs,     int *subs);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>   | <p><b>array_ptr</b><br/>Pointer to an mxArray.</p> <p><b>nsubs</b><br/>The number of elements in the subs array. Typically, you set nsubs equal to the number of dimensions in the mxArray that array_ptr points to.</p> <p><b>subs</b><br/>An array of integers. Each value in the array should specify that dimension's subscript. The value in subs[0] specifies the row subscript, and the value in subs[1] specifies the column subscript. Note that mxCalcSingleSubscript views 0 as the first element of an mxArray, but MATLAB sees 1 as the first element of an mxArray. For example, in MATLAB, (1, 1) denotes the starting element of a two-dimensional mxArray; however, to express the starting element of a two-dimensional mxArray in subs, you must set subs[0] to 0 and subs[1] to 0.</p> |
| <b>Returns</b>     | <p>The number of elements between the start of the mxArray and the specified subscript. This returned number is called an “index”; many mx routines (for example, mxGetField) require an index as an argument.</p> <p>If subs describes the starting element of an mxArray, mxCalcSingleSubscript returns 0. If subs describes the final element of an mxArray, then mxCalcSingleSubscript returns N-1 (where N is the total number of elements).</p>                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | Call mxCalcSingleSubscript to determine how many elements there are between the beginning of the mxArray and a given element of that mxArray. For example, given a subscript like (5, 7), mxCalcSingleSubscript returns the distance from the (0, 0) element of the array to the (5, 7) element. Remember that the mxArray data type internally represents all data elements in a one-dimensional array no matter how many dimensions the MATLAB mxArray appears to have.                                                                                                                                                                                                                                                                                                                                  |

MATLAB uses a column-major numbering scheme to represent data elements internally. That means that MATLAB internally stores data elements from the first column first, then data elements from the second column second, and so on through the last column. For example, suppose you create a 4-by-2 variable. It is helpful to visualize the data as shown below:

|   |   |
|---|---|
| A | E |
| B | F |
| C | G |
| D | H |

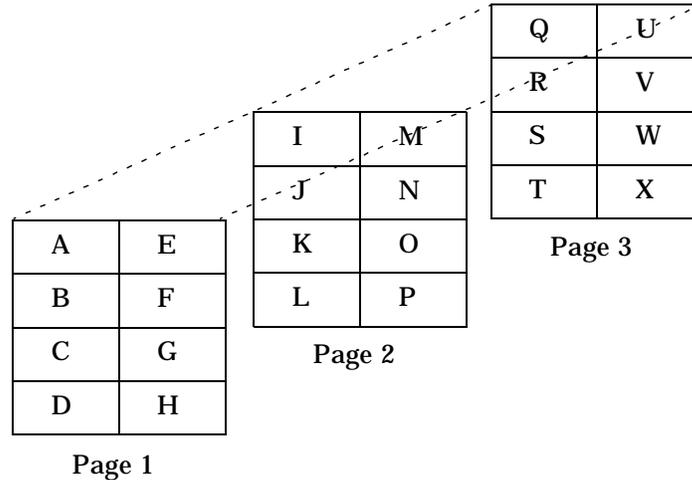
Although in fact, MATLAB internally represents the data as the following:

|            |            |            |            |            |            |            |            |
|------------|------------|------------|------------|------------|------------|------------|------------|
| A          | B          | C          | D          | E          | F          | G          | H          |
| Index<br>0 | Index<br>1 | Index<br>2 | Index<br>3 | Index<br>4 | Index<br>5 | Index<br>6 | Index<br>7 |

Thus, the first column has indices 0 through 3 and the second column has indices 4 through 7.

If an `mxArray` is N-dimensional, then MATLAB represents the data in N-major order. For example, consider a three-dimensional array having dimensions 4-by-2-by-3. Although you can visualize the data as shown on the following page:

# mxCalcSingleSubscript



MATLAB internally represents the data for this three-dimensional array in the order shown below:

| A | B | C | D | E | F | G | H | I | J | K  | L  | M  | N  | O  | P  | Q  | R  | S  | T  | U  | V  | W  | X  |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

Thus, the indices of page 1 are lower than the indices of page 2. Within each page, the indices of the first column are lower than the indices of the second column. Within each column, the indices of the first row are lower than the indices of the second row.

`mxCalcSingleSubscript` provides an efficient way to get an individual offset. However, most applications do not need to get just a single offset. Rather, most applications have to walk through each element of data in an array. In such cases, avoid using `mxCalcSingleSubscript`. To walk through all elements of the array, it is far more efficient to find the array's starting address and then use pointer auto-incrementing to access successive elements. For example, to find the starting address of a numerical array, call `mxGetPr` or `mxGetPi`.

## Examples

Given a two-dimensional input array in `prhs[0]`, find the value stored at element (2, 3):

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int nsubs = 2;
 int subs[] = {2, 3};
 double *start_of_real_data;
 int index;
 double value;

 /* Find the index of location (2, 3).
 Note that (2, 3) corresponds to (3, 4) in MATLAB. */
 index = mxCalcSingleSubscript(prhs[0], nsubs, subs);

 /* Get the start of the real data in the array. */
 if (mxIsDouble(prhs[0])) {
 start_of_real_data = (double *)mxGetPr(prhs[0]);

 /* Get the address for location (2, 3), then dereference it,
 and print it. */

 value = *(start_of_real_data + index);
 mexPrintf("The value at MATLAB's (%d,%d) is %g\n",
 subs[0]+1, subs[1]+1, value);
 }
 else
 mexErrMsgTxt("Input array must be a double.");
}
```

Given a three-dimensional array in `prhs[0]`, setting

```
int nsubs = 3;
int subs[] = {5, 12, 9};
```

returns the offset of (5, 12, 9) from the (0, 0, 0) element. Given a three-dimensional array in `prhs[0]`, setting

```
int nsubs = 2;
int subs[] = {5, 12, 9};
```

## mxCalcSingleSubscript

---

returns the offset of (5, 12, 0) from the (0, 0, 0) element. For an additional example, see `mxcalcSingleSubscript.c` in the `mx` subdirectory of the `examples` directory.

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Allocate dynamic memory using MATLAB's memory manager                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" #include &lt;stdlib.h&gt; void *mxMalloc(size_t n, size_t size);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <p><b>n</b><br/>Number of elements to allocate. This must be a nonnegative number.</p> <p><b>size</b><br/>Number of bytes per element. (The C <code>sizeof</code> operator calculates the number of bytes per element.)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Returns</b>     | <p>A pointer to the start of the allocated dynamic memory, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxMalloc</code> returns NULL. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt.</p> <p><code>mxMalloc</code> is unsuccessful when there is insufficient free heap space.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>MATLAB applications should always call <code>mxMalloc</code> rather than <code>calloc</code> to allocate memory. Note that <code>mxMalloc</code> works differently in MEX-files than in stand-alone MATLAB applications.</p> <p>In MEX-files, <code>mxMalloc</code> automatically</p> <ul style="list-style-type: none"><li>• Allocates enough contiguous heap space to hold <code>n</code> elements.</li><li>• Initializes all <code>n</code> elements to 0.</li><li>• Registers the returned heap space with the MATLAB memory management facility.</li></ul> <p>The MATLAB memory management facility maintains a list of all memory allocated by <code>mxMalloc</code>. The MATLAB memory management facility automatically frees (deallocates) all of a MEX-file's parcels when control returns to the MATLAB prompt.</p> <p>In stand-alone MATLAB applications, <code>mxMalloc</code> defaults to calling the ANSI C <code>calloc</code> function. If this default behavior is unacceptable, you can write your own memory allocation routine, and then register this routine with</p> |

`mxSetAllocFns`. Then, whenever `mxCall oc` is called, `mxCall oc` calls your memory allocation routine instead of `call oc`.

By default, in a MEX-file, `mxCall oc` generates nonpersistent `mxCall oc` data. In other words, the memory management facility automatically deallocates the memory as soon as the MEX-file ends. If you want the memory to persist after the MEX-file completes, call `mexMakeMemoryPersistent` after calling `mxCall oc`. If you write a MEX-file with persistent memory, be sure to register a `mexAtExit` function to free allocated memory in the event your MEX-file is cleared.

When you finish using the memory allocated by `mxCall oc`, call `mxFree`. `mxFree` deallocates the memory.

## Examples

This example uses `mxCall oc` to allocate enough heap space to hold a string.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 char *buf;
 int buflen;
 int status;

 if (mxIsString(prhs[0])) {
 /* Find out how long the input string array is. */
 buflen = mxGetN(prhs[0])+1;
 /* Allocate enough memory to hold the converted string. */
 buf = mxCalloc(buflen, sizeof(char));
 /* Copy the string data into buf. */
 status = mxGetString(prhs[0], buf, buflen);
 /* Manipulate the string. */
 ...

 /* When finished using the string, deallocate it. */
 mxFree(buf);
 }
 else
 mexErrMsgTxt("Input argument must be a string.");
}
```

For an additional example, see `mxcalloc.c` in the `mx` subdirectory of the `examples` directory.

**See Also**

`mxFree`, `mxDestroyArray`, `mxMakeArrayPersistent`,  
`mxMakeMemoryPersistent`, `mxMalloc`, `mxSetAllocFcns`

# mxChar

---

|                     |                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>      | Data type that string <code>mxArrays</code> use to store their data elements                                                                                                                     |
| <b>C Definition</b> | <code>typedef Uint16 mxChar;</code>                                                                                                                                                              |
| <b>Description</b>  | All string <code>mxArrays</code> store their data elements as <code>mxChar</code> rather than as <code>char</code> . The MATLAB API defines an <code>mxChar</code> as a 16-bit unsigned integer. |
| <b>See Also</b>     | <code>mxCreateCharArray</code>                                                                                                                                                                   |

**Purpose** Enumerated data type that identifies an `mxArray`'s class (category)

**C Definition**

```
typedef enum {
 mxCELL_CLASS = 1,
 mxSTRUCT_CLASS,
 mxOBJECT_CLASS,
 mxCHAR_CLASS,
 mxSPARSE_CLASS,
 mxDOUBLE_CLASS,
 mxSINGLE_CLASS,
 mxINT8_CLASS,
 mxUINT8_CLASS,
 mxINT16_CLASS,
 mxUINT16_CLASS,
 mxINT32_CLASS,
 mxUINT32_CLASS,
 mxINT64_CLASS, /* place holder - future enhancements */
 mxUINT64_CLASS, /* place holder - future enhancements */
 mxUNKNOWN_CLASS = -1
} mxClassID;
```

**Constants**

`mxCELL_CLASS`  
Identifies a cell `mxArray`.

`mxSTRUCT_CLASS`  
Identifies a structure `mxArray`.

`mxOBJECT_CLASS`  
Identifies a user-defined (nonstandard) `mxArray`.

`mxCHAR_CLASS`  
Identifies a string `mxArray`; that is an `mxArray` whose data is represented as `mxCHAR`'s.

`mxSPARSE_CLASS`  
Identifies a sparse `mxArray`; that is, an `mxArray` that only stores its nonzero elements.

# mxClassID

---

`mxDOUBLE_CLASS`

Identifies a numeric `mxArray` whose data is stored as double-precision, floating-point numbers.

`mxSINGLE_CLASS`

Identifies a numeric `mxArray` whose data is stored as single-precision, floating-point numbers.

`mxINT8_CLASS`

Identifies a numeric `mxArray` whose data is stored as signed 8-bit integers.

`mxUINT8_CLASS`

Identifies a numeric `mxArray` whose data is stored as unsigned 8-bit integers.

`mxINT16_CLASS`

Identifies a numeric `mxArray` whose data is stored as signed 16-bit integers.

`mxUINT16_CLASS`

Identifies a numeric `mxArray` whose data is stored as unsigned 16-bit integers.

`mxINT32_CLASS`

Identifies a numeric `mxArray` whose data is stored as signed 32-bit integers.

`mxUINT32_CLASS`

Identifies a numeric `mxArray` whose data is stored as unsigned 32-bit integers.

`mxINT64_CLASS`

Reserved for possible future use.

`mxUINT64_CLASS`

Reserved for possible future use.

`mxUNKNOWN_CLASS = -1`

The class cannot be determined. You cannot specify this category for an `mxArray`; however, `mxGetClassID` can return this value if it cannot identify the class.

## Description

Various `mx` calls require or return an `mxClassID` argument. `mxClassID` identifies the way in which the `mxArray` represents its data elements.

## See Also

`mxCreateNumericArray`

|                    |                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Clear the logical flag                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxClearLogical (mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>   | <p>array_ptr<br/>Pointer to an mxArray having a numeric class.</p>                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | <p>Use mxClearLogical to turn off the mxArray's logical flag. This flag tells MATLAB that the mxArray's data is to be treated as numeric data rather than as Boolean data. If the logical flag is on, then MATLAB treats a 0 value as meaning false and a nonzero value as meaning true.</p> <p>Call mxSetLogical to turn on the mxArray's logical flag.</p> |
| <b>Example</b>     | See mxclearlogical.c in the mx subdirectory of the examples directory.                                                                                                                                                                                                                                                                                       |
| <b>See Also</b>    | mxIsLogical, mxSetLogical                                                                                                                                                                                                                                                                                                                                    |

# mxComplexity

---

|                     |                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>      | Flag that specifies whether an <code>mxArray</code> has imaginary components                                                                                                                    |
| <b>C Definition</b> | <pre>typedef enum mxComplexity {mxREAL=0, mxCOMPLEX};</pre>                                                                                                                                     |
| <b>Constants</b>    | <p><code>mxREAL</code><br/>Identifies an <code>mxArray</code> with no imaginary components.</p> <p><code>mxCOMPLEX</code><br/>Identifies an <code>mxArray</code> with imaginary components.</p> |
| <b>Description</b>  | Various <code>mx</code> calls require an <code>mxComplexity</code> argument. You can set an <code>mxComplex</code> argument to either <code>mxREAL</code> or <code>mxCOMPLEX</code> .           |
| <b>See Also</b>     | <code>mxCreateNumericArray</code> , <code>mxCreateDoubleMatrix</code> , <code>mxCreateSparse</code>                                                                                             |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated N-dimensional cell <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateCellArray(int ndim, const int *dims);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>   | <p><code>ndim</code><br/>The desired number of dimensions in the created cell. For example, to create a three-dimensional cell <code>mxArray</code>, set <code>ndim</code> to 3.</p> <p><code>dims</code><br/>The dimensions array. Each element in the dimensions array contains the size of the <code>mxArray</code> in that dimension. For example, setting <code>dims[0]</code> to 5 and <code>dims[1]</code> to 7 establishes a 5-by-7 <code>mxArray</code>. In most cases, there should be <code>ndim</code> elements in the <code>dims</code> array.</p> |
| <b>Returns</b>     | <p>A pointer to the created cell <code>mxArray</code>, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCreateCellArray</code> returns NULL. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. Causes of failure include</p> <ul style="list-style-type: none"><li>• Insufficient free heap space.</li><li>• Specifying a value for <code>ndim</code> that is greater than the number of values in the <code>dims</code> array.</li></ul>                                      |
| <b>Description</b> | <p>Use <code>mxCellArray</code> to create a cell <code>mxArray</code> whose size is defined by <code>ndim</code> and <code>dims</code>. For example, to establish a three-dimensional cell <code>mxArray</code> having dimensions 4-by-8-by-7, set</p> <pre>ndim = 3; dims[0] = 4; dims[1] = 8; dims[2] = 7;</pre> <p>The created cell <code>mxArray</code> is unpopulated; that is, <code>mxCreateCellArray</code> initializes each cell to NULL. To put data into a cell, call <code>mxSetCell</code>.</p>                                                    |

# mxCreateCellArray

---

## Examples

Create a two-dimensional 2-by-2 cell mxArray named amoeba, then populate two of its 4 cells:

```
int ndim=2, dims[]={2, 2};
int index, nsubs=2, subs[2];
double real_pr[] = {5.23, 7.45, 8.17, 9.79};
double *pr;
mxArray *cell_array_ptr, *string_array_ptr, *vector_ptr;

/* Create a 2-by-2 cell array. */
cell_array_ptr = mxCreateCellArray(ndim, dims);
mxSetName(cell_array_ptr, "amoeba");

/* Create a string array. */
string_array_ptr = mxCreateString("Hello friends.");
/* Place the string array into cell element (1,1). */
subs[0]=0; subs[1]=0;
index = mxCalcSingleSubscript(cell_array_ptr, nsubs, subs);
mxSetCell(cell_array_ptr, index, string_array_ptr);

/* Create a 1-by-4 vector array of doubles. */
vector_ptr = mxCreateDoubleMatrix(1, 4, mxREAL);
pr = mxGetPr(vector_ptr);
memcpy((void *)pr, (const void *)real_pr, 4*sizeof(double));
/* Place the vector array into cell element (2,2). */
subs[0]=1; subs[1]=1;
index = mxCalcSingleSubscript(cell_array_ptr, nsubs, subs);
mxSetCell(cell_array_ptr, index, vector_ptr);
```

The code leaves cell array elements (1, 2) and (2, 1) unpopulated.

For an additional example, see `mxcreatcelarray.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateCellMatrix`, `mxGetCell`, `mxSetCell`, `mxIsCell`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated 2-dimensional cell mxArray                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateCellMatrix(int m, int n);</pre>                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <p><b>m</b><br/>The desired number of rows.</p> <p><b>n</b><br/>The desired number of columns.</p>                                                                                                                                                                                                                                                                                                                                                        |
| <b>Returns</b>     | A pointer to the created cell mxArray, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, mxCreateCellMatrix returns NULL. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. Insufficient free heap space is the only reason for mxCreateCellMatrix to be unsuccessful.                                                                                                            |
| <b>Description</b> | <p>Use mxCreateCellMatrix to create an m-by-n two-dimensional cell mxArray. The created cell mxArray is empty; that is, mxCreateCellMatrix initializes each cell to NULL. To put data into cells, call mxSetCell.</p> <p>mxCreateCellMatrix is identical to mxCreateCellArray except that mxCreateCellMatrix can create two-dimensional mxArrays only, but mxCreateCellArray can create mxArrays having any number of dimensions greater than 1.</p>      |
| <b>Examples</b>    | <p>Create an unpopulated 2-by-2 cell mxArray:</p> <pre>int      rows=2, cols=2; mxArray *cell_array_ptr;  /* Create a 2-by-2 cell mxArray named Paramecium. */ cell_array_ptr = mxCreateCellMatrix(rows, cols); mxSetName(cell_array_ptr, "Paramecium");  ...</pre> <p>For an additional example, see mxcreatecellmatrix.c in the mx subdirectory of the examples directory. For an example of how to populate a cell mxArray, see mxCreateCellArray.</p> |

# mxCreateCellMatrix

---

## See Also

`mxCreateCellArray`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated N-dimensional string <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateCharArray(int ndim, const int *dims);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>   | <p><code>ndim</code><br/>The desired number of dimensions in the string <code>mxArray</code>. You must specify a positive number. If you specify 0, 1, or 2, <code>mxCreateCharArray</code> creates a two-dimensional <code>mxArray</code>.</p> <p><code>dims</code><br/>The dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, setting <code>dims[0]</code> to 5 and <code>dims[1]</code> to 7 establishes a 5-by-7 <code>mxArray</code>. The <code>dims</code> array must have at least <code>ndim</code> elements.</p> |
| <b>Returns</b>     | A pointer to the created string <code>mxArray</code> , if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCreateCharArray</code> returns NULL. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. Insufficient free heap space is the only reason for <code>mxCreateCharArray</code> to be unsuccessful.                                                                                                                                                                                                       |
| <b>Description</b> | Call <code>mxCreateCharArray</code> to create an unpopulated N-dimensional string <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

# mxCreateCharArray

---

## Examples

Call `mxCreateCharArray` to create a 3-by-80-by-2 string `mxArray`. Then, populate the string `mxArray` based on the values in `Shake` and `Twain`.

```
#define ROWS 3
#define COLS 80
#define PAGES 2
int ndim = 3, dims[]={ROWS, COLS, PAGES};
mxArray *array_ptr;
static char *Shake[ROWS] = {"Hamlet",
 "The Tempest",
 "The Twelfth Night"};
static char *Twain[ROWS] = {
 "A Connecticut Yankee in King Arthur's Court",
 "The Adventures of Huckleberry Finn",
 "Tom Sawyer"};
mxChar *pr, *original_pr;
char *ptr_to_seed_data;
int c;

/* Create a 3-Dimensional character mxArray. */
array_ptr = mxCreateCharArray(ndim, dims);
if (array_ptr == NULL)
 mexErrMsgTxt("Could not create Character mxArray.");

/* Copy Shake into the mxArray one character at a time. */
original_pr = (mxChar *)mxGetPr(array_ptr);
for (c=0; c<ROWS; c++) {
 pr = original_pr + c;
 ptr_to_seed_data = Shake[c];
 while (*ptr_to_seed_data) {
 *pr = (mxChar)*ptr_to_seed_data;
 pr += ROWS;
 ptr_to_seed_data++;
 }
}

/* Copy Twain into the mxArray one character at a time. */
original_pr = (mxChar *)mxGetPr(array_ptr);
original_pr += (ROWS * COLS); /* move forward one page. */
for (c=0; c<ROWS; c++) {
```

```
pr = original_pr + c;
ptr_to_seed_data = Twain[c];
while (*ptr_to_seed_data) {
 *pr = (mxChar)*ptr_to_seed_data;
 pr += ROWS;
 ptr_to_seed_data++;
}
}
```

For an additional example, see `mxcreatechararray.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxCreateCharMatrixFromStrings`, `mxCreateString`

# mxCreateCharMatrixFromStrings

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create a populated 2-dimensional string <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateCharMatrixFromStrings(int m, const char **str);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>   | <p><code>m</code><br/>The desired number of rows in the created string <code>mxArray</code>. The value you specify for <code>m</code> should equal the number of strings in <code>str</code>.</p> <p><code>str</code><br/>A pointer to a list of strings. The <code>str</code> array must contain at least <code>m</code> strings.</p>                                                                                                                                                                                       |
| <b>Returns</b>     | A pointer to the created string <code>mxArray</code> , if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCreateCharMatrixFromStrings</code> returns <code>NULL</code> . If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. Insufficient free heap space is the primary reason for <code>mxCreateCharArray</code> to be unsuccessful. Another possible reason for failure is that <code>str</code> contains fewer than <code>m</code> strings. |
| <b>Description</b> | <p>Use <code>mxCreateCharMatrixFromStrings</code> to create a two-dimensional string <code>mxArray</code>, where each row is initialized to a string from <code>str</code>. The created <code>mxArray</code> has dimensions <code>m-by-max</code>, where <code>max</code> is the length of the longest string in <code>str</code>.</p> <p>Note that string <code>mxArrays</code> represent their data elements as <code>mxChar</code> rather than as <code>char</code>.</p>                                                  |

## Examples

Create a 3-by-22 string mxArray initialized to all the three substrings of Shake:

```
#include "mex.h"
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
/* BEGIN */
#define ROWS 3
static const char *Shake[ROWS] = {"To be or not to be.",
"Aye. There's the rub.",
"Out, out, damn spot."};

mxArray *array_ptr;
/* END */
int rows, columns;
/* BEGIN */
/* Create a 2-Dimensional string mxArray initialized to Shake. */
array_ptr = mxCreateCharMatrixFromStrings(ROWS, Shake);
/*END*/
mxSetName(array_ptr, "s");

rows = mxGetM(array_ptr);
columns = mxGetN(array_ptr);
mexPrintf("Rows = %d; Columns = %d\n", rows, columns);

/* Place the string array in the MATLAB workspace, then invoke
a MATLAB string manipulation function on it. */
mxPutArray(array_ptr, "caller");
mexEvalString("s2 = upper(s)");

/*BEGIN */

/* PONGO */

/* When finished with the string array, free its memory. */
mxDestroyArray(array_ptr);
/* END */
}
```

# mxCreateCharMatrixFromStrings

---

For an additional example, see `mxcreatecharmatrix.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateCharArray`, `mxCreateString`, `mxGetString`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated 2-dimensional, double-precision, floating-point mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateDoubleMatrix(int m, int n,                                mxComplexity ComplexFlag);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>   | <p><b>m</b><br/>The desired number of rows.</p> <p><b>n</b><br/>The desired number of columns.</p> <p><b>ComplexFlag</b><br/>Specify either <code>mxREAL</code> or <code>mxCOMPLEX</code>. If the data you plan to put into the mxArray has no imaginary components, specify <code>mxREAL</code>. If the data has some imaginary components, specify <code>mxCOMPLEX</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Returns</b>     | A pointer to the created mxArray, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCreateDoubleMatrix</code> returns <code>NULL</code> . If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. <code>mxCreateDoubleMatrix</code> is unsuccessful when there is not enough free heap space to create the mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | <p>Use <code>mxCreateDoubleMatrix</code> to create an m-by-n mxArray. <code>mxCreateDoubleMatrix</code> initializes each element in the pr array to 0. If you set <code>ComplexFlag</code> to <code>mxCOMPLEX</code>, <code>mxCreateDoubleMatrix</code> also initializes each element in the pi array to 0.</p> <p>If you set <code>ComplexFlag</code> to <code>mxREAL</code>, <code>mxCreateDoubleMatrix</code> allocates enough memory to hold m-by-n real elements. If you set <code>ComplexFlag</code> to <code>mxCOMPLEX</code>, <code>mxCreateDoubleMatrix</code> allocates enough memory to hold m-by-n real elements and m-by-n imaginary elements.</p> <p>Call <code>mxDestroyArray</code> when you finish using the mxArray. <code>mxDestroyArray</code> deallocates the mxArray and its associated real and complex elements.</p> |

# mxCreateDoubleMatrix

---

## Examples

Create a 2-by-4 double-precision, floating-point mxArray, then populate it:

```
int rows=2, cols=4;
double pr_data[] = {5.2, 7.9, 1.3, 4.2, 6.7, 5.9, 1.9, 9.4};
double pi_data[] = {3.4, 6.5, 2.2, 9.1, 8.3, 4.7, 2.5, 7.5};
double *start_of_pr, *start_of_pi;
mxArray *array_ptr;

/* Create a 2-by-4 real double matrix named "B". */
array_ptr = mxCreateDoubleMatrix(rows, cols, mxCOMPLEX);
mxSetName(array_ptr, "B");

/* Populate the real part of the created array. */
start_of_pr = (double *)mxGetPr(array_ptr);
memcpy(start_of_pr, pr_data, rows * cols * sizeof(double));

/* Populate the imaginary part of the created array. */
start_of_pi = (double *)mxGetPi(array_ptr);
memcpy(start_of_pi, pi_data, rows * cols * sizeof(double));
```

For an additional example, see `mxcreatedoublematrix.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateNumericArray`, `mxComplexity`

**V4 Compatible** This function is obsolete; MATLAB 5 does not support it. To use this function in existing code, use the `-V4` option of the `mex` script.

Call `mxCreateDoubleMatrix` instead of `mxCreateFull`.

**See Also** `mxCreateDoubleMatrix`

# mxCreateNumericArray

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated N-dimensional numeric <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateNumericArray(int ndim, const int *dims,                                mxClassID class, mxComplexity Compl exFl ag);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>   | <p><code>ndim</code><br/>Number of dimensions. If you specify a value for <code>ndims</code> that is less than 2, <code>mxCreateNumericArray</code> automatically sets the number of dimensions to 2.</p> <p><code>dims</code><br/>The dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, setting <code>dims[0]</code> to 5 and <code>dims[1]</code> to 7 establishes a 5-by-7 <code>mxArray</code>. In most cases, there should be <code>ndim</code> elements in the <code>dims</code> array.</p> <p><code>class</code><br/>The way in which the numerical data is to be represented in memory. For example, specifying <code>mxINT16_CLASS</code> causes each piece of numerical data in the <code>mxArray</code> to be represented as a 16-bit signed integer. You can specify any class except for <code>mxNUMERIC_CLASS</code>, <code>mxSTRUCT_CLASS</code>, <code>mxCELL_CLASS</code>, or <code>mxOBJECT_CLASS</code>.</p> <p><code>Compl exFl ag</code><br/>Specify either <code>mxREAL</code> or <code>mxCOMPLEX</code>. If the data you plan to put into the <code>mxArray</code> has no imaginary components, specify <code>mxREAL</code>. If the data will have some imaginary components, specify <code>mxCOMPLEX</code>.</p> |
| <b>Returns</b>     | A pointer to the created <code>mxArray</code> , if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCreateNumericArray</code> returns <code>NULL</code> . If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt. <code>mxCreateNumericArray</code> is unsuccessful when there is not enough free heap space to create the <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | Call <code>mxCreateNumericArray</code> to create an N-dimensional <code>mxArray</code> in which all data elements have the numeric data type specified by <code>class</code> . After creating the <code>mxArray</code> , <code>mxCreateNumericArray</code> initializes all its real data elements to 0. If <code>Compl exFl ag</code> equals <code>mxCOMPLEX</code> , <code>mxCreateNumericArray</code> also initializes all                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

its imaginary data elements to 0. `mxCreateNumericArray` differs from `mxCreateDoubleMatrix` in two important respects

- All data elements in `mxCreateDoubleMatrix` are double-precision, floating-point numbers. The data elements in `mxCreateNumericArray` could be any numerical type, including different integer precisions.
- `mxCreateDoubleMatrix` can create two-dimensional arrays only; `mxCreateNumericArray` can create arrays of two or more dimensions.

`mxCreateNumericArray` allocates dynamic memory to store the created `mxArray`. When you finish with the created `mxArray`, call `mxDestroyArray` to deallocate its memory.

# mxCreateNumericArray

---

## Examples

Call `mxCreateNumericArray` to create a 2-by-3-by-2 `mxArray` of unsigned 8-bit integers. Then, call a combination of functions to populate the `mxArray`:

```
#define FIRST_DIM 2
#define SECOND_DIM 3
#define THIRD_DIM 2
#define TOTAL_ELEMENTS (FIRST_DIM * SECOND_DIM * THIRD_DIM)
int ndim = 3, dims[3] = {FIRST_DIM, SECOND_DIM, THIRD_DIM};
unsigned char real_data[] = {9, 7, 5, 2, 6, 3, 4, 8, 2, 1, 10, 5};
unsigned char *start_of_pr;
unsigned char *start_of_pi;
mxArray *array_ptr;
size_t bytes_to_copy;

/* Create a 2-by-3-by-2 array of unsigned 8-bit integers. */
array_ptr = mxCreateNumericArray(ndim, dims, mxUINT8_CLASS,
 mxREAL);

if (array_ptr == NULL)
 mexErrMsgTxt("Could not create mxArray.\n");

/* Populate the real part of the created array. */
start_of_pr = (unsigned char *)mxGetPr(array_ptr);
bytes_to_copy = TOTAL_ELEMENTS * mxGetElementSize(array_ptr);
memcpy(start_of_pr, real_data, bytes_to_copy);

/* Populate the imaginary part of the created array. */
start_of_pi = (unsigned char *)mxGetPi(array_ptr);
memcpy(start_of_pi, real_data, bytes_to_copy);
```

For an additional example, see `mxcreatenumeri carray. c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateDoubleMatrix`, `mxCreateSparse`, `mxCreateString`, `mxComplexity`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create a 2-dimensional unpopulated sparse mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateSparse(int m, int n, int nzmax,                         mxComplexity ComplexFlag);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>   | <p><b>m</b><br/>The desired number of rows.</p> <p><b>n</b><br/>The desired number of columns.</p> <p><b>nzmax</b><br/>The number of elements that mxCreateSparse should allocate to hold the pr, ir, and, if ComplexFlag is mxCOMPLEX, pi arrays. Set the value of nzmax to be greater than or equal to the number of nonzero elements you plan to put into the mxArray, but make sure that nzmax is less than or equal to m*n.</p> <p><b>ComplexFlag</b><br/>Set this value to mxREAL or mxCOMPLEX. If the mxArray you are creating is to contain imaginary data, then set ComplexFlag to mxCOMPLEX; otherwise, set ComplexFlag to mxREAL.</p> |
| <b>Returns</b>     | A pointer to the created sparse mxArray on success; returns NULL on failure. The most likely reason for failure is insufficient free heap space. If that happens, try reducing nzmax, m, or n.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Call mxCreateSparse to create an unpopulated sparse mxArray. The returned sparse mxArray contains no sparse information and cannot be passed as an argument to any MATLAB sparse functions. In order to make the returned sparse mxArray useful, you must initialize the pr, ir, jc, and (if it exists) pi array.</p> <p>mxCreateSparse allocates space for</p> <ul style="list-style-type: none"><li>• A pr array of m-by-n elements.</li><li>• A pi array of m-by-n elements (but only if ComplexFlag is mxCOMPLEX).</li><li>• An ir array of nzmax elements.</li><li>• A jc array of m elements.</li></ul>                                 |

# mxCreateSparse

---

When you finish using the sparse mxArray, call `mxDestroyArray` to reclaim all its heap space.

## Examples

Create a sparse mxArray of real data. Although the mxArray is 3-by-2, only four of the six elements are occupied by nontrivial data.

```
#define NZMAX 4
#define ROWS 4
#define COLS 2
int rows=ROWS, cols=COLS;
mxArray *ptr_array; /* Pointer to created sparse array. */
static double static_pr_data[NZMAX] = {5.8, 6.2, 5.9, 6.1};
static int static_ir_data[NZMAX] = {0, 2, 1, 3};
static int static_jc_data[COLS+1] = {0, 2, 4};
double *start_of_pr;
int *start_of_ir, *start_of_jc;
mxArray *array_ptr;

/* Create a sparse array and name it "Sparrow". */
array_ptr = mxCreateSparse(rows, cols, NZMAX, mxREAL);
mxSetName(array_ptr, "Sparrow");

/* Place pr data into the newly created sparse array. */
start_of_pr = (double *)mxGetPr(array_ptr);
memcpy(start_of_pr, static_pr_data, NZMAX*sizeof(double));

/* Place ir data into the newly created sparse array. */
start_of_ir = (int *)mxGetIr(array_ptr);
memcpy(start_of_ir, static_ir_data, NZMAX*sizeof(int));

/* Place jc data into the newly created sparse array. */
start_of_jc = (int *)mxGetJc(array_ptr);
memcpy(start_of_jc, static_jc_data, NZMAX*sizeof(int));

/* ... Use the sparse array in some fashion. */
/* When finished with the mxArray, deallocate it. */
mxDestroyArray(array_ptr);
```

For an additional example, see `mxcreatesparse.c` in the `mx` subdirectory of the `examples` directory.

**See Also**

`mxDestroyArray`, `mxSetNzmax`, `mxSetPr`, `mxSetPi`, `mxSetIr`, `mxSetJc`,  
`mxComplexity`

# mxCreateString

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create a 1-by-n string <code>mxArray</code> initialized to the specified string                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateString(const char *str);</pre>                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>   | <code>str</code><br>The C string that is to serve as the <code>mxArray</code> 's initial data.                                                                                                                                                                                                                                                                                 |
| <b>Returns</b>     | A pointer to the created string <code>mxArray</code> , if successful; otherwise, returns <code>NULL</code> . The most likely cause of failure is insufficient free heap space.                                                                                                                                                                                                 |
| <b>Description</b> | Use <code>mxCreateString</code> to create a string <code>mxArray</code> initialized to <code>str</code> . Many MATLAB functions (for example, <code>strcmp</code> and <code>upper</code> ) require string array inputs.<br><br>Free the string <code>mxArray</code> when you are finished using it. To free a string <code>mxArray</code> , call <code>mxDestroyArray</code> . |

**Examples** Create a string `mxArray` named `s` containing the value of string `idiom`:

```
mxArray *array_ptr;
const char idiom[] = "Everyone loves MATLAB. ";

/* Create a string array. */
array_ptr = mxCreateString(idiom);

/* Name the string array "s". */
mxSetName(array_ptr, "s");

/* Place the string array in the MATLAB workspace, then invoke
a MATLAB string manipulation function on it. */
mexPutArray(array_ptr, "caller");
mexEvalString("us = upper(s)");

/* When finished with the string array, free its memory. */
mxDestroyArray(array_ptr);
```

For an additional example, see `mxcreatestri ng. c` in the `mx` subdirectory of the `examples` directory.

**See Also**

`mxCreateCharMatrixFromStrings`, `mxCreateCharArray`

# mxCreateStructArray

---

**Purpose** Create an unpopulated N-dimensional structure `mxArray`

**C Syntax**

```
#include "matrix.h"
mxArray *mxCreateStructArray(int ndim, const int *dims, int nfields,
 const char **field_names);
```

**Arguments**

`ndim`  
Number of dimensions. If you set `ndims` to be less than 2, `mxCreateNumericArray` creates a two-dimensional `mxArray`.

`dims`  
The dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, setting `dims[0]` to 5 and `dims[1]` to 7 establishes a 5-by-7 `mxArray`. Typically, the `dims` array should have `ndim` elements.

`nfields`  
The desired number of fields in each element.

`field_names`  
The desired list of field names.

**Returns** A pointer to the created structure `mxArray`, if successful; otherwise, returns NULL. The most likely cause of failure is insufficient heap space to hold the returned `mxArray`.

**Description** Call `mxCreateStructArray` to create an unpopulated structure `mxArray`. Each element of a structure `mxArray` contains the same number of fields (specified in `nfields`). Each field has a name; the list of names is specified in `field_names`. A structure `mxArray` in MATLAB is conceptually identical to an array of structs in the C language.

Each field holds one `mxArray` pointer. `mxCreateStructArray` initializes each field to NULL. Call `mxSetField` or `mxSetFieldByNumber` to place a non-NULL `mxArray` pointer in a field.

When you finish using the returned structure `mxArray`, call `mxDestroyArray` to reclaim its space.

**Examples** Create a 2-by-3 structure `mxArray`, in which each element contains two fields (the "name" field and the "grade" field). Then, populate the field with data.

Each "name" field holds a string mxArray and each "grade" field holds a numeric mxArray.

```
#define rows 3
#define cols 2
#define TOTAL_ELEMENTS (rows * cols)
int ndim = 2, dims[2] = {rows, cols};
int number_of_fields=2;
const char *field_names[] = {"name", "grade"};
const char *names_values[] = {"Rachel", "Aysha", "Maria",
 "Per", "Martin", "Bob"};
double grades_values[] = {100, 95, 95, 97, 98, 96};
double *pr;
mxArray *field_value, *struct_array_ptr;
int index;
unsigned char *start_of_pr, *start_of_pi;
mxArray *array_ptr;
size_t bytes_to_copy;
/* Create a 2-by-3 array of structs. */
struct_array_ptr = mxCreateStructArray(ndim, dims,
 number_of_fields, field_names);

/* Populate the 6 name fields. */
for (index=0; index<TOTAL_ELEMENTS; index++) {
 field_value = mxCreateString(names_values[index]);
 mxSetField(struct_array_ptr, index, "name", field_value);
 /* mxDestroyArray(field_value); */
}

/* Populate the 6 grade fields. */
for (index=0; index<TOTAL_ELEMENTS; index++) {
 field_value = mxCreateDoubleMatrix(1, 1, mxREAL);
 pr = mxGetPr(field_value);
 pr[0] = grades_values[index];
 mxSetField(struct_array_ptr, index, "grade", field_value);
 /* mxDestroyArray(field_value); */
}
```

For an additional example, see `mxcreatestructarray.c` in the `mx` subdirectory of the `examples` directory.

# mxCreateStructArray

---

## See Also

`mxCreateFull`, `mxDestroyArray`, `mxSetNzmax`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Create an unpopulated 2-dimensional structure <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxCreateStructMatrix(int m, int n, int nfields,                                const char **field_names);</pre>                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | <p><code>m</code><br/>The desired number of rows. This must be a positive integer.</p> <p><code>n</code><br/>The desired number of columns. This must be a positive integer.</p> <p><code>nfields</code><br/>The desired number of fields in each element.</p> <p><code>field_names</code><br/>The desired list of field names.</p>                                                                                                                                                                                                    |
| <b>Returns</b>     | A pointer to the created structure <code>mxArray</code> , if successful; otherwise, returns <code>NULL</code> . The most likely cause of failure is insufficient heap space to hold the returned <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | <code>mxCreateStructMatrix</code> and <code>mxCreateStructArray</code> are almost identical. The only difference is that <code>mxCreateStructMatrix</code> can only create two-dimensional <code>mxArrays</code> , while <code>mxCreateStructArray</code> can create <code>mxArrays</code> having two or more dimensions.                                                                                                                                                                                                              |
| <b>Examples</b>    | Create a two-dimensional structure <code>mxArray</code> having the dimensions 5-by-7:<br><pre>int      rows=100, cols=1; int      number_of_fields=4; const char *fieldnames[] = {"pressure", "wind speed",                              "wind direction", "dewpoint"}; mxArray *struct_array_ptr;  /* Create a 100-by-1 structure mxArray. */ struct_array_ptr = mxCreateStructMatrix(rows, cols,                                          number_of_fields, fieldnames);  /* Populate the structure mxArray with data. */ ... </pre> |

## mxCreateStructMatrix

---

The created structure `mxArray` is unpopulated. See the `mxCreateStructArray` reference page for sample code demonstrating how to populate a structure `mxArray`. For an additional example of `mxCreateStructMatrix`, see `mxcreatestructmatrix.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxCreateStructArray`, `mxGetFieldByNumber`, `mxGetFieldNameByNumber`, `mxGetFieldNumber`, `mxIsStruct`

**Purpose** Free dynamic memory allocated by an `mxCreate` routine

**C Syntax**

```
#include "matrix.h"
void mxDestroyArray(mxArray *array_ptr);
```

**Arguments**

`array_ptr`  
Pointer to the `mxArray` that you want to free.

**Description**

`mxDestroyArray` deallocates the memory occupied by the specified `mxArray`. `mxDestroyArray` not only deallocates the memory occupied by the `mxArray`'s characteristics fields (such as `m` and `n`), but also deallocates all the `mxArray`'s associated data arrays (such as `pr`, `pi`, `ir`, and/or `jc`). You should not call `mxDestroyArray` on an `mxArray` you are returning on the left hand side.

**Examples**

```
int rows=2, cols=4;
double pr_data[] = {5.2, 7.9, 1.3, 4.2, 6.7, 5.9, 1.9, 9.4};
double pi_data[] = {3.4, 6.5, 2.2, 9.1, 8.3, 4.7, 2.5, 7.5};
double *pr, *pi;
mxArray *array_ptr;

/* Create a 2-by-4 populated complex matrix. */
array_ptr = mxCreateDoubleMatrix(rows, cols, mxCOMPLEX);
pr = mxGetPr(array_ptr);
pi = mxGetPi(array_ptr);
memcpy((void *)pr, (const void *)pr_data,
 rows*cols*sizeof(double));
memcpy((void *)pi, (const void *)pi_data,
 rows*cols*sizeof(double));

/* Use the array in some fashion. */
...

/* When finished using the array, deallocate its space. */
mxDestroyArray(array_ptr);
```

For an additional example, see `mxdestroyarray.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxMalloc`, `mxFree`, `mexMakeArrayPersistent`, `mexMakeMemoryPersistent`

# mxDuplicateArray

---

**Purpose** Make a deep copy of an array

**C Syntax**

```
#include "matrix.h"
mxArray *mxDuplicateArray(const mxArray *in);
```

**Arguments**

`in`  
Pointer to the array's copy.

**Description** `mxDuplicateArray` makes a deep copy of an array, and returns a pointer to the copy. A deep copy refers to a copy in which all levels of data are copied. For example, a deep copy of a cell array copies each cell, and the contents of the each cell (if any), and so on.

**Example** See `mxduplicatearray.c` in the `mx` subdirectory of the `examples` directory.

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Free dynamic memory allocated by <code>mxCall oc</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxFree(void *ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>   | <code>ptr</code><br>Pointer to the beginning of any memory parcel allocated by <code>mxCall oc</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | <p>To deallocate heap space, MATLAB applications should always call <code>mxFree</code> rather than the ANSI C <code>free</code> function.</p> <p><code>mxFree</code> works differently in MEX-files than in stand-alone MATLAB applications.</p> <p>In MEX-files, <code>mxFree</code> automatically</p> <ul style="list-style-type: none"><li>• Calls the ANSI C <code>free</code> function, which deallocates the contiguous heap space that begins at address <code>ptr</code>.</li><li>• Removes this memory parcel from the MATLAB memory management facility's list of memory parcels.</li></ul> <p>The MATLAB memory management facility maintains a list of all memory allocated by <code>mxCall oc</code> (and by the <code>mxCreate</code> calls). The MATLAB memory management facility automatically frees (deallocates) all of a MEX-file's parcels when control returns to the MATLAB prompt.</p> <p>By default, when <code>mxFree</code> appears in stand-alone MATLAB applications, <code>mxFree</code> simply calls the ANSI C <code>free</code> function. If this default behavior is unacceptable, you can write your own memory deallocation routine and register this routine with <code>mxSetAll ocFcns</code>. Then, whenever <code>mxFree</code> is called, <code>mxFree</code> calls your memory allocation routine instead of <code>free</code>.</p> <p>In a MEX-file, your use of <code>mxFree</code> depends on whether the specified memory parcel is persistent or nonpersistent. By default, memory parcels created by <code>mxCall oc</code> are nonpersistent. However, if an application calls <code>mexMakeMemoryPersistent</code>, then the specified memory parcel becomes persistent.</p> <p>The MATLAB memory management facility automatically frees all nonpersistent memory whenever a MEX-file completes. Thus, even if you do not call <code>mxFree</code>, MATLAB takes care of freeing the memory for you.</p> |

## mxFree

---

Nevertheless, it is a good programming practice to deallocate memory just as soon as you are through using it. Doing so generally makes the entire system run more efficiently.

When a MEX-file completes, the MATLAB memory management facility does not free persistent memory parcels. Therefore, the only way to free a persistent memory parcel is to call `mxFree`. Typically, MEX-files call `mexAtExit` to register a clean-up handler. Then, the clean-up handler calls `mxFree`.

### Example

See the examples on the `mxMalloc` and `mexAtExit` reference pages.

### See Also

`mxMalloc`, `mxDestroyArray`, `mxMalloc`, `mexMakeArrayPersistent`,  
`mexMakeMemoryPersistent`

**V4 Compatible** This function is obsolete; MATLAB 5 does not support it. To use this function in existing code, use the `-V4` option of the `mex` script.

Call `mxDestroyArray` instead of `mxFreeMatrix`.

**See Also** `mxDestroyArray`

# mxGetCell

---

**Purpose** Get a cell's contents

**C Syntax**

```
#include "matrix.h"
mxArray *mxGetCell(const mxArray *array_ptr, int index);
```

**Arguments**

`array_ptr`  
Pointer to a cell mxArray.

`index`  
The number of elements in the cell mxArray between the first element and the desired one. See `mxCalcSingleSubscript` for details on calculating an index.

**Returns** A pointer to the `i`th cell mxArray, if successful; otherwise, returns NULL. Causes of failure include

- Insufficient free heap space to hold the returned cell mxArray.
- Specifying an `array_ptr` that does not point to a cell mxArray.
- Specifying an `index` greater than the number of elements in the cell.

**Description** Call `mxGetCell` to get a pointer to the mxArray held in the `index`th element of the cell mxArray. **Note:** Changing data contained within the cell may cause unpredictable results.

## Examples

Consider a MEX-file named `OneTwo` that calls `mxGetCell` to get the (1, 2) cell of an input cell `mxArray`. If cell (1, 2) contains a string `mxArray`, the MEX-file converts its data to C string format.

```

void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int nsubs=2, subs[]={0, 1}, index, buflen, status;
 mxArray *cell_element_ptr;
 char *buf;

 /* Is the first input argument a cell mxArray. */
 if (mxIsCell(prhs[0])) {

 /* Get the cell at (1, 2). */
 index = mxCalcSingleSubscript(prhs[0], nsubs, subs);
 cell_element_ptr = mxGetCell(prhs[0], index);

 /* If the cell at (1, 2) holds a string, print the string. */
 if (mxIsChar(cell_element_ptr)) {

 /* Find out how long the input string array is. */
 buflen = (mxGetM(cell_element_ptr) *
 mxGetN(cell_element_ptr)) + 1;

 /* Allocate enough memory to hold the converted string. */
 buf = mxMalloc(buflen, sizeof(char));
 if (buf == NULL)
 mexErrMsgTxt("Not enough heap space to hold string");
 else {
 /* Copy the string data into buf. */
 status = mxGetString(cell_element_ptr, buf, buflen);

 /* Manipulate the string. */
 ...
 }
 }
 }
}

```

## mxGetCell

---

In MATLAB, create a 2-by-2 cell mxArray named A:

```
>> A(1, 1) = {[1 4 3; 0 5 8; 7 2 9]};
>> A(1, 2) = {'Marilyn'};
>> A(2, 1) = {3+7i};
>> A(2, 2) = {-pi : pi / 10 : pi }
```

Passing A as an argument to OneTwo

```
>> OneTwo(A)
```

causes OneTwo to convert the contents of cell (1, 2) to the C string Marilyn. For an additional example, see `mxgetCell.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxCreateCellArray`, `mxIsCell`, `mxSetCell`

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>   | Get (as an enumerated constant) an <code>mxArray</code> 's class                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>C Syntax</b>  | <pre>#include "matrix.h" mxClassID mxGetClassID(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b> | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Returns</b>   | The class (category) of the <code>mxArray</code> that <code>array_ptr</code> points to. Classes are:<br><br><code>mxCELL_CLASS</code><br>Identifies a cell <code>mxArray</code> .<br><br><code>mxSTRUCT_CLASS</code><br>Identifies a structure <code>mxArray</code> .<br><br><code>mxOBJECT_CLASS</code><br>Identifies a user-defined (nonstandard) <code>mxArray</code> .<br><br><code>mxCHAR_CLASS</code><br>Identifies a string <code>mxArray</code> ; that is an <code>mxArray</code> whose data is represented as <code>mxCHAR</code> 's.<br><br><code>mxSPARSE_CLASS</code><br>Identifies a sparse <code>mxArray</code> ; that is, an <code>mxArray</code> that only stores its nonzero elements.<br><br><code>mxDOUBLE_CLASS</code><br>Identifies a numeric <code>mxArray</code> whose data is stored as double-precision, floating-point numbers.<br><br><code>mxSINGLE_CLASS</code><br>Identifies a numeric <code>mxArray</code> whose data is stored as single-precision, floating-point numbers.<br><br><code>mxINT8_CLASS</code><br>Identifies a numeric <code>mxArray</code> whose data is stored as signed 8-bit integers.<br><br><code>mxUINT8_CLASS</code><br>Identifies a numeric <code>mxArray</code> whose data is stored as unsigned 8-bit integers.<br><br><code>mxINT16_CLASS</code> |

# mxGetClassID

---

Identifies a numeric mxArray whose data is stored as signed 16-bit integers.

`mxUINT16_CLASS`

Identifies a numeric mxArray whose data is stored as unsigned 16-bit integers.

`mxINT32_CLASS`

Identifies a numeric mxArray whose data is stored as signed 32-bit integers.

`mxUINT32_CLASS`

Identifies a numeric mxArray whose data is stored as unsigned 32-bit integers.

`mxINT64_CLASS`

Reserved for possible future use.

`mxUINT64_CLASS`

Reserved for possible future use.

`mxUNKNOWN_CLASS = -1`

The class cannot be determined. You cannot specify this category for an mxArray; however, `mxGetClassID` can return this value if it cannot identify the class.

## Description

Use `mxGetClassID` to determine the class of an mxArray. The class of an mxArray identifies the kind of data the mxArray is holding. For example, if `array_ptr` points to a sparse mxArray, then `mxGetClassID` returns `mxSPARSE_CLASS`.

`mxGetClassID` is similar to `mxGetClassName`, except that the former returns the class as an enumerated value and the latter returns the class as a string.

## Examples

Consider a MEX-file that can accept any kind of `mxArray` as its first input argument. If you intend to take action on the first input argument depending on its class, the first step is to call `mxGetClassID`.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 mxClassID category;

 category = mxGetClassID(prhs[0]);
 switch(category) {
 case mxCHAR_CLASS: analyze_string(prhs[0]); break;
 case mxSTRUCT_CLASS: analyze_structure(prhs[0]); break;
 case mxSPARSE_CLASS: analyze_sparse(prhs[0]); break;
 case mxCELL_CLASS: analyze_cell(prhs[0]); break;
 case mxUNKNOWN_CLASS: mexWarnMsgTxt("Unknown class."); break;
 default: analyze_numeric(prhs[0]); break;
 }
}
```

## See Also

`mxGetClassName`

# mxGetClassName

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get (as a string) an mxArray' s class                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>C Syntax</b>    | <pre>#include "matrix.h" const char *mxGetClassName(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Returns</b>     | The class (as a string) of array_ptr.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p>Call <code>mxGetClassName</code> to determine the class of an mxArray. The class of an mxArray identifies the kind of data the mxArray is holding. For example, if <code>array_ptr</code> points to a sparse mxArray, then <code>mxGetClassName</code> returns "sparse".</p> <p><code>mxGetClassID</code> is similar to <code>mxGetClassName</code>, except that the former returns the class as an enumerated value and the latter returns the class as a string.</p> |
| <b>Example</b>     | See <code>mxgetclassname.c</code> in the <code>mx</code> subdirectory of the <code>examples</code> directory.                                                                                                                                                                                                                                                                                                                                                             |
| <b>See Also</b>    | <code>mxGetClassID</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

|                    |                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get pointer to data                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void *mxGetData(const mxArray *array_ptr);</pre>                                                                |
| <b>Arguments</b>   | <pre>array_ptr</pre> Pointer to an mxArray.                                                                                              |
| <b>Description</b> | Similar to mxGetPr, except mxGetData returns a void *. Use mxGetData on numeric arrays with contents other than double.                  |
| <b>Example</b>     | <pre>type = mxGetClass(prhs[0]); switch type case mxUNIT8_CLASS:     data = (unsigned char *)mxGetData(prhs[0]);     .     .     .</pre> |
| <b>See Also</b>    | mxGetPr                                                                                                                                  |

# mxGetDimensions

---

**Purpose** Get a pointer to the dimensions array

**C Syntax**

```
#include "matrix.h"
const int *mxGetDimensions(const mxArray *array_ptr);
```

**Arguments**

array\_ptr  
Pointer to an mxArray.

**Returns** The address of the first element in a dimension array. Each integer in the dimensions array represents the number of elements in a particular dimension. The array is not NULL-terminated.

**Description** Use `mxGetDimensions` to determine how many elements are in each dimension of the mxArray that `array_ptr` points to. Call `mxGetNumberOfDimensions` to get the number of dimensions in the mxArray.

**Example** Consider a MEX-file named `GetSize` that returns the size of each dimension of an input mxArray:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int number_of_dims, c;
 const int *dim_array;

 number_of_dims = mxGetNumberOfDimensions(prhs[0]);
 dim_array = mxGetDimensions(prhs[0]);
 for (c=0; c<number_of_dims; c++)
 mexPrintf("%d\n", *dim_array++);
}
```

In MATLAB, create a variable named `my4d`, then pass `my4d` to `GetSize`:

```
>> my4d = rand(3, 5, 2, 4);
```

Call `GetSize` to return the size of each dimension in `my4d`:

```
>> GetSize(my4d)
3
5
2
4
```

## See Also

`mxGetNumberOfDimensions`

# mxGetElementSize

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the number of bytes required to store each data element                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxGetElementSize(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Returns</b>     | The number of bytes required to store one element of the specified <code>mxArray</code> , if successful. Returns 0 on failure. The primary reason for failure is that <code>array_ptr</code> points to an <code>mxArray</code> having an unrecognized class. If <code>array_ptr</code> points to a cell <code>mxArray</code> or a structure <code>mxArray</code> , then <code>mxGetElementSize</code> returns the size of a pointer (not the size of all the elements in each cell or structure field).                                                                                                               |
| <b>Description</b> | <p>Call <code>mxGetElementSize</code> to determine the number of bytes in each data element of the <code>mxArray</code>. For example, if the <code>mxClassID</code> of an <code>mxArray</code> is <code>mxINT16_CLASS</code>, then the <code>mxArray</code> stores each data element as a 16-bit (2 byte) signed integer. Thus, <code>mxGetElementSize</code> returns 2.</p> <p><code>mxGetElementSize</code> is particularly helpful when using a nonMATLAB routine to manipulate data elements. For example, <code>memcpy</code> requires (for its third argument) the size of the elements you intend to copy.</p> |
| <b>Examples</b>    | Consider a MEX-file that calls <code>memcpy</code> to make a copy of whatever kind of data gets passed to it. The third argument to <code>memcpy</code> is the number of bytes to copy.                                                                                                                                                                                                                                                                                                                                                                                                                               |

In order to determine the number of bytes, you must first call `mxGetElementSize`.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int bytes_per_element, total_elements;
 void *pr, *mycopy;

 /* Get the characteristics of the input mxArray. */
 bytes_per_element = mxGetElementSize(prhs[0]);
 total_elements = mxGetM(prhs[0]) * mxGetN(prhs[0]);
 pr = mxGetPr(prhs[0]);

 /* Allocate enough heap to hold a copy of the real elements of
 the input mxArray. Then copy them. */
 mycopy = mxMalloc(total_elements, bytes_per_element);
 memcpy(mycopy, pr, bytes_per_element * total_elements);

 ...
}
```

For an additional example, see `mxgetelementsize.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetM`, `mxGetN`

# mxGetEps

---

|                    |                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get value of eps                                                                                                                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" double mxGetEps(void);</pre>                                                                                                                                                                                                                                                                      |
| <b>Returns</b>     | The value of the MATLAB eps variable.                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | Call <code>mxGetEps</code> to return the value of MATLAB's eps variable. This variable holds the distance from 1.0 to the next largest floating point number. As such, it is a measure of floating-point accuracy. MATLAB's <code>PINV</code> and <code>RANK</code> functions use <code>eps</code> as a default tolerance. |
| <b>See Also</b>    | <code>mxGetInf</code> , <code>mxGetNaN</code>                                                                                                                                                                                                                                                                              |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get a field value, given a field name and an index in a structure array                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" mxArray *mxGetField(const mxArray *array_ptr, int index,                     const char *field_name);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a structure <code>mxArray</code>.</p> <p><code>index</code><br/>The desired element. The first element of an <code>mxArray</code> has an <code>index</code> of 0, the second element has an <code>index</code> of 1, and the last element has an <code>index</code> of <code>N-1</code>, where <code>N</code> is the total number of elements in the structure <code>mxArray</code>.</p> <p><code>field_name</code><br/>The name of the field whose value you want to extract.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Returns</b>     | <p>A pointer to the <code>mxArray</code> in the specified field at the specified <code>field_name</code>, on success. Returns <code>NULL</code> otherwise. One possibility is that there is no value assigned to the specified field. Another possibility is that there is a value, but the call failed. Common causes of failure include</p> <ul style="list-style-type: none"><li>• Specifying an <code>array_ptr</code> that does not point to a structure <code>mxArray</code>. To determine if <code>array_ptr</code> points to a structure <code>mxArray</code>, call <code>mxIsStruct</code>.</li><li>• Specifying an out-of-range <code>index</code> to an element past the end of the <code>mxArray</code>. For example, given a structure <code>mxArray</code> that contains 10 elements, you cannot specify an <code>index</code> greater than 9.</li><li>• Specifying a nonexistent <code>field_name</code>. Call <code>mxGetFieldByName</code> or <code>mxGetFieldNumber</code> to get existing <code>field</code> names.</li><li>• Insufficient heap space to hold the returned <code>mxArray</code>.</li></ul> |
| <b>Description</b> | <p>Call <code>mxGetField</code> to get the value held in the specified element of the specified field. In pseudo-C terminology, <code>mxGetField</code> returns the value at</p> <pre>array_ptr[index].field_name</pre> <p><code>mxGetFieldByIndex</code> is similar to <code>mxGetField</code>. Both functions return the same value. The only difference is in the way you specify the field. <code>mxGetFieldByIndex</code> takes <code>field_num</code> as its third argument, and <code>mxGetField</code> takes <code>field_name</code> as its third argument.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

# mxGetField

---

When you finish using the returned `mxArray`, call `mxDestroyArray` to deallocate it. **Note:** Changing data contained within the field may cause unpredictable results.

## Examples

Consider a MEX-file named `WhatBill` that prints the value of the "Billing" field in each element of an input structure `mxArray`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 mxArray *field_array_ptr;
 double value_in_billing_field;
 int index, number_of_elements;

 /* Find the values of the billing field. */
 if (mxIsStruct(prhs[0])) {

 /* Loop through each element of the structure. */
 number_of_elements = mxGetM(prhs[0]) * mxGetN(prhs[0]);
 for (index=0; index<number_of_elements; index++) {

 /* Get the value in the "billing" field at this index. */
 field_array_ptr = mxGetField(prhs[0], index, "billing");

 /* The returned "field" is a pointer to a scalar mxArray.
 Get the value associated with the scalar mxArray. */
 if (field_array_ptr) {
 value_in_billing_field = mxGetScalar(field_array_ptr);
 mexPrintf("%g\n", value_in_billing_field);
 }
 else
 mexWarnMsgTxt("Missing 'billing' field.");
 }
 }
 else
 mexErrMsgTxt("You must pass a structure mxArray.");
}
```

In MATLAB, create a structure named `patient` initialized to

```
>> patient(1).name=' Cheryl Doe' ;
>> patient(1).billing=827;
>> patient(2).name=' Scott Woe' ;
>> patient(2).billing=435;
>> patient(3).name=' Cleve Roe' ;
>> patient(3).billing=256;
```

Passing `patient` as the first argument to `WhatBill` yields

```
>> WhatBill(patient)
827
435
256
```

For an additional example, see `mxgetfield.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetFieldByNumber`, `mxGetFieldByNameByNumber`, `mxGetFieldNumber`,  
`mxGetNumberOfFields`, `mxIsStruct`, `mxSetField`, `mxSetFieldByNumber`

# mxGetFieldByNumber

---

**Purpose** Get a field value, given a field number and an index in a structure array

**C Syntax**

```
#include "matrix.h"
mxArray *mxGetFieldNumber(const mxArray *array_ptr, int index,
 int *field_number);
```

**Arguments**

`array_ptr`  
Pointer to a structure `mxArray`.

`index`  
The desired element. The first element of an `mxArray` has an `index` of 0, the second element has an `index` of 1, and the last element has an `index` of `N-1`, where `N` is the total number of elements in the structure `mxArray`. See `mxCalcSingleSubscript` for more details on calculating an `index`.

`field_number`  
The position of the field whose value you want to extract. The first field within each element has a field number of 0, the second field has a field number of 1, and so on. The last field has a field number of `N-1`, where `N` is the number of fields.

**Returns** A pointer to the `mxArray` in the specified field at the specified `field_name`, on success. Returns `NULL` otherwise. One possibility is that there is no value assigned to the specified field. Another possibility is that there is a value, but the call failed. Common causes of failure include

- Specifying an `array_ptr` that does not point to a structure `mxArray`. Call `mxIsStruct` to determine if `array_ptr` points to is a structure `mxArray`.
- Specifying an out-of-range `index` to an element past the end of the `mxArray`. For example, given a structure `mxArray` that contains 10 elements, you cannot specify an `index` greater than 9.
- Specifying a nonexistent field name. Call `mxGetFieldNameByNumber` or `mxGetFieldNumber` to determine existing field names.
- Insufficient heap space to hold the returned `mxArray`.

**Description** Call `mxGetFieldByNumber` to get the value held in the specified `field_number` at the `index`-th element.

When you finish using the returned `mxArray`, call `mxDestroyArray` to deallocate it. **Note:** Changing data contained within the field may cause unpredictable results.

## Examples

Consider a MEX-file that gathers a pointer to the `mxArray` stored at each field of each element in a structure `mxArray`. For example, given an input 12-by-1 structure `mxArray` in which each element contains three fields, the MEX-file calls `mxGetFieldByNumber` 36 times:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int total_array_elements, number_of_fields;
 int index, field_num;
 const char *field_name;
 mxArray *field_array_ptr;

 if (mxIsStruct(prhs[0])) {

 total_array_elements = mxGetM(prhs[0]) * mxGetN(prhs[0]);
 number_of_fields = mxGetNumberOfFields(prhs[0]);

 for (index=0; index < total_array_elements; index++) {
 for (field_num=0; field_num<number_of_fields;
 field_num++){
 field_array_ptr = mxGetFieldByNumber(prhs[0], index,
 field_num);
 /* Code to analyze field_array_ptr. */
 }
 }
 }
 else
 mexErrMsgTxt("You must specify a structure array.");
}
```

This MEX-file does not contain code to analyze the returned `field_array_ptr`. (The online example `expl ore. c` does contain such code.) For an additional

# mxGetFieldByNumber

---

example, see `mxgetfieldbynumber.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetField`, `mxGetFieldByName`, `mxGetFieldNumber`,  
`mxGetNumberOfFields`, `mxSetField`, `mxSetFieldByNumber`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get a field name, given a field number in a structure array                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" const char *mxGetFieldNameByNumber(const mxArray *array_ptr,     int field_number);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a structure <code>mxArray</code>.</p> <p><code>field_number</code><br/>The position of the desired field. For instance, to get the name of the first field, set <code>field_number</code> to 0; to get the name of the second field, set <code>field_number</code> to 1; and so on.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Returns</b>     | <p>A pointer to the <code>n</code>th field name, on success. Returns NULL on failure. Common causes of failure include:</p> <ul style="list-style-type: none"><li>• Specifying an <code>array_ptr</code> that does not point to a structure <code>mxArray</code>. Call <code>mxIsStruct</code> to determine if <code>array_ptr</code> points to a structure <code>mxArray</code>.</li><li>• Specifying a value of <code>field_number</code> greater than or equal to the number of fields in the structure <code>mxArray</code>. (Remember that <code>field_number</code> 0 symbolizes the first field, so <code>index N-1</code> symbolizes the last field.)</li></ul> <p>You must allocate space to hold the returned field name. You should allocate enough memory to hold a string <code>mxMAXNAM</code> characters long.</p> |
| <b>Description</b> | <p>Call <code>mxGetFieldNameByNumber</code> to get the name of a field in the given structure <code>mxArray</code>. A typical use of <code>mxGetFieldNameByNumber</code> is to call it inside a loop in order to get the names of all the fields in a given <code>mxArray</code>.</p> <p>Consider a MATLAB structure initialized to</p> <pre>&gt;&gt; patient.name = 'John Doe'; &gt;&gt; patient.billing = 127.00; &gt;&gt; patient.test = [79 75 73; 180 178 177.5; 220 210 205];</pre> <p>The <code>field_number</code> 0 represents the field name "name"; <code>field_number</code> 1 represents field name "billing"; <code>field_number</code> 2 represents field name "test". A <code>field_number</code> other than 0, 1, or 2 causes <code>mxGetFieldNameByNumber</code> to return NULL.</p>                            |

# mxGetFieldNameByNumber

---

## Examples

Create a MEX-file named `SeeStruc` that tells you the names of each field in an input structure `mxArray`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int number_of_fields, field_num;
 const char *field_name;

 if (mxIsStruct(prhs[0])) {

 number_of_fields = mxGetNumberOfFields(prhs[0]);

 if (number_of_fields == 0)
 mexWarnMsgTxt("This structure has no fields.");
 else {
 /* Get the first field name, then the second, and so on. */
 for (field_num=0; field_num<number_of_fields; field_num++)
 {
 field_name = mxGetFieldNameByNumber(prhs[0], field_num);
 mexPrintf("%s\n", field_name);
 }
 }
 }
 else
 mexErrMsgTxt("You must pass a structure mxArray.");
}
```

In MATLAB, create a structure named `patient`

```
>> patient.name = 'John Doe';
>> patient.billing = 127.00;
>> patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

Passing `patient` to `SeeStruc` yields

```
>> SeeStruc(patient)
The three fields of patient are:
 name
 billing
 test
```

For an additional example, see `mxgetfieldnamebynumber.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxGetField`, `mxIsStruct`, `mxSetField`

# mxGetFieldName

---

**Purpose** Get a field number, given a field name in a structure array

**C Syntax**

```
#include "matrix.h"
int mxGetFieldName(const mxArray *array_ptr,
 const char *field_name);
```

**Arguments**

`array_ptr`  
Pointer to a structure `mxArray`.

`field_name`  
The name of a field in the structure `mxArray`.

**Returns** The field number of the specified `field_name`, on success. The first field has a field number of 0, the second field has a field number of 1, and so on. Returns `-1` on failure. Common causes of failure include:

- Specifying an `array_ptr` that does not point to a structure `mxArray`. Call `mxIsStruct` to determine if `array_ptr` points to a structure `mxArray`.
- Specifying the `field_name` of a nonexistent field.

**Description** If you know the name of a field but do not know its field number, call `mxGetFieldName`. Conversely, if you know the field number but do not know its field name, call `mxGetFieldNameByNumber`.

For example, consider a MATLAB structure initialized to

```
>> patient.name = 'John Doe';
>> patient.billing = 127.00;
>> patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

The `field_name` "name" has a field number of 0; the `field_name` "billing" has a `field_number` of 1; and the `field_name` "test" has a field number of 2. If you call `mxGetFieldName` and specify a `field_name` of anything other than "name", "billing", or "test", then `mxGetFieldName` returns `-1`.

**Example** See `mxgetfieldnumber.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxGetField`, `mxGetFieldByNumber`, `mxGetFieldNameByNumber`, `mxGetNumberOfFields`, `mxSetField`, `mxSetFieldByNumber`

|                    |                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get pointer to imaginary data of an mxArray                                                                          |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void *mxGetImagData(const mxArray *array_ptr);</pre>                                        |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                  |
| <b>Description</b> | Similar to mxGetPi, except it returns a void *. Use mxGetImagData on numeric arrays with contents other than double. |
| <b>See Also</b>    | mxGetPi                                                                                                              |

# mxGetInf

---

**Purpose** Get the value of infinity

**C Syntax**

```
#include "matrix.h"
double mxGetInf(void);
```

**Returns** The value of infinity on your system.

**Description** Call `mxGetInf` to return the value of the MATLAB internal `inf` variable. `inf` is a permanent variable representing IEEE arithmetic positive infinity. The value of `inf` is built into the system; you cannot modify it.

Operations that return infinity include

- Division by 0. For example, `5/0` returns infinity.
- Operations resulting in overflow. For example, `exp(10000)` returns infinity because the result is too large to be represented on your machine.

**Example** See `mxgetinf.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxGetEps`, `mxGetNaN`

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the <code>i r</code> array of a sparse matrix                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int *mxGetIr(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Returns</b>     | A pointer to the first element in the <code>i r</code> array, if successful. Otherwise, returns <code>NULL</code> . Possible causes of failure include <ul style="list-style-type: none"><li>• Specifying a full (nonsparse) <code>mxArray</code>.</li><li>• Specifying a <code>NULL</code> <code>array_ptr</code>. (This usually means that an earlier call to <code>mxCreateSparse</code> failed.)</li></ul>                                                                                                                                                                                                                                                                                             |
| <b>Description</b> | Use <code>mxGetIr</code> to obtain the starting address of the <code>i r</code> array. The <code>i r</code> array is an array of integers; the length of the <code>i r</code> array is typically <code>nzmax</code> values. For example, if <code>nzmax</code> equals 100, then the <code>i r</code> array should contain 100 integers.<br><br>Each value in an <code>i r</code> array indicates a row (offset by 1) at which a nonzero element can be found. (The <code>j c</code> array is an index that indirectly specifies a column where nonzero elements can be found.)<br><br>For details on the <code>i r</code> and <code>j c</code> arrays, see <code>mxSetIr</code> and <code>mxSetJc</code> . |

## Examples

Consider a MEX-file named PrSparse that displays the positions and values of all nonzero elements in the input sparse mxArray.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double *pr;
 int *ir, *jc;
 int row, col, total=0, number_of_columns;
 int starting_row_index, stopping_row_index,
current_row_index;

 /* Get the starting positions of the data in the sparse array. */
 pr = mxGetPr(prhs[0]);
 ir = mxGetIr(prhs[0]);
 jc = mxGetJc(prhs[0]);

 /* Display the nonzero elements of the sparse array. */
 number_of_columns = mxGetN(prhs[0]);
 for (col=0; col<number_of_columns; col++) {
 starting_row_index = jc[col];
 stopping_row_index = jc[col+1];
 if (starting_row_index == stopping_row_index)
 continue;
 else {
 for (current_row_index = starting_row_index;
 current_row_index < stopping_row_index;
 current_row_index++)
 mexPrintf("(%d,%d) = %g\n", ir[current_row_index]+1,
 col+1, pr[total++]);
 }
 }
}
```

In MATLAB, create a sparse mxArray named Sparrow containing four nonzero elements:

```
>> Sparrow=sparse(zeros(100, 3));
>> Sparrow(50, 1)=1;
>> Sparrow(23, 2)=1;
>> Sparrow(37, 2)=1;
>> Sparrow(92, 2)=1;
```

Passing Sparrow as the first argument to PrSparse yields:

```
>> PrSparse(Sparrow)
(50, 1) = 1
(23, 2) = 1
(37, 2) = 1
(92, 2) = 1
```

For an additional example, see `mxgetir.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetJc`, `mxGetNzmax`, `mxSetIr`, `mxSetJc`, `mxSetNzmax`

# mxGetJc

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the <code>j c</code> array of a sparse matrix                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int *mxGetJc(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Returns</b>     | A pointer to the first element in the <code>j c</code> array, if successful; otherwise, returns <code>NULL</code> . The most likely cause of failure is specifying an <code>array_ptr</code> that points to a full (nonsparse) <code>mxArray</code> .                                                                                                                                                                                            |
| <b>Description</b> | Use <code>mxGetJc</code> to obtain the starting address of the <code>j c</code> array. The <code>j c</code> array is an integer array having <code>n+1</code> elements where <code>n</code> is the number of columns in the sparse <code>mxArray</code> . The values in the <code>j c</code> array indirectly indicate columns containing nonzero elements. For a detailed explanation of the <code>j c</code> array, see <code>mxSetJc</code> . |

**Examples**

Consider a MEX-file named `SecndCol` that displays the number of nonzero elements in the second column of an input `mxArray`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int *jc;
 int starting_row_index, stopping_row_index;
 int elements_in_second_col;

 /* Get the starting positions of the data in the sparse array. */
 jc = mxGetJc(prhs[0]);

 /* How many elements are in the second column of the sparse array.
 */
 starting_row_index = *(jc + 1);
 stopping_row_index = *(jc + 2);
 elements_in_second_col = stopping_row_index -
starting_row_index;
 mexPrintf("There are %d elements in the second column.\n",
 elements_in_second_col);
}
```

In MATLAB, create a sparse `mxArray` and put four nontrivial elements in the second column:

```
>> sp = sparse(eye(100));
>> sp(23, 2)=1;
>> sp(57, 2)=1;
>> sp(84, 2)=1;
```

Now call `SecndCol`, passing the sparse `mxArray` as an input argument:

```
>> SecndCol(sp);
There are 4 elements in the second column.
```

For an additional example, see `mxgetjc.c` in the `mx` subdirectory of the `examples` directory.

**See Also**

`mxGetIr`, `mxSetIr`, `mxSetJc`

# mxGetM

---

**Purpose** Get the number of rows

**C Syntax**

```
#include "matrix.h"
int mxGetM(const mxArray *array_ptr);
```

**Arguments**

array\_ptr  
Pointer to an array.

**Returns** The number of rows in the mxArray to which array\_ptr points.

**Description** mxGetM returns the number of rows in the specified array. The term “rows” always means the first dimension of the array no matter how many dimensions the array has. For example, if array\_ptr points to a four-dimensional array having dimensions 8-by-9-by-5-by-3, then mxGetM returns 8.

**Examples** Consider a MEX-file that accepts two input mxArray arguments. Both mxArray arguments must have the same number of rows in order for the calculation to work.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int rows;

 rows = mxGetM(prhs[0]);
 if (rows != 1)
 mexErrMsgTxt("First input array must be a vector.");
 else
 ...
}
```

For an additional example, see mxgetm.c in the mx subdirectory of the examples directory.

**See Also** mxGetN, mxSetM, mxSetN

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the total number of columns in a two-dimensional mxArray or the total number of elements in dimensions 2 through N for an m-by-n array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxGetN(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Returns</b>     | The number of columns in the mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Call mxGetN to determine the number of columns in the specified mxArray.</p> <p>If array_ptr is an N-dimensional mxArray, mxGetN is the product of dimensions 2 through N. For example, if array_ptr points to a four-dimensional mxArray having dimensions 13-by-5-by-4-by-6, then mxGetN returns the value 120 (5x4x6). If the specified mxArray has more than two dimensions and you need to know exactly how many elements are in each dimension, then call mxGetDimensions.</p> <p>If array_ptr points to a sparse mxArray, mxGetN still returns the number of columns, not the number of occupied columns.</p> |
| <b>Examples</b>    | <p>Find the total number of elements in an mxArray:</p> <pre>#include "mex.h"  void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {     int rows, cols, total_elements;      rows = mxGetM(prhs[0]);     cols = mxGetN(prhs[0]);     total_elements = rows * cols;     printf("This matrix has %d elements.\n", total_elements); }</pre>                                                                                                                                                                                                                                                      |

## mxGetN

---

For an additional example, see `mxgetn.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxGetM`, `mxGetNumberOfDimensions`, `mxSetM`, `mxSetN`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the name of the specified mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>C Syntax</b>    | <pre>#include "matrix.h" const char *mxGetName(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>   | <p>array_ptr<br/>Pointer to an mxArray.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Returns</b>     | A pointer to the start of the name field. If the mxArray has no name, the first element in the name field is <code>\0</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | <p>Use <code>mxGetName</code> to determine the name of the mxArray that <code>array_ptr</code> points to. The returned name is a NULL-terminated character string. MATLAB variable names are stored in fixed-length character arrays of length <code>mxMAXNAM+1</code>, where <code>mxMAXNAM</code> is defined in the file <code>mxArray.h</code>. Thus variable names can be any length up to <code>mxMAXNAM</code>. The actual length is determined by the NULL terminator.</p> <p><code>mxGetName</code> passes back a pointer to an existing section of memory; therefore, your application should not allocate space to hold the returned name string. Do not attempt to deallocate or free the returned string.</p> |
| <b>Examples</b>    | <p>Consider a MEX-file named <code>ret_name</code> that returns the name of the first input mxArray</p> <pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {     const char *array_name;      array_name = mxGetName(prhs[0]);     if (*array_name == '\0')         mexPrintf("This mxArray is unnamed.");     else         mexPrintf("The name of this mxArray is %s.\n", array_name); }</pre>                                                                                                                                                                                                                                                                                            |

# mxGetName

---

If you pass a named `mxArray` to `ret_name`, then `ret_name` prints the `mxArray`'s name; for example:

```
>> crandon = rand(6, 1);
>> ret_name(myarray)
The name of this mxArray is crandon.
```

However, passing an unnamed `mxArray`

```
>> mxGetName(rand(6, 1))
This mxArray is unnamed.
```

For an additional example, see `mxgetname.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxSetName`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the value of Not-a-Number                                                                                                                                                                                                                                                                                                                                                                          |
| <b>C Syntax</b>    | <pre>#include "matrix.h" double mxGetNaN(void);</pre>                                                                                                                                                                                                                                                                                                                                                  |
| <b>Returns</b>     | The value of Not-a-Number on your system.                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b> | <p>Call <code>mxGetNaN</code> to return the value of NaN for your system. NaN is the IEEE arithmetic representation for Not-a-Number. Certain mathematical operations return NaN as a result, for example:</p> <ul style="list-style-type: none"><li>• <code>0.0/0.0</code></li><li>• <code>inf-inf</code></li></ul> <p>The value of Not-a-Number is built in to the system; you cannot modify it.</p> |
| <b>See Also</b>    | <code>mxGetEps</code> , <code>mxGetInf</code>                                                                                                                                                                                                                                                                                                                                                          |

# mxGetNumberOfDimensions

---

**Purpose** Get the number of dimensions

**C Syntax**

```
#include "matrix.h"
int mxGetNumberOfDimensions(const mxArray *array_ptr);
```

**Arguments**

array\_ptr  
Pointer to an mxArray.

**Returns** The number of dimensions in the specified mxArray. The returned value is always 2 or greater.

**Description** Use mxGetNumberOfDimensions to determine how many dimensions are in the specified array. To determine how many elements are in each dimension, call mxGetDimensions.

## Examples

Consider a MEX-file named `CountDim` that calls `mxGetNumberOfDimensions` to determine how many dimensions are in the first input argument.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int number_of_dimensions;
 int *dims;
 int c;

 /* Look at the number of dimensions in the input mxArray. */
 number_of_dimensions = mxGetNumberOfDimensions(prhs[0]);

 /* Create a dimensions array having the same number of
 dimensions as the input mxArray. Arbitrarily make the length of
 each dimension be 2. */

 dims = mxMalloc(number_of_dimensions, sizeof(int));
 for (c=0; c<number_of_dimensions; c++)
 dims[c]=2;

 /* Create an mxArray of signed 16-bit integers. */
 mxCreateNumericArray(number_of_dimensions, dims,
mxINT16_CLASS, mxREAL);
 ...
}
```

In MATLAB, create a three-dimensional `mxArray` named `td`. Then pass `td` as an argument to `CountDim`

```
>> td = rand(6, 4, 2);
>> CountDim(td)
```

Since `td` is a 3-dimensional `mxArray`, `CountDim` creates a 2-by-2-by-2 `mxArray`. If `td` had been a 4-dimensional `mxArray`, `CountDim` would have created a 2-by-2-by-2-by-2.

For an additional example, see `mxgetnumberofdimensions.c` in the `mx` subdirectory of the `examples` directory.

# mxGetNumberOfDimensions

---

## See Also

`mxSetM`, `mxSetN`

|                    |                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get number of elements in an array                                                                                                                                                                                    |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxGetNumberOfElements(const mxArray *array_ptr);</pre>                                                                                                                                   |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                        |
| <b>Returns</b>     | Number of elements in the specified <code>mxArray</code> .                                                                                                                                                            |
| <b>Description</b> | <code>mxGetNumberOfElements</code> tells you how many “pieces” an array has. Use <code>mxGetClassID</code> to find out what the pieces are. These two functions provide the highest-level information about an array. |
| <b>Example</b>     | See <code>mxgetnumberofelements.c</code> in the <code>mx</code> subdirectory of the <code>examples</code> directory.                                                                                                  |
| <b>See Also</b>    | <code>mxGetDimensions</code> , <code>mxGetM</code> , <code>mxGetN</code> , <code>mxGetClassID</code> , <code>mxGetClassName</code>                                                                                    |

# mxGetNumberOfFields

---

|                    |                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the number of fields in a structure <code>mxArray</code>                                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxGetNumberOfFields(const mxArray *array_ptr);</pre>                                                                                                                                                                                 |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to a structure <code>mxArray</code> .                                                                                                                                                                                           |
| <b>Returns</b>     | The number of fields, on success. Returns 0 on failure. The most common cause of failure is that <code>array_ptr</code> is not a structure <code>mxArray</code> . Call <code>mxIsStruct</code> to determine if <code>array_ptr</code> is a structure.             |
| <b>Description</b> | Call <code>mxGetNumberOfFields</code> to determine how many fields are in the specified structure <code>mxArray</code> .<br><br>Once you know the number of fields in a structure, it is easy to loop through every field in order to set or to get field values. |
| <b>Examples</b>    | See the example in the <code>mxGetFieldNameByNumber</code> reference page, or <code>mxgetnumberoffields.c</code> in the <code>mx</code> subdirectory of the <code>examples</code> directory.                                                                      |
| <b>See Also</b>    | <code>mxGetField</code> , <code>mxIsStruct</code> , <code>mxSetField</code>                                                                                                                                                                                       |

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the number of elements in the <code>i r</code> , <code>pr</code> , and (if it exists) <code>pi</code> arrays                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxGetNzmax(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Returns</b>     | The number of elements allocated to hold nonzero entries in the specified sparse <code>mxArray</code> , on success. Returns an indeterminate value on error. The most likely cause of failure is that <code>array_ptr</code> points to a full (nonsparse) <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | <p>Use <code>mxGetNzmax</code> to get the value of the <code>nzmax</code> field. The <code>nzmax</code> field holds an integer value that signifies the number of elements in the <code>i r</code>, <code>pr</code>, and, if it exists, the <code>pi</code> arrays. The value of <code>nzmax</code> is always greater than or equal to the number of nonzero elements in a sparse <code>mxArray</code>. In addition, the value of <code>nzmax</code> is always less than or equal to the number of rows times the number of columns.</p> <p>As you adjust the number of nonzero elements in a sparse <code>mxArray</code>, MATLAB often adjusts the value of the <code>nzmax</code> field. MATLAB adjusts <code>nzmax</code> in order to reduce the number of costly reallocations and in order to optimize its use of heap space.</p> |

# mxGetNzmax

---

## Examples

Consider a MEX-file named SparCnt that displays

- The number of nonzero elements in the mxArray.
- The value of nzmax.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int nzmax, nnz, columns;

 if (mxIsSparse(prhs[0])) {
 nzmax = mxGetNzmax(prhs[0]);
 columns = mxGetN(prhs[0]);
 nnz = *(mxGetJc(prhs[0]) + columns);
 mexPrintf("Contains %d nonzero elements.\n", nnz);
 mexPrintf("Allocates memory to hold %d elements.\n", nzmax);
 }
 else
 mexErrMsgTxt("First argument must be a sparse array.");
}
```

In MATLAB, create a sparse mxArray named sparrow. Then, pass sparrow as an argument to SparCnt:

```
>> sparrow = sparse(eye(100));
>> SparCnt(sparrow)
Contains 100 nonzero elements.
Allocates memory to hold 100 elements.
```

Adding one nonzero element to sparrow causes MATLAB to increase nzmax by 10

```
>> sparrow(50, 75)=1;
>> mxGetNzmax(sparrow)
Contains 101 nonzero elements.
Allocates memory to hold 110 elements.
```

For an additional example, see `mxgetnzmax.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxSetNzmax`

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get an mxArray's imaginary data elements                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>C Syntax</b>    | <pre>#include "matrix.h" double *mxGetPi(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>     | The imaginary data elements of the specified mxArray, on success. Returns NULL if there is no imaginary data or if there is an error.                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | <p>The <code>pi</code> field points to an array containing the imaginary data of the mxArray. Call <code>mxGetPi</code> to get the contents of the <code>pi</code> field; that is, to get the starting address of this imaginary data.</p> <p>The best way to determine if an mxArray is purely real is to call <code>mxIsComplex</code>.</p> <p>The imaginary parts of all input matrices to a MATLAB function are allocated if any of the input matrices are complex.</p> |

# mxGetPi

---

## Examples

Consider a MEX-file named `MyImag` that displays the contents of the imaginary component of the {3, 1, 2} element of a three-dimensional `mxArray`

```
#include "mex.h"

void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double *imag_data_ptr, imag_value;
 int nsubs=3, subs[]={3, 1, 2}, index;

 /* The input array must be mxDOUBLE_CLASS. */
 if (mxIsDouble(prhs[0])) {
 /* Get starting address of imaginary data in input array. */
 imag_data_ptr = (double *)mxGetPi(prhs[0]);

 /* Get index to {3, 1, 2}. */
 index = mxCalcSingleSubscript(prhs[0], nsubs, subs);

 /* Get the imaginary component at {3, 1, 2} */
 imag_value = *(imag_data_ptr + index);
 mexPrintf("%g\n", imag_value);
 }
 else
 mexErrMsgTxt("First argument must be a double array.");
}
```

In MATLAB, create a three-dimensional `mxArray` named `t` containing imaginary data parts:

```
>> t=sqrt(randn(4, 4, 6));
```

Call `MyImag`, passing `t` as an argument

```
>> MyImag(t)
0.395875
```

For an additional example, see `mxgetpi.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetPr`, `mxSetPi`, `mxSetPr`

---

|                    |                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get an mxArray's real data elements                                                                                                                                                                        |
| <b>C Syntax</b>    | <pre>#include "matrix.h" double *mxGetPr(const mxArray *array_ptr);</pre>                                                                                                                                  |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                        |
| <b>Returns</b>     | The address of the first element of the real data. Returns NULL if there is no real data.                                                                                                                  |
| <b>Description</b> | Call mxGetPr to determine the starting address of the real data in the mxArray that array_ptr points to. Once you have the starting address, it is fairly easy to access any other element in the mxArray. |

## Examples

Consider a MEX-file named `DisplayReal` that displays the value of every real element in the input `mxArray`

```
#include "mex.h"

void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 int c, total_num_of_elements;
 double *real_data_ptr;

 if (mxIsDouble(prhs[0])) {
 /* Get starting address of real data in input array. */
 real_data_ptr = (double *)mxGetPr(prhs[0]);

 /* Using pointer auto-increment, display every element in
 the array. */
 total_num_of_elements = mxGetM(prhs[0]) * mxGetN(prhs[0]);

 /* Display the contents of every real value. */
 for (c = 0; c < total_num_of_elements; c++)
 mexPrintf("%g\n", *real_data_ptr++);
 }
 else
 mexErrMsgTxt("First argument must be a double array.");
}
```

In MATLAB, create Values:

```
>> Values = [2 3; 5 7]
Values =
 2 3
 5 7
```

Pass `Values` as an argument to `DispReal`

```
>> DispReal (Values)
2
5
3
7
```

For additional examples, see `mxgetpr.c` and `mxgetpr2.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxGetPi`, `mxSetPi`, `mxSetPr`

# mxGetScalar

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Get the real component of an <code>mxArray</code> 's first data element                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" double mxGetScalar(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> other than a cell <code>mxArray</code> or a structure <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Returns</b>     | <p>The value of the first real (nonimaginary) element of the <code>mxArray</code>. Notice that <code>mxGetScalar</code> returns a <code>double</code>. Therefore, if real elements in the <code>mxArray</code> are stored as something other than <code>double</code>s, <code>mxGetScalar</code> automatically converts the scalar value into a <code>double</code>. To preserve the original data representation of the scalar, you must cast the return value to the desired data type.</p> <p>If <code>array_ptr</code> points to a structure <code>mxArray</code> or a cell <code>mxArray</code>, <code>mxGetScalar</code> returns 0.0.</p> <p>If <code>array_ptr</code> points to a sparse <code>mxArray</code>, <code>mxGetScalar</code> returns the value of the first nonzero real element in the <code>mxArray</code>.</p> <p>If <code>array_ptr</code> points to an empty <code>mxArray</code>, <code>mxGetScalar</code> returns an indeterminate value.</p> |
| <b>Description</b> | <p>Call <code>mxGetScalar</code> to get the value of the first real (nonimaginary) element of the <code>mxArray</code>.</p> <p>In most cases, you call <code>mxGetScalar</code> when <code>array_ptr</code> points to an <code>mxArray</code> containing only one element (a scalar). However, <code>array_ptr</code> can point to an <code>mxArray</code> containing many elements. If <code>array_ptr</code> points to an <code>mxArray</code> containing multiple elements, <code>mxGetScalar</code> returns the value of the first real element. If <code>array_ptr</code> points to a two-dimensional <code>mxArray</code>, <code>mxGetScalar</code> returns the value of the (1, 1) element; if <code>array_ptr</code> points to a three-dimensional <code>mxArray</code>, <code>mxGetScalar</code> returns the value of the (1, 1, 1) element; and so on.</p>                                                                                                   |

## Examples

Get the first real element of the first two input arguments to a MEX-file:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double interest_rate, starting_principal;

 interest_rate = mxGetScalar(prhs[0]);
 starting_principal = mxGetScalar(prhs[1]);
 ...
}
```

`mxGetScalar` does not return imaginary data. The easiest way to get an imaginary scalar is to dereference the pointer returned by `mxGetPi`. For example, consider the MEX-file entitled `ImagScalar`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double imag_scalar;
 double *ptr_to_start_of_imag_data;

 ptr_to_start_of_imag_data = mxGetPi(prhs[0]);
 imag_scalar = *ptr_to_start_of_imag_data;
 mexPrintf("The first imag element is %g\n", imag_scalar);
 ...
}
```

Given an `mxArray` named `b` containing some imaginary elements

```
>> b = sqrt([-5 -3; 7 -8]);
>> ImagScalar(b)
The first imaginary element is 2.23607
```

For additional examples, see `mxgetscalar.c` and `mxgetscalar2.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetM`, `mxGetN`

# mxGetString

---

**Purpose** Copy a string `mxArray`'s data into a C-style string

**C Syntax**

```
#include "matrix.h"
int mxGetString(const mxArray *array_ptr, char *buf, int buflen);
```

**Arguments**

`array_ptr`  
Pointer to a string `mxArray`; that is, a pointer to an `mxArray` having the `mxCHAR_CLASS` class.

`buf`  
The starting location into which the string should be written. `mxGetString` writes the character data into `buf` and then terminates the string with a NULL character (in the manner of C strings). `buf` can either point to dynamic or static memory.

`buflen`  
Maximum number of characters to read into `buf`. Typically, you set `buflen` to 1 plus the number of elements in the string `mxArray` to which `array_ptr` points. (See the `mxGetM` and `mxGetN` reference pages to find out how to get the number of elements.)

**Note:** Users of multibyte character sets should be aware that MATLAB packs multibyte characters into an `mxChar` (16-bit unsigned integer). When allocating space for the return string, to avoid possible truncation you should set

```
buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0]) * sizeof(mxChar)) + 1
```

**Returns** 0 on success, and 1 on failure. Possible reasons for failure include

- Specifying an `mxArray` that is not a string `mxArray`.
- Specifying `buflen` with less than the number of characters needed to store the entire `mxArray` pointed to by `array_ptr`. If this is the case, 1 is returned and the string is truncated.

**Description** Call `mxGetString` to copy the character data of a string `mxArray` into a C-style string. The copied C-style string starts at `buf` and contains no more than `buflen - 1` characters. The C-style string is always terminated with a NULL character.

If the string array contains several rows, they are copied, one column at a time, into one long string array.

## Examples

Use `mxGetString` to convert the data from a string array into a C string named `buf`:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 char *buf;
 int buflen;
 int status;

 /* Find out how long the input string array is. Note: Users of
 multibyte character sets should uncomment and use the second
 buflen statement that follows and should comment out the first
 buflen statement that follows. */

 buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0])) + 1;
 /* buflen = (mxGetM(prhs[0])*mxGetN(prhs[0])*sizeof(mxChar))+1 */

 /* Allocate enough memory to hold the converted string. */
 buf = mxMalloc(buflen, sizeof(char));
 if (buf == NULL)
 mexErrMsgTxt("Not enough heap space to hold converted
string.");

 /* Copy the string data from prhs[0] and place it into buf. */
 status = mxGetString(prhs[0], buf, buflen);
 if (status == 0)
 mexPrintf("The converted string is \n%s.\n", buf);
 else
 mexErrMsgTxt("Could not convert string data.");

 /* Manipulate buf as you would manipulate any C string. */
 ...
}
```

For an additional example, see `mxgetstring.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCreateCharArray`, `mxCreateCharMatrixFromStrings`, `mxCreateString`

# mxIsCell

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if a cell mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsCell(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | <pre>array_ptr</pre> Pointer to an array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Returns</b>     | true if array_ptr points to an array having the class mxCELL_CLASS; otherwise, returns false.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | Use mxIsCell to determine if the specified array is a cell array.<br>Do not confuse a cell array with a cell element. Remember that a cell array contains various cell elements, and that most cell elements are not cell arrays.                                                                                                                                                                                                                                                                                            |
| <b>Examples</b>    | Consider a MEX-file that expects its first input argument to be a cell array. If it is not a cell array, the MEX-file issues an error message.<br><pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {     if (mxIsCell(prhs[0])) {         /* ... Yes, it is a cell array. */     }     else /* No, this is not a cell array. */         mexErrMsgTxt("You must specify a cell array."); }</pre> For an additional example, see mxiscell.c in the mx subdirectory of the examples directory. |
| <b>See Also</b>    | mxIsClass                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if a string mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsChar(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>   | <p>array_ptr<br/>Pointer to an mxArray.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>     | true if array_ptr points to an array having the class mxCHAR_CLASS; otherwise, returns false.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b> | <p>Use mxIsChar to determine if array_ptr points to string mxArray.<br/>Calling mxIsChar is equivalent to calling</p> <pre>mxGetClassID(array_ptr) == mxCHAR_CLASS</pre>                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Example</b>     | <p>Consider a MEX-file that expects its first input argument to be a string mxArray. If it is not a string mxArray, the MEX-file issues an error message.</p> <pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {     if (mxIsChar(prhs[0])) {         /* ... Yes, it is a string mxArray. */     }     else /* No, this is not a string array. */         mexErrMsgTxt("You must specify a string array"); }</pre> <p>For an additional example, see mxischar.c in the mx subdirectory of the examples directory.</p> |
| <b>See Also</b>    | mxIsClass, mxGetClassID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

# mxIsClass

---

**Purpose** True if mxArray is a member of the specified class

**C Syntax**

```
#include "matrix.h"
bool mxIsClass(const mxArray *array_ptr, const char *name);
```

**Arguments**

**array\_ptr**  
Pointer to an array.

**name**  
The array category that you are testing. Specify name as a string (not as an enumerated constant). You can specify any one of the predefined constants, which are:

| Value of Name | Corresponding Class |
|---------------|---------------------|
| "double"      | mxDOUBLE_CLASS      |
| "sparse"      | mxSPARSE_CLASS      |
| "char"        | mxCHAR_CLASS        |
| "cell"        | mxCELL_CLASS        |
| "struct"      | mxSTRUCT_CLASS      |
| "single"      | mxSINGLE_CLASS      |
| "int8"        | mxINT8_CLASS        |
| "uint8"       | mxUINT8_CLASS       |
| "int16"       | mxINT16_CLASS       |
| "uint16"      | mxUINT16_CLASS      |
| "int32"       | mxINT32_CLASS       |
| "uint32"      | mxUINT32_CLASS      |
| "object"      | mxOBJECT_CLASS      |
| "unknown"     | mxUNKNOWN_CLASS     |

Or, you can specify one of your own class names.

- Returns** true if `array_ptr` points to an array having category name; otherwise, returns false.
- Description** Each `mxArray` is tagged as being a certain type. Call `mxIsClass` to determine if the specified `mxArray` has this type.
- Example** See `mxIsClass.c` in the `mx` subdirectory of the `examples` directory.
- See Also** `mxIsEmpty`, `mxGetClassID`, `mxClassID`

# mxIsComplex

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if data is complex                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsComplex(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Returns</b>     | true if array_ptr is a numeric array containing complex data; otherwise, returns false. If array_ptr points to a cell array or a structure array, then mxIsComplex returns false.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Use mxIsComplex to determine whether or not an imaginary part is allocated for an mxArray. The imaginary pointer pi is NULL if an mxArray is purely real and does not have any imaginary data. If an mxArray is complex, pi points to an array of numbers.</p> <p>When a MEX-file is called, MATLAB automatically examines all the input (right-hand side) arrays. If any input array is complex, then MATLAB automatically allocates memory to hold imaginary data for all other input arrays. For example, suppose a user passes three input variables (apricot, banana, and carambola) to a MEX-file named Jest:</p> <pre>&gt;&gt; apricot = 7; &gt;&gt; banana = sqrt(-5:5); &gt;&gt; carambola = magic(2); &gt;&gt; Jest(apricot, banana, carambola);</pre> <p>banana is complex. Therefore, even though array apricot is purely real, MATLAB automatically allocates space (one element) to hold an imaginary value of apricot. MATLAB also automatically allocates space (four-elements) to hold the nonexistent imaginary values of carambola.</p> <p>In other words, MATLAB forces every input array to be real or every input array to be complex.</p> |
| <b>Example</b>     | See mxIsComplex.c in the mx subdirectory of the examples directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>See Also</b>    | mxIsNumeric                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as double-precision, floating-point numbers                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsDouble(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>     | true if the <code>mxArray</code> stores its data as double-precision, floating-point numbers; otherwise, returns false.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Call <code>mxIsDouble</code> to determine whether or not the specified <code>mxArray</code> represents its real and imaginary data as double-precision, floating-point numbers.</p> <p>Older versions of MATLAB store all <code>mxArray</code> data as double-precision, floating-point numbers. However, starting with MATLAB 5, MATLAB can store real and imaginary data in a variety of numerical formats.</p> <p>Calling <code>mxIsDouble</code> is equivalent to calling</p> <pre>mxGetClassID(array_ptr == mxDOUBLE_CLASS)</pre> |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

# mxIsEmpty

---

**Purpose** True if mxArray is empty

**C Syntax**

```
#include "matrix.h"
bool mxIsEmpty(const mxArray *array_ptr);
```

**Arguments** `array_ptr`  
Pointer to an array.

**Returns** true if the mxArray is empty; otherwise, returns false.

**Description** Use `mxIsEmpty` to determine if an mxArray is empty. An mxArray is empty if the size of any of its dimensions is 0.

Attempts to access empty mxArray cause undesirable behavior. To avoid accessing empty arrays, test them by calling `mxIsEmpty`.

Note that `mxIsEmpty` is not the opposite of `mxIsFull`.

**Examples** `mxGetScalar` returns an indeterminate value if passed an empty mxArray. To avoid this situation, call `mxIsEmpty` first:

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 double value;

 /* If array is empty, it would not be good to access it. */
 if (mxIsEmpty(prhs[0]))
 mexErrMsgTxt("You cannot pass an empty array.\n");
 else {
 value = mxGetScalar(prhs[0]);
 mexPrintf("First real value is %g\n", value);
 }
}
```

For an additional example, see `mxisempty.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxIsClass`

|                    |                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if value is finite                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsFinite(double value);</pre>                                                                                                                                                                 |
| <b>Arguments</b>   | value<br>The double-precision, floating-point number that you are testing.                                                                                                                                                    |
| <b>Returns</b>     | true if value is finite; otherwise, returns false.                                                                                                                                                                            |
| <b>Description</b> | Call <code>mxIsFinite</code> to determine whether or not <code>value</code> is finite. A number is finite if it is not equal to <ul style="list-style-type: none"><li>• <code>Inf</code></li><li>• <code>NaN</code></li></ul> |
| <b>See Also</b>    | <code>mxIsInf</code> , <code>mxIsNaN</code>                                                                                                                                                                                   |

# mxIsFromGlobalWS

---

|                    |                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if the mxArray was copied from MATLAB's global workspace                                                                                                                                                                                                                                     |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsFromGlobalWS(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                   |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                                                                               |
| <b>Returns</b>     | true if the array was copied out of the global workspace; otherwise, returns false.                                                                                                                                                                                                               |
| <b>Description</b> | mxIsFromGlobalWS is useful for stand-alone mat and engine programs. mexIsGlobal tells you if the pointer you pass actually points into the global workspace.                                                                                                                                      |
| <b>Example</b>     | <pre>bool f; . . . f = mexIsGlobal(prhs[0]); if (f) {     other_array = mxDuplicateArray(prhs[0]);     f = mxIsFromGlobalWS(other_array);     if (f)         mexPrintf("The copy was made from the global WS.");     else         mexPrintf("The copy was not made from the global WS."); }</pre> |
| <b>See Also</b>    | mexIsGlobal                                                                                                                                                                                                                                                                                       |

**V4 Compatible** This function is obsolete; MATLAB 5 does not support it. To use this function in existing code, use the `-V4` option of the `mex` script.

In MATLAB 5 MEX-files, you should call

```
if(!mxIsSparse(prhs[0]))
```

instead of

```
if(mxIsFull(prhs[0]))
```

**See Also** `mxIsSparse`

# mxIsInf

---

**Purpose** True if value is infinite

**C Syntax**

```
#include "matrix.h"
bool mxIsInf(double value);
```

**Arguments** `value`  
The double-precision, floating-point number that you are testing.

**Returns** `true` if value is infinite; otherwise, returns `false`.

**Description** Call `mxIsInf` to determine whether or not `value` is equal to infinity. MATLAB stores the value of infinity in a permanent variable named `inf`, which represents IEEE arithmetic positive infinity. The value of `inf` is built into the system; you cannot modify it.

Operations that return infinity include

- Division by 0. For example, `5/0` returns infinity.
- Operations resulting in overflow. For example, `exp(10000)` returns infinity because the result is too large to be represented on your machine.

If `value` equals NaN (Not-a-Number), then `mxIsInf` returns `false`. In other words, NaN is not equal to infinity.

**Example** See `mxIsInf.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxIsFinite`, `mxIsNaN`

---

|                    |                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as signed 8-bit integers                                                                                                                                                                                  |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsInt8(const mxArray *array_ptr);</pre>                                                                                                                                                                                    |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                             |
| <b>Returns</b>     | true if the array stores its data as signed 8-bit integers; otherwise, returns false.                                                                                                                                                                      |
| <b>Description</b> | Use <code>mxIsInt8</code> to determine whether or not the specified array represents its real and imaginary data as 8-bit signed integers.<br>Calling <code>mxIsInt8</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxINT8_CLASS</pre> |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code>                                                                                                                                                                                                         |

# mxIsInt16

---

|                    |                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if mxArray represents its data as signed 16-bit integers                                                                                                                                                                            |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsInt16(const mxArray *array_ptr);</pre>                                                                                                                                                                 |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                      |
| <b>Returns</b>     | true if the array stores its data as signed 16-bit integers; otherwise, returns false.                                                                                                                                                   |
| <b>Description</b> | Use mxIsInt16 to determine whether or not the specified array represents its real and imaginary data as 16-bit signed integers.<br><br>Calling mxIsInt16 is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxINT16_CLASS</pre> |
| <b>See Also</b>    | mxIsClass, mxGetClassID                                                                                                                                                                                                                  |

---

|                    |                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as signed 32-bit integers                                                                                                                                                                                         |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsInt32(const mxArray *array_ptr);</pre>                                                                                                                                                                                           |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                     |
| <b>Returns</b>     | true if the array stores its data as signed 32-bit integers; otherwise, returns false.                                                                                                                                                                             |
| <b>Description</b> | Use <code>mxIsInt32</code> to determine whether or not the specified array represents its real and imaginary data as 32-bit signed integers.<br><br>Calling <code>mxIsInt32</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxINT32_CLASS</pre> |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code>                                                                                                                                                                                                                 |

# mxIsLogical

---

|                    |                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> is Boolean                                                                                                                                                                                                                                                                                    |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsLogical(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                 |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                             |
| <b>Returns</b>     | true if the <code>mxArray</code> 's logical flag is on; otherwise, returns false. If an <code>mxArray</code> does not hold numerical data (for instance, if <code>array_ptr</code> points to a structure <code>mxArray</code> or a cell <code>mxArray</code> ), then <code>mxIsLogical</code> automatically returns False. |
| <b>Description</b> | Use <code>mxIsLogical</code> to determine whether MATLAB treats the data in the <code>mxArray</code> as Boolean (logical) or numerical (not logical).<br><br>If an <code>mxArray</code> is logical, then MATLAB treats all zeros as meaning false and all nonzero values as meaning true.                                  |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxSetLogical</code>                                                                                                                                                                                                                                                                         |

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if value is Not-a-Number                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsNaN(double value);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>   | <p>value</p> <p>The double-precision, floating-point number that you are testing.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Returns</b>     | true if value is Not-a-Number; otherwise, returns false.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p>Call <code>mxIsNaN</code> to determine whether or not <code>value</code> is equal to NaN. NaN is the IEEE arithmetic representation for Not-a-Number. A NaN is obtained as a result of mathematically undefined operations such as</p> <ul style="list-style-type: none"><li>• <code>0.0/0.0</code></li><li>• <code>inf-inf</code></li></ul> <p>The system understands a family of bit patterns as being equivalent to NaN. In other words, NaN is not a single value, rather it is a family of numbers that MATLAB (and other IEEE-compliant applications) interpret as being equal to Not-a-Number.</p> |
| <b>See Also</b>    | <code>mxIsFinite</code> , <code>mxIsInf</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

# mxIsNumeric

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if mxArray is numeric                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsNumeric(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Returns</b>     | True if the array's storage type is <ul style="list-style-type: none"><li>• mxDOUBLE_CLASS</li><li>• mxSPARSE_CLASS</li><li>• mxSINGLE_CLASS</li><li>• mxINT8_CLASS</li><li>• mxUINT8_CLASS</li><li>• mxINT16_CLASS</li><li>• mxUINT16_CLASS</li><li>• mxINT32_CLASS</li><li>• mxUINT32_CLASS</li></ul> Returns False if the array's storage type is <ul style="list-style-type: none"><li>• mxCELL_CLASS</li><li>• mxCHAR_CLASS</li><li>• mxOBJECT_CLASS</li><li>• mxSTRUCT_CLASS</li><li>• mxUNKNOWN_CLASS</li></ul> |
| <b>Description</b> | Call mxIsNumeric to determine if the specified array contains numeric data. If the specified array is a cell, string, or a structure, then mxIsNumeric returns False. Otherwise, mxIsNumeric returns True.<br><br>Call mxGetClassID to determine the exact storage type.                                                                                                                                                                                                                                               |
| <b>See Also</b>    | mxGetClassID                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

|                    |                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as single-precision, floating-point numbers                                                                                                                                                                                            |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsSingle(const mxArray *array_ptr);</pre>                                                                                                                                                                                                               |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                          |
| <b>Returns</b>     | true if the array stores its data as single-precision, floating-point numbers; otherwise, returns false.                                                                                                                                                                                |
| <b>Description</b> | Use <code>mxIsSingle</code> to determine whether or not the specified array represents its real and imaginary data as single-precision, floating-point numbers.<br><br>Calling <code>mxIsDouble</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxSINGLE_CLASS</pre> |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code>                                                                                                                                                                                                                                      |

# mxIsSparse

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if a sparse mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsSparse(const mxArray *array_ptr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>     | 1 if array_ptr points to a sparse mxArray; otherwise, returns 0. A return value of 0 means that array_ptr points to a full mxArray or that array_ptr does not point to a legal mxArray.                                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b> | Use mxIsSparse to determine if array_ptr points to a sparse mxArray. Many routines (for example, mxGetIr and mxGetJc) require a sparse mxArray as input.                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Examples</b>    | <p>Consider a MEX-file that requires the first input argument to be a sparse mxArray. The first part of the MEX-file might be</p> <pre>void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {     if (mxIsSparse(prhs[0])) {         /* Yes, this is a sparse mxArray. */         ...     }     else         mexErrMsgTxt("First argument must be a sparse array.");</pre> <p>For an additional example, see <code>mxissparse.c</code> in the <code>mx</code> subdirectory of the <code>examples</code> directory.</p> |
| <b>See Also</b>    | <code>mxGetIr</code> , <code>mxGetJc</code> , <code>mxIsFull</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

**V4 Compatible** This function is obsolete; MATLAB 5 does not support it. To use this function in existing code, use the `-V4` option of the `mex` script.

Use `mxIsChar` rather than `mxIsString`.

**See Also** `mxChar`, `mxIsChar`

# mxIsStruct

---

|                    |                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if a structure mxArray                                                                                                                                                    |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsStruct(const mxArray *array_ptr);</pre>                                                                                                      |
| <b>Arguments</b>   | array_ptr<br>Pointer to an mxArray.                                                                                                                                            |
| <b>Returns</b>     | true if array_ptr points to a structure array; otherwise, returns false.                                                                                                       |
| <b>Description</b> | Use mxIsStruct to determine if array_ptr points to a structure mxArray. Many routines (for example, mxGetFieldName and mxSetField) require a structure mxArray as an argument. |
| <b>See Also</b>    | mxCreateStructArray, mxCreateStructMatrix, mxGetNumberOfFields, mxGetField, mxSetField                                                                                         |

---

|                    |                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as unsigned 8-bit integers                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsInt8(const mxArray *array_ptr);</pre>                                                                                                                                                                                                           |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                    |
| <b>Returns</b>     | true if the <code>mxArray</code> stores its data as unsigned 8-bit integers; otherwise, returns false.                                                                                                                                                                            |
| <b>Description</b> | Use <code>mxIsInt8</code> to determine whether or not the specified <code>mxArray</code> represents its real and imaginary data as 8-bit unsigned integers.<br><br>Calling <code>mxIsUint8</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxUINT8_CLASS</pre> |
| <b>See Also</b>    | <code>mxGetClassID</code> , <code>mxIsClass</code> , <code>mxIsInt8</code> , <code>mxIsInt16</code> , <code>mxIsInt32</code> , <code>mxIsUint16</code> , <code>mxIsUint32</code>                                                                                                  |

# mxIsUint16

---

|                    |                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as unsigned 16-bit integers                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsUint16(const mxArray *array_ptr);</pre>                                                                                                                                                                                                              |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                         |
| <b>Returns</b>     | true if the <code>mxArray</code> stores its data as unsigned 16-bit integers; otherwise, returns false.                                                                                                                                                                                |
| <b>Description</b> | Use <code>mxIsUint16</code> to determine whether or not the specified <code>mxArray</code> represents its real and imaginary data as 16-bit unsigned integers.<br><br>Calling <code>mxIsUint16</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxUINT16_CLASS</pre> |
| <b>See Also</b>    | <code>mxGetClassID</code> , <code>mxIsClass</code> , <code>mxIsInt8</code> , <code>mxIsInt16</code> , <code>mxIsInt32</code> , <code>mxIsUint16</code> , <code>mxIsUint32</code>                                                                                                       |

|                    |                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | True if <code>mxArray</code> represents its data as unsigned 32-bit integers                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" bool mxIsUint32(const mxArray *array_ptr);</pre>                                                                                                                                                                                                              |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                         |
| <b>Returns</b>     | true if the <code>mxArray</code> stores its data as unsigned 32-bit integers; otherwise, returns false.                                                                                                                                                                                |
| <b>Description</b> | Use <code>mxIsUint32</code> to determine whether or not the specified <code>mxArray</code> represents its real and imaginary data as 32-bit unsigned integers.<br><br>Calling <code>mxIsUint32</code> is equivalent to calling<br><pre>mxGetClassID(array_ptr) == mxUINT32_CLASS</pre> |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code> , <code>mxIsUint16</code> , <code>mxIsUint8</code> , <code>mxIsInt32</code> , <code>mxIsInt16</code> , <code>mxIsInt8</code>                                                                                                        |

# mxMalloc

---

**Purpose** Allocate dynamic memory using MATLAB's memory manager

**C Syntax**

```
#include "matrix.h"
#include <stdlib.h>
void *mxMalloc(size_t n);
```

**Arguments**

n  
Number of bytes to allocate.

**Returns** A pointer to the start of the allocated dynamic memory, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, `mxMalloc` returns NULL. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt.

`mxMalloc` is unsuccessful when there is insufficient free heap space.

**Description** MATLAB applications should always call `mxMalloc` rather than `malloc` to allocate memory. Note that `mxMalloc` works differently in MEX-files than in stand-alone MATLAB applications.

In MEX-files, `mxMalloc` automatically

- Allocates enough contiguous heap space to hold `n` bytes.
- Registers the returned heap space with the MATLAB memory management facility.

The MATLAB memory management facility maintains a list of all memory allocated by `mxMalloc`. The MATLAB memory management facility automatically frees (deallocates) all of a MEX-file's parcels when control returns to the MATLAB prompt.

In stand-alone MATLAB applications, `mxMalloc` defaults to calling the ANSI C `malloc` function. If this default behavior is unacceptable, you can write your own memory allocation routine, and then register this routine with `mxSetAllocFns`. Then, whenever `mxMalloc` is called, `mxMalloc` calls your memory allocation routine instead of `malloc`.

By default, in a MEX-file, `mxMalloc` generates nonpersistent `mxMalloc` data. In other words, the memory management facility automatically deallocates the memory as soon as the MEX-file ends. If you want the memory to persist after

the MEX-file completes, call `mxMakeMemoryPersistent` after calling `mxMalloc`. If you write a MEX-file with persistent memory, be sure to register a `mxAtExit` function to free allocated memory in the event your MEX-file is cleared.

When you finish using the memory allocated by `mxMalloc`, call `mxFree`. `mxFree` deallocates the memory.

## Examples

This example uses `mxMalloc` to allocate enough heap space to hold a string.

```
void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 char *buf;
 int buflen;
 int status;

 if (mxIsString(prhs[0])) {
 /* Find out how long the input string array is. */
 buflen = mxGetN(prhs[0])+1;
 /* Allocate enough memory to hold the converted string. */
 buf = mxMalloc(buflen);
 /* Copy the string data into buf. */
 status = mxGetString(prhs[0], buf, buflen);
 /* Manipulate the string. */
 ...

 /* When finished using the string, deallocate it. */
 mxFree(buf);
 }
 else
 mexErrMsgTxt("Input argument must be a string.");
}
```

For an additional example, see `mxmalloc.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCalloc`, `mxFree`, `mxDestroyArray`, `mxMakeArrayPersistent`,  
`mxMakeMemoryPersistent`, `mxSetAlllocFns`

# mxRealloc

---

**Purpose** Reallocate memory

**C Syntax**

```
#include "matrix.h"
#include <stdlib.h>
void *mxRealloc(void *ptr, size_t size);
```

**Arguments**

`ptr`  
Pointer to a block of memory allocated by `mxCalloc`, or by a previous call to `mxRealloc`.

`size_t`  
New size of allocated memory, in bytes.

**Description** `mxRealloc` reallocates the memory routine for the managed list. If `mxRealloc` fails to allocate a block, you must free the block since the ANSI definition of `realloc` states that the block remains allocated. `mxRealloc` returns `NULL` in this case, and in subsequent calls to `mxRealloc` of the form:

```
x = mxRealloc(x, size);
```

**Note:** This will cause memory leaks if `mxRealloc` fails and returns `NULL`.

**See Also** `mxCalloc`, `mxFree`, `mxMalloc`, `mxSetAllocFns`

**Purpose** Register your own memory allocation and deallocation functions in a stand-alone engine or MAT application

**C Syntax**

```
#include "matrix.h"
#include <stdlib.h>
void mxSetAllocFcns(calloc_proc callocfcn, free_proc freefcn,
 realloc_proc reallocfcn, malloc_proc mallocfcn);
```

**Arguments**

**callocfcn**  
The name of the function that `mxCalloc` uses to perform memory allocation operations. The function you specify is ordinarily a wrapper around the ANSI C `calloc` function. The `callocfcn` you write must have the prototype:

```
void * callocfcn(size_t nmemb, size_t size);
```

**nmemb** The number of contiguous elements that you want the matrix library to allocate on your behalf.

**size** The size of each element. To get the size, you typically use the `sizeof` operator or the `mxGetElementSize` routine.

The `callocfcn` you specify must create memory in which all allocated memory has been initialized to zero.

**freefcn**  
The name of the function that `mxFree` uses to perform memory deallocation (freeing) operations. The `freefcn` you write must have the prototype:

```
void freefcn(void *ptr);
```

**ptr** Pointer to beginning of the memory parcel to deallocate.

The `freefcn` you specify must contain code to determine if `ptr` is `NULL`. If `ptr` is `NULL`, then your `freefcn` must not attempt to deallocate it.

# mxSetAllocFcns

---

`reallocfcn`

The name of the function that `mxRealloc` uses to perform memory reallocation operations. The `reallocfcn` you write must have the prototype:

```
void * reallocfcn(void *ptr, size_t size);
```

`ptr`            Pointer to beginning of the memory parcel to reallocate.

`size`           The size of each element. To get the size, you typically use the `sizeof` operator or the `mxGetElementSize` routine.

`mallocfcn`

The name of the function the API functions should call in place of `malloc` to perform memory reallocation operations. The `mallocfcn` you write must have the prototype:

```
void * mallocfcn(size_t n);
```

`n`              The number of bytes to allocate.

The `mallocfcn` you specify doesn't necessarily need to initialize the memory it allocates.

## Description

Call `mxSetAllocFcns` to establish your own memory allocation and deallocation routines in a stand-alone (nonMEX) application.

It is illegal to call `mxSetAllocFcns` from a MEX-file; doing so causes a compiler error.

In a stand-alone application, if you do not call `mxSetAllocFcns`, then

- `mxCalloc` simply calls the ANSI C `calloc` routine.
- `mxFree` calls a free function, which calls the ANSI C `free` routine if a NULL pointer is not passed.
- `mxRealloc` simply calls the ANSI C `realloc` routine.

Writing your own `callocfcn`, `mallocfcn`, `freefcn`, and `reallocfcn` allows you to customize memory allocation and deallocation.

## Example

See `mxsetallocfcns.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxCalloc`, `mxFree`, `mxMalloc`, `mxRealloc`

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the value of one cell                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetCell(mxArray *array_ptr, int index, mxArray *value);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>   | <p><b>array_ptr</b><br/>Pointer to a cell mxArray.</p> <p><b>index</b><br/>Index from the beginning of the mxArray. Specify the number of elements between the first cell of the mxArray and the cell you wish to set. The easiest way to calculate index is to call <code>mxCalcSingleSubscript</code>.</p> <p><b>value</b><br/>The new value of the cell. You can put any kind of mxArray into a cell. In fact, you can even put another cell mxArray into a cell.</p>                                                                                                                                                                                                              |
| <b>Description</b> | <p>Call <code>mxSetCell</code> to put the designated value into a particular cell of a cell mxArray. Use <code>mxSetCell</code> to assign new values to unpopulated cells or to overwrite the value of an existing cell.</p> <p>If the specified cell is already occupied, then <code>mxSetCell</code> assigns the new value. However, the old cell value remains in memory until you call <code>mxDestroyArray</code>.</p> <p>Note: Inputs to a MEX-file are constant read-only mxArrays and should not be modified. Using <code>mxSetCell*</code> or <code>mxSetField*</code> to modify the cells or fields of an argument passed from MATLAB will cause unpredictable results.</p> |
| <b>Examples</b>    | For an example of using <code>mxSetCell</code> to populate cells in a freshly created cell mxArray, see <code>mxCreateCharArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## mxSetCell

---

Consider a function that calls `mxSetCell` to change the value already held in the first cell element.

```
void
modify_first_cell(mxArray *cell_array_ptr, mxArray *new_value)
{
 int index_of_first_cell=0;
 mxArray *old_value;

 /* Get pointer to old cell. */
 old_value = mxGetCell(cell_array_ptr, index_of_first_cell);

 /* Free the memory that the old_value was using. */
 mxDestroyArray(old_value);

 /* Assign the new value to the first cell. */
 mxSetCell(cell_array_ptr, index_of_first_cell, new_value);
}
```

For an additional example, see `mxsetcell.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxCreateCellArray`, `mxCreateCellMatrix`, `mxGetCell`, `mxIsCell`

|                    |                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Convert a MATLAB structure array to a MATLAB object array by specifying a class name to associate with the object                                                                                                                                                                                                                                                                 |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxSetClassName(mxArray *array_ptr, const char *classname);</pre>                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to an <code>mxArray</code> of class <code>mxSTRUCT_CLASS</code>.</p> <p><code>classname</code><br/>The object class to which to convert <code>array_ptr</code>.</p>                                                                                                                                                                         |
| <b>Returns</b>     | 0 if successful; nonzero otherwise.                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <code>mxSetClassName</code> converts a structure array to an object array, to be saved subsequently to a MAT-file. The object is not registered or validated by MATLAB until it is loaded via the <code>LOAD</code> command. If the specified <code>classname</code> is an undefined class within MATLAB, <code>LOAD</code> converts the object back to a simple structure array. |
| <b>See Also</b>    | <code>mxIsClass</code> , <code>mxGetClassID</code>                                                                                                                                                                                                                                                                                                                                |

# mxSetData

---

**Purpose** Set pointer to data

**C Syntax**

```
#include "matrix.h"
void mxSetData(mxArray *array_ptr, void *data_ptr);
```

**Arguments**

array\_ptr  
Pointer to an mxArray.

data\_ptr  
Pointer to data.

**Description** mxSetData is similar to mxSetPr, except it returns a void \*. Use this on numeric arrays with contents other than double.

**See Also** mxSetPr

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Modify the number of dimensions and/or the size of each dimension                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>C Syntax</b>    | <pre>#include "matrix.h" int mxSetDimensions(mxArray *array_ptr, const int *size, int ndims);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to an <code>mxArray</code>.</p> <p><code>ndims</code><br/>The desired number of dimensions.</p> <p><code>dims</code><br/>The first element of an array of <code>ints</code>. Each element in the array holds the size of a particular dimension. The first element holds the number of elements in the first dimension (rows); the second element holds the number of elements in the second dimension, and so on.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Returns</b>     | 0 on success, and 1 on failure. <code>mxSetDimensions</code> allocates heap space to hold the input size array. So it is possible (though extremely unlikely) that increasing the number of dimensions can cause the system to run out of heap space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | <p>Call <code>mxSetDimensions</code> to reshape an existing <code>mxArray</code>. <code>mxSetDimensions</code> is similar to <code>mxSetM</code> and <code>mxSetN</code>; however, <code>mxSetDimensions</code> provides greater control for reshaping <code>mxArrays</code> that have more than two-dimensions.</p> <p><code>mxSetDimensions</code> does not allocate or deallocate any space for the <code>pr</code> or <code>pi</code> arrays. Consequently, if your call to <code>mxSetDimensions</code> increases the number of elements in the <code>mxArray</code>, then you must enlarge the <code>pr</code> (and <code>pi</code>, if it exists) arrays accordingly.</p> <p>If your call to <code>mxSetDimensions</code> reduces the number of elements in the <code>mxArray</code>, then you can optionally reduce the size of the <code>pr</code> and <code>pi</code> arrays.</p> |

# mxSetDimensions

---

## Examples

Create a populated 3-by-2 mxArray. Then, call `mxSetDimensions` to expand the mxArray to 4-by-3-by-2. Preserve the original 3-by-2 data as the first page of the expanded mxArray.

```
#define ROWS 3
#define COLS 2
#define TOTAL_ELEMENTS (ROWS * COLS)

#define NEW_PAGES 4
#define NEW_ROWS 3
#define NEW_COLS 2
#define TOTAL_NEW_ELEMENTS (NEW_PAGES * NEW_ROWS * NEW_COLS)

mxArray *array_ptr;
static double real_data[] = {5.2, 7.8, 4.3, 9.3, 8.2, 7.1};
int new_ndims=3;
int new_dims[3]={NEW_PAGES, NEW_ROWS, NEW_COLS};
double *start_of_new_pr, *start_of_old_pr, *pr;

/* Create a 3-by-2 array named "Apricot" */
array_ptr = mxCreateDoubleMatrix(ROWS, COLS, mxREAL);
pr = mxGetPr(array_ptr);
memcpy((void *)pr, (const void *)real_data,
 ROWS*COLS*sizeof(double));
mxSetName(array_ptr, "Apricots");

...

/* Change the dimensions of "Apricots" from 3-by-2 to
4-by-3-by-2. */
mxSetDimensions(array_ptr, new_ndims, new_dims);

/* Allocate space to hold 24 data elements. */
start_of_new_pr = mxMalloc(TOTAL_NEW_ELEMENTS,
sizeof(double));

/* Copy the old real_data to the beginning of the new section. */
start_of_old_pr = mxGetPr(array_ptr);
memcpy(start_of_new_pr, start_of_old_pr, TOTAL_ELEMENTS);
```

```
/* Deallocate the space held by the old pr. */
mxFree(start_of_old_pr);

/* Associate the new data section with the array_ptr. */
mxSetPr(array_ptr, start_of_new_pr);

...
```

For an additional example, see `mxsetdimensions.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxGetNumberOfDimensions`, `mxSetM`, `mxSetN`

# mxSetField

---

**Purpose** Set a field value of a structure array, given a field name and an index

**C Syntax**

```
#include "matrix.h"
void mxSetField(mxArray *array_ptr, int index,
 const char *field_name, mxArray *value);
```

**Arguments**

`array_ptr`  
Pointer to a structure mxArray. Call `mxIsStruct` to determine if `array_ptr` points to a structure mxArray.

`index`  
The desired element. The first element of an mxArray has an index of 0, the second element has an index of 1, and the last element has an index of  $N - 1$ , where  $N$  is the total number of elements in the structure mxArray. See `mxCalcSingleSubscript` for details on calculating an index.

`field_name`  
The name of the field whose value you are assigning. Call `mxGetFieldNameByNumber` or `mxGetFieldNumber` to determine existing `field_names`.

`value`  
Pointer to the mxArray you are assigning.

**Description** Use `mxSetField` to assign a `value` to the specified element of the specified field. In pseudo-C terminology, `mxSetField` performs the assignment

```
array_ptr[index].field_name = value;
```

If there is already a value at the given position, the `value` pointer you specified overwrites the old value pointer. However, `mxSetField` does not free the dynamic memory that the old value pointer pointed to. Consequently, you should typically free this old mxArray immediately before or after calling `mxSetField`.

**Note:** Inputs to a MEX-file are constant read-only mxArrays and should not be modified. Using `mxSetCell*` or `mxSetField*` to modify the cells or fields of an argument passed from MATLAB will cause unpredictable results.

**Examples**

Consider a function that expects to receive a structure `mxArray` containing a speed field. The speed field contains a numeric scalar. The function calls `mxSetField` to change the value in the speed field to a string `mxArray`.

```
void RedoTheSpeedField(mxArray *array_ptr)
{
 mxArray *old_field, *low_array_ptr, *high_array_ptr;
 double wind_speed_as_a_number;
 int index, number_of_elements;

 high_array_ptr = mxCreateString("high");
 low_array_ptr = mxCreateString("low");

 /* Loop through each element of the structure. */
 number_of_elements = mxGetM(array_ptr) * mxGetN(array_ptr);
 for (index=0; index<number_of_elements; index++) {

 /* Get the value in the "speed" field at this index. */
 old_field = mxGetField(array_ptr, index, "speed");

 /* The returned "field" is a pointer to a scalar mxArray.
 Get the value associated with the scalar mxArray. */
 if (old_field) {
 wind_speed_as_a_number = mxGetScalar(old_field);

 /* Assign the new field to the structure. */
 if (wind_speed_as_a_number > 15)
 mxSetField(array_ptr, index, "speed", high_array_ptr);
 else
 mxSetField(array_ptr, index, "speed", low_array_ptr);

 /* Deallocate the memory taken up by the previous field value. */
 mxDestroyArray(old_field);
 }
 else
 /* Issue a warning message about a missing 'speed' field. */

```

For an additional example, see `mxsetfield.c` in the `mx` subdirectory of the `examples` directory.

# mxSetField

---

## See Also

`mxCreateStructArray`, `mxCreateStructMatrix`, `mxGetField`,  
`mxGetFieldByNumber`, `mxGetFieldNameByNumber`, `mxGetFieldNumber`,  
`mxGetNumberOfFields`, `mxIsStruct`, `mxSetFieldByNumber`

**Purpose** Set a field value in a structure array, given a field number and an index

**C Syntax**

```
#include "matrix.h"
void mxSetFieldByNumber(mxArray *array_ptr, int index,
 int field_number, mxArray *value);
```

**Arguments**

**array\_ptr**  
Pointer to a structure mxArray. Call `mxIsStruct` to determine if `array_ptr` points to a structure mxArray.

**index**  
The desired element. The first element of an mxArray has an index of 0, the second element has an index of 1, and the last element has an index of  $N-1$ , where  $N$  is the total number of elements in the structure mxArray. See `mxCalcSingleSubscript` for details on calculating an index.

**field\_number**  
The position of the field whose value you want to extract. The first field within each element has a `field_number` of 0, the second field has a `field_number` of 1, and so on. The last field has a `field_number` of  $N-1$ , where  $N$  is the number of fields.

**value**  
The value you are assigning.

**Note:** Inputs to a MEX-file are constant read-only mxArrays and should not be modified. Using `mxSetCell*` or `mxSetField*` to modify the cells or fields of an argument passed from MATLAB will cause unpredictable results.

**Description** Use `mxSetFieldByNumber` to assign a value to the specified element of the specified field. `mxSetFieldByNumber` is almost identical to `mxSetField`; however, the former takes a field number as its third argument and the latter takes a field name as its third argument.

**See Also** `mxCreateStructArray`, `mxCreateStructMatrix`, `mxGetField`, `mxGetFieldByNumber`, `mxGetFieldNameByNumber`, `mxGetFieldNumber`, `mxGetNumberOfFields`, `mxIsStruct`, `mxSetField`

# mxSetImagData

---

**Purpose** Set imaginary data pointer for an mxArray

**C Syntax**

```
#include "matrix.h"
void mxSetImagData(mxAarray *array_ptr, void *pi);
```

**Arguments**

array\_ptr  
Pointer to an mxArray.

pi  
Pointer to the first element of an array. Each element in the array contains the imaginary component of a value. The array must be in dynamic memory; call mxMalloc to allocate this dynamic memory. If pi points to static memory, memory leaks and other memory errors may result.

**Description** mxSetImagData is similar to mxSetPi, except it returns a void \*. Use this on numeric arrays with contents other than double.

**See Also** mxSetPi

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the <code>ir</code> array of a sparse <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetIr(mxArray *array_ptr, int *ir);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a sparse <code>mxArray</code>.</p> <p><code>ir</code><br/>Pointer to the <code>ir</code> array. The <code>ir</code> array must be sorted in column-major order.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | <p>Use <code>mxSetIr</code> to specify the <code>ir</code> array of a sparse <code>mxArray</code>. The <code>ir</code> array is an array of integers; the length of the <code>ir</code> array should equal the value of <code>nzmax</code>.</p> <p>Each element in the <code>ir</code> array indicates a row (offset by 1) at which a nonzero element can be found. (The <code>jc</code> array is an index that indirectly specifies a column where nonzero elements can be found. See <code>mxSetJc</code> for more details on <code>jc</code>.)</p> <p>For example, suppose you create a 7-by-3 sparse <code>mxArray</code> named <code>Sparrow</code> containing six nonzero elements by typing</p> <pre>&gt;&gt; Sparrow=zeros(7,3); &gt;&gt; Sparrow(2,1)=1; &gt;&gt; Sparrow(5,1)=1; &gt;&gt; Sparrow(3,2)=1; &gt;&gt; Sparrow(2,3)=2; &gt;&gt; Sparrow(5,3)=1; &gt;&gt; Sparrow(6,3)=1; &gt;&gt; Sparrow=sparse(Sparrow);</pre> <p>The <code>pr</code> array holds the real data for the sparse matrix, which in <code>Sparrow</code> is the five 1s and the one 2. If there is any nonzero imaginary data, then it is in a <code>pi</code> array.</p> |

| Subscript | ir | pr | jc | Comments                            |
|-----------|----|----|----|-------------------------------------|
| (2, 1)    | 1  | 1  | 0  | Column 1; ir is 1 because row is 2. |
| (5, 1)    | 4  | 1  | 2  | Column 1; ir is 4 because row is 5. |
| (3, 2)    | 2  | 1  | 3  | Column 2; ir is 2 because row is 3. |
| (2, 3)    | 1  | 2  | 6  | Column 3; ir is 1 because row is 2. |
| (5, 3)    | 4  | 1  |    | Column 3; ir is 4 because row is 5. |
| (6, 3)    | 5  | 1  |    | Column 3; ir is 5 because row is 6. |

Notice how each element of the `ir` array is always 1 less than the row of the corresponding nonzero element. For instance, the first nonzero element is in row 2; therefore, the first element in `ir` is 1 (that is, 2-1). The second nonzero element is in row 5; therefore, the second element in `ir` is 4 (5-1).

The `ir` array must be in column-major order. That means that the `ir` array must define the row positions in column 1 (if any) first, then the row positions in column 2 (if any) second, and so on through column N. Within each column, row position 1 must appear prior to row position 2, and so on.

`mxSetIr` does not sort the `ir` array for you; you must specify an `ir` array that is already sorted.

## Examples

See the examples on the `mxSetNzmax` reference page, and `mxsetir.c` in the `mx` subdirectory of the `examples` directory

## See Also

`mxCreateSparse`, `mxGetIr`, `mxGetJc`, `mxSetJc`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the <code>j c</code> array of a sparse <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetJc(mxArray *array_ptr, int *j c);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a sparse <code>mxArray</code>.</p> <p><code>j c</code><br/>Pointer to the <code>j c</code> array.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>Use <code>mxSetJc</code> to specify a new <code>j c</code> array for a sparse <code>mxArray</code>. The <code>j c</code> array is an integer array having <math>n+1</math> elements where <math>n</math> is the number of columns in the sparse <code>mxArray</code>. The values in the <code>j c</code> array have the meanings:</p> <ul style="list-style-type: none"> <li>• <code>j c[j]</code> is the index in <code>i r</code>, <code>pr</code> (and <code>pi</code> if it exists) of the first nonzero entry in the <math>j</math>th column.</li> <li>• <code>j c[j+1]-1</code> is the index of the last nonzero entry in the <math>j</math>th column.</li> <li>• <code>j c[number of columns + 1]</code> is equal to <code>nnz</code>, which is the number of nonzero entries in the entire sparse <code>mxArray</code>.</li> </ul> <p>The number of nonzero elements in any column (denoted as column <code>C</code>) is</p> $j c[C] - j c[C-1];$ <p>For example, consider a 7-by-3 sparse <code>mxArray</code> named <code>Sparrow</code> containing six nonzero elements, created by typing</p> <pre>&gt;&gt; Sparrow=zeros(7,3); &gt;&gt; Sparrow(2,1)=1; &gt;&gt; Sparrow(5,1)=1; &gt;&gt; Sparrow(3,2)=1; &gt;&gt; Sparrow(2,3)=2; &gt;&gt; Sparrow(5,3)=1; &gt;&gt; Sparrow(6,3)=1; &gt;&gt; Sparrow=sparse(Sparrow);</pre> |

The contents of the `ir`, `jc`, and `pr` arrays are

| Subscript | ir | pr | jc | Comment                                                                                      |
|-----------|----|----|----|----------------------------------------------------------------------------------------------|
| (2, 1)    | 1  | 1  | 0  | Column 1 contains 2 entries, at <code>ir[0]</code> , <code>ir[1]</code>                      |
| (5, 1)    | 4  | 1  | 2  | Column 2 contains 1 entry, at <code>ir[2]</code>                                             |
| (3, 2)    | 2  | 1  | 3  | Column 3 contains 3 entries, at <code>ir[3]</code> , <code>ir[4]</code> , <code>ir[5]</code> |
| (2, 3)    | 1  | 2  | 6  | There are 6 nonzero elements.                                                                |
| (5, 3)    | 4  | 1  |    |                                                                                              |
| (6, 3)    | 5  | 1  |    |                                                                                              |

Now onto a much sparser `mxArray`. Consider an 8,000 element sparse `mxArray` named `Spacious` containing only 3 nonzero elements. The `ir`, `pr`, and `jc` arrays contain

| Subscript | ir | pr | jc | Comment                                          |
|-----------|----|----|----|--------------------------------------------------|
| (73, 2)   | 72 | 1  | 0  | Column 1 contains 0 entries                      |
| (50, 3)   | 49 | 1  | 0  | Column 2 contains 1 entry, at <code>ir[0]</code> |
| (64, 5)   | 63 | 1  | 1  | Column 3 contains 1 entry, at <code>ir[1]</code> |
|           |    |    | 2  | Column 4 contains 0 entries.                     |
|           |    |    | 2  | Column 5 contains 1 entry, at <code>ir[3]</code> |
|           |    |    | 3  | Column 6 contains 0 entries.                     |
|           |    |    | 3  | Column 7 contains 0 entries.                     |
|           |    |    | 3  | Column 8 contains 0 entries.                     |
|           |    |    | 3  | There are 3 nonzero elements.                    |

## Example

See the example on the `mxSetNzmax` reference page.

**See Also**      `mxGetIr`, `mxGetJc`, `mxSetIr`

# mxSetLogical

---

|                    |                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the logical flag                                                                                                                                                                                                                                                                                 |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetLogical (mxArray *array_ptr);</pre>                                                                                                                                                                                                                               |
| <b>Arguments</b>   | <code>array_ptr</code><br>Pointer to an <code>mxArray</code> having a numeric class.                                                                                                                                                                                                                 |
| <b>Description</b> | Use <code>mxSetLogical</code> to turn on an <code>mxArray</code> 's logical flag. This flag tells MATLAB that the array's data is to be treated as Boolean. If the logical flag is on, then MATLAB treats a 0 value as meaning <code>false</code> and a nonzero value as meaning <code>true</code> . |
| <b>Example</b>     | See <code>mxsetlogical.c</code> in the <code>mx</code> subdirectory of the <code>examples</code> directory.                                                                                                                                                                                          |
| <b>See Also</b>    | <code>mxClearLogical</code> , <code>mxIsLogical</code>                                                                                                                                                                                                                                               |

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the number of rows                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetM(mxArray *array_ptr, int m);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | <p><b>m</b><br/>The desired number of rows.</p> <p><b>array_ptr</b><br/>Pointer to an mxArray.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b> | <p>Call <code>mxSetM</code> to set the number of rows in the specified mxArray. The term “rows” means the first dimension of an mxArray, regardless of the number of dimensions. Call <code>mxSetN</code> to set the number of columns.</p> <p>You typically use <code>mxSetM</code> to change the shape of an existing mxArray. Note that <code>mxSetM</code> does not allocate or deallocate any space for the <code>pr</code>, <code>pi</code>, <code>ir</code>, or <code>jc</code> arrays. Consequently, if your calls to <code>mxSetM</code> and <code>mxSetN</code> increase the number of elements in the mxArray, then you must enlarge the <code>pr</code>, <code>pi</code>, <code>ir</code>, and/or <code>jc</code> arrays. Call <code>mxRealloc</code> to enlarge them. (See the <code>mxSetN</code> reference page for an example of doing this.)</p> <p>If your calls to <code>mxSetM</code> and <code>mxSetN</code> end up reducing the number of elements in the array, then you do can optionally reduce the sizes of the <code>pr</code>, <code>pi</code>, <code>ir</code>, and/or <code>jc</code> arrays in order to use heap space more efficiently.</p> |

## Examples

Reshape a 3-by-2 mxArray into a 6-by-1 mxArray for more efficient use of memory without changing the data held by the mxArray.

```
static double real_data[] = {5.2, 7.8, 4.3, 9.3, 8.2, 7.1};
int old_rows = 3, old_cols = 2;
int new_rows = 6, new_cols = 1;
double *pr;
mxArray *array_ptr;

/* Create a 3-by-2 array named "Apricot" */
array_ptr = mxCreateDoubleMatrix(old_rows, old_cols, mxREAL);
pr = mxGetPr(array_ptr);
memcpy((void *)pr, (const void *)real_data,
 old_rows*old_cols*sizeof(double));
mxSetName(array_ptr, "Apricots");

...

/* Change the dimensions of "Apricots" from 3-by-2 to 6-by-1. */
mxSetM(array_ptr, new_rows);
mxSetN(array_ptr, new_cols);

...
```

The data in Apricot starts out as

```
5.2 9.3
7.8 8.2
4.3 7.1
```

After reshaping, Apricot's data becomes

```
5.2
7.8
4.3
9.3
8.2
7.1
```

For an additional example, see `mxsetm.c` in the `mx` subdirectory of the `examples` directory.

**See Also**      `mxGetM`, `mxGetN`, `mxSetN`

# mxSetN

---

**Purpose** Set the number of columns

**C Syntax**

```
#include "matrix.h"
void mxSetN(mxArray *array_ptr, int n);
```

**Arguments**

`array_ptr`  
Pointer to an `mxArray`.

`n`  
The desired number of columns.

**Description** Call `mxSetN` to set the number of columns in the specified `mxArray`. The term “columns” always means the second dimension of a matrix. Calling `mxSetN` forces an `mxArray` to have two dimensions. For example, if `array_ptr` points to an `mxArray` having three dimensions, calling `mxSetN` reduces the `mxArray` to two dimensions.

You typically use `mxSetN` to change the shape of an existing `mxArray`. Note that `mxSetN` does not allocate or deallocate any space for the `pr`, `pi`, `ir`, or `jc` arrays. Consequently, if your calls to `mxSetN` and `mxSetM` increase the number of elements in the `mxArray`, then you must enlarge the `pr`, `pi`, `ir`, and/or `jc` arrays.

If your calls to `mxSetM` and `mxSetN` end up reducing the number of elements in the `mxArray`, then you may want to reduce the size of the `pr`, `pi`, `ir`, or `jc` arrays in order to reduce heap space usage. However, reducing the size is not mandatory.

**Examples** Consider a 3-by-3 `mxArray` containing

|    |    |    |
|----|----|----|
| 10 | 14 | 20 |
| 12 | 17 | 22 |
| 13 | 18 | 23 |

Suppose you enlarge the `mxArray` to 4-by-4 but preserve the positions of the original 3-by-3 to yield an `mxArray` containing

|    |    |    |   |
|----|----|----|---|
| 10 | 14 | 20 | 0 |
| 12 | 17 | 22 | 0 |
| 13 | 18 | 23 | 0 |
| 0  | 0  | 0  | 0 |

The code to create the original 3-by-3 mxArray and the expanded 4-by-4 mxArray is

```
#define COLS_IN_OLD_ARRAY 3
#define ROWS_IN_OLD_ARRAY 3
#define ROWS_IN_NEW_ARRAY 4
 mxArray *array_ptr;
 double old_pr[9] = {10, 12, 13, 14, 17, 18, 20, 22, 23};
 double *dest, *src, *new_heap_pr, *pr;
 int col;

/* Create a 3-by-3 real-only mxArray of doubles. */
array_ptr = mxCreateDoubleMatrix(3, 3, mxREAL);
pr = mxGetPr(array_ptr);
memcpy((void *)pr, (const void *)old_pr, 9*sizeof(double));
mxSetName(array_ptr, "Apricots");

...

/* Reshape Apricots into a 4-by-4 mxArray. */
mxSetM(array_ptr, 4);
mxSetN(array_ptr, 4);

/* Redo pr to respond to the new size of Apricots; preserve the
positions of the original 3-by-3. */

/* First, allocate heap to hold 16 real elements. */
new_heap_pr = (double *)mxCalloc(16, sizeof(double));

/* Next, copy the original 9 elements. */
for (col=0; col<COLS_IN_OLD_ARRAY; col++) {
 dest = new_heap_pr + (ROWS_IN_NEW_ARRAY * col);
 src = &old_pr[ROWS_IN_OLD_ARRAY * col];
 memcpy(dest, src, ROWS_IN_OLD_ARRAY * sizeof(double));
}

/* First, deallocate the old pr data. */
mxFree(mxGetPr(array_ptr));

/* Then, associate the new data with the real data of Apricots. */
mxSetPr(array_ptr, new_heap_pr);
```

For an additional example, see `mxsetn.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxGetM`, `mxGetN`, `mxSetM`

# mxSetName

---

**Purpose** Set the name of an mxArray

**C Syntax**

```
#include "matrix.h"
void mxSetName(mxArray *array_ptr, const char *name);
```

**Arguments**

`array_ptr`  
Pointer to an mxArray.

`name`  
The name you are assigning to the mxArray. The specified name can be up to `mxMAXNAM` characters, where `mxMAXNAM` is a constant defined in the `matrix.h` header file. If you specify a name longer than `mxMAXNAM-1` characters, then `mxSetName` assigns only the first `mxMAXNAM-1` characters to the name.

**Description** Call `mxSetName` to establish a name for an mxArray or to change an existing name.

`mxSetName` assigns the characters in `name` to a fixed-width section of memory. Do not deallocate this memory.

**Examples** Create an mxArray. Then, give it a name.

```
mxArray *array_ptr;

/* Create a 5-by-7 real array. */
array_ptr = mxCreateDoubleMatrix(5, 7, mxREAL);

/* Name the array "Grapes" */
mxSetName(array_ptr, "Grapes");

...
```

For an additional example, see `mxsetname.c` in the `mx` subdirectory of the `examples` directory.

**See Also** `mxGetName`

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set the storage space for nonzero elements                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetNzmax(mxArray *array_ptr, int nzmax);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a sparse <code>mxArray</code>.</p> <p><code>nzmax</code><br/>The number of elements that <code>mxCreateSparse</code> should allocate to hold the arrays pointed to by <code>ir</code>, <code>pr</code>, and <code>pi</code> (if it exists). You should set <code>nzmax</code> greater than or equal to the number of nonzero elements in the <code>mxArray</code>, but you should set it to be less than or equal to the number of rows times the number of columns. If you specify an <code>nzmax</code> value of 0, <code>mxSetNzmax</code> sets the value of <code>nzmax</code> to 1.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | <p>Use <code>mxSetNzmax</code> to assign a new value to the <code>nzmax</code> field of the specified sparse <code>mxArray</code>. The <code>nzmax</code> field holds the maximum possible number of nonzero elements in the sparse <code>mxArray</code>.</p> <p>The number of elements in the <code>ir</code>, <code>pr</code>, and <code>pi</code> (if it exists) arrays must be equal to <code>nzmax</code>. Therefore, after calling <code>mxSetNzmax</code>, you must change the size of the <code>ir</code>, <code>pr</code>, and <code>pi</code> arrays. To change the size of one of these arrays, you should</p> <ol style="list-style-type: none"><li>1 Call <code>mxMalloc</code>, setting <code>n</code> to the new value of <code>nzmax</code>.</li><li>2 Call the ANSI C routine <code>memcpy</code> to copy the contents of the old array to the new area allocated in Step 1.</li><li>3 Call <code>mxFree</code> to free the memory occupied by the old array.</li><li>4 Call the appropriate <code>mxSet</code> routine (<code>mxSetIr</code>, <code>mxSetPr</code>, or <code>mxSetPi</code>) to establish the new memory area as the current one.</li></ol> <p>How big should <code>nzmax</code> be? One philosophy is that you should set <code>nzmax</code> equal to or slightly greater than the number of nonzero elements in a sparse <code>mxArray</code>. This philosophy conserves precious heap space. Another philosophy is that you should make <code>nzmax</code> equal to the total number of elements in an <code>mxArray</code>. This philosophy eliminates (or, at least reduces) expensive reallocations.</p> |

## Examples

Create a sparse mxArray, then reduce its nzmax value to conserve heap space:

```
int starting_nzmax=5000, rows=2500, cols=2, new_nzmax=4;
static double static_pr_data[] = {5.8, 6.2, 5.9, 6.1};
static int static_ir_data[] = {0, 1, 1, 2};
static int static_jc_data[] = {0, 2, 4};
double *start_of_pr, *new_pr_ptr;
int *start_of_ir, *start_of_jc, *new_ir_ptr;
mxArray *array_ptr;

/* Create a sparse array and name it "Sparrow". */
array_ptr = mxCreateSparse(rows, cols, starting_nzmax,
mxREAL);
mxSetName(array_ptr, "Sparrow");

/* Place pr data into Sparrow. */
start_of_pr = (double *)mxGetPr(array_ptr);
memcpy(start_of_pr, static_pr_data, sizeof(static_pr_data));

/* Place ir data into Sparrow. */
start_of_ir = (int *)mxGetIr(array_ptr);
memcpy(start_of_ir, static_ir_data, sizeof(static_ir_data));
/* Place jc data into Sparrow. */
start_of_jc = (int *)mxGetJc(array_ptr);
memcpy(start_of_jc, static_jc_data, sizeof(static_jc_data));
...
/* Decrease the size of nzmax. */
mxSetNzmax(array_ptr, new_nzmax);
/* Adjust ir accordingly. */
new_ir_ptr = mxMalloc(new_nzmax, sizeof(double));
memcpy(new_ir_ptr, mxGetIr(array_ptr),
new_nzmax*sizeof(double));
mxFree(mxGetIr(array_ptr));
mxSetIr(array_ptr, new_ir_ptr);
/* Adjust pr accordingly. */
new_pr_ptr = mxMalloc(new_nzmax, sizeof(double));
memcpy(new_pr_ptr, mxGetPr(array_ptr),
new_nzmax*sizeof(double));
mxFree(mxGetPr(array_ptr));
mxSetPr(array_ptr, new_pr_ptr);
```

There is no need to adjust  $\pi$  because Sparrow is purely real. For an additional example, see `mxsetnzmax.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxGetNzmax`

# mxSetPi

---

**Purpose** Set new imaginary data for an `mxArray`

**C Syntax**

```
#include "matrix.h"
void mxSetPi(mxArray *array_ptr, double *pi);
```

**Arguments**

`array_ptr`  
Pointer to a full (nonsparse) `mxArray`.

`pi`  
Pointer to the first element of an array. Each element in the array contains the imaginary component of a value. The array must be in dynamic memory; call `mxMalloc` to allocate this dynamic memory. If `pi` points to static memory, memory leaks and other memory errors may result.

**Description** Use `mxSetPi` to set the imaginary data of the specified `mxArray`.

Most `mxCreate` functions optionally allocate heap space to hold imaginary data. If you tell an `mxCreate` function to allocate heap space (for example, by setting the `ComplexFlag` to `mxComplex` or by setting `pi` to a non-NULL value), then you do not ordinarily use `mxSetPi` to initialize the created `mxArray`'s imaginary elements. Rather, you typically call `mxSetPi` to replace the initial imaginary values with new ones.

**Examples**

Create a 1-by-5 mxArray. Then, grow the mxArray to 1-by-6, seeding it with the five elements of the 1-by-5 mxArray.

```

mxArray *array_ptr;
double start_pr[5] = {3.2, 4.6, 5.1, 6.8, 7.3};
double start_pi[5] = {4.5, 4.2, 4.4, 4.8, 4.6};
double *new_heap_pr, *new_heap_pi, *pr, *pi;

/* Create a 1-by-5 complex array of doubles. */
array_ptr = mxCreateDoubleMatrix(1, 5, mxCOMPLEX);
pr = mxGetPr(array_ptr);
pi = mxGetPi(array_ptr);
memcpy((void *)pr, (const void *)start_pr, 5*sizeof(double));
memcpy((void *)pi, (const void *)start_pi, 5*sizeof(double));
mxSetName(array_ptr, "Apricots");
...

/* Add a sixth element to Apricots. */

/* First, allocate heap to hold six complex elements. */
new_heap_pr = (double *)mxCalloc(6, sizeof(double));
new_heap_pi = (double *)mxCalloc(6, sizeof(double));

/* Next, copy the old five complex elements to a new section. */
memcpy(new_heap_pr, start_pr, 5*sizeof(double));
memcpy(new_heap_pi, start_pi, 5*sizeof(double));

/* Next, assign the 6th complex element. */
new_heap_pr[5] = 8.9;
new_heap_pi[5] = 4.7;

/* Next, free the heap required to hold the old pr and pi. */
mxFree(mxGetPr(array_ptr));
mxFree(mxGetPi(array_ptr));

/* Now, change Apricot's dimensions to 1-by-6. */
mxSetN(array_ptr, 6);

/* Finally, associate the new numbers with Apricots. */
mxSetPr(array_ptr, new_heap_pr);

```

## mxSetPi

---

```
mxSetPi (array_ptr, new_heap_pi);
```

For an additional example, see `mxsetpi.c` in the `mx` subdirectory of the `examples` directory.

### See Also

`mxGetPi`, `mxGetPr`, `mxSetPr`

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Set new real data for an mxArray                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>C Syntax</b>    | <pre>#include "matrix.h" void mxSetPr(mxAarray *array_ptr, double *pr);</pre>                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>   | <p><code>array_ptr</code><br/>Pointer to a full (nonsparse) mxArray.</p> <p><code>pr</code><br/>Pointer to the first element of an array. Each element in the array contains the real component of a value. The array must be in dynamic memory; call <code>mxMalloc</code> to allocate this dynamic memory. If <code>pr</code> points to static memory, then memory leaks and other memory errors may result.</p> |
| <b>Description</b> | <p>Use <code>mxSetPr</code> to set the real data of the specified mxArray.</p> <p>All <code>mxCreate</code> calls allocate heap space to hold real data. Therefore, you do not ordinarily use <code>mxSetPr</code> to initialize the real elements of a freshly-created mxArray. Rather, you typically call <code>mxSetPr</code> to replace the initial real values with new ones.</p>                             |

## Examples

Create a populated 2-by-3 real-only mxArray. Then, change its data.

```
mxArray *array_ptr;
double start_pr[6] = {3.2, 4.6, 5.1, 6.8, 7.3, 8.1};
double *start_of_real_data, *next_data, *pr;
int c;

/* Create a 2-by-3 real-only array of doubles. */
array_ptr = mxCreateDoubleMatrix(2, 3, mxREAL);
pr = mxGetPr(array_ptr);
memcpy((void *)pr, (const void *)start_pr, 6*sizeof(double));
mxSetName(array_ptr, "Apricot");

...

/* Redo "Apricot", changing its size and its real data. */

/* First, deallocate the old pr data. */
mxFree(mxGetPr(array_ptr));

/* Next, change Apricot's dimensions to 2-by-4. */
mxSetN(array_ptr, 4);

/* Next, allocate memory to hold 8 real elements. */
start_of_real_data =
 (double *)mxMalloc(8, mxGetElementSize(array_ptr));

/* Place 8 values in the newly allocated memory. */
next_data = start_of_real_data;
for (c=12; c<20; c++)
 *next_data++ = sqrt(c);

/* Associate the new data with the real data of Apricots. */
mxSetPr(array_ptr, start_of_real_data);
```

Apricot initially contains

|        |        |        |
|--------|--------|--------|
| 3.2000 | 5.1000 | 7.3000 |
| 4.6000 | 6.8000 | 8.1000 |

After calling `mxSetPr`, `Apricot` contains

|        |        |        |
|--------|--------|--------|
| 3.4641 | 3.7417 | 4.0000 |
| 3.6056 | 3.8730 | 4.1231 |

For an additional example, see `mxsetpr.c` in the `mx` subdirectory of the `examples` directory.

## See Also

`mxGetPr`, `mxGetPi`, `mxSetPi`

# engClose

---

**Purpose**               Quit a MATLAB engine session

**Fortran Syntax**    integer\*4 function engClose(ep)  
                  integer\*4 ep

**Arguments**        ep  
                  Engine pointer.

**Description**     This routine allows you to quit a MATLAB engine session.  
  
engClose sends a quit command to the MATLAB engine session and closes the connection. It returns 0 on success, and 1 otherwise. Possible failure includes attempting to terminate a MATLAB engine session that was already terminated.

**Example**           See `fengdemo.f` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Evaluate expression in character array                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <pre>integer*4 function engEvalString(ep, command) integer*4 ep character*(*) command</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>      | <p>ep<br/>Engine pointer.</p> <p>command<br/>Character array to execute.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>    | <p>engEvalString evaluates the expression contained in <code>command</code> for the MATLAB engine session, <code>ep</code>, previously started by <code>engOpen</code>. It returns a nonzero value if the MATLAB session is no longer running, and zero otherwise.</p> <p>On UNIX systems, <code>engEvalString</code> sends commands to MATLAB by writing down a pipe connected to MATLAB's <i>stdin</i>. Any output resulting from the command that ordinarily appears on the screen is read back from <i>stdout</i> into the buffer defined by <code>engOutputBuffer</code>.</p> |
| <b>Example</b>        | See <code>fengdemo.f</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.                                                                                                                                                                                                                                                                                                                                                                  |

# engGetFull

---

**Purpose** Read full `mxArrays` from an engine

**Fortran Syntax** `integer*4 function engGetFull(ep, name, m, n, pr, pi)`  
`integer*4 ep, m, n, pr, pi`  
`character*(*) name`

**Arguments**

`ep`  
Engine pointer.

`name`  
Name of `mxArray` to get or put into engine's workspace.

`m`  
Row dimension.

`n`  
Column dimension.

`pr`  
Pointer to real part.

`pi`  
Pointer to imaginary part.

**Description** Most MATLAB applications work only with full (nonsparse) `mxArrays`. This routine provides an easy way to copy a full `mxArray` from a MATLAB engine process. It offers an alternative to `engGetMatrix`, which does not require use of the `mxArray` structure.

`engGetFull` reads the named `mxArray` from the engine pointed to by `ep` and places the row dimensions, column dimensions, real array pointer, and imaginary array pointer into the locations specified by `m`, `n`, `pr`, and `pi`, respectively.

`engGetFull` returns 0 if successful, and 1 otherwise.

`engGetFull` allocates memory for the real and imaginary arrays using `mxMalloc`; use `mxFree` to return it when you are done.

If the `mxArray` is purely real, the imaginary pointer is given 0.

**Example**

See `fengdemo.f` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.

# engGetMatrix

---

**Purpose** Read mxArray from a MATLAB engine's workspace

**Fortran Syntax** integer\*4 function engGetMatrix(ep, name)  
integer\*4 ep  
character\*(\*) name

**Arguments** ep  
Engine pointer.  
  
name  
Name of mxArray to get from engine.

**Description** This routine allows you to copy an mxArray out of a MATLAB engine's workspace.

engGetMatrix reads the named mxArray from the engine pointed to by ep and returns a pointer to a newly allocated mxArray structure, or 0 if the attempt fails.

Be careful in your code to free the mxArray created by this routine when you are finished with it.

On UNIX systems, engGetMatrix issues the command `save stdout name` to MATLAB, causing MATLAB to write the named mxArray down its *stdout* pipe, which is in turn caught and decoded by engGetMatrix.

**Example** See `fengdemo.f` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Start a MATLAB engine session                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Fortran Syntax</b> | <pre>integer*4 function engOpen(startcmd) integer*4 ep character*(*) startcmd</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>      | <p>ep<br/>Engine pointer.</p> <p>startcmd<br/>Character array to start MATLAB process.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>    | <p>This routine allows you to start a MATLAB process for the purpose of using MATLAB as a computational engine.</p> <p>engOpen(startcmd) starts a MATLAB process using the command specified in startcmd, establishes a connection, and returns a unique engine identifier, or 0 if the open fails.</p> <p>On the UNIX system, if startcmd is empty, engOpen starts MATLAB on the current host using the command matlab. If startcmd is a hostname, engOpen starts MATLAB on the designated host by embedding the specified hostname string into the larger string:</p> <pre>"rsh hostname \"/bin/csh -c 'setenv DISPLAY\ hostname:0; matlab' \\"</pre> <p>If startcmd is anything else (has white space in it, or nonalphanumeric characters), it is executed literally to start MATLAB.</p> <p>engOpen performs the following steps:</p> <ol style="list-style-type: none"> <li>1 Creates two pipes.</li> <li>2 Forks a new process and sets up the pipes to pass <i>stdin</i> and <i>stdout</i> from the child to two file descriptors in the parent.</li> <li>3 Executes a command to run MATLAB (rsh for remote execution).</li> </ol> |
| <b>Example</b>        | See fengdemo.f in the eng_mat subdirectory of the examples directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

# engOutputBuffer

---

**Purpose** Specify buffer for MATLAB output

**Fortran Syntax** `subroutine engOutputBuffer(ep, p, n)`  
`integer*4 ep, n`  
`character *(*) p`

**Arguments**

`ep`  
Engine pointer.

`n`  
Length of buffer `p`.

`p`  
Character buffer of length `n`.

**Description** `engOutputBuffer` defines a character buffer for `engEvalString` to return any output that ordinarily appears on the screen.

The default behavior of `engEvalString` is to discard any standard output caused by the command it is executing. `engOutputBuffer(ep, p, n)` tells any subsequent calls to `engEvalString` to save the first `n` characters of output in the character buffer `p`.

**Example** See `fengdemo.f` in the `eng_mat` subdirectory of the `examples` directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Write full <code>mxArrays</code> into the workspace of an engine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Fortran Syntax</b> | <pre>integer*4 function engPutFull (ep, name, m, n, pr, pi) integer*4 ep, m, n, pr, pi character*(*) name</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>      | <p><code>ep</code><br/>Engine pointer.</p> <p><code>name</code><br/>Name of <code>mxArray</code> to put into engine's workspace.</p> <p><code>m</code><br/>Row dimension.</p> <p><code>n</code><br/>Column dimension.</p> <p><code>pr</code><br/>Pointer to real part.</p> <p><code>pi</code><br/>Pointer to imaginary part.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>    | <p>Most MATLAB applications work only with full (nonsparse) <code>mxArrays</code>. This routine provides an easy way to write a full <code>mxArray</code> into a MATLAB engine process. It offers an alternative to <code>engPutMatrix</code>, which does not require use of the <code>mxArray</code> structure.</p> <p><code>engPutFull</code> writes the <code>mxArray</code> with dimensions <code>m</code>-by-<code>n</code>, real data <code>pr</code>, and imaginary data <code>pi</code> into the workspace of engine <code>ep</code> with the specified name.</p> <p>If the <code>mxArray</code> does not exist in the engine's workspace, it is created. If an <code>mxArray</code> with the same name already exists in the workspace, the existing <code>mxArray</code> is replaced with the new <code>mxArray</code>.</p> |
| <b>Example</b>        | See <code>fengdemo.f</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

# engPutMatrix

---

**Purpose** Write mxArray into a MATLAB engine's workspace

**Fortran Syntax** integer\*4 function engPutMatrix(ep, mp)  
integer\*4 mp, ep

**Arguments**

ep  
Engine pointer.

mp  
mxArray pointer.

**Description** This routine allows you to write an mxArray into a MATLAB engine's workspace.

engPutMatrix writes mxArray mp to the engine ep. If the mxArray does not exist in the workspace, it is created. If an mxArray with the same name already exists in the workspace, the existing mxArray is replaced with the new mxArray.

engPutMatrix returns 0 if successful and 1 if an error occurs.

Be careful in your code to free the mxArray created by this routine when you are finished with it.

On UNIX systems, engPutMatrix issues the command load stdio name to MATLAB and sends the data down the *stdin* pipe.

**Example** See fengdemo.f in the eng\_mat subdirectory of the examples directory for a sample program that illustrates how to call the MATLAB engine functions from a Fortran program.

---

|                       |                                                                                                                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Closes a MAT-file                                                                                                                                                                                                                  |
| <b>Fortran Syntax</b> | <code>integer*4 function matClose(mfp)</code><br><code>integer*4 mfp</code>                                                                                                                                                        |
| <b>Arguments</b>      | <code>mfp</code><br>Pointer to MAT-file information.                                                                                                                                                                               |
| <b>Description</b>    | <code>matClose</code> closes the MAT-file associated with <code>mfp</code> . It returns -1 for a write error, and 0 if successful.                                                                                                 |
| <b>Example</b>        | See <code>matdemo1.f</code> and <code>matdemo2.f</code> in the <code>eng_mat</code> subdirectory of the <code>examples</code> directory for sample programs that illustrate how to use this MAT-file routine in a Fortran program. |

# matDeleteMatrix

---

**Purpose** Delete named mxArray from MAT-file

**Fortran Syntax** subroutine matDeleteMatrix(mfp, name)  
integer\*4 mfp  
character\*(\*) name

**Arguments** mfp  
Pointer to MAT-file information.  
  
name  
Name of mxArray to delete.

**Description** matDeleteMatrix deletes the named mxArray from the MAT-file pointed to by mfp. The file is rewritten to accomplish this task. matDeleteMatrix returns 0 if successful, and nonzero if an error occurs.

**Example** See matdemo1.f in the eng\_mat subdirectory of the examples directory for a sample program that illustrates how to use this MAT-file routine in a Fortran program.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get directory of <code>mxArrays</code> in a MAT-file                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Fortran Syntax</b> | <pre>integer*4 function matGetDir(mfp, num) integer*4 mfp, num</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>      | <p><code>mfp</code><br/>Pointer to MAT-file information.</p> <p><code>num</code><br/>Address of the variable to contain the number of <code>mxArrays</code> in the MAT-file.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>    | <p>This routine allows you to get a list of the names of the <code>mxArrays</code> contained within a MAT-file.</p> <p><code>matGetDir</code> returns a pointer to an internal array containing pointers to the names of the <code>mxArrays</code> in the MAT-file pointed to by <code>mfp</code>. The length of the internal array (number of <code>mxArrays</code> in the MAT-file) is placed into <code>num</code>. The internal array is allocated using a single <code>mxCallLoc</code> and must be freed using <code>mxFree</code> when you are finished with it.</p> <p><code>matGetDir</code> returns 0 and sets <code>num</code> to a negative number if it fails. If <code>num</code> is zero, <code>mfp</code> contains no <code>mxArrays</code>.</p> <p>MATLAB variable names can be up to length 32.</p> |
| <b>Examples</b>       | <p>Print out a directory of the <code>mxArray</code> names contained within a MAT-file.</p> <pre> program main integer*4 mfp, dir, ndir, adir(100) character*20 names(100)  mfp = matOpen('foo.mat', 'r'); dir = matGetDir(mfp, ndir) if (dir .eq. 0) then     write (6,*) 'Can't read directory.'     stop end if</pre> <p>c Copy pointer into an array of pointers<br/>call <code>mxCopyPtrToInteger4(dir, adir, ndir)</code></p>                                                                                                                                                                                                                                                                                                                                                                                   |

```
c Copy pointer to character array
do 20 i = 1, ndir
 call mxCopyPtrToCharacter(adir(i), names(i), 20)
20 continue

write(6, *) 'Directory of MAT-file: '
do 30 i = 1, ndir
 write(6, *) names(i)
30 continue
matClose(mfp)
stop
end
```

See `matdemo2.f` in the `eng_mat` subdirectory of the `examples` directory for another sample program that illustrates how to use this MAT-file routine in a Fortran program.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Reads full <code>mxArrays</code> from MAT-files                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Fortran Syntax</b> | <pre>integer*4 function matGetFull(mfp, name, m, n, pr, pi) integer*4 mfp, m, n, pr, pi character*(*) name</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>      | <p><code>mfp</code><br/>Pointer to MAT-file information.</p> <p><code>m</code><br/>Row dimension.</p> <p><code>n</code><br/>Column dimension.</p> <p><code>name</code><br/>Name of <code>mxArray</code> to get or put to MAT-file.</p> <p><code>pi</code><br/>Pointer to imaginary part.</p> <p><code>pr</code><br/>Pointer to real part.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>    | <p>Most MATLAB applications work only with full (nonsparse) <code>mxArrays</code>. This routine provides an easy way to copy a full <code>mxArray</code> out of a MAT-file. It offers an alternative to <code>matGetMatrix</code>, which does not require use of the <code>mxArray</code> structure.</p> <p><code>matGetFull</code> reads the named <code>mxArray</code> from the MAT-file pointed to by <code>mfp</code> and places the row dimensions, column dimensions, real array pointer, and imaginary array pointer into the locations specified by <code>m</code>, <code>n</code>, <code>pr</code>, and <code>pi</code>, respectively.</p> <p><code>matGetFull</code> returns 0 if successful, and 1 if the named variable can't be found, the named variable is not a full <code>mxArray</code>, or there is a file read error.</p> <p><code>matGetFull</code> allocates memory for the real and imaginary arrays using <code>mxMalloc</code>; use <code>mxFree</code> to return the memory when you are done.</p> <p>If the <code>mxArray</code> is pure real, the imaginary pointer is 0.</p> |

# matGetFull

---

## Examples

Read the mxArray A from one MAT-file and write it out to another.

```
program main
integer matOpen, matClose, matPutFull, matGetFull
integer mf1, mf2, stat
integer m, n, pr, pi
mf1 = matOpen('foo.mat', 'r')
mf2 = matOpen('foo2.mat', 'w')
stat = matGetFull(mf1, 'A', m, n, pr, pi)
stat = matPutFull(mf2, 'A', m, n, pr, pi)
stat = matClose(mf1)
stat = matClose(mf2)
c
stop
end
```

Write a simple real mxArray into a MAT-file. Name the mxArray A and the MAT-file foo.mat.

```
integer matOpen, matClose, matPutFull
integer mfp, stat
double precision Areal(6)
data Areal / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 /
c
mfp = matOpen('foo.mat', 'w')
stat = matPutFull(mfp, 'A', 3, 2, Areal, 0)
stat = matClose(mfp)
c
stop
end
```

To test, run the second example; then go to MATLAB and enter:

```
load foo
A
A =
 1 4
 2 5
 3 6
```

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Reads mxArray from MAT-files                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Fortran Syntax</b> | <pre>integer*4 function matGetMatrix(mfp, name) integer*4 mfp character*(*) name</pre>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>      | <p>mfp<br/>Pointer to MAT-file information.</p> <p>name<br/>Name of mxArray to get from MAT-file.</p>                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b>    | <p>This routine allows you to copy an mxArray out of a MAT-file.</p> <p>matGetMatrix reads the named mxArray from the MAT-file pointed to by mfp and returns a pointer to a newly allocated mxArray structure, or 0 if the attempt fails.</p> <p>Be careful in your code to free the mxArray created by this routine when you are finished with it.</p>                                                                                                                                                         |
| <b>Examples</b>       | <p>Write a simple 3-by-2 real mxArray into a MAT-file. Name the mxArray A and the MAT-file foo.mat.</p> <pre> program main integer matOpen, mxCreateFull, matClose integer mxGetPr, matPutMatrix integer a, mfp, stat double precision Areal(6) data Areal / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 / c mfp = matOpen('foo.mat', 'w') a = mxCreateFull(3, 2, 0) call mxCopyReal8ToPtr(Areal, mxGetPr(a), 6) call mxSetName(a, 'A') stat = matPutMatrix(mfp, a) stat = matClose(mfp) call mxFreeMatrix(a) c stop end</pre> |

# matGetMatrix

---

To test, run this program; then go to MATLAB and enter:

```
load foo
A
A =
 1 4
 2 5
 3 6
```

See `matdemo1.f` in the `eng_mat` subdirectory of the `examples` directory for another sample program that illustrates how to use this MAT-file routine in a Fortran program.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get next <code>mxArray</code> from MAT-file                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <code>integer*4 function matGetNextMatrix(mfp)</code><br><code>integer*4 mfp</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>      | <code>mfp</code><br>Pointer to MAT-file information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>    | <p>This routine allows you to step sequentially through a MAT-file and read all the <code>mxArrays</code> in a single pass.</p> <p><code>matGetNextMatrix</code> reads the next <code>mxArray</code> from the MAT-file pointed to by <code>mfp</code> and returns a pointer to a newly allocated <code>mxArray</code> structure. Use it immediately after opening the MAT-file with <code>matOpen</code> and not in conjunction with other MAT-file routines; otherwise, the concept of the <i>next</i> <code>mxArray</code> is undefined.</p> <p><code>matGetNextMatrix</code> returns 0 when the end-of-file is reached or if there is an error condition.</p> <p>Be careful in your code to free the <code>mxArray</code> created by this routine when you are finished with it.</p> |
| <b>Examples</b>       | Print out the row dimensions of all the <code>mxArrays</code> in a MAT-file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

```

program main
 integer matOpen, matClose, mxGetM
 integer mp, mfp, stat, matGetNextMatrix
c
 mh = matOpen('foo.mat', 'r')
 do 10 i = 1, 1000
 mp = matGetNextMatrix(mfp)
 if (mp .eq. 0) then
 go to 20
 end if
 write(6, *) 'Row dimension is ', mxGetM(mp)
 mxFreeMatrix(mp)
 10 continue
 20 continue
 stat = matClose(mfp)
c

```

## matGetNextMatrix

---

```
stop
end
```

See `matdemo2.f` in the `eng_mat` subdirectory of the `examples` directory for another sample program that illustrates how to use this MAT-file routine in a Fortran program.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copy character mxArray from MAT-files                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Fortran Syntax</b> | <pre>integer*4 function matGetString(mfp, name, str, strlen) integer*4 mfp, strlen character*(*) name, str</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>      | <p><b>mfp</b><br/>Pointer to MAT-file information.</p> <p><b>name</b><br/>Name of mxArray to get from MAT-file.</p> <p><b>str</b><br/>Character array to read from MAT-file.</p> <p><b>strlen</b><br/>Length of the character array.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b>    | <p><code>matGetString</code> reads the character mxArray with the specified name into <code>str</code> from the MAT-file <code>mfp</code>. It returns zero if successful, and a nonzero value if an error occurs.</p> <p><code>matGetString</code> copies the character array from mxArray <code>name</code> on file <code>mfp</code> into the character array <code>str</code>.</p> <p>Only up to <code>strlen</code> characters are copied, so ordinarily <code>strlen</code> is set to the dimension of the character array to prevent writing past the end of the array. If the character mxArray contains several rows, they are copied, one column at a time, into one long character array.</p> <p><code>matGetString</code> returns 0 if the copy is successful, and 1 if the copy has failed because the mxArray is not a character mxArray, 2 if the length of the character array exceeds <code>strlen</code>, and 3 if there is a file read error.</p> |
| <b>Example</b>        | <pre>program main integer matOpen, matClose, matPutString integer mfp, stat c mfp = matOpen('foo.mat', 'w') stat = matPutString(mfp, 'A', 'Hello, world') stat = matClose(mfp)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

# matGetString

---

```
c
 stop
end
```

Then you can go to MATLAB and enter:

```
load foo
A
A =
 Hello, world
```

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------------------------------------------------------------|----|-------------------------------------------------------------------------|
| <b>Purpose</b>        | Opens a MAT-file                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| <b>Fortran Syntax</b> | <pre>integer*4 function matOpen(filename, mode) integer*4 mfp character*(*) filename, mode</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| <b>Arguments</b>      | <p>filename<br/>Name of file to open.</p> <p>mfp<br/>Pointer to MAT-file information.</p> <p>mode<br/>File opening mode.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| <b>Description</b>    | <p>This routine allows you to open MAT-files for reading and writing.</p> <p>matOpen opens the named file and returns a file handle, or 0 if the open fails. Legal values for mode are</p> <hr/> <table> <tr> <td>r</td> <td>Opens file for reading only; determines the current version of the MAT-file by inspecting the files and preserves the current version.</td> </tr> <tr> <td>u</td> <td>Opens file for update, both reading and writing, but does not create the file if the file does not exist (equivalent to the r+ mode of fopen); determines the current version of the MAT-file by inspecting the files and preserves the current version.</td> </tr> <tr> <td>w</td> <td>Opens file for writing only; deletes previous contents, if any.</td> </tr> <tr> <td>w4</td> <td>Creates a MATLAB 4 MAT-file, rather than the default MATLAB 5 MAT-file.</td> </tr> </table> <hr/> | r | Opens file for reading only; determines the current version of the MAT-file by inspecting the files and preserves the current version. | u | Opens file for update, both reading and writing, but does not create the file if the file does not exist (equivalent to the r+ mode of fopen); determines the current version of the MAT-file by inspecting the files and preserves the current version. | w | Opens file for writing only; deletes previous contents, if any. | w4 | Creates a MATLAB 4 MAT-file, rather than the default MATLAB 5 MAT-file. |
| r                     | Opens file for reading only; determines the current version of the MAT-file by inspecting the files and preserves the current version.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| u                     | Opens file for update, both reading and writing, but does not create the file if the file does not exist (equivalent to the r+ mode of fopen); determines the current version of the MAT-file by inspecting the files and preserves the current version.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| w                     | Opens file for writing only; deletes previous contents, if any.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| w4                    | Creates a MATLAB 4 MAT-file, rather than the default MATLAB 5 MAT-file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |
| <b>Example</b>        | See matdemo1.f and matdemo2.f in the eng_mat subdirectory of the examples directory for sample programs that illustrate how to use the MATLAB MAT-file routines in a Fortran program.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |   |                                                                                                                                        |   |                                                                                                                                                                                                                                                          |   |                                                                 |    |                                                                         |

# matPutFull

---

**Purpose** Writes full `mxArrays` into MAT-files

**Fortran Syntax** `integer*4 function matPutFull(mfp, name, m, n, pr, pi)`  
`integer*4 mfp, m, n, pr, pi`  
`character*(*) name`

**Arguments**

`mfp`  
Pointer to MAT-file information.

`m`  
Row dimension.

`n`  
Column dimension.

`name`  
Name of `mxArray` to write to MAT-file.

`pi`  
Pointer to imaginary part.

`pr`  
Pointer to real part.

**Description** Most MATLAB applications work only with full (nonsparse) `mxArrays`. This routine provides an easy way to write a full `mxArray` into a MAT-file. It offers an alternative to `matPutMatrix`, which does not require use of the `mxArray` structure.

`matPutFull` writes the `mxArray` with dimensions `m`-by-`n`, real data `pr`, and imaginary data `pi` onto the MAT-file `mfp` with the specified name.

If the `mxArray` does not exist on the MAT-file, it is appended to the end. If an `mxArray` with the same name already exists in the file, the existing `mxArray` is replaced with the new `mxArray` by rewriting the file.

**Examples** Read the `mxArray` `A` from one MAT-file and write it out to another

```
program main
 integer matOpen, matClose, matPutFull, matGetFull
 integer mf1, mf2, stat
 integer m, n, pr, pi
```

```

mf1 = matOpen('foo.mat', 'r')
mf2 = matOpen('foo2.mat', 'w')
stat = matGetFull(mf1, 'A', m, n, pr, pi)
stat = matPutFull(mf2, 'A', m, n, pr, pi)
stat = matClose(mf1)
stat = matClose(mf2)
c
stop
end

```

Write a simple real mxArray into a MAT-file. Name the mxArray A and the MAT-file foo.mat.

```

integer matOpen, matClose, matPutFull
integer mfp, stat
double precision Areal(6)
data Areal / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 /
c
mfp = matOpen('foo.mat', 'w')
stat = matPutFull(mfp, 'A', 3, 2, Areal, 0)
stat = matClose(mfp)
c
stop
end

```

To test, run the second example; then go to MATLAB and enter:

```

load foo
A
A =
 1 4
 2 5
 3 6

```

# matPutMatrix

---

**Purpose** Writes mxArray into MAT-files

**Fortran Syntax** `integer*4 function matPutMatrix(mfp, mp)`  
`integer*4 mp, mfp`

**Arguments** `mfp`  
Pointer to MAT-file information.

`mp`  
mxArray pointer.

**Description** This routine allows you to put an mxArray into a MAT-file.

`matPutMatrix` writes mxArray `mp` to the MAT-file `mfp`. If the mxArray does not exist in the MAT-file, it is appended to the end. If an mxArray with the same name already exists in the file, the existing mxArray is replaced with the new mxArray by rewriting the file. The size of the new mxArray can be different than the existing mxArray.

`matPutMatrix` returns 0 if successful and nonzero if an error occurs.

Be careful in your code to free the mxArray created by this routine when you are finished with it.

**Examples** Write a simple 3-by-2 real mxArray into a MAT-file. Name the mxArray `A` and the MAT-file `foo.mat`.

```
program main
 integer matOpen, mxCreateFull, matClose
 integer mxGetPr, matPutMatrix
 integer a, mfp, stat
 double precision Areal(6)
 data Areal / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 /
c
 mfp = matOpen('foo.mat', 'w')
 a = mxCreateFull(3, 2, 0)
 call mxCopyReal8ToPtr(Areal, mxGetPr(a), 6)
 call mxSetName(a, 'A')
 stat = matPutMatrix(mfp, a)
 stat = matClose(mfp)
 call mxFreeMatrix(a)
```

```
c
 stop
end
```

To test, run this program; then go to MATLAB and enter:

```
load foo
A
A =
 1 4
 2 5
 3 6
```

See `matdemo1.f` in the `eng_mat` subdirectory of the `examples` directory for another sample program that illustrates how to use this MAT-file routine in a Fortran program.

# matPutString

---

**Purpose** Write character `mxArrays` into MAT-files

**Fortran Syntax** `integer*4 function matPutString(mfp, name, str)`  
`integer*4 mfp`  
`character*(*) name, str`

**Arguments**

`mfp`  
Pointer to MAT-file information.

`name`  
Name of `mxArray` to write to MAT-file.

`str`  
Character array to write to MAT-file.

**Description** `matPutString` writes the `mxArray` with the specified name and `str` to the MAT-file `mfp`. It returns 0 if successful, and 1 if an error occurs.

If the `mxArray` does not exist on the MAT-file, it is appended to the end. If an `mxArray` with the same name already exists in the file, the existing `mxArray` is replaced with the new `mxArray` by rewriting the file.

## Example

```
program main
 integer matOpen, matClose, matPutString
 integer mfp, stat
c
 mfp = matOpen('foo.mat', 'w')
 stat = matPutString(mfp, 'A', 'Hello, world')
 stat = matClose(mfp)
c
 stop
end
```

Then you can go to MATLAB and enter:

```
load foo
A
A =
 Hello, world
```

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Register a subroutine to be called when the MEX-file is cleared or when MATLAB terminates                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Fortran Syntax</b> | <pre>integer*4 function mexAtExit (ExitFcn) subroutine ExitFcn()</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>      | ExitFcn<br>The exit function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Returns</b>        | Always returns 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b>    | <p>Use <code>mexAtExit</code> to register a subroutine to be called just before the MEX-file is cleared or MATLAB is terminated. <code>mexAtExit</code> gives your MEX-file a chance to perform an orderly shutdown of anything under its control.</p> <p>Each MEX-file can register only one active exit subroutine at a time. If you call <code>mexAtExit</code> more than once, MATLAB uses the <code>ExitFcn</code> from the more recent <code>mexAtExit</code> call as the exit function.</p> <p>If a MEX-file is locked, all attempts to clear the MEX-file will fail. Consequently, if a user attempts to clear a locked MEX-file, MATLAB does not call the <code>ExitFcn</code>.</p> <p>You must declare the <code>ExitFcn</code> as <code>external</code> in the Fortran routine that calls <code>mexAtExit</code> if it is not within the scope of the file.</p> |
| <b>See Also</b>       | <code>mexSetTrapFlag</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

# mexCallMATLAB

---

- Purpose** Call a MATLAB function, a user-defined M-file, or MEX-file
- Fortran Syntax** `integer*4 function mexCallMATLAB(nlhs, plhs, nrhs, prhs, name)`  
`integer*4 nlhs, nrhs, plhs(*), prhs(*)`  
`character*(*) name`
- On the Alpha and SGI64 platforms, use:
- `integer*8 function mexCallMATLAB(nlhs, plhs, nrhs, prhs, name)`  
`integer*4 nlhs, nrhs`  
`integer*8 plhs(*), prhs(*)`  
`character*(*) name`
- Arguments**
- `nlhs`  
Number of desired output arguments. This value must be less than or equal to 50.
- `plhs`  
Array of `mxArray` pointers that can be used to access the returned data from the function call. Once the data is accessed, you can then call `mxFree` to free the `mxArray` pointer. By default, MATLAB frees the pointer and any associated dynamic memory it allocates when you return from the `mexFunction` call.
- `nrhs`  
Number of input arguments. This value must be less than or equal to 50.
- `prhs`  
Array of pointers to input data.
- `name`  
Character array containing the name of the MATLAB built-in, operator, M-file, or MEX-file that you are calling. If `name` is an operator, just place the operator inside a pair of single quotes; for example, `'+'`.
- Returns** 0 if successful, and a nonzero value if unsuccessful and `mexSetTrapFlag` was previously called.
- Description** Call `mexCallMATLAB` to invoke internal MATLAB functions, MATLAB operators, M-files, or other MEX-files.

By default, if name detects an error, MATLAB terminates the MEX-file and returns control to the MATLAB prompt. If you want a different error behavior, turn on the trap flag by calling `mexSetTrapFlag`.

**See Also**

`mexFunction`, `mexSetTrapFlag`

# mexErrMsgTxt

---

**Purpose** Issue error message and return to the MATLAB prompt

**Fortran Syntax** `subroutine mexErrMsgTxt(error_msg)`  
`character*(*) error_msg`

**Arguments** `error_msg`  
Character array containing the error message to be displayed.

**Description** Call `mexErrMsgTxt` to write an error message to the MATLAB window. After the error message prints, MATLAB terminates the MEX-file and returns control to the MATLAB prompt.

Calling `mexErrMsgTxt` does not clear the MEX-file from memory. Consequently, `mexErrMsgTxt` does not invoke any registered exit routine to allocate memory.

If your application calls `mxCall loc` or one of the `mxCreate` routines to create `mxArray` pointers, `mexErrMsgTxt` automatically frees any associated memory allocated by these calls.

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Execute a MATLAB command in the workspace of the caller                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Fortran Syntax</b> | <code>integer*4 function mexEvalString(command)</code><br><code>character*(*) command</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Arguments</b>      | <code>command</code><br>A character array containing the MATLAB command to execute.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Returns</b>        | 0 if successful, and a nonzero value if unsuccessful.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b>    | <p>Call <code>mexEvalString</code> to invoke a MATLAB command in the workspace of the caller.</p> <p><code>mexEvalString</code> and <code>mexCallMATLAB</code> both execute MATLAB commands. However, <code>mexCallMATLAB</code> provides a mechanism for returning results (left-hand side arguments) back to the MEX-file; <code>mexEvalString</code> provides no way for return values to be passed back to the MEX-file.</p> <p>All arguments that appear to the right of an equals sign in the <code>command</code> array must already be current variables of the caller's workspace.</p> |
| <b>See Also</b>       | <code>mexCallMATLAB</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

# mexFunction

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | MATLAB entry point to a Fortran MEX-file                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Fortran Syntax</b> | <pre>subroutine mexFunction(nlhs, plhs, nrhs, prhs) integer*4 nlhs, nrhs, plhs(*), prhs(*)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>      | <p><code>nlhs</code><br/>The number of expected outputs.</p> <p><code>plhs</code><br/>Array of pointers to expected outputs.</p> <p><code>nrhs</code><br/>The number of inputs.</p> <p><code>prhs</code><br/>Array of pointers to input data. The input data is read only and should not be altered by your <code>mexFunction</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b>    | <p><code>mexFunction</code> is not a routine you call. Rather, <code>mexFunction</code> is the name of a subroutine you must write in every MEX-file. When you invoke a MEX-file, MATLAB searches for a subroutine named <code>mexFunction</code> inside the MEX-file. If it finds one, then the first executable line in <code>mexFunction</code> becomes the starting point of the MEX-file. If MATLAB cannot find a subroutine named <code>mexFunction</code> inside the MEX-file, MATLAB issues an error message.</p> <p>When you invoke a MEX-file, MATLAB automatically loads <code>nlhs</code>, <code>plhs</code>, <code>nrhs</code>, and <code>prhs</code> with the caller's information. In the syntax of the MATLAB language, functions have the general form</p> $[a, b, c, \dots] = \text{fun}(d, e, f, \dots)$ <p>where the <code>...</code> denotes more items of the same format. The <code>a, b, c...</code> are left-hand side arguments and the <code>d, e, f...</code> are right-hand side arguments. The arguments <code>nlhs</code> and <code>nrhs</code> contain the number of left-hand side and right-hand side arguments, respectively, with which the MEX-function is called. <code>prhs</code> is an array of <code>mxArray</code> pointers whose length is <code>nrhs</code>. <code>plhs</code> is a pointer to an array whose length is <code>nlhs</code>, where your function must set pointers for the returned left-hand side <code>mxArrays</code>.</p> |

|                       |                                                                                                                                                                                                         |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the value of eps                                                                                                                                                                                    |
| <b>Fortran Syntax</b> | real *8 function mexGetEps()                                                                                                                                                                            |
| <b>Arguments</b>      | none                                                                                                                                                                                                    |
| <b>Returns</b>        | The value of MATLAB's eps variable.                                                                                                                                                                     |
| <b>Description</b>    | The eps variable holds the distance between 1.0 and the next largest floating-point number. It is a measure of floating-point accuracy. MATLAB's PINV and RANK function use eps as a default tolerance. |
| <b>See Also</b>       | mexGetInf, mexGetNaN                                                                                                                                                                                    |

# mexGetFull

---

**Purpose** Routine to get component parts of a double-precision `mxArray` into a Fortran workspace

**Fortran Syntax** `integer*4 function mexGetFull (name, m, n, pr, pi)`  
`integer*4 m, n, pr, pi`  
`character*(*) name`

**Arguments**

`m`  
Row dimension.

`n`  
Column dimension.

`name`  
Name of `mxArray` to get from workspace.

`pi`  
Pointer to imaginary part.

`pr`  
Pointer to real part.

**Returns** 0 if successful, and 1 otherwise.

**Description** `mexGetFull` provides a way to copy data from a double-precision `mxArray` from the caller's workspace. It is an alternative to `mexGetMatrix`, which does not require use of the `mxArray` structure.

`mexGetFull` reads the named `mxArray` from the caller's workspace and places the row dimensions, column dimensions, real array pointer, and imaginary array pointer into the locations specified by `m`, `n`, `pr`, and `pi`, respectively. You can then use `mxCopyPtrToReal8` to copy the data from the pointer into the Fortran workspace.

`mexGetFull` allocates memory for the real and imaginary arrays using `mxMalloc`; use `mxFree` to return it when you are done.

If the `mxArray` is purely real, the imaginary pointer is given 0.

**See Also** `mxGetName`, `mxGetPr`, `mxGetPi`

|                       |                                                                                                                  |
|-----------------------|------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get a pointer to an mxArray from MATLAB's global workspace                                                       |
| <b>Fortran Syntax</b> | <code>integer*4 function mexGetGlobal (name)</code><br><code>character*(*) name</code>                           |
| <b>Arguments</b>      | <code>name</code><br>Name of mxArray to get from workspace.                                                      |
| <b>Returns</b>        | Pointer to global mxArray if successful, or 0 if it doesn't exist.                                               |
| <b>Description</b>    | <code>mexGetGlobal</code> gets an mxArray from MATLAB's global workspace instead of from the caller's workspace. |
| <b>See Also</b>       | <code>mxGetName</code> , <code>mxGetPr</code> , <code>mxGetPi</code>                                             |

# mexGetInf

---

**Purpose** Get the value of infinity

**Fortran Syntax** `real *8 function mexGetInf()`

**Arguments** none

**Returns** The value of infinity on your system.

**Description** Call `mexGetInf` to return the value of the MATLAB internal `inf` variable. `inf` is a permanent variable representing IEEE arithmetic positive infinity. The value of `inf` is built in to the system; you cannot modify it.

Operations that return infinity include:

- Division by 0. For example, `5/0` returns infinity.
- Operations resulting in overflow. For example, `exp(100000)` returns infinity because the result is too large to be represented on your machine.

**See Also** `mexGetEps`, `mexGetNaN`

|                       |                                                                                                                                                              |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copies an mxArray from the caller's workspace                                                                                                                |
| <b>Fortran Syntax</b> | <code>integer*4 function mexGetMatrix(name)</code><br><code>character*(*) name</code>                                                                        |
| <b>Returns</b>        | A pointer to a newly allocated mxArray if successful, 0 otherwise.                                                                                           |
| <b>Arguments</b>      | <code>name</code><br>Name of mxArray to get from workspace.                                                                                                  |
| <b>Description</b>    | <code>mexGetMatrix</code> reads the named mxArray from the caller's workspace, and returns a pointer to a newly allocated mxArray or 0 if the attempt fails. |

# mexGetMatrixPtr

---

**Purpose** Get the pointer to an mxArray in the caller's workspace

**Fortran Syntax** `integer*4 function mexGetMatrixPtr(name)`  
`character*(*) name`

**Returns** A pointer to an mxArray owned by MATLAB.

**Arguments** `name`  
Name of mxArray to get from caller's workspace.

**Description** `mexGetMatrixPtr` returns a pointer to the mxArray with the specified name in the workspace local to the calling function. It allows you to read or modify variables in the MATLAB workspace directly from a MEX-file.

Do not free or reallocate the memory associated with any part of an mxArray obtained with the `mexGetMatrixPtr` function, including the real part, imaginary part, and sparse structure. mxArrays obtained with this function are managed by MATLAB's own internal mechanisms and MATLAB will crash immediately if you change them.

`mexGetMatrixPtr` is meant to be used to read values from an mxArray in the workspace or to change those values, provided the mxArray remains the same size, complexity, and sparsity.

To get the pointer of a global variable that is not defined as global by the calling function, first declare it global with a call of the form `mexEvalString("global varname")`.

|                       |                                                                                                                                                                                                                                                                                                                    |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the value of Not-a-Number                                                                                                                                                                                                                                                                                      |
| <b>Fortran Syntax</b> | real *8 function mexGetNaN()                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>      | none                                                                                                                                                                                                                                                                                                               |
| <b>Returns</b>        | MATLAB's value of Not-a-Number.                                                                                                                                                                                                                                                                                    |
| <b>Description</b>    | <p>Call <code>mexGetNaN</code> to return the value of NaN for MATLAB. NaN is the IEEE arithmetic representation for Not-a-Number. Certain mathematical operations return NaN as a result, for example:</p> <ul style="list-style-type: none"><li>• <code>0.0/0.0</code></li><li>• <code>inf - inf</code></li></ul> |
| <b>See Also</b>       | <code>mexGetEps</code> , <code>mexGetInf</code>                                                                                                                                                                                                                                                                    |

# mexIsFinite

---

|                       |                                                                                                                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Determine whether or not a value is finite                                                                                                                                                                        |
| <b>Fortran Syntax</b> | <code>integer*4 function mexIsFinite(value)</code><br><code>real*8 value</code>                                                                                                                                   |
| <b>Arguments</b>      | <code>value</code><br>The double-precision, floating-point number you are testing.                                                                                                                                |
| <b>Returns</b>        | <code>true</code> if value is finite; otherwise, returns <code>false</code> .                                                                                                                                     |
| <b>Description</b>    | Call <code>mexIsFinite</code> to determine whether or not value is finite. A number is finite if it is not equal to <ul style="list-style-type: none"><li>• <code>Inf</code></li><li>• <code>Nan</code></li></ul> |
| <b>See Also</b>       | <code>mexIsInf</code> , <code>mexIsNaN</code>                                                                                                                                                                     |

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Determine whether or not a value is infinite                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Fortran Syntax</b> | <code>integer*4 function mexIsInf(value)</code><br><code>real*8 value</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>      | <code>value</code><br>The double-precision, floating-point number you are testing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Returns</b>        | true if value is infinite; otherwise, returns false.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>    | <p>Call <code>mexIsInf</code> to determine whether or not <code>value</code> is equal to infinity. MATLAB stores the value of infinity in a permanent variable named <code>inf</code>, which represents IEEE arithmetic positive infinity. The value of <code>inf</code> is built in to the system; you cannot modify it.</p> <p>Operations that return infinity include</p> <ul style="list-style-type: none"><li>• Division by 0. For example, <code>5/0</code> returns infinity.</li><li>• Operations resulting in overflow. For example, <code>exp(10000)</code> returns infinity because the result is too large to be represented on your machine.</li></ul> <p>If <code>value</code> equals NaN (Not-a-Number), then <code>mexIsInf</code> returns false. In other words, NaN is not equal to infinity.</p> |
| <b>See Also</b>       | <code>mexIsFinite</code> , <code>mexIsNaN</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

# mexIsNaN

---

**Purpose** Determine whether or not a value is Not-a-Number (NaN)

**Fortran Syntax** integer\*4 function mexIsNaN(value)  
real\*8 value

**Arguments** value  
The double-precision, floating-point number you are testing.

**Returns** true if value is Not-a-Number; otherwise, returns false.

**Description** Call mexIsNaN to determine whether or not value is equal to NaN, the IEEE arithmetic representation for Not-a-Number. A NaN is obtained as a result of mathematically undefined operations such as

- 0.0/0.0
- inf - inf

**See Also** mexIsFinite, mexIsInf, mexGetInf

|                       |                                                                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Print a character array                                                                                                                                                            |
| <b>Fortran Syntax</b> | subroutine mexPrintf(format, arg1, arg2, ...)<br>character*(*) format                                                                                                              |
| <b>Arguments</b>      | format<br>Format character array for output.<br><br>arg1, arg2, ...<br>Optional arguments.                                                                                         |
| <b>Description</b>    | mexPrintf prints a character array on the screen and in the diary (if the diary is in use). It provides a call-back to the standard C printf routine already linked inside MATLAB. |
| <b>See Also</b>       | mexErrMsgTxt                                                                                                                                                                       |

# mexPutFull

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Routine to create an <code>mxArray</code> from its component parts into a Fortran workspace                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Fortran Syntax</b> | <pre>integer*4 function mexPutFull (name, m, n, pr, pi) integer*4 m, n, pr, pi character*(*) name</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Arguments</b>      | <p><code>m</code><br/>Row dimension.</p> <p><code>n</code><br/>Column dimension.</p> <p><code>name</code><br/>Name of <code>mxArray</code> to put into workspace.</p> <p><code>pi</code><br/>Pointer to imaginary part.</p> <p><code>pr</code><br/>Pointer to real part.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Returns</b>        | 0 if successful, and 1 otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>    | <p>Most MATLAB applications work only with full (nonsparse) <code>mxArrays</code>. <code>mexPutFull</code> provides an easy way to write a full <code>mxArray</code> into a MEX-file's caller's workspace. It is an alternative to <code>mexPutMatrix</code>, which requires use of the <code>mxArray</code> structure.</p> <p><code>mexPutFull</code> writes the <code>mxArray</code> with dimensions <code>m</code>-by-<code>n</code>, real data <code>pr</code>, and imaginary data <code>pi</code> into the calling workspace with the specified name. If an <code>mxArray</code> with the same name already exists in the workspace, the existing <code>mxArray</code> is replaced with the new one.</p> |
| <b>See Also</b>       | <code>mxSetName</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|                       |                                                                                                                                                                                                                                                                         |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Writes an mxArray to the caller's workspace                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <code>integer*4 function mexPutMatrix(mp)</code><br><code>integer*4 mp</code>                                                                                                                                                                                           |
| <b>Arguments</b>      | <code>mp</code><br>Pointer to mxArray.                                                                                                                                                                                                                                  |
| <b>Returns</b>        | 0 if successful, and 1 if an error occurs.                                                                                                                                                                                                                              |
| <b>Description</b>    | <code>mexPutMatrix</code> writes mxArray <code>mp</code> to the caller's workspace. If the mxArray does not exist in the workspace, it is created. If an mxArray with the same name already exists in the workspace, the existing mxArray is replaced with the new one. |

# mexSetTrapFlag

---

**Purpose** Control response of mexCall MATLAB to errors

**Fortran Syntax** `subroutine mexSetTrapFlag(trap_flag)`  
`integer*4 trap_flag`

**Arguments** `trap_flag`  
Control flag. Currently, the only legal values are:

- 0 On error, control returns to the MATLAB prompt.
- 1 On error, control returns to your MEX-file.

**Description** Call `mexSetTrapFlag` to control MATLAB's response to errors in `mexCall MATLAB`.

If you do not call `mexSetTrapFlag`, then whenever MATLAB detects an error in a call to `mexCall MATLAB`, MATLAB automatically terminates the MEX-file and returns control to the MATLAB prompt. Calling `mexSetTrapFlag` with `trap_flag` set to 0 is equivalent to not calling `mexSetTrapFlag` at all.

If you call `mexSetTrapFlag` and set the `trap_flag` to 1, then whenever MATLAB detects an error in a call to `mexCall MATLAB`, MATLAB does not automatically terminate the MEX-file. Rather, MATLAB returns control to the line in the MEX-file immediately following the call to `mexCall MATLAB`. The MEX-file is then responsible for taking an appropriate response to the error.

**See Also** `mexAtExit`, `mexErrMsgTxt`

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Allocate dynamic memory using MATLAB's memory manager                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Fortran Syntax</b> | <pre>integer*4 function mxCalloc(n, size) integer*4 n, size</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>      | <p><b>n</b><br/>Number of elements to allocate. This must be a nonnegative number.</p> <p><b>size</b><br/>Number of bytes per element.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>        | <p>A pointer to the start of the allocated dynamic memory, if successful. If unsuccessful in a stand-alone (nonMEX-file) application, <code>mxCalloc</code> returns 0. If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt.</p> <p><code>mxCalloc</code> is unsuccessful when there is insufficient free heap space.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b>    | <p>The MATLAB memory management facility maintains a list of all memory allocated by <code>mxCalloc</code> (and by the <code>mxCreate</code> calls). The MATLAB memory management facility automatically frees (deallocates) all of a MEX-file's parcels when control returns to the MATLAB prompt.</p> <p>By default, in a MEX-file, <code>mxCalloc</code> generates nonpersistent <code>mxCalloc</code> data. In other words, the memory management facility automatically deallocates the memory as soon as the MEX-file ends. When you finish using the memory allocated by <code>mxCalloc</code>, call <code>mxFree</code>. <code>mxFree</code> deallocates the memory.</p> <p><code>mxCalloc</code> works differently in MEX-files than in stand-alone MATLAB applications. In MEX-files, <code>mxCalloc</code> automatically</p> <ul style="list-style-type: none"> <li>• Allocates enough contiguous heap space to hold <code>n</code> elements.</li> <li>• Initializes all <code>n</code> elements to 0.</li> <li>• Registers the returned heap space with the MATLAB memory management facility.</li> </ul> <p>In stand-alone MATLAB applications, MATLAB's memory manager is not used.</p> |
| <b>See Also</b>       | <code>mxFree</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

# mxCopyCharacterToPtr

---

**Purpose** Copy character values from a Fortran array to a pointer array

**Fortran Syntax** `subroutine mxCopyCharacterToPtr(y, px, n)`  
`character*(*) y`  
`integer*4 px, n`

**Arguments**

`n`  
Number of elements to copy.

`px`  
Pointer to character or name array.

`y`  
Character Fortran array.

**Description** `mxCopyCharacterToPtr` copies `n` character values from the Fortran character array `y` into the MATLAB string array pointed to by `px`. This subroutine is essential for copying character data between MATLAB's pointer arrays and ordinary Fortran character arrays.

**See Also** `mxCopyPtrToCharacter`

|                       |                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copy COMPLEX*16 values from a Fortran array to a pointer array                                                                                                                                                                                                                                                                                              |
| <b>Fortran Syntax</b> | <pre>subroutine mxCopyComplex16ToPtr(y, pr, pi, n)   complex*16 y(n)   integer*4 pr, pi, n</pre>                                                                                                                                                                                                                                                            |
| <b>Arguments</b>      | <p>n<br/>Number of elements to copy.</p> <p>pi<br/>Pointer to pi array.</p> <p>pr<br/>Pointer to pr array.</p> <p>y<br/>COMPLEX*16 Fortran array.</p>                                                                                                                                                                                                       |
| <b>Description</b>    | <p>mxCopyComplex16ToPtr copies n COMPLEX*16 values from the Fortran COMPLEX*16 array y into the MATLAB arrays pointed to by pr and pi. This subroutine is essential for use with Fortran compilers that do not support the %VAL construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.</p> |
| <b>See Also</b>       | mxCopyPtrToComplex16                                                                                                                                                                                                                                                                                                                                        |

# mxCopyInteger4ToPtr

---

**Purpose** Copy INTEGER\*4 values from a Fortran array to a pointer array

**Fortran Syntax** `subroutine mxCopyInteger4ToPtr(y, px, n)`  
`integer*4 y(n)`  
`integer*4 px, n`

**Arguments**

`n`  
Number of elements to copy.

`px`  
Pointer to `i r` or `j c` array.

`y`  
INTEGER\*4 Fortran array.

**Description** `mxCopyInteger4ToPtr` copies `n` INTEGER\*4 values from the Fortran INTEGER\*4 array `y` into the MATLAB array pointed to by `px`, either an `i r` or `j c` array. This subroutine is essential for use with Fortran compilers that do not support the `%VAL` construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.

Note: This function can only be used with sparse matrices.

**See Also** `mxCopyPtrToInteger4`

|                       |                                                                                                                                                                                                                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copy character values from a pointer array to a Fortran array                                                                                                                                                                                                                                                     |
| <b>Fortran Syntax</b> | <pre>subroutine mxCopyPtrToCharacter(px, y, n) character*(*) y integer*4 px, n</pre>                                                                                                                                                                                                                              |
| <b>Arguments</b>      | <p>n<br/>Number of elements to copy.</p> <p>px<br/>Pointer to character or name array.</p> <p>y<br/>Character Fortran array.</p>                                                                                                                                                                                  |
| <b>Description</b>    | <p><code>mxCopyPtrToCharacter</code> copies <code>n</code> character values from the MATLAB array pointed to by <code>px</code> into the Fortran character array <code>y</code>. This subroutine is essential for copying character data from MATLAB's pointer arrays into ordinary Fortran character arrays.</p> |
| <b>See Also</b>       | <code>mxCopyCharacterToPtr</code>                                                                                                                                                                                                                                                                                 |

# mxCopyPtrToComplex16

---

**Purpose** Copy COMPLEX\*16 values from a pointer array to a Fortran array

**Fortran Syntax**

```
subroutine mxCopyPtrToComplex16(pr, pi, y, n)
 complex*16 y(n)
 integer*4 pr, pi, n
```

**Arguments**

**n**  
Number of elements to copy.

**pi**  
Pointer to pi array.

**pr**  
Pointer to pr array.

**y**  
COMPLEX\*16 Fortran array.

**Description** `mxCopyPtrToComplex16` copies `n` COMPLEX\*16 values from the MATLAB arrays pointed to by `pr` and `pi` into the Fortran COMPLEX\*16 array `y`. This subroutine is essential for use with Fortran compilers that do not support the %VAL construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.

**See Also** `mxCopyComplex16ToPtr`

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copy INTEGER*4 values from a pointer array to a Fortran array                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Fortran Syntax</b> | <pre>subroutine mxCopyPtrToInteger4(px, y, n) integer*4 y(n) integer*4 px, n</pre>                                                                                                                                                                                                                                                                                                                                                              |
| <b>Arguments</b>      | <p>n<br/>Number of elements to copy.</p> <p>px<br/>Pointer to i r or j c array.</p> <p>y<br/>INTEGER*4 Fortran array.</p>                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>    | <p>mxCopyPtrToInteger4 copies n INTEGER*4 values from the MATLAB array pointed to by px, either an i r or j c array, into the Fortran INTEGER*4 array y. This subroutine is essential for use with Fortran compilers that do not support the %VAL construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.</p> <p>Note: This function can only be used with sparse matrices.</p> |
| <b>See Also</b>       | mxCopyInteger4ToPtr                                                                                                                                                                                                                                                                                                                                                                                                                             |

# mxCopyPtrToPtrArray

---

**Purpose** Copy pointer to a Fortran array

**Fortran Syntax** `subroutine mxCopyPtrToInteger4(px, y, n)`  
`integer*4 y(n)`  
`integer*4 px, n`

**Arguments**

`n`  
Number of elements to copy.

`px`  
Pointer to `i r` or `j c` array.

`y`  
`INTEGER*4` Fortran array.

**Description** `mxCopyPtrToPtrArray` copies a pointer to a Fortran array. This subroutine is essential for use with Fortran compilers that do not support the `%VAL` construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.

**See Also** `mxCopyInteger4ToPtr`

|                       |                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Copy REAL*8 values from a pointer array to a Fortran array                                                                                                                                                                                                                                                                                                 |
| <b>Fortran Syntax</b> | <pre>subroutine mxCopyPtrToReal8(px, y, n) real*8 y(n) integer*4 px, n</pre>                                                                                                                                                                                                                                                                               |
| <b>Arguments</b>      | <p>n<br/>Number of elements to copy.</p> <p>px<br/>Pointer to pr or pi array.</p> <p>y<br/>REAL*8 Fortran array.</p>                                                                                                                                                                                                                                       |
| <b>Description</b>    | mxCopyPtrToReal8 copies n REAL*8 values from the MATLAB array pointed to by px, either a pr or pi array, into the Fortran REAL*8 array y. This subroutine is essential for use with Fortran compilers that do not support the %VAL construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file. |
| <b>See Also</b>       | mxCopyReal8ToPtr                                                                                                                                                                                                                                                                                                                                           |

# mxCopyReal8ToPtr

---

**Purpose** Copy REAL\*8 values from a Fortran array to a pointer array

**Fortran Syntax** `subroutine mxCopyReal8ToPtr(y, px, n)`  
`real*8 y(n)`  
`integer*4 px, n`

**Arguments**

`n`  
Number of elements to copy.

`px`  
Pointer to pr or pi array.

`y`  
REAL\*8 Fortran array.

**Description** `mxCopyReal8ToPtr(y, px, n)` copies `n` REAL\*8 values from the Fortran REAL\*8 array `y` into the MATLAB array pointed to by `px`, either a `pr` or `pi` array. This subroutine is essential for use with Fortran compilers that do not support the `%VAL` construct in order to set up standard Fortran arrays for passing as arguments to the computation routine of a MEX-file.

**See Also** `mxCopyPtrToReal8`

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Create an unpopulated 2-dimensional mxArray                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Fortran Syntax</b> | <pre>integer*4 function mxCreateFull(m, n, ComplexFlag) integer*4 m, n, ComplexFlag</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>      | <p><b>m</b><br/>The desired number of rows.</p> <p><b>n</b><br/>The desired number of columns.</p> <p><b>ComplexFlag</b><br/>Specify <code>REAL = 0</code> if the data has no imaginary components; specify <code>COMPLEX = 1</code> if the data has some imaginary components.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Returns</b>        | An unpopulated, m-by-n mxArray if successful, 0 otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>    | <p>Use <code>mxCreateFull</code> to create an unpopulated mxArray of size m-by-n. <code>mxCreateFull</code> initializes each element in the <code>pr</code> array to 0. If you set <code>ComplexFlag</code> to 1, <code>mxCreateFull</code> also initializes each element in the <code>pi</code> array to 0.</p> <p>If you specify <code>REAL = 0</code>, <code>mxCreateFull</code> allocates enough memory to hold m-by-n real elements. If you specify <code>COMPLEX = 1</code>, <code>mxCreateFull</code> allocates enough memory to hold m-by-n real elements and m-by-n imaginary elements.</p> <p>Call <code>mxFreeMatrix</code> when you finish using the mxArray. <code>mxFreeMatrix</code> deallocates the mxArray and its associated real and complex elements.</p> |
| <b>See Also</b>       | <code>mxCreateSparse</code> , <code>mxFreeMatrix</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

# mxCreateSparse

---

**Purpose** Create a 2-dimensional unpopulated sparse mxArray

**Fortran Syntax** `integer*4 function mxCreateSparse(m, n, nzmax, ComplexFlag)`  
`integer*4 m, n, nzmax, ComplexFlag`

**Arguments** `m`  
The desired number of rows.

`n`  
The desired number of columns.

`nzmax`  
The number of elements that `mxCreateSparse` should allocate to hold the `pr`, `ir`, and, if `COMPLEX = 1`, `pi` arrays. Set the value of `nzmax` to be greater than or equal to the number of nonzero elements you plan to put into the mxArray, but make sure that `nzmax` is less than or equal to  $m*n$ .

`ComplexFlag`  
Specify `REAL = 0` if the data has no imaginary components; specify `COMPLEX = 1` if the data has some imaginary components.

**Returns** An unpopulated, sparse mxArray if successful, 0 otherwise.

**Description** Call `mxCreateSparse` to create an unpopulated sparse mxArray. The returned sparse mxArray contains no sparse information and cannot be passed as an argument to any MATLAB sparse functions. In order to make the returned sparse mxArray useful, you must initialize the `pr`, `ir`, `jc`, and (if it exists) `pi` array.

`mxCreateSparse` allocates space for

- A `pr` array of  $m$ -by- $n$  elements.
- A `pi` array of  $m$ -by- $n$  elements (but only if `ComplexFlag` is `COMPLEX = 1`).
- An `ir` array of `nzmax` elements.
- A `jc` array of  $m$  elements.

When you finish using the sparse mxArray, call `mxFreeMatrix` to reclaim all its heap space.

**See Also** `mxFreeMatrix`, `mxSetNzmax`, `mxSetPr`, `mxSetIr`, `mxSetJc`

|                       |                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Create a 1-by-n character array initialized to the specified string                                                                                                                                                                                                                                                                                                                                     |
| <b>Fortran Syntax</b> | <code>integer*4 function mxCreateString(str)</code><br><code>character*(*) str</code>                                                                                                                                                                                                                                                                                                                   |
| <b>Returns</b>        | A character array initialized to <code>str</code> if successful, 0 otherwise.                                                                                                                                                                                                                                                                                                                           |
| <b>Arguments</b>      | <code>str</code><br>The string that is to serve as the <code>mxArray</code> 's initial data.                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>    | Use <code>mxCreateString</code> to create a character <code>mxArray</code> initialized to <code>str</code> . Many MATLAB functions (for example, <code>strcmp</code> and <code>upper</code> ) require character <code>mxArray</code> inputs.<br><br>Free the character <code>mxArray</code> when you are finished using it. To free a character <code>mxArray</code> , call <code>mxFreeMatrix</code> . |

# mxFree

---

**Purpose** Free dynamic memory allocated by `mxCall oc`

**Fortran Syntax** `subroutine mxFree(ptr)`  
`integer*4 ptr`

**Arguments** `ptr`  
Pointer to the beginning of any memory parcel allocated by `mxCall oc`.

**Description** `mxFree` deallocates heap space. `mxFree` frees memory using MATLAB's own memory management facility. This ensures correct memory management in error and abort (**Ctrl-C**) conditions.

`mxFree` works differently in MEX-files than in stand-alone MATLAB applications. With MEX-files, `mxFree` returns to the heap any memory allocated using `mxCall oc`. If you do not free memory with this command, MATLAB frees it automatically on return from the MEX-file. In stand-alone MATLAB applications, you have to explicitly free memory, and MATLAB memory management is not used.

In a MEX-file, your use of `mxFree` depends on whether the specified memory parcel is persistent or nonpersistent. By default, memory parcels created by `mxCall oc` are nonpersistent.

The MATLAB memory management facility automatically frees all nonpersistent memory whenever a MEX-file completes. Thus, even if you do not call `mxFree`, MATLAB takes care of freeing the memory for you. Nevertheless, it is a good programming practice to deallocate memory just as soon as you are through using it. Doing so generally makes the entire system run more efficiently.

When a MEX-file completes, the MATLAB memory management facility does not free persistent memory parcels. Therefore, the only way to free a persistent memory parcel is to call `mxFree`. Typically, MEX-files call `mexAtExit` to register a clean-up handler. Then, the clean-up handler calls `mxFree`.

**See Also** `mxCall oc`, `mxFreeMatrix`

|                       |                                                                                                                                                                                                                              |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Free dynamic memory allocated by <code>mxCreateFull</code> and <code>mxCreateSparse</code>                                                                                                                                   |
| <b>Fortran Syntax</b> | <pre>subroutine mxFreeMatrix(pm) integer*4 pm</pre>                                                                                                                                                                          |
| <b>Arguments</b>      | <p><code>pm</code><br/>Pointer to the beginning of the <code>mxArray</code>.</p>                                                                                                                                             |
| <b>Description</b>    | <code>mxFreeMatrix</code> returns an <code>mxArray</code> to the heap for reuse, freeing any arrays ( <code>pr</code> , <code>pi</code> , <code>ir</code> , or <code>jc</code> ) allocated within the <code>mxArray</code> . |
| <b>See Also</b>       | <code>mxMalloc</code> , <code>mxFree</code>                                                                                                                                                                                  |

# mxGetIr

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the <code>i r</code> array                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetIr(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Returns</b>        | A pointer to the first element in the <code>i r</code> array, if successful. Otherwise, returns 0. Possible causes of failure include <ul style="list-style-type: none"><li>• Specifying a full (nonsparse) <code>mxArray</code>.</li><li>• An earlier call to <code>mxCreateSparse</code> failed.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>    | Use <code>mxGetIr</code> to obtain the starting address of the <code>i r</code> array. The <code>i r</code> array is an array of integers; the length of the <code>i r</code> array is typically <code>nzmax</code> values. For example, if <code>nzmax</code> equals 100, then the <code>i r</code> array should contain 100 integers.<br><br>Each value in an <code>i r</code> array indicates a row (offset by 1) at which a nonzero element can be found. (The <code>j c</code> array is an index that indirectly specifies a column where nonzero elements can be found.)<br><br>For details on the <code>i r</code> and <code>j c</code> arrays, see <code>mxSetIr</code> and <code>mxSetJc</code> . |
| <b>See Also</b>       | <code>mxGetJc</code> , <code>mxGetNzmax</code> , <code>mxSetIr</code> , <code>mxSetJc</code> , <code>mxSetNzmax</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the <code>j c</code> array                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetJc(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Returns</b>        | A pointer to the first element in the <code>j c</code> array, if successful; otherwise, returns 0. The most likely cause of failure is specifying a pointer that points to a full (nonsparse) <code>mxArray</code> .                                                                                                                                                                                                                             |
| <b>Description</b>    | Use <code>mxGetJc</code> to obtain the starting address of the <code>j c</code> array. The <code>j c</code> array is an integer array having <code>n+1</code> elements where <code>n</code> is the number of columns in the sparse <code>mxArray</code> . The values in the <code>j c</code> array indirectly indicate columns containing nonzero elements. For a detailed explanation of the <code>j c</code> array, see <code>mxSetJc</code> . |
| <b>See Also</b>       | <code>mxGetIr</code> , <code>mxSetIr</code> , <code>mxSetJc</code>                                                                                                                                                                                                                                                                                                                                                                               |

# mxGetM

---

**Purpose**                Get the number of rows

**Fortran Syntax**    integer\*4 function mxGetM(pm)  
                     integer\*4 pm

**Arguments**            pm  
                     Pointer to an array.

**Returns**                The number of rows in the mxArray to which pm points.

**Description**           mxGetM returns the number of rows in the specified array.

**See Also**                mxGetN, mxSetM, mxSetN

---

|                       |                                                                                                                                                                                                                                                                  |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the total number of columns                                                                                                                                                                                                                                  |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetN(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                          |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                          |
| <b>Returns</b>        | The number of columns in the <code>mxArray</code> .                                                                                                                                                                                                              |
| <b>Description</b>    | Call <code>mxGetN</code> to determine the number of columns in the specified <code>mxArray</code> .<br>If <code>pm</code> points to a sparse <code>mxArray</code> , <code>mxGetN</code> still returns the number of columns, not the number of occupied columns. |
| <b>See Also</b>       | <code>mxGetM</code> , <code>mxSetM</code> , <code>mxSetN</code>                                                                                                                                                                                                  |

# mxGetName

---

|                       |                                                                                                                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the name of the specified <code>mxArray</code>                                                                                                                                               |
| <b>Fortran Syntax</b> | <code>character*32 function mxGetName(pm)</code><br><code>integer*4 pm</code>                                                                                                                    |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                          |
| <b>Returns</b>        | A pointer to the start of the name field. If the <code>mxArray</code> has no name, <code>mxGetName</code> returns 0.                                                                             |
| <b>Description</b>    | Use <code>mxGetName</code> to determine the name of the <code>mxArray</code> that <code>pm</code> points to. The returned <code>mxArray</code> name is a character array with maximum length 32. |
| <b>See Also</b>       | <code>mxSetName</code>                                                                                                                                                                           |

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the number of elements in the <code>i r</code> , <code>pr</code> , and (if it exists) <code>pi</code> arrays                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetNzmax(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to a sparse <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Returns</b>        | The number of elements allocated to hold nonzero entries in the specified sparse <code>mxArray</code> , on success. Returns an indeterminate value on error. The most likely cause of failure is that <code>pm</code> points to a full (nonsparse) <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b>    | <p>Use <code>mxGetNzmax</code> to get the value of the <code>nzmax</code> field. The <code>nzmax</code> field holds an integer value that signifies the number of elements in the <code>i r</code>, <code>pr</code>, and, if it exists, the <code>pi</code> arrays. The value of <code>nzmax</code> is always greater than or equal to the number of nonzero elements in a sparse <code>mxArray</code>. In addition, the value of <code>nzmax</code> is always less than or equal to the number of rows times the number of columns.</p> <p>As you adjust the number of nonzero elements in a sparse <code>mxArray</code>, MATLAB often adjusts the value of the <code>nzmax</code> field. MATLAB adjusts <code>nzmax</code> in order to reduce the number of costly reallocations and in order to optimize its use of heap space.</p> |
| <b>See Also</b>       | <code>mxSetNzmax</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

# mxGetPi

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get an <code>mxArray</code> 's imaginary data elements                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <pre>integer*4 function mxGetPi (pm) integer*4 pm</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>        | The imaginary data elements of the specified <code>mxArray</code> , on success. Returns 0 if there is no imaginary data or if there is an error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b>    | <p>The <code>pi</code> field points to an array containing the imaginary data of the <code>mxArray</code>. Call <code>mxGetPi</code> to get the contents of the <code>pi</code> field; that is, to get the starting address of this imaginary data.</p> <p>The best way to determine if an <code>mxArray</code> is purely real is to call <code>mxIsComplex</code>.</p> <p>The imaginary parts of all input <code>mxArrays</code> to a MATLAB function are allocated if any of the input <code>mxArrays</code> is complex.</p> <p>If you use <code>mxGetPr</code> or <code>mxGetPi</code>, note that <code>mxFreeMatrix</code> frees <code>pr</code> and <code>pi</code> using <code>mxFree</code>, so <code>pr</code> and <code>pi</code> should only be set to memory allocated with <code>mxCallLoc</code>.</p> |
| <b>See Also</b>       | <code>mxGetPr</code> , <code>mxSetPi</code> , <code>mxSetPr</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get an <code>mxArray</code> 's real data elements                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetPr(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Returns</b>        | The address of the first element of the real data. Returns 0 if there is no real data.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b>    | Call <code>mxGetPr</code> to determine the starting address of the real data in the <code>mxArray</code> that <code>pm</code> points to. Once you have the starting address, it is fairly easy to access any other element in the <code>mxArray</code> .<br><br>If you use <code>mxGetPr</code> or <code>mxGetPi</code> , note that <code>mxFreeMatrix</code> frees <code>pr</code> and <code>pi</code> using <code>mxFree</code> , so <code>pr</code> and <code>pi</code> should only be set to memory allocated with <code>mxCall oc</code> . |
| <b>See Also</b>       | <code>mxGetPi</code> , <code>mxSetPi</code> , <code>mxSetPr</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

# mxGetScalar

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Get the real component of an <code>mxArray</code> 's first data element                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Fortran Syntax</b> | <pre>real *8 functi on mxGetScal ar(pm) integer*4 pm</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Returns</b>        | <p>The value of the first real (nonimaginary) element of the <code>mxArray</code>. If the <code>mxArray</code> is larger than 1-by-1, <code>mxGetScal ar</code> returns the value of the (1, 1) element.</p> <p>If <code>pm</code> points to a sparse <code>mxArray</code>, <code>mxGetScal ar</code> returns the value of the first nonzero real element in the <code>mxArray</code>.</p> <p>If <code>pm</code> points to an empty <code>mxArray</code>, <code>mxGetScal ar</code> returns an indeterminate value.</p>                                                                                                                                                        |
| <b>Description</b>    | <p>Call <code>mxGetScal ar</code> to get the value of the first real (nonimaginary) element of the <code>mxArray</code>.</p> <p>In most cases, you call <code>mxGetScal ar</code> when <code>pm</code> points to an <code>mxArray</code> containing only one element (a scalar). However, <code>pm</code> can point to an <code>mxArray</code> containing many elements. If <code>pm</code> points to an <code>mxArray</code> containing multiple elements, <code>mxGetScal ar</code> returns the value of the first real element. If <code>pm</code> points to a two-dimensional <code>mxArray</code>, <code>mxGetScal ar</code> returns the value of the (1, 1) element.</p> |
| <b>See Also</b>       | <code>mxGetM</code> , <code>mxGetN</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Create a character array from an <code>mxArray</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Fortran Syntax</b> | <code>integer*4 function mxGetString(pm, str, strlen)</code><br><code>integer*4 pm, strlen</code><br><code>character*(*) str</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .<br><br><code>str</code><br>Fortran character array.<br><br><code>strlen</code><br>Number of characters to retrieve from the <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Returns</b>        | The character array, on success. Returns 0 otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b>    | Call <code>mxGetString</code> to copy a character array from an <code>mxArray</code> . <code>mxGetString</code> copies and converts the character array from the <code>mxArray</code> <code>pm</code> into the character array <code>str</code> . Storage space for character array <code>str</code> must be allocated previously.<br><br>Only up to <code>strlen</code> characters are copied, so ordinarily, <code>strlen</code> is set to the dimension of the character array to prevent writing past the end of the array. Check the length of the character array in advance using <code>mxGetM</code> and <code>mxGetN</code> . If the character array contains several rows, they are copied, one column at a time, into one long character array. |
| <b>See Also</b>       | <code>mxCalloc</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

# mxIsComplex

---

**Purpose** Inquire if an mxArray is complex

**Fortran Syntax** `integer*4 function mxIsComplex(pm)`  
`integer*4 pm`

**Arguments** `pm`  
Pointer to an mxArray.

**Returns** 1 if complex, 0 otherwise.

**Description** Use `mxIsComplex` to determine whether or not an imaginary part is allocated for an mxArray. The imaginary pointer `pi` is 0 if an mxArray is purely real and does not have any imaginary data. If an mxArray is complex, `pi` points to an array of numbers.

When a MEX-file is called, MATLAB automatically examines all the input (right-hand side) arrays. If any input array is complex, then MATLAB automatically allocates memory to hold imaginary data for all other input arrays. For example, suppose a user passes three input variables (`apricot`, `banana`, and `carambola`) to a MEX-file named `Jest`:

```
>> apricot = 7;
>> banana = sqrt(-5:5);
>> carambola = magic(2);
>> Jest(apricot, banana, carambola);
```

`banana` is complex. Therefore, even though array `apricot` is purely real, MATLAB automatically allocates space (one element) to hold an imaginary value of `apricot`. MATLAB also automatically allocates space (four elements) to hold the nonexistent imaginary values of `carambola`.

In other words, MATLAB forces every input array to be real or every input array to be complex.

**See Also** `mxIsNumeric`

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Inquire if an <code>mxArray</code> is of type <code>double</code>                                                                                                                                                                                                                                                                                                                                             |
| <b>Fortran Syntax</b> | <code>integer*4</code> <code>function</code> <code>mxIsDouble(pm)</code><br><code>integer*4</code> <code>pm</code>                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                       |
| <b>Returns</b>        | 1 if true, 0 if false. If <code>mxIsDouble</code> returns 0, the array has no Fortran access functions and your Fortran program cannot use it.                                                                                                                                                                                                                                                                |
| <b>Description</b>    | Call <code>mxIsDouble</code> to determine whether or not the specified <code>mxArray</code> represents its real and imaginary data as double-precision, floating-point numbers.<br><br>Older versions of MATLAB store all <code>mxArray</code> data as double-precision, floating-point numbers. However, starting with MATLAB 5, MATLAB can store real and imaginary data in a variety of numerical formats. |

# mxIsFull

---

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Inquire if an mxArray is full                                                                |
| <b>Fortran Syntax</b> | <code>integer*4 function mxIsFull (pm)</code><br><code>integer*4 pm</code>                   |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an mxArray.                                                    |
| <b>Returns</b>        | 1 if the mxArray is full, 0 if it is sparse.                                                 |
| <b>Description</b>    | Call <code>mxIsFull</code> to determine if an mxArray is stored in full form or sparse form. |

|                       |                                                                                                              |
|-----------------------|--------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Inquire if an <code>mxArray</code> contains numeric data                                                     |
| <b>Fortran Syntax</b> | <code>integer*4 function mxIsNumeric(pm)</code><br><code>integer*4 pm</code>                                 |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                      |
| <b>Returns</b>        | 1 if the <code>mxArray</code> contains numeric data, 0 otherwise.                                            |
| <b>Description</b>    | Call <code>mxIsNumeric</code> to inquire whether or not the <code>mxArray</code> contains a character array. |
| <b>See Also</b>       | <code>mxIsString</code>                                                                                      |

# mxIsSparse

---

**Purpose** Inquire if an mxArray is sparse

**Fortran Syntax** integer\*4 function mxIsSparse(pm)  
integer\*4 pm

**Arguments** pm  
Pointer to an mxArray.

**Returns** 1 if the mxArray is sparse, 0 otherwise.

**Description** Use mxIsSparse to determine if an mxArray is stored in sparse form. Many routines (for example, mxGetIr and mxGetJc) require a sparse mxArray as input.

There are no corresponding set routines. Use mxCreateSparse to create sparse mxArrays.

**See Also** mxGetIr, mxGetJc, mxIsFull

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Inquire if an <code>mxArray</code> contains a character array                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Fortran Syntax</b> | <code>integer*4 function mxIsString(pm)</code><br><code>integer*4 pm</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>      | <code>pm</code><br>Pointer to an <code>mxArray</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Returns</b>        | 1 if the <code>mxArray</code> contains a character array, 0 otherwise.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>    | <p>Call <code>mxIsString</code> to inquire whether or not the <code>mxArray</code> contains a character array. The <code>DisplayMode</code> flag tells MATLAB whether to display the <code>mxArray</code> in numeric form or to interpret the elements as ASCII values and to display the <code>mxArray</code> as a character array, if the semicolon is omitted from a MATLAB statement.</p> <p>Use <code>mxGetString</code> and <code>mxCreateString</code> to extract and insert character arrays into <code>mxArrays</code>.</p> |
| <b>See Also</b>       | <code>mxCreateString</code> , <code>mxGetString</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

# mxSetIr

---

**Purpose** Set the `ir` array of a sparse `mxArray`

**Fortran Syntax** `subroutine mxSetIr(pm, ir)`  
`integer*4 pm, ir`

**Arguments** `pm`  
Pointer to a sparse `mxArray`.

`ir`  
Pointer to the `ir` array. The `ir` array must be sorted in column-major order.

**Description** Use `mxSetIr` to specify the `ir` array of a sparse `mxArray`. The `ir` array is an array of integers; the length of the `ir` array should equal the value of `nzmax`. Each element in the `ir` array indicates a row (offset by 1) at which a nonzero element can be found. (The `jc` array is an index that indirectly specifies a column where nonzero elements can be found. See `mxSetJc` for more details on `jc`.)

The `ir` array must be in column-major order. That means that the `ir` array must define the row positions in column 1 (if any) first, then the row positions in column 2 (if any) second, and so on through column `N`. Within each column, row position 1 must appear prior to row position 2, and so on.

`mxSetIr` does not sort the `ir` array for you; you must specify an `ir` array that is already sorted.

**See Also** `mxCreateSparse`, `mxGetIr`, `mxGetJc`, `mxSetJc`

---

|                       |                                                                                                                                                                                                                                                                         |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Set the <code>j c</code> array of a sparse <code>mxArray</code>                                                                                                                                                                                                         |
| <b>Fortran Syntax</b> | <pre>subroutine mxSetJc(pm, j c) integer*4 pm, j c</pre>                                                                                                                                                                                                                |
| <b>Arguments</b>      | <p><code>pm</code><br/>Pointer to a sparse <code>mxArray</code>.</p> <p><code>j c</code><br/>Pointer to the <code>j c</code> array.</p>                                                                                                                                 |
| <b>Description</b>    | Use <code>mxSetJc</code> to specify a new <code>j c</code> array for a sparse <code>mxArray</code> . The <code>j c</code> array is an integer array having <code>n+1</code> elements where <code>n</code> is the number of columns in the sparse <code>mxArray</code> . |
| <b>See Also</b>       | <code>mxGetIr</code> , <code>mxGetJc</code> , <code>mxSetIr</code>                                                                                                                                                                                                      |

# mxSetM

---

**Purpose** Set the number of rows

**Fortran Syntax** `subroutine mxSetM(pm, m)`  
`integer*4 pm, m`

**Arguments** `m`  
The desired number of rows.

`pm`  
Pointer to an `mxArray`.

**Description** Call `mxSetM` to set the number of rows in the specified `mxArray`. Call `mxSetN` to set the number of columns.

You typically use `mxSetM` to change the shape of an existing `mxArray`. Note that `mxSetM` does not allocate or deallocate any space for the `pr`, `pi`, `ir`, or `jc` arrays. Consequently, if your calls to `mxSetM` and `mxSetN` increase the number of elements in the `mxArray`, then you must enlarge the `pr`, `pi`, `ir`, and/or `jc` arrays.

If your calls to `mxSetM` and `mxSetN` end up reducing the number of elements in the array, then you can optionally reduce the sizes of the `pr`, `pi`, `ir`, and/or `jc` arrays in order to use heap space more efficiently.

**See Also** `mxGetM`, `mxGetN`, `mxSetN`

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Set the number of columns                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Fortran Syntax</b> | <pre>subroutine mxSetN(pm, n) integer*4 pm, n</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Arguments</b>      | <p><b>pm</b><br/>Pointer to an mxArray.</p> <p><b>n</b><br/>The desired number of columns.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b>    | <p>Call <code>mxSetN</code> to set the number of columns in the specified mxArray.</p> <p>You typically use <code>mxSetN</code> to change the shape of an existing mxArray. Note that <code>mxSetN</code> does not allocate or deallocate any space for the <code>pr</code>, <code>pi</code>, <code>ir</code>, or <code>jc</code> arrays. Consequently, if your calls to <code>mxSetN</code> and <code>mxSetM</code> increase the number of elements in the mxArray, then you must enlarge the <code>pr</code>, <code>pi</code>, <code>ir</code>, and/or <code>jc</code> arrays.</p> <p>If your calls to <code>mxSetM</code> and <code>mxSetN</code> end up reducing the number of elements in the mxArray, then you may want to reduce the size of the <code>pr</code>, <code>pi</code>, <code>ir</code>, or <code>jc</code> arrays in order to reduce heap space usage. However, reducing the size is not mandatory.</p> |
| <b>See Also</b>       | <code>mxGetM</code> , <code>mxGetN</code> , <code>mxSetM</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

# mxSetName

---

**Purpose** Set the name of an mxArray

**Fortran Syntax** `subroutine mxSetName(pm, name)`  
`integer*4 pm`  
`character*(32) name`

**Arguments**

`pm`  
Pointer to an mxArray.

`name`  
The name you are assigning to the mxArray. The specified name can be up to 32 characters. If you specify a name longer than 32 characters, mxSetName assigns only the first 32 characters to the name.

**Description** Call mxSetName to establish a name for an mxArray or to change an existing name.

mxSetName assigns the characters in name to a fixed-width section of memory. Do not deallocate this memory.

**See Also** mxGetName

**Purpose** Set the storage space for nonzero elements

**Fortran Syntax** `subroutine mxSetNzmax(pm, nzmax)`  
`integer*4 pm, nzmax`

**Arguments** `pm`  
 Pointer to a sparse `mxArray`.

`nzmax`  
 The number of elements that `mxCreateSparse` should allocate to hold the arrays pointed to by `ir`, `pr`, and `pi` (if it exists). You should set `nzmax` greater than or equal to the number of nonzero elements in the `mxArray`, but you should set it to be less than or equal to the number of rows times the number of columns. If you specify an `nzmax` value of 0, `mxSetNzmax` sets the value of `nzmax` to 1.

**Description** Use `mxSetNzmax` to assign a new value to the `nzmax` field of the specified sparse `mxArray`. The `nzmax` field holds the maximum possible number of nonzero elements in the sparse `mxArray`.

The number of elements in the `ir`, `pr`, and `pi` (if it exists) arrays must be equal to `nzmax`. Therefore, after calling `mxSetNzmax`, you must change the size of the `ir`, `pr`, and `pi` arrays.

How big should `nzmax` be? One philosophy is that you should set `nzmax` equal to or slightly greater than the number of nonzero elements in a sparse `mxArray`. This philosophy conserves precious heap space. Another philosophy is that you should make `nzmax` equal to the total number of elements in an `mxArray`. This philosophy eliminates (or, at least reduces) expensive reallocations.

**See Also** `mxGetNzmax`

# mxSetPi

---

**Purpose** Set new imaginary data for an `mxArray`

**Fortran Syntax** `subroutine mxSetPi (pm, pi)`  
`integer*4 pm, pi`

**Arguments**

`pm`  
Pointer to a full (nonsparse) `mxArray`.

`pi`  
Pointer to the first element of an array. Each element in the array contains the imaginary component of a value. The array must be in dynamic memory; call `mxCallLoc` to allocate this dynamic memory.

**Description** Use `mxSetPi` to set the imaginary data of the specified `mxArray`.

Most `mxCreate` functions optionally allocate heap space to hold imaginary data. If you tell an `mxCreate` function to allocate heap space (for example, by setting the `ComplexFlag` to `COMPLEX = 1` or by setting `pi` to a non-0 value), then you do not ordinarily use `mxSetPi` to initialize the created `mxArray`'s imaginary elements. Rather, you typically call `mxSetPi` to replace the initial imaginary values with new ones.

**See Also** `mxGetPi`, `mxGetPr`, `mxSetPr`

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>        | Set new real data for an mxArray                                                                                                                                                                                                                                                                                                                                                |
| <b>Fortran Syntax</b> | subroutine mxSetPr(pm, pr)<br>integer*4 pm, pr                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>      | pm<br>Pointer to a full (nonsparse) mxArray.<br><br>pr<br>Pointer to the first element of an array. Each element in the array contains the real component of a value. The array must be in dynamic memory; call <code>mxMalloc</code> to allocate this dynamic memory.                                                                                                          |
| <b>Description</b>    | Use <code>mxSetPr</code> to set the real data of the specified mxArray.<br><br>All <code>mxCreate</code> calls allocate heap space to hold real data. Therefore, you do not ordinarily use <code>mxSetPr</code> to initialize the real elements of a freshly created mxArray. Rather, you typically call <code>mxSetPr</code> to replace the initial real values with new ones. |
| <b>See Also</b>       | <code>mxGetPr</code> , <code>mxGetPi</code> , <code>mxSetPi</code>                                                                                                                                                                                                                                                                                                              |