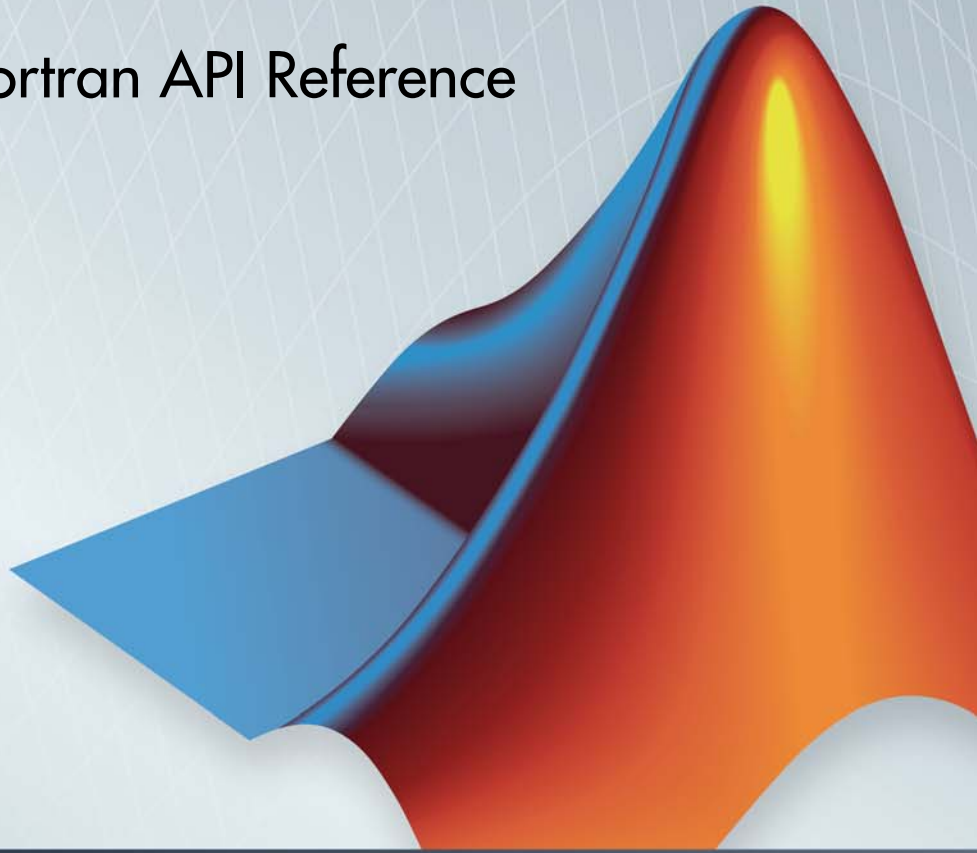


MATLAB®

C/C++ and Fortran API Reference

R2014a



MATLAB®



How to Contact MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



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



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

MATLAB® C/C++ and Fortran API Reference

© COPYRIGHT 1984–2014 by The MathWorks, Inc.

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.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

December 1996	First Printing	New for MATLAB 5 (Release 8)
May 1997	Online only	Revised for MATLAB 5.1 (Release 9)
January 1998	Online Only	Revised for MATLAB 5.2 (Release 10)
January 1999	Online Only	Revised for MATLAB 5.3 (Release 11)
September 2000	Online Only	Revised for MATLAB 6.0 (Release 12)
June 2001	Online only	Revised for MATLAB 6.1 (Release 12.1)
July 2002	Online only	Revised for MATLAB 6.5 (Release 13)
January 2003	Online only	Revised for MATLAB 6.5.1 (Release 13SP1)
June 2004	Online only	Revised for MATLAB 7.0 (Release 14)
October 2004	Online only	Revised for MATLAB 7.0.1 (Release 14SP1)
March 2005	Online only	Revised for MATLAB 7.0.4 (Release 14SP2)
September 2005	Online only	Revised for MATLAB 7.1 (Release 14SP3)
March 2006	Online only	Revised for MATLAB 7.2 (Release 2006a)
September 2006	Online only	Revised for MATLAB 7.3 (Release 2006b)
March 2007	Online only	Revised and renamed for MATLAB 7.4 (Release 2007a)
September 2007	Online only	Revised and renamed for MATLAB 7.5 (Release 2007b)
March 2008	Online only	Revised and renamed for MATLAB 7.6 (Release 2008a)
October 2008	Online only	Revised and renamed for MATLAB 7.7 (Release 2008b)
March 2009	Online only	Revised for MATLAB 7.8 (Release 2009a)
September 2009	Online only	Revised for MATLAB 7.9 (Release 2009b)
March 2010	Online only	Revised and renamed for MATLAB 7.10 (Release 2010a)
September 2010	Online only	Revised for MATLAB 7.11 (Release 2010b)
April 2011	Online only	Revised for MATLAB 7.12 (Release 2011a)
September 2011	Online only	Revised for MATLAB 7.13 (Release 2011b)
March 2012	Online only	Revised for MATLAB 7.14 (Release 2012a)
September 2012	Online only	Revised for MATLAB 8.0 (Release 2012b)
March 2013	Online only	Revised for MATLAB 8.1 (Release 2013a)
September 2013	Online only	Revised for MATLAB 8.2 (Release 2013b)
March 2014	Online only	Revised for MATLAB 8.3 (Release 2014a)

1 | API Reference

API Reference

engClose (C and Fortran)

Purpose Quit MATLAB engine session

C Syntax

```
#include "engine.h"
int engClose(Engine *ep);
```

Fortran Syntax

```
integer*4 engClose(ep)
mwPointer ep
```

Arguments ep
Engine pointer

Returns 0 on success, and 1 otherwise. Possible failure includes attempting to terminate an already-terminated MATLAB® engine session.

Description This routine sends a quit command to the MATLAB engine session and closes the connection.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- engdemo.c for a C example on UNIX® operating systems.
- engwindemo.c for a C example on Microsoft® Windows® operating systems.
- fengdemo.F for a Fortran example.

See Also engOpen

Purpose Evaluate expression in string

C Syntax

```
#include "engine.h"
int engEvalString(Engine *ep, const char *string);
```

Fortran Syntax

```
integer*4 engEvalString(ep, string)
mwPointer ep
character*(*) string
```

Arguments

- `ep`
Engine pointer
- `string`
String to execute

Returns 1 if the engine session is no longer running or the engine pointer is invalid or NULL. Otherwise, returns 0 even if the MATLAB engine session cannot evaluate the command.

Description `engEvalString` evaluates the expression contained in `string` for the MATLAB engine session, `ep`, previously started by `engOpen`.

UNIX Operating Systems

On UNIX systems, `engEvalString` sends commands to the MATLAB workspace by writing down a pipe connected to the MATLAB *stdin* process. MATLAB reads back from *stdout* any output resulting from the command that ordinarily appears on the screen, into the buffer defined by `engOutputBuffer`.

To turn off output buffering in C, use:

```
engOutputBuffer(ep, NULL, 0);
```

To turn off output buffering in Fortran, use:

```
engOutputBuffer(ep, '')
```

engEvalString (C and Fortran)

Microsoft Windows Operating Systems

On a Windows system, `engEvalString` communicates with MATLAB software using a Component Object Model (COM) interface.

Examples

See the following examples in `matlabroot/extern/examples/eng_mat`.

- `engdemo.c` for a C example on UNIX operating systems.
- `engwindemo.c` for a C example on Microsoft Windows operating systems.
- `fengdemo.F` for a Fortran example.

See Also

`engOpen`, `engOutputBuffer`

engGetVariable (C and Fortran)

Purpose	Copy variable from MATLAB engine workspace
C Syntax	<pre>#include "engine.h" mxArray *engGetVariable(Engine *ep, const char *name);</pre>
Fortran Syntax	<pre>mwPointer engGetVariable(ep, name) mwPointer ep character*(*) name</pre>
Arguments	<p>ep Engine pointer</p> <p>name Name of mxArray to get from MATLAB workspace</p>
Returns	Pointer to a newly allocated mxArray structure, or NULL if the attempt fails. <code>engGetVariable</code> fails if the named variable does not exist.
Description	<p><code>engGetVariable</code> reads the named mxArray from the MATLAB engine session associated with <code>ep</code>.</p> <p>Use <code>mxDestroyArray</code> to destroy the mxArray created by this routine when you are finished with it.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/eng_mat</i>.</p> <ul style="list-style-type: none">• <code>engdemo.c</code> for a C example on UNIX operating systems.• <code>engwindemo.c</code> for a C example on Microsoft Windows operating systems.
See Also	<code>engPutVariable</code> , <code>mxDestroyArray</code>

engGetVisible (C)

Purpose Determine visibility of MATLAB engine session

C Syntax

```
#include "engine.h"
int engGetVisible(Engine *ep, bool *value);
```

Arguments

ep
Engine pointer

value
Pointer to value returned from engGetVisible

Returns **Microsoft Windows Operating Systems Only**
0 on success, and 1 otherwise.

Description engGetVisible returns the current visibility setting for MATLAB engine session, ep. A *visible* engine session runs in a window on the Windows desktop, thus making the engine available for user interaction. MATLAB removes an invisible session from the desktop.

Examples The following code opens engine session ep and disables its visibility.

```
Engine *ep;
bool vis;

ep = engOpen(NULL);
engSetVisible(ep, 0);
```

To determine the current visibility setting, use:

```
engGetVisible(ep, &vis);
```

See Also engSetVisible

Purpose Type for MATLAB engine

Description A handle to a MATLAB engine object.

Engine is a C language opaque type.

You can call MATLAB software as a computational engine by writing C and Fortran programs that use the MATLAB engine library. Engine is the link between your program and the separate MATLAB engine process.

The header file containing this type is:

```
#include "engine.h"
```

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- `engdemo.c` shows how to call the MATLAB engine functions from a C program.
- `engwindemo.c` show how to call the MATLAB engine functions from a C program for Windows systems.
- `fengdemo.F` shows how to call the MATLAB engine functions from a Fortran program.

See Also `engOpen`

engOpen (C and Fortran)

Purpose	Start MATLAB engine session
C Syntax	<pre>#include "engine.h" Engine *engOpen(const char *startcmd);</pre>
Fortran Syntax	<pre>mwPointer engOpen(startcmd) character*(*) startcmd</pre>
Arguments	<p><code>startcmd</code> String to start the MATLAB process. On Windows systems, the <code>startcmd</code> string must be NULL.</p>
Returns	Pointer to an engine handle, or NULL if the open fails.
Description	<p>This routine allows you to start a MATLAB process for using MATLAB as a computational engine.</p> <p><code>engOpen</code> starts a MATLAB process using the command specified in the string <code>startcmd</code>, establishes a connection, and returns an engine pointer.</p> <p>On UNIX systems, if <code>startcmd</code> is NULL or the empty string, <code>engOpen</code> starts a MATLAB process on the current host using the command <code>matlab</code>. If <code>startcmd</code> is a hostname, <code>engOpen</code> starts a MATLAB process 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 <code>startcmd</code> is any other string (has white space in it, or nonalphanumeric characters), MATLAB executes the string literally.</p> <p>On UNIX systems, <code>engOpen</code> performs the following steps:</p> <ol style="list-style-type: none">1 Creates two pipes.

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

On Windows systems, `engOpen` opens a COM channel to MATLAB. The MATLAB software you registered during installation starts. If you did not register during installation, on the command line you can enter the command:

```
!matlab /regserver
```

See “MATLAB COM Integration” for additional details.

Examples

See the following examples in `matlabroot/extern/examples/eng_mat`.

- `engdemo.c` for a C example on UNIX operating systems.
- `engwindemo.c` for a C example on Microsoft Windows operating systems.
- `fengdemo.F` for a Fortran example.

engOpenSingleUse (C)

Purpose Start MATLAB engine session for single, nonshared use

C Syntax

```
#include "engine.h"
Engine *engOpenSingleUse(const char *startcmd, void *dcom,
    int *retstatus);
```

Arguments

startcmd
String to start MATLAB process. On Microsoft Windows systems, the startcmd string must be NULL.

dcom
Reserved for future use; must be NULL.

retstatus
Return status; possible cause of failure.

Returns **Microsoft Windows Operating Systems Only**

Pointer to an engine handle, or NULL if the open fails.

UNIX Operating Systems

Not supported on UNIX systems.

Description This routine allows you to start multiple MATLAB processes using MATLAB as a computational engine.

engOpenSingleUse starts a MATLAB process, establishes a connection, and returns a unique engine identifier, or NULL if the open fails. Each call to engOpenSingleUse starts a new MATLAB process.

engOpenSingleUse opens a COM channel to MATLAB. This starts the MATLAB software you registered during installation. If you did not register during installation, on the command line you can enter the command:

```
!matlab /regserver
```


`engOpenSingleUse` allows single-use instances of an engine server. `engOpenSingleUse` differs from `engOpen`, which allows multiple applications to use the same engine server.

See “MATLAB COM Integration” for additional details.

engOutputBuffer (C and Fortran)

Purpose Specify buffer for MATLAB output

C Syntax

```
#include "engine.h"
int engOutputBuffer(Engine *ep, char *p, int n);
```

Fortran Syntax

```
integer*4 engOutputBuffer(ep, p)
mwPointer ep
character*n p
```

Arguments

ep	Engine pointer
p	Pointer to character buffer
n	Length of buffer p

Returns 1 if you pass it a NULL engine pointer. Otherwise, returns 0.

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. A call to engOutputBuffer with a buffer of nonzero length tells any subsequent calls to engEvalString to save output in the character buffer pointed to by p.

To turn off output buffering in C, use:

```
engOutputBuffer(ep, NULL, 0);
```

To turn off output buffering in Fortran, use:

```
engOutputBuffer(ep, '')
```

Note The buffer returned by `engEvalString` is not NULL terminated.

Examples

See the following examples in `matlabroot/extern/examples/eng_mat`.

- `engdemo.c` for a C example on UNIX operating systems.
- `engwindemo.c` for a C example on Microsoft Windows operating systems.
- `fengdemo.F` for a Fortran example.

See Also

`engOpen`, `engEvalString`

engPutVariable (C and Fortran)

Purpose Put variable into MATLAB engine workspace

C Syntax

```
#include "engine.h"
int engPutVariable(Engine *ep, const char *name, const mxArray
    *pm);
```

Fortran Syntax

```
integer*4 engPutVariable(ep, name, pm)
mwPointer ep, pm
character*(*) name
```

Arguments

ep
Engine pointer

name
Name of mxArray in the engine workspace

pm
mxArray pointer

Returns 0 if successful and 1 if an error occurs.

Description engPutVariable writes mxArray pm to the engine ep, giving it the variable name name.

If the mxArray does not exist in the workspace, the function creates it. If an mxArray with the same name exists in the workspace, the function replaces the existing mxArray with the new mxArray.

Do not use MATLAB function names for variable names. Common variable names that conflict with function names include i, j, mode, char, size, or path. To determine whether a particular name is associated with a MATLAB function, use the which function.

The engine application owns the original mxArray and is responsible for freeing its memory. Although the engPutVariable function sends a copy of the mxArray to the MATLAB workspace, the engine application does not need to account for or free memory for the copy.

Examples

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `engdemo.c` for a C example on UNIX operating systems.
- `engwindemo.c` for a C example on Microsoft Windows operating systems.

See Also

`engGetVariable`

engSetVisible (C)

Purpose	Show or hide MATLAB engine session
C Syntax	<pre>#include "engine.h" int engSetVisible(Engine *ep, bool value);</pre>
Arguments	<p>ep Engine pointer</p> <p>value Value to set the Visible property to. Set value to 1 to make the engine window visible, or to 0 to make it invisible.</p>
Returns	Microsoft Windows Operating Systems Only 0 on success, and 1 otherwise.
Description	engSetVisible makes the window for the MATLAB engine session, ep, either visible or invisible on the Windows desktop. You can use this function to enable or disable user interaction with the MATLAB engine session.
Examples	<p>The following code opens engine session ep and disables its visibility.</p> <pre>Engine *ep; bool vis; ep = engOpen(NULL); engSetVisible(ep, 0);</pre> <p>To determine the current visibility setting, use:</p> <pre>engGetVisible(ep, &vis);</pre>
See Also	engGetVisible

Purpose	Close MAT-file
C Syntax	<pre>#include "mat.h" int matClose(MATFile *mfp);</pre>
Fortran Syntax	<pre>integer*4 matClose(mfp) mwPointer mfp</pre>
Arguments	<p>mfp Pointer to MAT-file information</p>
Returns	EOF in C (-1 in Fortran) for a write error, and 0 if successful.
Description	matClose closes the MAT-file associated with mfp.
Examples	<p>See the following examples in <i>matlabroot/extern/examples/eng_mat</i>.</p> <ul style="list-style-type: none">• matcreat.c• matdgn.c• matdemo1.F• matdemo2.F
See Also	matOpen

matDeleteVariable (C and Fortran)

Purpose	Delete array from MAT-file
C Syntax	<pre>#include "mat.h" int matDeleteVariable(MATFile *mfp, const char *name);</pre>
Fortran Syntax	<pre>integer*4 matDeleteVariable(mfp, name) mwPointer mfp character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Name of mxArray to delete</p>
Returns	0 if successful, and nonzero otherwise.
Description	matDeleteVariable deletes the named mxArray from the MAT-file pointed to by mfp.

Purpose

Type for MAT-file

Description

A handle to a MAT-file object. A MAT-file is the data file format MATLAB software uses for saving data to your disk.

MATFile is a C language opaque type.

The MAT-file interface library contains routines for reading and writing MAT-files. Call these routines from your own C/C++ and Fortran programs, using MATFile to access your data file.

The header file containing this type is:

```
#include "mat.h"
```

Examples

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matcreat.c`
- `matdgns.c`
- `matdemo1.F`
- `matdemo2.F`

See Also

`matOpen`, `matClose`, `matPutVariable`, `matGetVariable`, `mxDestroyArray`

matGetDir (C and Fortran)

Purpose	List of variables in MAT-file
C Syntax	<pre>#include "mat.h" char **matGetDir(MATFile *mfp, int *num);</pre>
Fortran Syntax	<pre>mwPointer matGetDir(mfp, num) mwPointer mfp integer*4 num</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>num Pointer to the variable containing the number of mxArray's in the MAT-file</p>
Returns	<p>Pointer to an internal array containing pointers to the names of the mxArray's in the MAT-file pointed to by mfp. In C, each name is a NULL-terminated string. The num output argument is the length of the internal array (number of mxArray's in the MAT-file). If num is zero, mfp contains no arrays.</p> <p>matGetDir returns NULL in C (0 in Fortran). If matGetDir fails, sets num to a negative number.</p>
Description	<p>This routine provides you with a list of the names of the mxArray's contained within a MAT-file.</p> <p>matGetDir allocates memory for the internal array of strings using a mxCalloc. You must free the memory using mxFree when you are finished with the array.</p> <p>MATLAB variable names can be up to length mxMAXNAM, defined in the C header file matrix.h.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/eng_mat</i>.</p> <ul style="list-style-type: none">• matcreat.c

- matdgn.c
- matdemo2.F

matGetFp (C)

Purpose	File pointer to MAT-file
C Syntax	<pre>#include "mat.h" FILE *matGetFp(MATFile *mfp);</pre>
Arguments	<p>mfp Pointer to MAT-file information</p>
Returns	C file handle to the MAT-file with handle mfp. Returns NULL if mfp is a handle to a MAT-file in HDF5-based format.
Description	Use matGetFp to obtain a C file handle to a MAT-file. Standard C library routines, like ferror and feof, use file handle to investigate errors.

matGetNextVariable (C and Fortran)

Purpose	Next array in MAT-file
C Syntax	<pre>#include "mat.h" mxArray *matGetNextVariable(MATFile *mfp, const char **name);</pre>
Fortran Syntax	<pre>mwPointer matGetNextVariable(mfp, name) mwPointer mfp character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Pointer to the variable containing the mxArray name</p>
Returns	<p>Pointer to a newly allocated mxArray structure representing the next mxArray from the MAT-file pointed to by mfp. The function returns the name of the mxArray in name.</p> <p>matGetNextVariable returns NULL in C (0 in Fortran) for end-of-file or if there is an error condition. In C, use feof and ferror from the Standard C Library to determine status.</p>
Description	<p>matGetNextVariable allows you to step sequentially through a MAT-file and read all the mxArrays in a single pass. The function reads and returns the next mxArray from the MAT-file pointed to by mfp.</p> <p>Use matGetNextVariable immediately after opening the MAT-file with matOpen and not with other MAT-file routines. Otherwise, the concept of the <i>next</i> mxArray is undefined.</p> <p>Use mxDestroyArray to destroy the mxArray created by this routine when you are finished with it.</p> <p>The order of variables returned from successive calls to matGetNextVariable is not guaranteed to be the same order in which the variables were written.</p>

matGetNextVariable (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matdgn.c`

See Also

`matGetNextVariableInfo`, `matGetVariable`, `mxDestroyArray`

matGetNextVariableInfo (C and Fortran)

Purpose	Array header information only
C Syntax	<pre>#include "mat.h" mxArray *matGetNextVariableInfo(MATFile *mfp, const char **name);</pre>
Fortran Syntax	<pre>mwPointer matGetNextVariableInfo(mfp, name) mwPointer mfp character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Pointer to the variable containing the mxArray name</p>
Returns	<p>Pointer to a newly allocated mxArray structure representing header information for the next mxArray from the MAT-file pointed to by mfp. The function returns the name of the mxArray in name.</p> <p>matGetNextVariableInfo returns NULL in C (0 in Fortran) when the end-of-file is reached or if there is an error condition. In C, use feof and ferror from the Standard C Library to determine status.</p>
Description	<p>matGetNextVariableInfo loads only the array header information, including everything except pr, pi, ir, and jc, from the current file offset.</p> <p>If pr, pi, ir, and jc are nonzero values when loaded with matGetVariable, matGetNextVariableInfo sets them to -1 instead. These headers are for informational use only. <i>Never</i> pass this data back to the MATLAB workspace or save it to MAT-files.</p> <p>Use mxDestroyArray to destroy the mxArray created by this routine when you are finished with it.</p> <p>The order of variables returned from successive calls to matGetNextVariableInfo is not guaranteed to be the same order in which the variables were written.</p>

matGetNextVariableInfo (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matdgn.c`

See Also

`matGetNextVariable`, `matGetVariableInfo`

matGetVariable (C and Fortran)

Purpose	Array from MAT-file
C Syntax	<pre>#include "mat.h" mxArray *matGetVariable(MATFile *mfp, const char *name);</pre>
Fortran Syntax	<pre>mwPointer matGetVariable(mfp, name) mwPointer mfp character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Name of mxArray to get from MAT-file</p>
Returns	<p>Pointer to a newly allocated mxArray structure representing the mxArray named by name from the MAT-file pointed to by mfp.</p> <p>matGetVariable returns NULL in C (0 in Fortran) if the attempt to return the mxArray named by name fails.</p>
Description	<p>This routine allows you to copy an mxArray out of a MAT-file.</p> <p>Use mxDestroyArray to destroy the mxArray created by this routine when you are finished with it.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/eng_mat</i>.</p> <ul style="list-style-type: none">• <code>matcreat.c</code>
See Also	<code>matPutVariable</code> , <code>mxDestroyArray</code>

matGetVariableInfo (C and Fortran)

Purpose	Array header information only
C Syntax	<pre>#include "mat.h" mxArray *matGetVariableInfo(MATFile *mfp, const char *name);</pre>
Fortran Syntax	<pre>mwPointer matGetVariableInfo(mfp, name) mwPointer mfp character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Name of mxArray to get from MAT-file</p>
Returns	<p>Pointer to a newly allocated mxArray structure representing header information for the mxArray named by name from the MAT-file pointed to by mfp.</p> <p>matGetVariableInfo returns NULL in C (0 in Fortran) if the attempt to return header information for the mxArray named by name fails.</p>
Description	<p>matGetVariableInfo loads only the array header information, including everything except pr, pi, ir, and jc. It recursively creates the cells and structures through their leaf elements, but does not include pr, pi, ir, and jc.</p> <p>If pr, pi, ir, and jc are nonzero values when loaded with matGetVariable, matGetVariableInfo sets them to -1 instead. These headers are for informational use only. <i>Never</i> pass this data back to the MATLAB workspace or save it to MAT-files.</p> <p>Use mxDestroyArray to destroy the mxArray created by this routine when you are finished with it.</p>
See Also	matGetVariable

Purpose Open MAT-file

C Syntax

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

Fortran Syntax

```
mwPointer matOpen(filename, mode)
character*(*) filename, mode
```

Arguments `filename`
Name of file to open

`mode`
File opening mode. The following table lists valid values for mode.

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. If the file does not exist, does not create a file (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 MAT-file compatible with MATLAB Versions 4 software and earlier.
wL	Opens file for writing character data using the default character set for your system. Use MATLAB Version 6 or 6.5 software to read the resulting MAT-file. If you do not use the wL mode switch, MATLAB writes character data to the MAT-file using Unicode® character encoding by default.

matOpen (C and Fortran)

wz	Opens file for writing compressed data. By default, the MATLAB save function compresses workspace variables as they are saved to a MAT-file. To use the same compression ratio when creating a MAT-file with the matOpen function, use the wz option.
w7.3	Creates a MAT-file in an HDF5-based format that can store objects that occupy more than 2 GB.

Returns File handle, or NULL in C (0 in Fortran) if the open fails.

Description This routine opens a MAT-file for reading and writing.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matcreat.c`
- `matdgns.c`
- `matdemo1.F`
- `matdemo2.F`

See Also `matClose`

Purpose	Array to MAT-file
C Syntax	<pre>#include "mat.h" int matPutVariable(MATFile *mfp, const char *name, const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 matPutVariable(mfp, name, pm) mwPointer mfp, pm character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Name of mxArray to put into MAT-file</p> <p>pm mxArray pointer</p>
Returns	0 if successful and nonzero if an error occurs. In C, use feof and ferror from the Standard C Library along with matGetFp to determine status.
Description	<p>This routine puts an mxArray into a MAT-file.</p> <p>matPutVariable writes mxArray pm to the MAT-file mfp. If the mxArray does not exist in the MAT-file, the function appends it to the end. If an mxArray with the same name exists in the file, the function replaces the existing mxArray with the new mxArray by rewriting the file.</p> <p>Do not use MATLAB function names for variable names. Common variable names that conflict with function names include i, j, mode, char, size, or path. To determine whether a particular name is associated with a MATLAB function, use the which function.</p> <p>The size of the new mxArray can be different from the existing mxArray.</p>

matPutVariable (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matcreat.c`

See Also

`matGetVariable`, `matGetFp`

matPutVariableAsGlobal (C and Fortran)

Purpose	Array to MAT-file as originating from global workspace
C Syntax	<pre>#include "mat.h" int matPutVariableAsGlobal(MATFile *mfp, const char *name, const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 matPutVariableAsGlobal(mfp, name, pm) mwPointer mfp, pm character*(*) name</pre>
Arguments	<p>mfp Pointer to MAT-file information</p> <p>name Name of mxArray to put into MAT-file</p> <p>pm mxArray pointer</p>
Returns	0 if successful and nonzero if an error occurs. In C, use feof and ferror from the Standard C Library with matGetFp to determine status.
Description	<p>This routine puts an mxArray into a MAT-file. <code>matPutVariableAsGlobal</code> is like <code>matPutVariable</code>, except that MATLAB software loads the array into the global workspace and sets a reference to it in the local workspace. If you write to a MATLAB 4 format file, <code>matPutVariableAsGlobal</code> does not load it as global and has the same effect as <code>matPutVariable</code>.</p> <p><code>matPutVariableAsGlobal</code> writes mxArray pm to the MAT-file mfp. If the mxArray does not exist in the MAT-file, the function appends it to the end. If an mxArray with the same name exists in the file, the function replaces the existing mxArray with the new mxArray by rewriting the file.</p> <p>Do not use MATLAB function names for variable names. Common variable names that conflict with function names include <code>i</code>, <code>j</code>, <code>mode</code>,</p>

matPutVariableAsGlobal (C and Fortran)

`char`, `size`, or `path`. To determine whether a particular name is associated with a MATLAB function, use the `which` function.

The size of the new `mxArray` can be different from the existing `mxArray`.

Examples

See the following examples in `matlabroot/extern/examples/eng_mat`.

- `matcreat.c`

See Also

`matPutVariable`, `matGetFp`

Purpose	Register function to call when MEX-function clears or MATLAB terminates
C Syntax	<pre>#include "mex.h" int mexAtExit(void (*ExitFcn)(void));</pre>
Fortran Syntax	<pre>#include "fintrf.h" integer*4 mexAtExit(ExitFcn) subroutine ExitFcn()</pre>
Arguments	ExitFcn Pointer to function you want to run on exit
Returns	Always returns 0.
Description	<p>Use <code>mexAtExit</code> to register a function to call just before clearing the MEX-function or terminating MATLAB. <code>mexAtExit</code> gives your MEX-function a chance to perform tasks such as freeing persistent memory and closing files. Other typical tasks include closing streams or sockets.</p> <p>Each MEX-function 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-function is locked, you cannot clear the MEX-file. Consequently, if you attempt to clear a locked MEX-file, MATLAB does not call the <code>ExitFcn</code>.</p> <p>In Fortran, 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>
Examples	See the following examples in <code>matlabroot/extern/examples/mex</code> .
	<ul style="list-style-type: none">• <code>mexatexit.c</code>
See Also	<code>mexLock</code> , <code>mexUnlock</code>

mexCallMATLAB (C and Fortran)

Purpose Call MATLAB function, user-defined function, or MEX-file

C Syntax

```
#include "mex.h"
int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs,
    mxArray *prhs[], const char *functionName);
```

Fortran Syntax

```
#include "fintrf.h"
integer*4 mexCallMATLAB(nlhs, plhs, nrhs, prhs, functionName)
integer*4 nlhs, nrhs
mwPointer plhs(*), prhs(*)
character*(*) functionName
```

Arguments

nlhs
Number of output arguments

plhs
Array of pointers to output arguments

nrhs
Number of input arguments

prhs
Array of pointers to input arguments

functionName
Character string containing name of the MATLAB built-in function, operator, user-defined function, or MEX-file you are calling

Returns 0 if successful, and a nonzero value if unsuccessful.

Description Call mexCallMATLAB to invoke internal MATLAB numeric functions, MATLAB operators, user-defined functions, or other MEX-files. Both mexCallMATLAB and mexEvalString execute MATLAB commands. Use mexCallMATLAB for returning results (left side arguments) back to the MEX-file. The mexEvalString function cannot return values to the MEX-file.

For a complete description of the input and output arguments passed to `functionName`, see `mexFunction`. When calling the `mexCallMATLAB` function, the number of output arguments `nlhs` and input arguments `nrhs` must be less than or equal to 50.

MATLAB allocates dynamic memory to store the `mxAArrays` in `plhs`. MATLAB automatically deallocates the dynamic memory when you clear the MEX-file. However, if heap space is at a premium, call `mxDestroyArray` when you are finished with the `mxAArrays` `plhs` points to.

If `functionName` is an operator, place the operator inside a pair of single quotes, for example, `'+'`.

Avoid using the `mexCallMATLAB` function in Simulink® S-functions. If you do, you must use the `mexMakeArrayPersistent` function to create the `plhs` `mxAArray` pointers returned by `mexCallMATLAB`, and free them manually using the `mxFree` function, in order to prevent intermittent crashes.

Note It is possible to generate an object of type `mxUNKNOWN_CLASS` using `mexCallMATLAB`.

This function returns two variables but only assigns one of them a value:

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

If you then call `foo` using `mexCallMATLAB`, the unassigned output variable is now type `mxUNKNOWN_CLASS`.

Error Handling

If `functionName` detects an error, MATLAB terminates the MEX-file and returns control to the MATLAB prompt. If you want to trap errors, use the `mexCallMATLABWithTrap` function.

mexCallMATLAB (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/mex*.

- `mexcallmatlab.c`
- `mexevalstring.c`
- `mexcallmatlabwithtrap.c`

See the following examples in *matlabroot/extern/examples/refbook*.

- `sincall.c`
- `sincall.F`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcreatecellmatrix.c`
- `mxcreatecellmatrixf.F`
- `mxisclass.c`

See Also

`mexFunction`, `mexCallMATLABWithTrap`, `mexEvalString`,
`mexMakeArrayPersistent`, `mxDestroyArray`

mexCallMATLABWithTrap (C and Fortran)

Purpose	Call MATLAB function, user-defined function, or MEX-file and capture error information
C Syntax	<pre>#include "mex.h" mxArray *mexCallMATLABWithTrap(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[], const char *functionName);</pre>
Fortran Syntax	<pre>#include "fintrf.h" mwPointer mexCallMATLABWithTrap(nlhs, plhs, nrhs, prhs, functionName) integer*4 nlhs, nrhs mwPointer plhs(*), prhs(*) character*(*) functionName</pre>
Arguments	<p>For more information about arguments, see <code>mexCallMATLAB</code>.</p> <p><code>nlhs</code> Number of desired output arguments.</p> <p><code>plhs</code> Array of pointers to output arguments.</p> <p><code>nrhs</code> Number of input arguments.</p> <p><code>prhs</code> Array of pointers to input arguments.</p> <p><code>functionName</code> Character string containing the name of the MATLAB built-in function, operator, function, or MEX-file that you are calling.</p>
Returns	NULL if no error occurred; otherwise, a pointer to an <code>mxArray</code> of class <code>MException</code> .
Description	The <code>mexCallMATLABWithTrap</code> function performs the same function as <code>mexCallMATLAB</code> . However, if MATLAB detects an error when executing <code>functionName</code> , MATLAB returns control to the line in the MEX-file

mexCallMATLABWithTrap (C and Fortran)

immediately following the call to `mexCallMATLABWithTrap`. For information about `MException`, see “Respond to an Exception”

See Also `mexCallMATLAB`, `MException`

mexErrMsgIdAndTxt (C and Fortran)

Purpose	Display error message with identifier and return to MATLAB prompt
C Syntax	<pre>#include "mex.h" void mexErrMsgIdAndTxt(const char *errorid, const char *errmsg, ...);</pre>
Fortran Syntax	<pre>#include "fintrf.h" subroutine mexErrMsgIdAndTxt(errorid, errmsg) character*(*) errorid, errmsg</pre>
Arguments	<p>errorid String containing a MATLAB message identifier. For information on creating identifiers, see “Message Identifiers”.</p> <p>errmsg String to display. In C, the string can include conversion specifications, used by the ANSI[®] C printf function.</p> <p>...</p> <p>In C, any arguments used in the message. Each argument must have a corresponding conversion specification.</p>
Description	<p>The mexErrMsgIdAndTxt function writes an error message to the MATLAB window. For more information, see the error function syntax statement using a message identifier. After the error message prints, MATLAB terminates the MEX-file and returns control to the MATLAB prompt.</p> <p>Calling mexErrMsgIdAndTxt does not clear the MEX-file from memory. Consequently, mexErrMsgIdAndTxt does not invoke the function registered through mexAtExit.</p> <p>If your application called mxMalloc or one of the mxCreate* routines to allocate memory, mexErrMsgIdAndTxt automatically frees the allocated memory.</p>

mexErrMsgIdAndTxt (C and Fortran)

Note If you get warnings when using `mexErrMsgIdAndTxt`, you might have a memory management compatibility problem. For more information, see “Memory Management Issues” in the External Interfaces documentation.

Remarks

In addition to the `errorid` and `errmsg`, the `mexerrmsgtxt` function determines where the error occurred, and displays the following information. For example, in the function `foo`, `mexerrmsgtxt` displays:

```
Error using foo
```

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillGetPr.c`
- `matrixDivide.c`
- `timestwo.F`
- `xtimesy.F`

Validate char Input

The following code snippet checks if input argument, `prhs[0]`, is a string. If not, the code displays a warning. If there is an error reading the input string, the code displays an error message and terminates the MEX-file.

```
char *buf;
int buflen;

if (mxIsChar(prhs[0])) {
    if (mxGetString(prhs[0], buf, buflen) == 0) {
        mexPrintf("The input string is: %s\n", buf);
    }
    else {
        mexErrMsgIdAndTxt("MyProg:ConvertString",
```



```
        "Could not convert string data.");
    // exit MEX-file
    }
}
else {
    mexWarnMsgIdAndTxt("MyProg:InputString",
        "Input should be a string to print properly.");
}

// continue with processing
```

See Also

mexWarnMsgIdAndTxt

Concepts

- “Memory Considerations For Class Destructors”

mexErrMsgTxt (C and Fortran)

Purpose Display error message and return to MATLAB prompt

Note mexErrMsgTxt is not recommended. Use mexErrMsgIdAndTxt instead.

C Syntax

```
#include "mex.h"
void mexErrMsgTxt(const char *errmsg);
```

Fortran Syntax

```
subroutine mexErrMsgTxt(errormsg)
character*(*) errormsg
```

Arguments

errormsg
String containing the error message to display

Description

mexErrMsgTxt writes 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 the function registered through mexAtExit.

If your application called mxMalloc or one of the mxCreate* routines to allocate memory, mexErrMsgTxt automatically frees the allocated memory.

Note If you get warnings when using mexErrMsgTxt, you might have a memory management compatibility problem. For more information, see “Memory Management Issues”.

Remarks

In addition to the errormsg, the mexerrmsgtxt function determines where the error occurred, and displays the following information. If

an error labeled Print my error message occurs in the function foo, mexerrmsgtxt displays:

```
Error using foo
Print my error message
```

See Also

mexErrMsgIdAndTxt, mexWarnMsgIdAndTxt

mexEvalString (C and Fortran)

Purpose	Execute MATLAB command in caller workspace
C Syntax	<pre>#include "mex.h" int mexEvalString(const char *command);</pre>
Fortran Syntax	<pre>#include "fintrf.h" integer*4 mexEvalString(command) character*(*) command</pre>
Arguments	command String containing MATLAB command to execute
Returns	0 if successful, and 1 if an error occurs.
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. Use <code>mexCallMATLAB</code> for returning results (left side arguments) back to the MEX-file. The <code>mexEvalString</code> function cannot return values to the MEX-file.</p> <p>All arguments that appear to the right of an equal sign in the command string must be current variables of the caller workspace.</p> <p>Do not use MATLAB function names for variable names. Common variable names that conflict with function names include <code>i</code>, <code>j</code>, <code>mode</code>, <code>char</code>, <code>size</code>, or <code>path</code>. To determine whether a particular name is associated with a MATLAB function, use the <code>which</code> function. For more information, see “Variable Names”.</p>
Error Handling	If <code>command</code> detects an error, MATLAB returns control to the MEX-file and <code>mexEvalString</code> returns 1. If you want to trap errors, use the <code>mexEvalStringWithTrap</code> function.

Examples

See the following examples in *matlabroot/extern/examples/mex*.

- `mexevalstring.c`

See Also

`mexCallMATLAB`, `mexEvalStringWithTrap`

mexEvalStringWithTrap (C and Fortran)

Purpose	Execute MATLAB command in caller workspace and capture error information
C Syntax	<pre>#include "mex.h" mxArray *mexEvalStringWithTrap(const char *command);</pre>
Fortran Syntax	<pre>#include "fintrf.h" mwPointer mexEvalStringWithTrap(command) character*(*) command</pre>
Arguments	command String containing the MATLAB command to execute
Returns	Object ME of class MException
Description	The mexEvalStringWithTrap function performs the same function as mexEvalString. However, if MATLAB detects an error when executing command, MATLAB returns control to the line in the MEX-file immediately following the call to mexEvalStringWithTrap.
See Also	mexEvalString, MException, mexCallMATLAB

Purpose	Entry point to C/C++ or Fortran MEX-file
C Syntax	<pre>#include "mex.h" void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])</pre>
Fortran Syntax	<pre>#include "fintrf.h" subroutine mexFunction(nlhs, plhs, nrhs, prhs) integer nlhs, nrhs mwPointer plhs(*), prhs(*)</pre>
Arguments	<p>nlhs Number of expected output mxArray</p> <p>plhs Array of pointers to the expected output mxArray</p> <p>nrhs Number of input mxArray</p> <p>prhs Array of pointers to the input mxArray. Do not modify any prhs values in your MEX-file. Changing the data in these read-only mxArray can produce undesired side effects.</p>
Description	<p>mexFunction is not a routine you call. Rather, mexFunction is the name of the gateway function in C (subroutine in Fortran) which every MEX-file requires. When you invoke a MEX-function, MATLAB software finds and loads the corresponding MEX-file of the same name. MATLAB then searches for a symbol named mexFunction within the MEX-file. If it finds one, it calls the MEX-function using the address of the mexFunction symbol. MATLAB displays an error message if it cannot find a routine named mexFunction inside the MEX-file.</p> <p>When you invoke a MEX-file, MATLAB automatically seeds nlhs, plhs, nrhs, and prhs with the caller's information. In the syntax of the MATLAB language, functions have the general form:</p>

mexFunction (C and Fortran)

```
[a,b,c,...] = fun(d,e,f,...)
```

where the ... denotes more items of the same format. The `a,b,c...` are left-side output arguments, and the `d,e,f...` are right-side input arguments. The arguments `nlhs` and `nrhs` contain the number of left side and right side arguments, respectively. `prhs` is an array of `mxArray` pointers whose length is `nrhs`. `plhs` is an array whose length is `nlhs`, where your function must set pointers for the output `mxArrays`.

Examples

See the following examples in `matlabroot/extern/examples/mex`.

- `mexfunction.c`
- `mexlockf.F`

Concepts

- “Introducing MEX-Files”

mexFunctionName (C and Fortran)

Purpose	Name of current MEX-function
C Syntax	<pre>#include "mex.h" const char *mexFunctionName(void);</pre>
Fortran Syntax	<pre>#include "fintrf.h" character*(*) mexFunctionName()</pre>
Returns	Name of the current MEX-function.
Description	<code>mexFunctionName</code> returns the name of the current MEX-function.
Examples	See the following examples in <i>matlabroot/extern/examples/mex</i> . <ul style="list-style-type: none">• <code>mexgetarray.c</code>

mexGet (C)

Purpose	Value of 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> Handle Graphics® property</p>
Returns	Value of the specified property in the specified graphics object on success. Returns NULL on failure. Do not modify the return argument from <code>mexGet</code> . Changing the data in a <code>const</code> (read-only) <code>mxArray</code> can 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 the MATLAB <code>get</code> function. To set a graphics property value, call <code>mexSet</code> .
Examples	See the following examples in <code>matlabroot/extern/examples/mex</code> . <ul style="list-style-type: none">• <code>mexget.c</code>
See Also	<code>mexSet</code>

Purpose	Copy of variable from specified workspace						
C Syntax	<pre>#include "mex.h" mxArray *mexGetVariable(const char *workspace, const char *varname);</pre>						
Fortran Syntax	<pre>#include "fintrf.h" mwPointer mexGetVariable(workspace, varname) character*(*) workspace, varname</pre>						
Arguments	<p>workspace Specifies where <code>mexGetVariable</code> searches for array <code>varname</code>. The possible values are:</p> <table><tr><td>base</td><td>Search for the variable in the base workspace.</td></tr><tr><td>caller</td><td>Search for the variable in the caller workspace.</td></tr><tr><td>global</td><td>Search for the variable in the global workspace.</td></tr></table> <p>varname Name of the variable to copy</p>	base	Search for the variable in the base workspace.	caller	Search for the variable in the caller workspace.	global	Search for the variable in the global workspace.
base	Search for the variable in the base workspace.						
caller	Search for the variable in the caller workspace.						
global	Search for the variable in the global workspace.						
Returns	Copy of the variable on success. Returns NULL in C (0 on Fortran) on failure. A common cause of failure is specifying a variable that is not currently in the workspace. Perhaps the variable was in the workspace at one time but has since been cleared.						
Description	<p>Call <code>mexGetVariable</code> to get a copy of the specified variable. The returned <code>mxArray</code> contains a copy of all the data and characteristics that the variable had in the other workspace. Modifications to the returned <code>mxArray</code> do not affect the variable in the workspace unless you write the copy back to the workspace with <code>mexPutVariable</code>.</p> <p>Use <code>mxDestroyArray</code> to destroy the <code>mxArray</code> created by this routine when you are finished with it.</p>						

mexGetVariable (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/mex*.

- `mexgetarray.c`

See Also

`mexGetVariablePtr`, `mexPutVariable`, `mxDestroyArray`

mexGetVariablePtr (C and Fortran)

Purpose	Read-only pointer to variable from another workspace						
C Syntax	<pre>#include "mex.h" const mxArray *mexGetVariablePtr(const char *workspace, const char *varname);</pre>						
Fortran Syntax	<pre>#include "fintrf.h" mwPointer mexGetVariablePtr(workspace, varname) character*(*) workspace, varname</pre>						
Arguments	<p>workspace Specifies which workspace you want mexGetVariablePtr to search. The possible values are</p> <table><tr><td>base</td><td>Search for the variable in the base workspace.</td></tr><tr><td>caller</td><td>Search for the variable in the caller workspace.</td></tr><tr><td>global</td><td>Search for the variable in the global workspace.</td></tr></table> <p>varname Name of a variable in another workspace. This is a variable name, not an mxArray pointer.</p>	base	Search for the variable in the base workspace.	caller	Search for the variable in the caller workspace.	global	Search for the variable in the global workspace.
base	Search for the variable in the base workspace.						
caller	Search for the variable in the caller workspace.						
global	Search for the variable in the global workspace.						
Returns	Read-only pointer to the mxArray on success. Returns NULL in C (0 in Fortran) on failure.						
Description	<p>Call mexGetVariablePtr to get a read-only pointer to the specified variable, varname, into your MEX-file workspace. This command is useful for examining an mxArray's data and characteristics. If you want to change data or characteristics, use mexGetVariable (along with mexPutVariable) instead of mexGetVariablePtr.</p> <p>If you simply want to examine data or characteristics, mexGetVariablePtr offers superior performance because the caller wants to pass only a pointer to the array.</p>						

mexGetVariablePtr (C and Fortran)

Examples

See the following examples in *matlabroot/extern/examples/mx*.

- `mxislogical.c`

See Also

`mexGetVariable`

Purpose	Determine if variable has global scope
Compatibility	<code>mexIsGlobal</code> has been removed. Use <code>mxIsFromGlobalWS</code> instead.
C Syntax	<pre>#include "matrix.h" bool mexIsGlobal(const mxArray *pm);</pre>
Fortran Syntax	<pre>#include "fintrf.h" integer*4 mexIsGlobal(pm) mwPointer pm</pre>
Arguments	<code>pm</code> Pointer to an <code>mxArray</code>
Returns	Logical 0 (false).
Description	Use <code>mexIsGlobal</code> to determine if the specified <code>mxArray</code> has global scope.
Examples	See the following examples in <code>matlabroot/extern/examples/mx</code> . <ul style="list-style-type: none">• <code>mxislogical.c</code>
See Also	<code>mexGetVariable</code> , <code>mexGetVariablePtr</code> , <code>mexPutVariable</code> , <code>global</code>

mexIsLocked (C and Fortran)

Purpose	Determine if MEX-file is locked
C Syntax	<pre>#include "mex.h" bool mexIsLocked(void);</pre>
Fortran Syntax	<pre>#include "fintrf.h" integer*4 mexIsLocked()</pre>
Returns	Logical 1 (true) if the MEX-file is locked; logical 0 (false) if the file is unlocked.
Description	<p>Call <code>mexIsLocked</code> to determine if the MEX-file is locked. By default, MEX-files are unlocked, meaning you can clear the MEX-file at any time.</p> <p>To unlock a MEX-file, call <code>mexUnlock</code>.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mex</code>.</p> <ul style="list-style-type: none">• <code>mexlock.c</code>• <code>mexlockf.F</code>
See Also	<code>mexLock</code> , <code>mexMakeArrayPersistent</code> , <code>mexMakeMemoryPersistent</code> , <code>mexUnlock</code> , <code>clear</code>

Purpose	Prevent clearing MEX-file from memory
C Syntax	<pre>#include "mex.h" void mexLock(void);</pre>
Fortran Syntax	<pre>#include "fintrf.h" subroutine mexLock()</pre>
Description	<p>By default, MEX-files are unlocked, meaning you can clear them at any time. Call <code>mexLock</code> to prohibit clearing a MEX-file.</p> <p>To unlock a MEX-file, you must call <code>mexUnlock</code>. Do not use the <code>munlock</code> function.</p> <p><code>mexLock</code> increments a lock count. If you call <code>mexLock</code> <i>n</i> times, call <code>mexUnlock</code> <i>n</i> times to unlock your MEX-file.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mex</code>.</p> <ul style="list-style-type: none">• <code>mexlock.c</code>• <code>mexlockf.F</code>
See Also	<code>mexIsLocked</code> , <code>mexMakeArrayPersistent</code> , <code>mexMakeMemoryPersistent</code> , <code>mexUnlock</code> , <code>clear</code>

mexMakeArrayPersistent (C and Fortran)

Purpose Make array persist after MEX-file completes

C Syntax

```
#include "mex.h"
void mexMakeArrayPersistent(mxArray *pm);
```

Fortran Syntax

```
#include "fintrf.h"
subroutine mexMakeArrayPersistent(pm)
mwPointer pm
```

Arguments

pm
Pointer to an mxArray created by an mxCreate* function

Description

By default, an mxArray allocated by an mxCreate* function is not persistent. The MATLAB memory management facility automatically frees a nonpersistent mxArray when the MEX-function finishes. If you want the mxArray to persist through multiple invocations of the MEX-function, you must call the mexMakeArrayPersistent function.

Note If you create a persistent mxArray, you are responsible for destroying it using mxDestroyArray when the MEX-file is cleared. If you do not destroy a persistent mxArray, MATLAB leaks memory. See mexAtExit to see how to register a function that gets called when the MEX-file is cleared. See mexLock to see how to lock your MEX-file so that it is never cleared.

See Also mexAtExit, mxDestroyArray, mexLock, mexMakeMemoryPersistent, and the mxCreate* functions

mexMakeMemoryPersistent (C and Fortran)

Purpose Make memory allocated by MATLAB software persist after MEX-function completes

C Syntax

```
#include "mex.h"
void mexMakeMemoryPersistent(void *ptr);
```

Fortran Syntax

```
#include "fintrf.h"
subroutine mexMakeMemoryPersistent(ptr)
mwPointer ptr
```

Arguments

ptr
Pointer to the beginning of memory allocated by one of the MATLAB memory allocation routines

Description By default, memory allocated by MATLAB software is nonpersistent, so it is freed automatically when the MEX-function finishes. If you want the memory to persist, you must call `mexMakeMemoryPersistent`.

Note If you create persistent memory, you are responsible for freeing it when the MEX-function is cleared. If you do not free the memory, MATLAB leaks memory. To free memory, use `mxFree`. See `mexAtExit` to see how to register a function that gets called when the MEX-function is cleared. See `mexLock` to see how to lock your MEX-function so that it is never cleared.

See Also `mexAtExit`, `mexLock`, `mexMakeArrayPersistent`, `mxMalloc`, `mxFree`, `mxMalloc`, `mxRealloc`

mexPrintf (C and Fortran)

Purpose	ANSI C PRINTF-style output routine
C Syntax	<pre>#include "mex.h" int mexPrintf(const char *message, ...);</pre>
Fortran Syntax	<pre>#include "fintrf.h" integer*4 mexPrintf(message) character*(*) message</pre>
Arguments	<p>message String to display. In C, the string can include conversion specifications, used by the ANSI C <code>printf</code> function.</p> <p>... In C, any arguments used in the message. Each argument must have a corresponding conversion specification.</p>
Returns	Number of characters printed including characters specified with backslash codes, such as <code>\n</code> and <code>\b</code> .
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 <code>printf</code> routine already linked inside MATLAB software, which avoids linking the entire <code>stdio</code> library into your MEX-file.</p> <p>In a C MEX-file, you must call <code>mexPrintf</code> instead of <code>printf</code> to display a string.</p> <hr/> <p>Note If you want the literal % in your message, use %% in the message string since % has special meaning to <code>printf</code>. Failing to do so causes unpredictable results.</p> <hr/>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mex</code>.</p> <ul style="list-style-type: none">• <code>mexfunction.c</code>

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`

See Also

`mexErrMsgIdAndTxt`, `mexErrMsgTxt`, `mexWarnMsgIdAndTxt`,
`mexWarnMsgTxt`

mexPutVariable (C and Fortran)

Purpose

Array from MEX-function into specified workspace

C Syntax

```
#include "mex.h"
int mexPutVariable(const char *workspace, const char *varname,
    const mxArray *pm);
```

Fortran Syntax

```
#include "fintrf.h"
integer*4 mexPutVariable(workspace, varname, pm)
character*(*) workspace, varname
mwPointer pm
```

Arguments

workspace

Specifies scope of the array you are copying. Values for workspace are:

base	Copy mxArray to the base workspace.
caller	Copy mxArray to the caller workspace.
global	Copy mxArray to the list of global variables.

varname

Name of mxArray in the workspace

pm

Pointer to the mxArray

Returns

0 on success; 1 on failure. A possible cause of failure is that pm is NULL in C (0 in Fortran).

Description

Call mexPutVariable to copy the mxArray, at pointer pm, from your MEX-function into the specified workspace. MATLAB software gives the name, varname, to the copied mxArray in the receiving workspace.

mexPutVariable makes the array accessible to other entities, such as MATLAB, user-defined functions, or other MEX-functions.

If a variable of the same name exists in the specified workspace, mexPutVariable 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 `mexPutVariable` to copy `Peaches` into the same workspace:

```
mexPutVariable("base", "Peaches", pm)
```

The value passed by `mexPutVariable` replaces the old value of `Peaches`.

Do not use MATLAB function names for variable names. Common variable names that conflict with function names include `i`, `j`, `mode`, `char`, `size`, or `path`. To determine whether a particular name is associated with a MATLAB function, use the `which` function.

Examples

See the following examples in `matlabroot/extern/examples/mex`.

- `mexgetarray.c`

See Also

`mexGetVariable`

mexSet (C)

Purpose Set value of specified Handle Graphics property

C Syntax

```
#include "mex.h"
int mexSet(double handle, const char *property,
           mxArray *value);
```

Arguments

handle
Handle to a particular graphics object

property
String naming a Handle Graphics property

value
Pointer to an mxArray holding the new value to assign to the property

Returns 0 on success; 1 on failure. Possible causes of failure include:

- Specifying a nonexistent property.
- Specifying an illegal value for that property, for example, specifying a string value for a numerical property.

Description Call `mexSet` to set the value of the property of a certain graphics object. `mexSet` is the API equivalent of the MATLAB `set` function. To get the value of a graphics property, call `mexGet`.

Examples See the following examples in the `matlabroot/extern/examples/mex` folder.

- `mexcallmatlab.c`
- `mexget.c`
- `mexgetarray.c`

See Also `mexGet`

Purpose Control response of MEXCALLMATLAB to errors

C Syntax

```
#include "mex.h"
void mexSetTrapFlag(int trapflag);
```

Note mexSetTrapFlag will be removed in a future version. Use mexCallMATLABWithTrap instead.

Fortran Syntax

```
subroutine mexSetTrapFlag(trapflag)
integer*4 trapflag
```

Arguments

trapflag
Control flag. Possible 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 the MATLAB response to errors in mexCallMATLAB.

If you do not call mexSetTrapFlag, then whenever MATLAB detects an error in a call to mexCallMATLAB, MATLAB automatically terminates the MEX-file and returns control to the MATLAB prompt. Calling mexSetTrapFlag with trapflag set to 0 is equivalent to not calling mexSetTrapFlag at all.

If you call mexSetTrapFlag and set the trapflag to 1, then whenever MATLAB detects an error in a call to mexCallMATLAB, MATLAB does not automatically terminate the MEX-file. Rather, MATLAB returns control to the line in the MEX-file immediately following the call to mexCallMATLAB. The MEX-file is then responsible for taking an appropriate response to the error.

If you call mexSetTrapFlag, the value of the trapflag you set remains in effect until the next call to mexSetTrapFlag within that MEX-file or,

mexSetTrapFlag (C and Fortran)

if there are no more calls to `mexSetTrapFlag`, until the MEX-file exits.
If a routine defined in a MEX-file calls another MEX-file, MATLAB:

- 1** Saves the current value of the `trapflag` in the first MEX-file.
- 2** Calls the second MEX-file with the `trapflag` initialized to 0 within that file.
- 3** Restores the saved value of `trapflag` in the first MEX-file when the second MEX-file exits.

See Also

`mexCallMATLAB`, `mexCallMATLABWithTrap`, `mexAtExit`, `mexErrMsgTxt`

Purpose	Allow clearing MEX-file from memory
C Syntax	<pre>#include "mex.h" void mexUnlock(void);</pre>
Fortran Syntax	<pre>#include "fintrf.h" subroutine mexUnlock()</pre>
Description	<p>By default, MEX-files are unlocked, meaning you can clear them at any time. Calling <code>mexLock</code> locks a MEX-file so that you cannot clear it from memory. Call <code>mexUnlock</code> to remove the lock.</p> <p><code>mexLock</code> increments a lock count. If you called <code>mexLock</code> <code>n</code> times, call <code>mexUnlock</code> <code>n</code> times to unlock your MEX-file.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mex</code>.</p> <ul style="list-style-type: none">• <code>mexlock.c</code>• <code>mexlockf.F</code>
See Also	<code>mexIsLocked</code> , <code>mexLock</code> , <code>mexMakeArrayPersistent</code> , <code>mexMakeMemoryPersistent</code> , <code>clear</code>

mexWarnMsgIdAndTxt (C and Fortran)

Purpose	Warning message with identifier
C Syntax	<pre>#include "mex.h" void mexWarnMsgIdAndTxt(const char *warningid, const char *warningmsg, ...);</pre>
Fortran Syntax	<pre>#include "fintrf.h" subroutine mexWarnMsgIdAndTxt(warningid, warningmsg) character*(*) warningid, warningmsg</pre>
Arguments	<p><code>warningid</code> String containing a MATLAB message identifier. For information on creating identifiers, see “Message Identifiers”.</p> <p><code>warningmsg</code> String to display. In C, the string can include conversion specifications, used by the ANSI C <code>printf</code> function.</p> <p>...</p> <p>In C, any arguments used in the message. Each argument must have a corresponding conversion specification.</p>
Description	<p>The <code>mexWarnMsgIdAndTxt</code> function writes a warning message to the MATLAB window. For more information, see the <code>warning</code> function syntax statement using a message identifier.</p> <p>Unlike <code>mexErrMsgIdAndTxt</code>, calling <code>mexWarnMsgIdAndTxt</code> does not terminate the MEX-file.</p>
See Also	<code>mexErrMsgIdAndTxt</code>

mexWarnMsgTxt (C and Fortran)

Purpose Warning message

Note mexWarnMsgTxt is not recommended. Use mexWarnMsgIdAndTxt instead.

C Syntax

```
#include "mex.h"
void mexWarnMsgTxt(const char *warningmsg);
```

Fortran Syntax

```
subroutine mexWarnMsgTxt(warningmsg)
character*(*) warningmsg
```

Arguments

warningmsg
String containing the warning message to display

Description

mexWarnMsgTxt causes MATLAB software to display the contents of warningmsg. mexWarnMsgTxt does not terminate the MEX-file.

See Also mexErrMsgIdAndTxt, mexWarnMsgIdAndTxt

mwIndex (C and Fortran)

Purpose Type for index values

Description `mwIndex` is a type that represents index values, such as indices into arrays. Use this function for cross-platform flexibility. By default, `mwIndex` is equivalent to `int` in C. When using the `mex -largeArrayDims` switch, `mwIndex` is equivalent to `size_t` in C. In Fortran, `mwIndex` is similarly equivalent to `INTEGER*4` or `INTEGER*8`, based on platform and compilation flags.

The C header file containing this type is:

```
#include "matrix.h"
```

In Fortran, `mwIndex` is a preprocessor macro. The Fortran header file containing this type is:

```
#include "fintrf.h"
```

See Also `mex`, `mwSize`, `mwSignedIndex`

Purpose

Platform-independent pointer type

Description

`mwPointer` is a preprocessor macro that declares the appropriate Fortran type representing a pointer to an `mxArray` or to other data that is not of a native Fortran type, such as memory allocated by `mxMalloc`. On 32-bit platforms, the Fortran type that represents a pointer is `INTEGER*4`; on 64-bit platforms, it is `INTEGER*8`. The Fortran preprocessor translates `mwPointer` to the Fortran declaration that is appropriate for the platform on which you compile your file.

If your Fortran compiler supports preprocessing, you can use `mwPointer` to declare functions, arguments, and variables that represent pointers. If you cannot use `mwPointer`, you must ensure that your declarations have the correct size for the platform on which you are compiling Fortran code.

The Fortran header file containing this type is:

```
#include "fintfrf.h"
```

Examples

This example declares the arguments for `mexFunction` in a Fortran MEX-file:

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
mwPointer plhs(*), prhs(*)
integer nlhs, nrhs
```

For additional examples, see the Fortran files with names ending in `.F` in the `matlabroot/extern/examples` folder.

mwSignedIndex (C and Fortran)

Purpose Signed integer type for size values

Description mwSignedIndex is a signed integer type that represents size values, such as array dimensions. Use this function for cross-platform flexibility. By default, mwSignedIndex is equivalent to ptrdiff_t in C++. In Fortran, mwSignedIndex is similarly equivalent to INTEGER*4 or INTEGER*8, based on platform and compilation flags.

The C header file containing this type is:

```
#include "matrix.h"
```

The Fortran header file containing this type is:

```
#include "fintrf.h"
```

See Also mwSize, mwIndex

Purpose

Type for size values

Description

mwSize is a type that represents size values, such as array dimensions. Use this function for cross-platform flexibility. By default, mwSize is equivalent to int in C. When using the mex -largeArrayDims switch, mwSize is equivalent to size_t in C. In Fortran, mwSize is similarly equivalent to INTEGER*4 or INTEGER*8, based on platform and compilation flags.

The C header file containing this type is:

```
#include "matrix.h"
```

In Fortran, mwSize is a preprocessor macro. The Fortran header file containing this type is:

```
#include "fintrf.h"
```

See Also

mex, mwIndex, mwSignedIndex

mxAddField (C and Fortran)

Purpose Add field to structure array

C Syntax

```
#include "matrix.h"
extern int mxAddField(mxArray *pm, const char *fieldname);
```

Fortran Syntax

```
integer*4 mxAddField(pm, fieldname)
mwPointer pm
character*(*) fieldname
```

Arguments

pm
 Pointer to a structure mxArray

fieldname
 Name of the field you want to add

Returns Field number on success, or -1 if inputs are invalid or an out-of-memory condition occurs.

Description Call `mxAddField` to add a field to a structure array. Create the values with the `mxCreate*` functions and use `mxSetFieldByNumber` to set the individual values for the field.

See Also `mxRemoveField`, `mxSetFieldByNumber`

Purpose

Type for MATLAB array

Description

The fundamental type underlying MATLAB data. For information on how the MATLAB array works with MATLAB-supported variables, see “MATLAB Data”.

`mxArray` is a C language opaque type.

All C MEX-files start with a gateway routine, called `mexFunction`, which requires `mxArray` for both input and output parameters. For information about the C MEX-file gateway routine, see “Components of MEX-File”.

Once you have MATLAB data in your MEX-file, use functions in the MX Matrix Library to manipulate the data, and functions in the MEX Library to perform operations in the MATLAB environment. You use `mxArray` to pass data to and from these functions.

Use any of the `mxCreate*` functions to create data, and the corresponding `mxDestroyArray` function to free memory.

The header file containing this type is:

```
#include "matrix.h"
```

Example

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcreatecharmatrixfromstr.c`

Tips

- For information about data in MATLAB language scripts and functions, see “Data Types”.

See Also

`mexFunction`, `mxClassID`, `mxCreateDoubleMatrix`,
`mxCreateNumericArray`, `mxCreateString`, `mxDestroyArray`,
`mxGetData`, `mxSetData`

mxArrayToString (C)

Purpose Array to string

C Syntax

```
#include "matrix.h"
char *mxArrayToString(const mxArray *array_ptr);
```

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

Returns C-style string. Returns NULL on failure. Possible reasons for failure include out of memory and specifying an mxArray that is not a string mxArray.

Description Call mxArrayToString to copy the character data of a string mxArray into a C-style string. 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. This function is similar to mxGetString, except that

- It does not require the length of the string as an input.
- It supports multibyte encoded characters.

mxArrayToString does not free the dynamic memory that the char pointer points to. Consequently, you should typically free the string (using mxFree) immediately after you have finished using it.

Examples See the following examples in *matlabroot/extern/examples/mex*.

- mexatexit.c

See the following examples in *matlabroot/extern/examples/mx*.

- mxcreatecharmatrixfromstr.c

- `mxislogical.c`

See Also

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

mxAssert (C)

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 Like the ANSI C assert macro, mxAssert checks the value of an assertion, and continues execution only if the assertion holds. If expr evaluates to logical 1 (true), mxAssert does nothing. If expr evaluates to logical 0 (false), mxAssert terminates the MEX-file and prints an error to the MATLAB command window. The error contains 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 do not want a description to follow the failed assertion message.

The mex script turns off these assertions when building optimized MEX-functions, so use this for debugging purposes only. Build the MEX-file using the syntax `mex -g filename` in order to use mxAssert.

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; 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. You can use these checks during development and then remove them when the code works properly, letting you use them for troubleshooting during development without slowing down the final product.

See Also mxAssertS, mexErrMsgIdAndTxt

Purpose	Check assertion value without printing assertion text
C Syntax	<pre>#include "matrix.h" void mxAssertS(int expr, char *error_message);</pre>
Arguments	<p>expr Value of assertion</p> <p>error_message Description of why assertion failed</p>
Description	mxAssertS is like mxAssert, except mxAssertS does not print the text of the failed assertion.
See Also	mxAssert

mxCalcSingleSubscript (C and Fortran)

Purpose Offset from first element to desired element

C Syntax

```
#include "matrix.h"
mwIndex mxCalcSingleSubscript(const mxArray *pm, mwSize nsubs,
                               mwIndex *subs);
```

Fortran Syntax

```
mwIndex mxCalcSingleSubscript(pm, nsubs, subs)
mwPointer pm
mwSize nsubs
mwIndex subs
```

Arguments

pm Pointer to an mxArray

nsubs Number of elements in the subs array. Typically, you set nsubs equal to the number of dimensions in the mxArray that pm points to.

subs An array of integers. Each value in the array specifies that dimension's subscript. In C syntax, the value in subs[0] specifies the row subscript, and the value in subs[1] specifies the column subscript. Use zero-based indexing for subscripts. For example, to express the starting element of a two-dimensional mxArray in subs, set subs[0] to 0 and subs[1] to 0.

In Fortran syntax, the value in subs(1) specifies the row subscript, and the value in subs(2) specifies the column subscript. Use 1-based indexing for subscripts. For example, to express the starting element of a two-dimensional mxArray in subs, set subs(1) to 1 and subs(2) to 1.

Returns The number of elements, or *index*, between the start of the mxArray and the specified subscript. This is the linear index equivalent of

mxCalcSingleSubscript (C and Fortran)

the subscripts. Many MX Matrix Library routines (for example, `mxGetField`) require an index as an argument.

If `subs` describes the starting element of an `mxArray`, `mxCalcSingleSubscript` returns 0. If `subs` describes the final element of an `mxArray`, `mxCalcSingleSubscript` returns `N-1` (where `N` is the total number of elements).

Description

Call `mxCalcSingleSubscript` to determine how many elements there are between the beginning of the `mxArray` and a given element of that `mxArray`. The function converts subscripts to linear indices.

For example, given a subscript like `(5,7)`, `mxCalcSingleSubscript` returns the distance from the first 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. For examples showing the internal representation, see “Data Storage”.

Avoid using `mxCalcSingleSubscript` to traverse the elements of an array. In C, it is more efficient to do this by finding the array’s starting address and then using pointer autoincrementing to access successive elements. For example, to find the starting address of a numerical array, call `mxGetPr` or `mxGetPi`.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcalsinglesubscript.c`

See Also

`mxGetCell`, `mxSetCell`

mxCalloc (C and Fortran)

Purpose Allocate dynamic memory for array, initialized to 0, using MATLAB memory manager

C Syntax

```
#include "matrix.h"
#include <stdlib.h>
void *mxCalloc(mwSize n, mwSize size);
```

Fortran Syntax

```
mwPointer mxCalloc(n, size)
mwSize n, size
```

Arguments

n Number of elements to allocate. This must be a nonnegative number.

size Number of bytes per element. (The C `sizeof` operator calculates the number of bytes per element.)

Returns Pointer to the start of the allocated dynamic memory, if successful. If unsuccessful in a MAT or engine standalone application, `mxCalloc` returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt.

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

Description `mxCalloc` allocates contiguous heap space sufficient to hold `n` elements of `size` bytes each, and initializes this newly allocated memory to 0. Use `mxCalloc` instead of the ANSI C `calloc` function to allocate memory in MATLAB applications.

In MEX-files, but not MAT or engine applications, `mxCalloc` registers the allocated memory with the MATLAB memory manager. When control returns to the MATLAB prompt, the memory manager then automatically frees, or *deallocates*, this memory.

How you manage the memory created by this function depends on the purpose of the data assigned to it. If you assign it to an output argument

in `plhs[]` using the `mxSetPr` function, MATLAB is responsible for freeing the memory.

If you use the data internally, the MATLAB memory manager maintains a list of all memory allocated by the function and automatically frees (deallocates) the memory when control returns to the MATLAB prompt. In general, we recommend that MEX-file functions destroy their own temporary arrays and free their own dynamically allocated memory. It is more efficient to perform this cleanup in the source MEX-file than to rely on the automatic mechanism. Therefore, when you finish using the memory allocated by this function, call `mxFree` to deallocate the memory.

If you do not assign this data to an output argument, and you want it to persist after the MEX-file completes, call `mexMakeMemoryPersistent` after calling this function. 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.

Examples

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillSetData.c`
- `phonebook.c`
- `revord.c`

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcalcsinglesubscript.c`
- `mxsetdimensions.c`

See Also

`mexAtExit`, `mexMakeArrayPersistent`, `mexMakeMemoryPersistent`, `mxDestroyArray`, `mxFree`, `mxMalloc`, `mxRealloc`

mxChar (C)

Purpose Type for string array

Description MATLAB stores an mxArray string as type mxChar to represent the C-style char type. MATLAB uses 16-bit unsigned integer character encoding for Unicode characters.

The header file containing this type is:

```
#include "matrix.h"
```

Examples See the following examples in *matlabroot/extern/examples/mx*.

- `mxmalloc.c`
- `mxcreatecharmatrixfromstr.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

Tips

- For information about data in MATLAB language scripts and functions, see “Data Types”.

See Also `mxCreateCharArray`

Purpose Enumerated value identifying class of array

C Syntax

```
typedef enum {  
    mxUNKNOWN_CLASS,  
    mxCELL_CLASS,  
    mxSTRUCT_CLASS,  
    mxLOGICAL_CLASS,  
    mxCHAR_CLASS,  
    mxVOID_CLASS,  
    mxDOUBLE_CLASS,  
    mxSINGLE_CLASS,  
    mxINT8_CLASS,  
    mxUINT8_CLASS,  
    mxINT16_CLASS,  
    mxUINT16_CLASS,  
    mxINT32_CLASS,  
    mxUINT32_CLASS,  
    mxINT64_CLASS,  
    mxUINT64_CLASS,  
    mxFUNCTION_CLASS  
} mxClassID;
```

Constants

`mxUNKNOWN_CLASS`

Undetermined class. You cannot specify this category for an `mxArray`; however, if `mxGetClassID` cannot identify the class, it returns this value.

`mxCELL_CLASS`

Identifies a cell `mxArray`.

`mxSTRUCT_CLASS`

Identifies a structure `mxArray`.

`mxLOGICAL_CLASS`

Identifies a logical `mxArray`, an `mxArray` of `mxLogical` data.

`mxCHAR_CLASS`

Identifies a string `mxArray`, an `mxArray` whose data is represented as `mxChar`.

mxClassID (C)

<code>mxVOID_CLASS</code>	Reserved.
<code>mxDOUBLE_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxSINGLE_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxINT8_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxUINT8_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxINT16_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxUINT16_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxINT32_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxUINT32_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxINT64_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.
<code>mxUINT64_CLASS</code>	Identifies a numeric <code>mxArray</code> whose data is stored as the type specified in the MATLAB Primitive Types table.

`mxFUNCTION_CLASS`
Identifies a function handle `mxArray`.

Description

Various MX Matrix Library functions require or return an `mxClassID` argument. `mxClassID` identifies the way in which the `mxArray` represents its data elements.

The following table shows MATLAB types with their equivalent C types. Use the type from the right-most column for reading `mxArrays` with the `mxClassID` value shown in the left column.

MATLAB Primitive Types

mxClassID Value	MATLAB Type	MEX Type	C Primitive Type
<code>mxINT8_CLASS</code>	<code>int8</code>	<code>int8_T</code>	char, byte
<code>mxUINT8_CLASS</code>	<code>uint8</code>	<code>uint8_T</code>	unsigned char, byte
<code>mxINT16_CLASS</code>	<code>int16</code>	<code>int16_T</code>	short
<code>mxUINT16_CLASS</code>	<code>uint16</code>	<code>uint16_T</code>	unsigned short
<code>mxINT32_CLASS</code>	<code>int32</code>	<code>int32_T</code>	int
<code>mxUINT32_CLASS</code>	<code>uint32</code>	<code>uint32_T</code>	unsigned int
<code>mxINT64_CLASS</code>	<code>int64</code>	<code>int64_T</code>	long long
<code>mxUINT64_CLASS</code>	<code>uint64</code>	<code>uint64_T</code>	unsigned long long
<code>mxSINGLE_CLASS</code>	<code>single</code>	<code>float</code>	float
<code>mxDOUBLE_CLASS</code>	<code>double</code>	<code>double</code>	double

Examples

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See Also

`mxGetClassID`, `mxCreateNumericArray`

mxClassIDFromClassName (Fortran)

Purpose	Identifier corresponding to class
Fortran Syntax	<pre>integer*4 mxClassIDFromClassName(classname) character*(*) classname</pre>
Arguments	<pre>classname</pre> <p>character array specifying a MATLAB class name. For a list of valid <code>classname</code> choices, see the <code>mxIsClass</code> reference page.</p>
Returns	Numeric identifier used internally by MATLAB software to represent the MATLAB class, <code>classname</code> . Returns unknown if <code>classname</code> is not a recognized MATLAB class.
Description	Use <code>mxClassIDFromClassName</code> to obtain an identifier for any MATLAB class. This function is most commonly used to provide a <code>classid</code> argument to <code>mxCreateNumericArray</code> and <code>mxCreateNumericMatrix</code> .
Examples	See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• <code>matsqint8.F</code>
See Also	<code>mxGetClassName</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxIsClass</code>

Purpose	Flag specifying whether array has imaginary components
C Syntax	<pre>typedef enum mxComplexity {mxREAL=0, mxCOMPLEX};</pre>
Constants	<p><code>mxREAL</code> Identifies an mxArray with no imaginary components.</p> <p><code>mxCOMPLEX</code> Identifies an mxArray with imaginary components.</p>
Description	Various MX Matrix Library functions require an <code>mxComplexity</code> argument. You can set an <code>mxComplex</code> argument to either <code>mxREAL</code> or <code>mxCOMPLEX</code> .
Examples	See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• <code>mxcalcsinglesubscript.c</code>
See Also	<code>mxCreateNumericArray</code> , <code>mxCreateDoubleMatrix</code> , <code>mxCreateSparse</code>

mxCopyCharacterToPtr (Fortran)

Purpose CHARACTER values from Fortran array to pointer array

Fortran Syntax subroutine mxCopyCharacterToPtr(y, px, n)
character*(*) y
mwPointer px
mwSize n

Arguments

y	character Fortran array
px	Pointer to character or name array
n	Number of elements to copy

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 pointer arrays and ordinary Fortran character arrays.

See Also mxCopyPtrToCharacter, mxCreateCharArray, mxCreateString, mxCreateCharMatrixFromStrings

mxCopyComplex16ToPtr (Fortran)

Purpose	COMPLEX*16 values from Fortran array to pointer array
Fortran Syntax	<pre>subroutine mxCopyComplex16ToPtr(y, pr, pi, n) complex*16 y(n) mwPointer pr, pi mwSize n</pre>
Arguments	<p><i>y</i> COMPLEX*16 Fortran array</p> <p><i>pr</i> Pointer to the real data of a double-precision MATLAB array</p> <p><i>pi</i> Pointer to the imaginary data of a double-precision MATLAB array</p> <p><i>n</i> Number of elements to copy</p>
Description	<p><code>mxCopyComplex16ToPtr</code> copies <i>n</i> COMPLEX*16 values from the Fortran COMPLEX*16 array <i>y</i> into the MATLAB arrays pointed to by <i>pr</i> and <i>pi</i>.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>convec.F</code>
See Also	<code>mxCopyPtrToComplex16</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxGetData</code> , <code>mxGetImagData</code>

mxCopyComplex8ToPtr (Fortran)

Purpose	COMPLEX*8 values from Fortran array to pointer array
Fortran Syntax	<pre>subroutine mxCopyComplex8ToPtr(y, pr, pi, n) complex*8 y(n) mwPointer pr, pi mwSize n</pre>
Arguments	<p>y COMPLEX*8 Fortran array</p> <p>pr Pointer to the real data of a single-precision MATLAB array</p> <p>pi Pointer to the imaginary data of a single-precision MATLAB array</p> <p>n Number of elements to copy</p>
Description	<p>mxCopyComplex8ToPtr copies n COMPLEX*8 values from the Fortran COMPLEX*8 array y into the MATLAB arrays pointed to by pr and pi.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
See Also	<p>mxCopyPtrToComplex8, mxCreateNumericArray, mxCreateNumericMatrix, mxGetData, mxGetImagData</p>

Purpose	INTEGER*1 values from Fortran array to pointer array
Fortran Syntax	<pre>subroutine mxCopyInteger1ToPtr(y, px, n) integer*1 y(n) mwPointer px mwSize n</pre>
Arguments	<p>y INTEGER*1 Fortran array</p> <p>px Pointer to the real or imaginary data of the array</p> <p>n Number of elements to copy</p>
Description	<p>mxCopyInteger1ToPtr copies n INTEGER*1 values from the Fortran INTEGER*1 array y into the MATLAB array pointed to by px, either a real or an imaginary array.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• matsqint8.F
See Also	<p>mxCopyPtrToInteger1, mxCreateNumericArray, mxCreateNumericMatrix</p>

mxCopyInteger2ToPtr (Fortran)

Purpose INTEGER*2 values from Fortran array to pointer array

Fortran Syntax

```
subroutine mxCopyInteger2ToPtr(y, px, n)
integer*2 y(n)
mwPointer px
mwSize n
```

Arguments

y
INTEGER*2 Fortran array

px
Pointer to the real or imaginary data of the array

n
Number of elements to copy

Description mxCopyInteger2ToPtr copies n INTEGER*2 values from the Fortran INTEGER*2 array y into the MATLAB array pointed to by px, either a real or an imaginary array.

Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

See Also mxCopyPtrToInteger2, mxCreateNumericArray, mxCreateNumericMatrix

Purpose INTEGER*4 values from Fortran array to pointer array

Fortran Syntax

```
subroutine mxCopyInteger4ToPtr(y, px, n)
integer*4 y(n)
mwPointer px
mwSize n
```

Arguments

y
INTEGER*4 Fortran array

px
Pointer to the real or imaginary data of the array

n
Number of elements to copy

Description mxCopyInteger4ToPtr copies n INTEGER*4 values from the Fortran INTEGER*4 array y into the MATLAB array pointed to by px, either a real or an imaginary array.

Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

See Also mxCopyPtrToInteger4, mxCreateNumericArray, mxCreateNumericMatrix

mxCopyPtrToCharacter (Fortran)

Purpose CHARACTER values from pointer array to Fortran array

Fortran Syntax
subroutine mxCopyPtrToCharacter(px, y, n)
mwPointer px
character*(*) y
mwSize n

Arguments

px	Pointer to character or name array
y	character Fortran array
n	Number of elements to copy

Description mxCopyPtrToCharacter copies n character values from the MATLAB array pointed to by px into the Fortran character array y. This subroutine is essential for copying character data from MATLAB pointer arrays into ordinary Fortran character arrays.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- matdemo2.F

See Also mxCopyCharacterToPtr, mxCreateCharArray, mxCreateString, mxCreateCharMatrixFromStrings

mxCopyPtrToComplex16 (Fortran)

Purpose	COMPLEX*16 values from pointer array to Fortran array
Fortran Syntax	<pre>subroutine mxCopyPtrToComplex16(pr, pi, y, n) mwPointer pr, pi complex*16 y(n) mwSize n</pre>
Arguments	<p><code>pr</code> Pointer to the real data of a double-precision MATLAB array</p> <p><code>pi</code> Pointer to the imaginary data of a double-precision MATLAB array</p> <p><code>y</code> COMPLEX*16 Fortran array</p> <p><code>n</code> Number of elements to copy</p>
Description	<p><code>mxCopyPtrToComplex16</code> copies <code>n</code> COMPLEX*16 values from the MATLAB arrays pointed to by <code>pr</code> and <code>pi</code> into the Fortran COMPLEX*16 array <code>y</code>.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>convec.F</code>
See Also	<code>mxCopyComplex16ToPtr</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxGetData</code> , <code>mxGetImagData</code>

mxCopyPtrToComplex8 (Fortran)

Purpose COMPLEX*8 values from pointer array to Fortran array

**Fortran
Syntax** subroutine mxCopyPtrToComplex8(pr, pi, y, n)
 mwPointer pr, pi
 complex*8 y(n)
 mwSize n

Arguments pr
 Pointer to the real data of a single-precision MATLAB array

 pi
 Pointer to the imaginary data of a single-precision MATLAB array

 y
 COMPLEX*8 Fortran array

 n
 Number of elements to copy

Description mxCopyPtrToComplex8 copies n COMPLEX*8 values from the MATLAB arrays pointed to by pr and pi into the Fortran COMPLEX*8 array y.

 Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

See Also mxCopyComplex8ToPtr, mxCreateNumericArray,
 mxCreateNumericMatrix, mxGetData, mxGetImagData

Purpose	INTEGER*1 values from pointer array to Fortran array
Fortran Syntax	<pre>subroutine mxCopyPtrToInteger1(px, y, n) mwPointer px integer*1 y(n) mwSize n</pre>
Arguments	<p>px Pointer to the real or imaginary data of the array</p> <p>y INTEGER*1 Fortran array</p> <p>n Number of elements to copy</p>
Description	<p>mxCopyPtrToInteger1 copies n INTEGER*1 values from the MATLAB array pointed to by px, either a real or imaginary array, into the Fortran INTEGER*1 array y.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• matsqint8.F
See Also	<p>mxCopyInteger1ToPtr, mxCreateNumericArray, mxCreateNumericMatrix</p>

mxCopyPtrToInteger2 (Fortran)

Purpose INTEGER*2 values from pointer array to Fortran array

Fortran Syntax

```
subroutine mxCopyPtrToInteger2(px, y, n)
mwPointer px
integer*2 y(n)
mwSize n
```

Arguments

px Pointer to the real or imaginary data of the array

y INTEGER*2 Fortran array

n Number of elements to copy

Description mxCopyPtrToInteger2 copies n INTEGER*2 values from the MATLAB array pointed to by px, either a real or an imaginary array, into the Fortran INTEGER*2 array y.

Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

See Also mxCopyInteger2ToPtr, mxCreateNumericArray, mxCreateNumericMatrix

Purpose INTEGER*4 values from pointer array to Fortran array

Fortran Syntax

```
subroutine mxCopyPtrToInteger4(px, y, n)
mwPointer px
integer*4 y(n)
mwSize n
```

Arguments

px
Pointer to the real or imaginary data of the array

y
INTEGER*4 Fortran array

n
Number of elements to copy

Description mxCopyPtrToInteger4 copies n INTEGER*4 values from the MATLAB array pointed to by px, either a real or an imaginary array, into the Fortran INTEGER*4 array y.

Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

See Also mxCopyInteger4ToPtr, mxCreateNumericArray, mxCreateNumericMatrix

mxCopyPtrToPtrArray (Fortran)

Purpose Pointer values from pointer array to Fortran array

Fortran Syntax

```
subroutine mxCopyPtrToPtrArray(px, y, n)
mwPointer px
mwPointer y(n)
mwSize n
```

Arguments

px Pointer to pointer array

y Fortran array of mwPointer values

n Number of pointers to copy

Description mxCopyPtrToPtrArray copies n pointers from the MATLAB array pointed to by px into the Fortran array y. This subroutine is essential for copying the output of matGetDir into an array of pointers. After calling this function, each element of y contains a pointer to a string. You can convert these strings to Fortran character arrays by passing each element of y as the first argument to mxCopyPtrToCharacter.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- matdemo2.F

See Also matGetDir, mxCopyPtrToCharacter

Purpose	REAL*4 values from pointer array to Fortran array
Fortran Syntax	<pre>subroutine mxCopyPtrToReal4(px, y, n) mwPointer px real*4 y(n) mwSize n</pre>
Arguments	<p>px Pointer to the real or imaginary data of a single-precision MATLAB array</p> <p>y REAL*4 Fortran array</p> <p>n Number of elements to copy</p>
Description	<p>mxCopyPtrToReal4 copies n REAL*4 values from the MATLAB array pointed to by px, either a pr or pi array, into the Fortran REAL*4 array y.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
See Also	<code>mxCopyReal4ToPtr</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxGetData</code> , <code>mxGetImagData</code>

mxCopyPtrToReal8 (Fortran)

Purpose	REAL*8 values from pointer array to Fortran array
Fortran Syntax	<pre>subroutine mxCopyPtrToReal8(px, y, n) mwPointer px real*8 y(n) mwSize n</pre>
Arguments	<p>px Pointer to the real or imaginary data of a double-precision MATLAB array</p> <p>y REAL*8 Fortran array</p> <p>n Number of elements to copy</p>
Description	<p>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.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/eng_mat</i>.</p> <ul style="list-style-type: none">• fengdemo.F <p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• timestwo.F• xtimesy.F
See Also	<code>mxCopyReal8ToPtr</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxGetData</code> , <code>mxGetImagData</code>

Purpose	REAL*4 values from Fortran array to pointer array
Fortran Syntax	<pre>subroutine mxCopyReal4ToPtr(y, px, n) real*4 y(n) mwPointer px mwSize n</pre>
Arguments	<p>y REAL*4 Fortran array</p> <p>px Pointer to the real or imaginary data of a single-precision MATLAB array</p> <p>n Number of elements to copy</p>
Description	<p>mxCopyReal4ToPtr copies n REAL*4 values from the Fortran REAL*4 array y into the MATLAB array pointed to by px, either a pr or pi array.</p> <p>Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.</p>
See Also	<code>mxCopyPtrToReal4</code> , <code>mxCreateNumericArray</code> , <code>mxCreateNumericMatrix</code> , <code>mxGetData</code> , <code>mxGetImagData</code>

mxCopyReal8ToPtr (Fortran)

Purpose REAL*8 values from Fortran array to pointer array

Fortran Syntax

```
subroutine mxCopyReal8ToPtr(y, px, n)
real*8 y(n)
mwPointer px
mwSize n
```

Arguments

y
REAL*8 Fortran array

px
Pointer to the real or imaginary data of a double-precision MATLAB array

n
Number of elements to copy

Description mxCopyReal8ToPtr 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. Sets up standard Fortran arrays for passing as arguments to or from the computation routine of a MEX-file. Use this subroutine with Fortran compilers that do not support the %VAL construct.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- matdemo1.F
- fengdemo.F

See the following examples in *matlabroot/extern/examples/refbook*.

- timestwo.F
- xtimesy.F

See Also mxCopyPtrToReal8, mxCreateNumericArray, mxCreateNumericMatrix, mxGetData, mxGetImagData

mxCreateCellArray (C and Fortran)

Purpose	N-D cell array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateCellArray(mwSize ndim, const mwSize *dims);</pre>
Fortran Syntax	<pre>mwPointer mxCreateCellArray(ndim, dims) mwSize ndim mwSize dims(ndim)</pre>
Arguments	<p>ndim Number of dimensions in the created cell. For example, to create a three-dimensional cell mxArray, set ndim to 3.</p> <p>dims Dimensions array. Each element in the dimensions array contains the size of the mxArray in that dimension. For example, in C, setting dims[0] to 5 and dims[1] to 7 establishes a 5-by-7 mxArray. In Fortran, setting dims(1) to 5 and dims(2) to 7 establishes a 5-by-7 mxArray. In most cases, there should be ndim elements in the dims array.</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Use mxCreateCellArray to create a cell mxArray with size defined by ndim and dims. For example, in C, to establish a three-dimensional cell mxArray having dimensions 4-by-8-by-7, set:</p> <pre>ndim = 3; dims[0] = 4; dims[1] = 8; dims[2] = 7;</pre> <p>In Fortran, to establish a three-dimensional cell mxArray having dimensions 4-by-8-by-7, set:</p>

mxCreateCellArray (C and Fortran)

```
ndim = 3;  
dims(1) = 4; dims(2) = 8; dims(3) = 7;
```

The created cell mxArray is unpopulated; `mxCreateCellArray` initializes each cell to `NULL`. To put data into a cell, call `mxSetCell`.

MATLAB automatically removes any trailing singleton dimensions specified in the `dims` argument. For example, if `ndim` equals 5 and `dims` equals `[4 1 7 1 1]`, the resulting array has the dimensions 4-by-1-by-7.

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `phonebook.c`

See Also

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

mxCreateCellMatrix (C and Fortran)

Purpose	2-D cell array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateCellMatrix(mwSize m, mwSize n);</pre>
Fortran Syntax	<pre>mwPointer mxCreateCellMatrix(m, n) mwSize m, n</pre>
Arguments	<p>m Number of rows</p> <p>n Number of columns</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Use <code>mxCreateCellMatrix</code> to create an m-by-n two-dimensional cell mxArray. The created cell mxArray is unpopulated; <code>mxCreateCellMatrix</code> initializes each cell to NULL in C (0 in Fortran). To put data into cells, call <code>mxSetCell</code>.</p> <p><code>mxCreateCellMatrix</code> is identical to <code>mxCreateCellArray</code> except that <code>mxCreateCellMatrix</code> can create two-dimensional mxArrays only, but <code>mxCreateCellArray</code> can create mxArrays having any number of dimensions greater than 1.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxcreatecellmatrix.c</code>• <code>mxcreatecellmatrixf.F</code>
See Also	<code>mxCreateCellArray</code>

mxCreateCharArray (C and Fortran)

Purpose	N-D string array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateCharArray(mwSize ndim, const mwSize *dims);</pre>
Fortran Syntax	<pre>mwPointer mxCreateCharArray(ndim, dims) mwSize ndim mwSize dims(ndim)</pre>
Arguments	<p>ndim Number of dimensions in the string mxArray. You must specify a positive number. If you specify 0, 1, or 2, mxCreateCharArray creates a two-dimensional mxArray.</p> <p>dims Dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, in C, setting <code>dims[0]</code> to 5 and <code>dims[1]</code> to 7 establishes a 5-by-7 mxArray. In Fortran, setting <code>dims(1)</code> to 5 and <code>dims(2)</code> to 7 establishes a 5-by-7 character mxArray. The <code>dims</code> array must have at least <code>ndim</code> elements.</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Call mxCreateCharArray to create an N-dimensional string mxArray. The created mxArray is unpopulated; that is, mxCreateCharArray initializes each cell to NULL in C (0 in Fortran).</p> <p>MATLAB automatically removes any trailing singleton dimensions specified in the <code>dims</code> argument. For example, if <code>ndim</code> equals 5 and <code>dims</code> equals <code>[4 1 7 1 1]</code>, the resulting array has the dimensions 4-by-1-by-7.</p>

mxCreateCharArray (C and Fortran)

See Also `mxCreateCharMatrixFromStrings`, `mxCreateString`

mxCreateCharMatrixFromStrings (C and Fortran)

Purpose 2-D string array initialized to specified value

C Syntax

```
#include "matrix.h"
mxArray *mxCreateCharMatrixFromStrings(mwSize m, const char **str);
```

Fortran Syntax

```
mwPointer mxCreateCharMatrixFromStrings(m, str)
mwSize m
character*(*) str(m)
```

Arguments

m Number of rows in the created string mxArray. The value you specify for **m** is the number of strings in **str**.

str In C, an array of strings containing at least **m** strings. In Fortran, a `character*n` array of size **m**, where each element of the array is **n** bytes.

Returns Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray. Another possible reason for failure is that **str** contains fewer than **m** strings.

Description Use `mxCreateCharMatrixFromStrings` to create a two-dimensional string mxArray, where each row is initialized to a string from **str**. In C, the created mxArray has dimensions **m-by-max**, where **max** is the length of the longest string in **str**. In Fortran, the created mxArray has dimensions **m-by-n**, where **n** is the number of characters in `str(i)`. String mxArrays represent their data elements as `mxChar` rather than as C `char`.

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxcreatecharmatrixfromstr.c`

mxCreateCharMatrixFromStrings (C and Fortran)

See Also `mxCreateCharArray`, `mxCreateString`, `mxGetString`

mxCreateDoubleMatrix (C and Fortran)

Purpose	2-D, double-precision, floating-point array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateDoubleMatrix(mwSize m, mwSize n, mxComplexity ComplexFlag);</pre>
Fortran Syntax	<pre>mwPointer mxCreateDoubleMatrix(m, n, ComplexFlag) mwSize m, n integer*4 ComplexFlag</pre>
Arguments	<p>m Number of rows</p> <p>n Number of columns</p> <p>ComplexFlag If the mxArray you are creating is to contain imaginary data, set ComplexFlag to mxCOMPLEX in C (1 in Fortran). Otherwise, set ComplexFlag to mxREAL in C (0 in Fortran).</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Use mxCreateDoubleMatrix to create an m-by-n mxArray. mxCreateDoubleMatrix initializes each element in the pr array to 0. If you set ComplexFlag to mxCOMPLEX in C (1 in Fortran), mxCreateDoubleMatrix also initializes each element in the pi array to 0.</p> <p>If you set ComplexFlag to mxREAL in C (0 in Fortran), mxCreateDoubleMatrix allocates enough memory to hold m-by-n real elements. If you set ComplexFlag to mxCOMPLEX in C (1 in Fortran), mxCreateDoubleMatrix allocates enough memory to hold m-by-n real elements and m-by-n imaginary elements.</p>

mxCreateDoubleMatrix (C and Fortran)

Call `mxDestroyArray` when you finish using the `mxArray`. `mxDestroyArray` deallocates the `mxArray` and its associated real and complex elements.

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `convec.c`
- `findnz.c`
- `matrixDivide.c`
- `sincall.c`
- `timestwo.c`
- `timestwoalt.c`
- `xtimesy.c`

For Fortran examples, see:

- `convec.F`
- `dblmat.F`
- `matsq.F`
- `timestwo.F`
- `xtimesy.F`

See Also

`mxCreateNumericArray`

mxCreateDoubleScalar (C and Fortran)

Purpose	Scalar, double-precision array initialized to specified value
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateDoubleScalar(double value);</pre>
Fortran Syntax	<pre>mwPointer mxCreateDoubleScalar(value) real*8 value</pre>
Arguments	<p>value</p> <p>Value to which you want to initialize the array</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	Call <code>mxCreateDoubleScalar</code> to create a scalar double mxArray. When you finish using the mxArray, call <code>mxDestroyArray</code> to destroy it.
Alternatives	<p>C Language</p> <p>In C, you can replace the statements:</p> <pre>pa = mxCreateDoubleMatrix(1, 1, mxREAL); *mxGetPr(pa) = value;</pre> <p>with a call to <code>mxCreateDoubleScalar</code>:</p> <pre>pa = mxCreateDoubleScalar(value);</pre> <p>Fortran Language</p> <p>In Fortran, you can replace the statements:</p> <pre>pm = mxCreateDoubleMatrix(1, 1, 0) mxCopyReal8ToPtr(value, mxGetPr(pm), 1)</pre>

mxCreateDoubleScalar (C and Fortran)

with a call to mxCreateDoubleScalar:

```
pm = mxCreateDoubleScalar(value)
```

See Also

mxGetPr, mxCreateDoubleMatrix

mxCreateLogicalArray (C)

Purpose	N-D logical array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateLogicalArray(mwSize ndim, const mwSize *dims);</pre>
Arguments	<p>ndim Number of dimensions. If you specify a value for <code>ndim</code> that is less than 2, <code>mxCreateLogicalArray</code> automatically sets the number of dimensions to 2.</p> <p>dims 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>. There are <code>ndim</code> elements in the <code>dims</code> array.</p>
Returns	Pointer to the created <code>mxArray</code> , if successful. If unsuccessful in a standalone (non-MEX-file) application, returns <code>NULL</code> in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the <code>mxArray</code> .
Description	<p>Call <code>mxCreateLogicalArray</code> to create an N-dimensional <code>mxArray</code> of <code>mxLogical</code> elements. After creating the <code>mxArray</code>, <code>mxCreateLogicalArray</code> initializes all its elements to logical 0. <code>mxCreateLogicalArray</code> differs from <code>mxCreateLogicalMatrix</code> in that the latter can create two-dimensional arrays only.</p> <p><code>mxCreateLogicalArray</code> allocates dynamic memory to store the created <code>mxArray</code>. When you finish with the created <code>mxArray</code>, call <code>mxDestroyArray</code> to deallocate its memory.</p> <p>MATLAB automatically removes any trailing singleton dimensions specified in the <code>dims</code> argument. For example, if <code>ndim</code> equals 5 and <code>dims</code> equals [4 1 7 1 1], the resulting array has the dimensions 4-by-1-by-7.</p>

See Also

`mxCreateLogicalMatrix`, `mxCreateSparseLogicalMatrix`,
`mxCreateLogicalScalar`

mxCreateLogicalMatrix (C)

Purpose	2-D logical array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateLogicalMatrix(mwSize m, mwSize n);</pre>
Arguments	<p>m Number of rows</p> <p>n Number of columns</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Use <code>mxCreateLogicalMatrix</code> to create an m-by-n mxArray of <code>mxLogical</code> elements. <code>mxCreateLogicalMatrix</code> initializes each element in the array to logical 0.</p> <p>Call <code>mxDestroyArray</code> when you finish using the mxArray. <code>mxDestroyArray</code> deallocates the mxArray.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxislogical.c</code>
See Also	<code>mxCreateLogicalArray</code> , <code>mxCreateSparseLogicalMatrix</code> , <code>mxCreateLogicalScalar</code>

Purpose	Scalar, logical array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateLogicalScalar(mxLogical value);</pre>
Arguments	value Logical value to which you want to initialize the array
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p>Call <code>mxCreateLogicalScalar</code> to create a scalar logical mxArray. <code>mxCreateLogicalScalar</code> is a convenience function that replaces the following code:</p> <pre>pa = mxCreateLogicalMatrix(1, 1); *mxGetLogicals(pa) = value;</pre> <p>When you finish using the mxArray, call <code>mxDestroyArray</code> to destroy it.</p>
See Also	<code>mxCreateLogicalArray</code> , <code>mxCreateLogicalMatrix</code> , <code>mxIsLogicalScalar</code> , <code>mxIsLogicalScalarTrue</code> , <code>mxGetLogicals</code> , <code>mxDestroyArray</code>

mxCreateNumericArray (C and Fortran)

Purpose

N-D numeric array

C Syntax

```
#include "matrix.h"
mxArray *mxCreateNumericArray(mwSize ndim, const mwSize *dims,
                              mxClassID classid, mxComplexity ComplexFlag);
```

Fortran Syntax

```
mwPointer mxCreateNumericArray(ndim, dims, classid,
                               ComplexFlag)
mwSize ndim
mwSize dims(ndim)
integer*4 classid, ComplexFlag
```

Arguments

ndim

Number of dimensions. If you specify a value for `ndim` that is less than 2, `mxCreateNumericArray` automatically sets the number of dimensions to 2.

dims

Dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, in C, setting `dims[0]` to 5 and `dims[1]` to 7 establishes a 5-by-7 `mxArray`. In Fortran, setting `dims(1)` to 5 and `dims(2)` to 7 establishes a 5-by-7 `mxArray`. In most cases, there are `ndim` elements in the `dims` array.

classid

Identifier for the class of the array, which determines the way the numerical data is represented in memory. For example, specifying `mxINT16_CLASS` in C causes each piece of numerical data in the `mxArray` to be represented as a 16-bit signed integer. In Fortran, use the function `mxClassIDFromClassname` to derive the `classid` value from a MATLAB class name. See the Description section for more information.

ComplexFlag

If the `mxArray` you are creating is to contain imaginary data, set `ComplexFlag` to `mxCOMPLEX` in C (1 in Fortran). Otherwise, set `ComplexFlag` to `mxREAL` in C (0 in Fortran).

mxCreateNumericArray (C and Fortran)

Returns

Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.

Description

Call `mxCreateNumericArray` to create an N-dimensional mxArray in which all data elements have the numeric data type specified by `classid`. After creating the mxArray, `mxCreateNumericArray` initializes all its real data elements to 0. If `ComplexFlag` equals `mxCOMPLEX` in C (1 in Fortran), `mxCreateNumericArray` also initializes all its imaginary data elements to 0. `mxCreateNumericArray` differs from `mxCreateDoubleMatrix` as follows:

- All data elements in `mxCreateDoubleMatrix` are double-precision, floating-point numbers. The data elements in `mxCreateNumericArray` can 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.

MATLAB automatically removes any trailing singleton dimensions specified in the `dims` argument. For example, if `ndim` equals 5 and `dims` equals [4 1 7 1 1], the resulting array has the dimensions 4-by-1-by-7.

The following table shows the C `classid` values and the Fortran data types that are equivalent to MATLAB classes.

MATLAB Class Name	C classid Value	Fortran Type
int8	mxINT8_CLASS	BYTE
uint8	mxUINT8_CLASS	

mxCreateNumericArray (C and Fortran)

MATLAB Class Name	C classid Value	Fortran Type
int16	mxINT16_CLASS	INTEGER*2
uint16	mxUINT16_CLASS	
int32	mxINT32_CLASS	INTEGER*4
uint32	mxUINT32_CLASS	
int64	mxINT64_CLASS	INTEGER*8
uint64	mxUINT64_CLASS	
single	mxSINGLE_CLASS	REAL*4
double	mxDOUBLE_CLASS	REAL*8
single, with imaginary components	mxSINGLE_CLASS	COMPLEX*8
double, with imaginary components	mxDOUBLE_CLASS	COMPLEX*16

Examples

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`
- `doubleelement.c`
- `matrixDivide.c`
- `matsqint8.F`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxisfinite.c`

See Also

`mxClassId`, `mxClassIdFromClassName`, `mxComplexity`,
`mxCreateNumericMatrix`

mxCreateNumericMatrix (C and Fortran)

Purpose	2-D numeric matrix
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateNumericMatrix(mwSize m, mwSize n, mxClassID classid, mxComplexity ComplexFlag);</pre>
Fortran Syntax	<pre>mwPointer mxCreateNumericMatrix(m, n, classid, ComplexFlag) mwSize m, n integer*4 classid, ComplexFlag</pre>
Arguments	<p>m Number of rows</p> <p>n Number of columns</p> <p>classid Identifier for the class of the array, which determines the way the numerical data is represented in memory. For example, specifying <code>mxINT16_CLASS</code> in C causes each piece of numerical data in the <code>mxArray</code> to be represented as a 16-bit signed integer. In Fortran, use the function <code>mxClassIDFromClassname</code> to derive the <code>classid</code> value from a MATLAB class name.</p> <p>ComplexFlag If the <code>mxArray</code> you are creating is to contain imaginary data, set <code>ComplexFlag</code> to <code>mxCOMPLEX</code> in C (1 in Fortran). Otherwise, set <code>ComplexFlag</code> to <code>mxREAL</code> in C (0 in Fortran).</p>
Returns	Pointer to the created <code>mxArray</code> , if successful. If unsuccessful in a standalone (non-MEX-file) application, returns <code>NULL</code> in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the <code>mxArray</code> .
Description	Call <code>mxCreateNumericMatrix</code> to create a 2-D <code>mxArray</code> in which all data elements have the numeric data type specified by <code>classid</code> . After

mxCreateNumericMatrix (C and Fortran)

creating the `mxArray`, `mxCreateNumericMatrix` initializes all its real data elements to 0. If `ComplexFlag` equals `mxCOMPLEX` in C (1 in Fortran), `mxCreateNumericMatrix` also initializes all its imaginary data elements to 0. `mxCreateNumericMatrix` allocates dynamic memory to store the created `mxArray`. When you finish using the `mxArray`, call `mxDestroyArray` to destroy it.

The following table shows the C `classid` values and the Fortran data types that are equivalent to MATLAB classes.

MATLAB Class Name	C classid Value	Fortran Type
<code>int8</code>	<code>mxINT8_CLASS</code>	BYTE
<code>uint8</code>	<code>mxUINT8_CLASS</code>	
<code>int16</code>	<code>mxINT16_CLASS</code>	INTEGER*2
<code>uint16</code>	<code>mxUINT16_CLASS</code>	
<code>int32</code>	<code>mxINT32_CLASS</code>	INTEGER*4
<code>uint32</code>	<code>mxUINT32_CLASS</code>	
<code>int64</code>	<code>mxINT64_CLASS</code>	INTEGER*8
<code>uint64</code>	<code>mxUINT64_CLASS</code>	
<code>single</code>	<code>mxSINGLE_CLASS</code>	REAL*4
<code>double</code>	<code>mxDOUBLE_CLASS</code>	REAL*8
<code>single, with imaginary components</code>	<code>mxSINGLE_CLASS</code>	COMPLEX*8
<code>double, with imaginary components</code>	<code>mxDOUBLE_CLASS</code>	COMPLEX*16

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillGetPr.c`

mxCreateNumericMatrix (C and Fortran)

The following Fortran statements create a 4-by-3 matrix of REAL*4 elements having no imaginary components:

```
C      Create 4x3 mxArray of REAL*4
      mxCreateNumericMatrix(4, 3,
+                          mxClassIDFromClassName('single'), 0)
```

See Also

`mxClassId`, `mxClassIdFromClassName`, `mxComplexity`,
`mxCreateNumericArray`

mxCreateSparse (C and Fortran)

Purpose 2-D sparse array

C Syntax

```
#include "matrix.h"
mxArray *mxCreateSparse(mwSize m, mwSize n, mwSize nzmax,
                        mxComplexity ComplexFlag);
```

Fortran Syntax

```
mwPointer mxCreateSparse(m, n, nzmax, ComplexFlag)
mwSize m, n, nzmax
integer*4 ComplexFlag
```

Arguments

m
Number of rows

n
Number of columns

nzmax
Number of elements that `mxCreateSparse` should allocate to hold the `pr`, `ir`, and, if `ComplexFlag` is `mxCOMPLEX` in C (1 in Fortran), `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
If the `mxArray` you are creating is to contain imaginary data, set `ComplexFlag` to `mxCOMPLEX` in C (1 in Fortran). Otherwise, set `ComplexFlag` to `mxREAL` in C (0 in Fortran).

Returns Pointer to the created `mxArray`, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns `NULL` in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the `mxArray`. In that case, try reducing `nzmax`, `m`, or `n`.

Description Call `mxCreateSparse` to create an unpopulated sparse double `mxArray`. The returned sparse `mxArray` contains no sparse information and cannot be passed as an argument to any MATLAB sparse functions. To

make the returned sparse mxArray useful, you must initialize the `pr`, `ir`, `jc`, and (if it exists) `pi` arrays.

`mxCreateSparse` allocates space for:

- A `pr` array of length `nzmax`.
- A `pi` array of length `nzmax`, but only if `ComplexFlag` is `mxCOMPLEX` in C (1 in Fortran).
- An `ir` array of length `nzmax`.
- A `jc` array of length `n+1`.

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

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `fulltosparse.c`
- `fulltosparse.F`

See Also

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

mxCreateSparseLogicalMatrix (C)

Purpose 2-D, sparse, logical array

C Syntax

```
#include "matrix.h"
mxArray *mxCreateSparseLogicalMatrix(mwSize m, mwSize n,
    mwSize nzmax);
```

Arguments

m
Number of rows

n
Number of columns

nzmax
Number of elements that mxCreateSparseLogicalMatrix should allocate to hold the data. 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 $m*n$.

Returns Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.

Description Use mxCreateSparseLogicalMatrix to create an m-by-n mxArray of mxLogical elements. mxCreateSparseLogicalMatrix initializes each element in the array to logical 0.

Call mxDestroyArray when you finish using the mxArray. mxDestroyArray deallocates the mxArray and its elements.

See Also mxCreateLogicalArray, mxCreateLogicalMatrix, mxCreateLogicalScalar, mxCreateSparse, mxIsLogical

Purpose	1-N array initialized to specified string
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateString(const char *str);</pre>
Fortran Syntax	<pre>mwPointer mxCreateString(str) character*(*) str</pre>
Arguments	<p><code>str</code></p> <p>String that is to serve as the <code>mxArray</code>'s initial data</p>
Returns	Pointer to the created <code>mxArray</code> , if successful. If unsuccessful in a standalone (non-MEX-file) application, returns <code>NULL</code> in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the <code>mxArray</code> .
Description	<p>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.</p> <p>Free the string <code>mxArray</code> when you are finished using it. To free a string <code>mxArray</code>, call <code>mxDestroyArray</code>.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>revord.c</code>• <code>revord.F</code> <p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxcreatestructarray.c</code>• <code>mxisclass.c</code> <p>See the following examples in <code>matlabroot/extern/examples/eng_mat</code>.</p>

mxCreateString (C and Fortran)

- `matdemo1.F`

See Also

`mxCreateCharMatrixFromStrings`, `mxCreateCharArray`

mxCreateStructArray (C and Fortran)

Purpose

N-D structure array

C Syntax

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

Fortran Syntax

```
mwPointer mxCreateStructArray(ndim, dims, nfields,
    fieldnames)
mwSize ndim
mwSize dims(ndim)
integer*4 nfields
character*(*) fieldnames(nfields)
```

Arguments

`ndim`

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

`dims`

Dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, in C, setting `dims[0]` to 5 and `dims[1]` to 7 establishes a 5-by-7 `mxArray`. In Fortran, setting `dims(1)` to 5 and `dims(2)` to 7 establishes a 5-by-7 `mxArray`. Typically, the `dims` array should have `ndim` elements.

`nfields`

Number of fields in each element

`fieldnames`

List of field names

Each structure field name must begin with a letter and is case sensitive. The rest of the name may contain letters, numerals, and underscore characters. Use the `namelengthmax` function to determine the maximum length of a field name.

mxCreateStructArray (C and Fortran)

Returns

Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the 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 `fieldnames`. A MATLAB structure mxArray is conceptually identical to an array of structs in the C language.

Each field holds one mxArray pointer. `mxCreateStructArray` initializes each field to NULL in C (0 in Fortran). 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.

Any trailing singleton dimensions specified in the `dims` argument are automatically removed from the resulting array. For example, if `ndim` equals 5 and `dims` equals [4 1 7 1 1], the resulting array is given the dimensions 4-by-1-by-7.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcreatestructarray.c`

See Also

`mxDestroyArray`, `mxAddField`, `mxRemoveField`, `mxSetField`, `mxSetFieldByNumber`

mxCreateStructMatrix (C and Fortran)

Purpose	2-D structure array
C Syntax	<pre>#include "matrix.h" mxArray *mxCreateStructMatrix(mwSize m, mwSize n, int nfields, const char **fieldnames);</pre>
Fortran Syntax	<pre>mwPointer mxCreateStructMatrix(m, n, nfields, fieldnames) mwSize m, n integer*4 nfields character*(*) fieldnames(nfields)</pre>
Arguments	<p>m Number of rows. This must be a positive integer.</p> <p>n Number of columns. This must be a positive integer.</p> <p>nfields Number of fields in each element.</p> <p>fieldnames List of field names.</p> <p>Each structure field name must begin with a letter and is case sensitive. The rest of the name may contain letters, numerals, and underscore characters. Use the <code>namelengthmax</code> function to determine the maximum length of a field name.</p>
Returns	Pointer to the created <code>mxArray</code> , if successful. If unsuccessful in a standalone (non-MEX-file) application, returns <code>NULL</code> in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the <code>mxArray</code> .
Description	<code>mxCreateStructMatrix</code> and <code>mxCreateStructArray</code> are almost identical. The only difference is that <code>mxCreateStructMatrix</code> can create only two-dimensional <code>mxArrays</code> , while <code>mxCreateStructArray</code> can create <code>mxArrays</code> having two or more dimensions.

mxCreateStructMatrix (C and Fortran)

C Examples

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`

See Also

`mxCreateStructArray`

mxDestroyArray (C and Fortran)

Purpose	Free dynamic memory allocated by MXCREATE* functions
C Syntax	<pre>#include "matrix.h" void mxDestroyArray(mxArray *pm);</pre>
Fortran Syntax	<pre>subroutine mxDestroyArray(pm) mwPointer pm</pre>
Arguments	pm Pointer to the mxArray 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 and pi for complex arrays, ir and jc for sparse arrays, fields of structure arrays, and cells of cell arrays. Do not call mxDestroyArray on an mxArray you are returning in a left side argument.
Examples	See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• matrixDivide.c• matrixDivideComplex.c• sincall.c• sincall.F See the following examples in <i>matlabroot/extern/examples/mex</i> . <ul style="list-style-type: none">• mexcallmatlab.c• mexgetarray.c See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• mxisclass.c

mxDestroyArray (C and Fortran)

- `mxcreatecellmatrixf.F`

See Also

`mxCalloc`, `mxMalloc`, `mxFree`, `mexMakeArrayPersistent`,
`mexMakeMemoryPersistent`

mxDuplicateArray (C and Fortran)

Purpose	Make deep copy of array
C Syntax	<pre>#include "matrix.h" mxArray *mxDuplicateArray(const mxArray *in);</pre>
Fortran Syntax	<pre>mwPointer mxDuplicateArray(in) mwPointer in</pre>
Arguments	<p><i>in</i></p> <p>Pointer to the mxArray you want to copy</p>
Returns	Pointer to the created mxArray, if successful. If unsuccessful in a standalone (non-MEX-file) application, returns NULL in C (0 in Fortran). If unsuccessful in a MEX-file, the MEX-file terminates and returns control to the MATLAB prompt. The function is unsuccessful when there is not enough free heap space to create the mxArray.
Description	<p><code>mxDuplicateArray</code> 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 each cell (if any), and so on.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/mex</i>.</p> <ul style="list-style-type: none">• <code>mexget.c</code> <p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• <code>phonebook.c</code> <p>See the following examples in <i>matlabroot/extern/examples/mx</i>.</p> <ul style="list-style-type: none">• <code>mxcreatecellmatrix.c</code>• <code>mxcreatecellmatrixf.F</code>• <code>mxgetinf.c</code>

mxDuplicateArray (C and Fortran)

- `mxsetdimensions.c`
- `mxsetdimensionsf.F`
- `mxsetnzmax.c`

Purpose	Free dynamic memory allocated by <code>MXCALLOC</code> , <code>MXMALLOC</code> , or <code>MXREALLOC</code> functions
C Syntax	<pre>#include "matrix.h" void mxFree(void *ptr);</pre>
Fortran Syntax	<pre>subroutine mxFree(ptr) mwPointer ptr</pre>
Arguments	<p><code>ptr</code></p> <p>Pointer to the beginning of any memory parcel allocated by <code>mxMalloc</code>, <code>mxMalloc</code>, or <code>mxRealloc</code>.</p>
Description	<p><code>mxFree</code> deallocates heap space using the MATLAB memory management facility. This ensures correct memory management in error and abort (Ctrl+C) conditions.</p> <p>To deallocate heap space, MATLAB applications in C should always call <code>mxFree</code> rather than the ANSI C <code>free</code> function.</p> <p>In MEX-files, but excluding MAT or engine standalone applications, the MATLAB memory management facility maintains a list of all memory allocated by <code>mxMalloc</code>, <code>mxMalloc</code>, and <code>mxRealloc</code>. The memory management facility automatically deallocates all of a MEX-file's managed parcels when the MEX-file completes and control returns to the MATLAB prompt. <code>mxFree</code> also removes the memory parcel from the memory management facility's list of memory parcels.</p> <p>When <code>mxFree</code> appears in a MAT or engine standalone MATLAB application, it simply deallocates the contiguous heap space that begins at address <code>ptr</code>.</p> <p>In MEX-files, your use of <code>mxFree</code> depends on whether the specified memory parcel is persistent or nonpersistent. By default, memory parcels created by <code>mxMalloc</code>, <code>mxMalloc</code>, and <code>mxRealloc</code> are nonpersistent. The 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</p>

mxFree (C and Fortran)

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

If an application calls `mexMakeMemoryPersistent`, the specified memory parcel becomes persistent. When a MEX-file completes, the 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 cleanup handler. The cleanup handler calls `mxFree`.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcalcsinglesubscript.c`
- `mxcreatecharmatrixfromstr.c`
- `mxisfinite.c`
- `mxmalloc.c`
- `mxsetdimensions.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillGetPrDynamicData.c`
- `phonebook.c`

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See Also

`mexAtExit`, `mexMakeArrayPersistent`, `mexMakeMemoryPersistent`, `mxMalloc`, `mxRealloc`, `mxCallocc`, `mxDestroyArray`, `mxMalloc`, `mxRealloc`

Purpose	Get contents of cell array
C Syntax	<pre>#include "matrix.h" mxArray *mxGetCell(const mxArray *pm, mwIndex index);</pre>
Fortran Syntax	<pre>mwPointer mxGetCell(pm, index) mwPointer pm mwIndex index</pre>
Arguments	<p><code>pm</code> Pointer to a cell mxArray</p> <p><code>index</code> Number of elements in the cell mxArray between the first element and the desired one. See <code>mxCalcSingleSubscript</code> for details on calculating an index in a multidimensional cell array.</p>
Returns	<p>Pointer to the <code>i</code>th cell mxArray if successful. Otherwise, returns NULL in C (0 in Fortran). Causes of failure include:</p> <ul style="list-style-type: none">• Specifying the index of a cell array element that has not been populated.• Specifying a <code>pm</code> that does not point to a cell mxArray.• Specifying an <code>index</code> greater than the number of elements in the cell.• Insufficient heap space. <p><code>mxGetCell</code> allocates an empty, double mxArray if the pointer in the cell is NULL.</p>
Description	Call <code>mxGetCell</code> to get a pointer to the mxArray held in the indexed element of the cell mxArray.

mxGetCell (C and Fortran)

Note Inputs to a MEX-file are constant read-only mxArray's. Do not modify the inputs. Using `mxSetCell*` or `mxSetField*` functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

Examples

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See Also

`mxCreateCellArray`, `mxIsCell`, `mxSetCell`

Purpose	Pointer to character array data
C Syntax	<pre>#include "matrix.h" mxChar *mxGetChars(const mxArray *array_ptr);</pre>
Arguments	<pre>array_ptr Pointer to an mxArray</pre>
Returns	Pointer to the first character in the mxArray. Returns NULL if the specified array is not a character array.
Description	Call mxGetChars to access the first character in the mxArray that array_ptr points to. Once you have the starting address, you can access any other element in the mxArray.
See Also	<pre>mxGetString</pre>

mxGetClassID (C and Fortran)

Purpose	Class of array
C Syntax	<pre>#include "matrix.h" mxClassID mxGetClassID(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxGetClassID(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Numeric identifier of the class (category) of the mxArray that pm points to. For a list of C-language class identifiers, see the mxClassID reference page.
Description	<p>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 pm points to a logical mxArray, then mxGetClassId returns mxLOGICAL_CLASS (in C).</p> <p>mxGetClassId is like mxGetClassName, except that the former returns the class as an integer identifier and the latter returns the class as a string.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/mex</i>.</p> <ul style="list-style-type: none">• explore.c <p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• phonebook.c
See Also	mxClassID, mxGetClassName

mxGetClassName (C and Fortran)

Purpose Class of array as string

C Syntax

```
#include "matrix.h"
const char *mxGetClassName(const mxArray *pm);
```

Fortran Syntax

```
character*(*) mxGetClassName(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Class (as a string) of the mxArray pointed to by pm.

Description Call `mxGetClassName` to determine the class of an mxArray. The class of an mxArray identifies the kind of data the mxArray is holding. For example, if pm points to a logical mxArray, `mxGetClassName` returns `logical`.

`mxGetClassID` is like `mxGetClassName`, except that the former returns the class as an integer identifier, as listed in the `mxClassID` reference page, and the latter returns the class as a string, as listed in the `mxIsClass` reference page.

Examples See the following examples in `matlabroot/extern/examples/mex`.

- `mexfunction.c`

See the following examples in `matlabroot/extern/examples/mx`.

- `mxisclass.c`

See Also `mxGetClassID`, `mxIsClass`

mxGetData (C and Fortran)

Purpose	Pointer to real numeric data elements in array
C Syntax	<pre>#include "matrix.h" void *mxGetData(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetData(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Pointer to the first element of the real data. Returns NULL in C (0 in Fortran) if there is no real data.
Description	<p>In C, mxGetData returns a void pointer (void *). Since void pointers point to a value that has no type, you must cast the return value to the pointer type that matches the type specified by pm. To see how MATLAB types map to their equivalent C types, see the table on the mxClassID reference page.</p> <p>In Fortran, to copy values from the returned pointer, use one of the mxCopyPtrTo* functions in the following manner:</p> <pre>C Get the data in mxArray, pm mxCopyPtrToReal8(mxGetData(pm), data, + mxGetNumberOfElements(pm))</pre>

Examples See the following examples in *matlabroot/extern/examples/mex*.

- explore.c

See the following examples in *matlabroot/extern/examples/refbook*.

- matrixDivide.c
- matrixDivideComplex.c
- phonebook.c

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcreatecharmatrixfromstr.c`
- `mxisfinite.c`

See Also

`mxGetImagData`, `mxGetPr`, `mxClassID`

mxGetDimensions (C and Fortran)

Purpose Pointer to dimensions array

C Syntax

```
#include "matrix.h"
const mwSize *mxGetDimensions(const mxArray *pm);
```

Fortran Syntax

```
mwPointer mxGetDimensions(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray.

Returns Pointer to the first element in the dimensions 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 pm points to. Call `mxGetNumberOfDimensions` to get the number of dimensions in the mxArray.

To copy the values to Fortran, use `mxCopyPtrToInteger4` in the following manner:

```
C      Get dimensions of mxArray, pm
      mxCopyPtrToInteger4(mxGetDimensions(pm), dims,
+                          mxGetNumberOfDimensions(pm))
```

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxcalcsinglesubscript.c`
- `mxgeteps.c`
- `mxisfinite.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `findnz.c`

mxGetDimensions (C and Fortran)

- `phonebook.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

`mxGetNumberOfDimensions`

mxGetElementSize (C and Fortran)

Purpose Number of bytes required to store each data element

C Syntax

```
#include "matrix.h"
size_t mxGetElementSize(const mxArray *pm);
```

Fortran Syntax

```
mwPointer mxGetElementSize(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Number of bytes required to store one element of the specified mxArray, if successful. Returns 0 on failure. The primary reason for failure is that pm points to an mxArray having an unrecognized class. If pm points to a cell mxArray or a structure mxArray, mxGetElementSize returns the size of a pointer (not the size of all the elements in each cell or structure field).

Description Call mxGetElementSize to determine the number of bytes in each data element of the mxArray. For example, if the MATLAB class of an mxArray is int16, the mxArray stores each data element as a 16-bit (2-byte) signed integer. Thus, mxGetElementSize returns 2.

mxGetElementSize is helpful when using a non-MATLAB routine to manipulate data elements. For example, the C function memcpy requires (for its third argument) the size of the elements you intend to copy.

Note Fortran does not have an equivalent of size_t. mwPointer is a preprocessor macro that provides the appropriate Fortran type. The value returned by this function, however, is not a pointer.

Examples See the following examples in *matlabroot/extern/examples/refbook*.

- doubleelement.c

mxGetElementSize (C and Fortran)

- `phonebook.c`

See Also

`mxGetM`, `mxGetN`

mxGetEps (C and Fortran)

Purpose	Value of EPS
C Syntax	<pre>#include "matrix.h" double mxGetEps(void);</pre>
Fortran Syntax	<pre>real*8 mxGetEps</pre>
Returns	Value of the MATLAB eps variable
Description	Call mxGetEps to return the value of the MATLAB 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. The MATLAB pinv and rank functions use eps as a default tolerance.
Examples	See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• mxgeteps.c• mxgetepsf.F
See Also	mxGetInf, mxGetNan

Purpose Get field value from structure array, given index and field name

C Syntax

```
#include "matrix.h"
mxArray *mxGetField(const mxArray *pm, mwIndex index,
                    const char *fieldname);
```

Fortran Syntax

```
mwPointer mxGetField(pm, index, fieldname)
mwPointer pm
mwIndex index
character*(*) fieldname
```

Arguments

`pm`
Pointer to a structure `mxArray`

`index`
Index of the desired element.

In C, the first element of an `mxArray` has an index of 0. The index of the last element is `N - 1`, where `N` is the number of elements in the array. In Fortran, the first element of an `mxArray` has an index of 1. The index of the last element is `N`, where `N` is the number of elements in the array.

`fieldname`
Name of the field whose value you want to extract.

Returns Pointer to the `mxArray` in the specified field at the specified `fieldname`, on success. Returns `NULL` in C (`0` in Fortran) if passed an invalid argument or if there is no value assigned to the specified field. Common causes of failure include:

- Specifying an array pointer `pm` that does not point to a structure `mxArray`. To determine whether `pm` points to a structure `mxArray`, call `mxIsStruct`.
- Specifying an `index` to an element outside the bounds of the `mxArray`. For example, given a structure `mxArray` that contains ten elements, you cannot specify an `index` greater than `9` in C (`10` in Fortran).

mxGetField (C and Fortran)

- Specifying a nonexistent `fieldname`. Call `mxGetFieldNameByNumber` or `mxGetFieldNumber` to get existing field names.
- Insufficient heap space.

`mxGetField` allocates an empty, double `mxArray` if the pointer in the field is `NULL`.

Description

Call `mxGetField` to get the value held in the specified element of the specified field. In pseudo-C terminology, `mxGetField` returns the value at:

```
pm[index].fieldname
```

`mxGetFieldByNumber` is like `mxGetField`. Both functions return the same value. The only difference is in the way you specify the field. `mxGetFieldByNumber` takes a field number as its third argument, and `mxGetField` takes a field name as its third argument.

Note Inputs to a MEX-file are constant read-only `mxArrays`. Do not modify the inputs. Using `mxSetCell*` or `mxSetField*` functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

In C, calling:

```
mxGetField(pa, index, "field_name");
```

is equivalent to calling:

```
field_num = mxGetFieldNumber(pa, "field_name");  
mxGetFieldByNumber(pa, index, field_num);
```

where `index` is 0 if you have a 1-by-1 structure.

In Fortran, calling:

```
mxGetField(pm, index, 'fieldname')
```

is equivalent to calling:

```
fieldnum = mxGetFieldNumber(pm, 'fieldname')  
mxGetFieldByNumber(pm, index, fieldnum)
```

where `index` is 1 if you have a 1-by-1 structure.

See Also

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

mxGetFieldByNumber (C and Fortran)

Purpose Get field value from structure array, given index and field number

C Syntax

```
#include "matrix.h"
mxArray *mxGetFieldByNumber(const mxArray *pm, mwIndex index,
                             int fieldnumber);
```

Fortran Syntax

```
mwPointer mxGetFieldByNumber(pm, index, fieldnumber)
mwPointer pm
mwIndex index
integer*4 fieldnumber
```

Arguments

pm
Pointer to a structure mxArray

index
Index of the desired element.

In C, the first element of an mxArray has an index of 0. The index of the last element is N-1, where N is the number of elements in the array. In Fortran, the first element of an mxArray has an index of 1. The index of the last element is N, where N is the number of elements in the array.

See mxCalcSingleSubscript for more details on calculating an index.

fieldnumber
Position of the field whose value you want to extract

In C, 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.

In Fortran, the first field within each element has a field number of 1, the second field has a field number of 2, and so on. The last field has a field number of N, where N is the number of fields.

mxGetFieldByNumber (C and Fortran)

Returns

Pointer to the mxArray in the specified field for the desired element, on success. Returns NULL in C (0 in Fortran) if passed an invalid argument or if there is no value assigned to the specified field. Common causes of failure include:

- Specifying an array pointer `pm` that does not point to a structure mxArray. Call `mxIsStruct` to determine whether `pm` points to a structure mxArray.
- Specifying an `index` to an element outside the bounds of the mxArray. For example, given a structure mxArray that contains ten elements, you cannot specify an `index` greater than 9 in C (10 in Fortran).
- Specifying a nonexistent field number. Call `mxGetFieldNumber` to determine the field number that corresponds to a given field name.

Description

Call `mxGetFieldByNumber` to get the value held in the specified `fieldnumber` at the indexed element.

Note Inputs to a MEX-file are constant read-only mxArrays. Do not modify the inputs. Using `mxSetCell*` or `mxSetField*` functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

In C, calling:

```
mxGetField(pa, index, "field_name");
```

is equivalent to calling:

```
field_num = mxGetFieldNumber(pa, "field_name");  
mxGetFieldByNumber(pa, index, field_num);
```

where `index` is 0 if you have a 1-by-1 structure.

In Fortran, calling:

mxGetFieldByNumber (C and Fortran)

```
mxGetField(pm, index, 'fieldname')
```

is equivalent to calling:

```
fieldnum = mxGetFieldNumber(pm, 'fieldname')
mxGetFieldByNumber(pm, index, fieldnum)
```

where `index` is 1 if you have a 1-by-1 structure.

Examples

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxisclass.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

`mxGetField`, `mxGetFieldNameByNumber`, `mxGetFieldNumber`,
`mxGetNumberOfFields`, `mxIsStruct`, `mxSetField`, `mxSetFieldByNumber`

mxGetFieldNameByNumber (C and Fortran)

Purpose

Get field name from structure array, given field number

C Syntax

```
#include "matrix.h"
const char *mxGetFieldNameByNumber(const mxArray *pm,
                                   int fieldnumber);
```

Fortran Syntax

```
character*(*) mxGetFieldNameByNumber(pm, fieldnumber)
mwPointer pm
integer*4 fieldnumber
```

Arguments

pm

Pointer to a structure mxArray

fieldnumber

Position of the desired field. For instance, in C, to get the name of the first field, set `fieldnumber` to 0; to get the name of the second field, set `fieldnumber` to 1; and so on. In Fortran, to get the name of the first field, set `fieldnumber` to 1; to get the name of the second field, set `fieldnumber` to 2; and so on.

Returns

Pointer to the *n*th field name, on success. Returns NULL in C (0 in Fortran) on failure. Common causes of failure include

- Specifying an array pointer `pm` that does not point to a structure mxArray. Call `mxIsStruct` to determine whether `pm` points to a structure mxArray.
- Specifying a value of `fieldnumber` outside the bounds of the number of fields in the structure mxArray. In C, `fieldnumber` 0 represents the first field, and `fieldnumber` *N*-1 represents the last field, where *N* is the number of fields in the structure mxArray. In Fortran, `fieldnumber` 1 represents the first field, and `fieldnumber` *N* represents the last field.

Description

Call `mxGetFieldNameByNumber` to get the name of a field in the given structure mxArray. A typical use of `mxGetFieldNameByNumber` is to call

mxGetFieldNameByNumber (C and Fortran)

it inside a loop in order to get the names of all the fields in a given mxArray.

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];
```

In C, the field number 0 represents the field name; field number 1 represents field `billing`; field number 2 represents field `test`. A field number other than 0, 1, or 2 causes `mxGetFieldNameByNumber` to return `NULL`.

In Fortran, the field number 1 represents the field name; field number 2 represents field `billing`; field number 3 represents field `test`. A field number other than 1, 2, or 3 causes `mxGetFieldNameByNumber` to return 0.

Examples

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxisclass.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

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

mxGetFieldName (C and Fortran)

Purpose

Get field number from structure array, given field name

C Syntax

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

Fortran Syntax

```
integer*4 mxGetFieldName(pm, fieldname)
mwPointer pm
character*(*) fieldname
```

Arguments

`pm`
Pointer to a structure mxArray

`fieldname`
Name of a field in the structure mxArray

Returns

Field number of the specified `fieldname`, on success. In C, the first field has a field number of 0, the second field has a field number of 1, and so on. In Fortran, the first field has a field number of 1, the second field has a field number of 2, and so on. Returns -1 in C (0 in Fortran) on failure. Common causes of failure include

- Specifying an array pointer `pm` that does not point to a structure mxArray. Call `mxIsStruct` to determine whether `pm` points to a structure mxArray.
- Specifying the `fieldname` 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];
```

mxGetFieldName (C and Fortran)

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

Calling:

```
mxGetField(pa, index, "field_name");
```

is equivalent to calling:

```
field_num = mxGetFieldName(pa, "field_name");  
mxGetFieldByNumber(pa, index, field_num);
```

where `index` is 0 if you have a 1-by-1 structure.

In Fortran, the field `name` has a field number of 1; the field `billing` has a field number of 2; and the field `test` has a field number of 3. If you call `mxGetFieldName` and specify a field name of anything other than `name`, `billing`, or `test`, `mxGetFieldName` returns 0.

Calling:

```
mxGetField(pm, index, 'fieldname');
```

is equivalent to calling:

```
fieldnum = mxGetFieldName(pm, 'fieldname');  
mxGetFieldByNumber(pm, index, fieldnum);
```

where `index` is 1 if you have a 1-by-1 structure.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxcreatestructarray.c`

See Also

`mxGetField`, `mxGetFieldByNumber`, `mxGetFieldNameByNumber`,
`mxGetNumberOfFields`, `mxIsStruct`, `mxSetField`, `mxSetFieldByNumber`

mxGetImagData (C and Fortran)

Purpose	Pointer to imaginary data elements in array
C Syntax	<pre>#include "matrix.h" void *mxGetImagData(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetImagData(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray</p>
Returns	Pointer to the first element of the imaginary data. Returns NULL in C (0 in Fortran) if there is no imaginary data or if there is an error.
Description	This function is like mxGetPi, except that in C it returns a void *. For more information, see the description for the mxGetData function.
Examples	<p>See the following examples in <i>matlabroot/extern/examples/mex</i>.</p> <ul style="list-style-type: none">• explore.c <p>See the following examples in <i>matlabroot/extern/examples/mx</i>.</p> <ul style="list-style-type: none">• mxisfinite.c
See Also	mxGetData, mxGetPi

mxGetInf (C and Fortran)

Purpose	Value of infinity
C Syntax	<pre>#include "matrix.h" double mxGetInf(void);</pre>
Fortran Syntax	<pre>real*8 mxGetInf</pre>
Returns	Value of infinity on your system.
Description	<p>Call <code>mxGetInf</code> to return the value of the MATLAB internal <code>inf</code> variable. <code>inf</code> is a permanent variable representing IEEE® arithmetic positive infinity. Your system specifies the value of <code>inf</code>; you cannot modify it.</p> <p>Operations that return infinity include:</p> <ul style="list-style-type: none">• Division by 0. For example, <code>5/0</code> returns infinity.• Operations resulting in overflow. For example, <code>exp(10000)</code> returns infinity because the result is too large to be represented on your machine.
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxgetinf.c</code>
See Also	<code>mxGetEps</code> , <code>mxGetNaN</code>

Purpose	Sparse matrix IR array
C Syntax	<pre>#include "matrix.h" mwIndex *mxGetIr(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetIr(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to a sparse mxArray</p>
Returns	<p>Pointer to the first element in the <code>ir</code> array, if successful, and NULL in C (0 in Fortran) otherwise. Possible causes of failure include:</p> <ul style="list-style-type: none">• Specifying a full (nonsparse) mxArray.• Specifying a value for <code>pm</code> that is NULL in C (0 in Fortran). This usually means that an earlier call to <code>mxCreateSparse</code> failed.
Description	<p>Use <code>mxGetIr</code> to obtain the starting address of the <code>ir</code> array. The <code>ir</code> array is an array of integers. The length of <code>ir</code> is <code>nzmax</code>, the storage allocated for the sparse array, or <code>nnz</code>, the number of nonzero matrix elements. For example, if <code>nzmax</code> equals 100, the <code>ir</code> array contains 100 integers.</p> <p>Each value in an <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.)</p> <p>For details on the <code>ir</code> and <code>jc</code> arrays, see <code>mxSetIr</code> and <code>mxSetJc</code>.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>fulltosparse.c</code>• <code>fulltosparse.F</code> <p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p>

mxGetlr (C and Fortran)

- `mxsetdimensions.c`
- `mxsetnzmax.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

`mxGetJc`, `mxGetNzmax`, `mxSetIr`, `mxSetJc`, `mxSetNzmax`, `nzmax`, `nnz`

Purpose	Sparse matrix JC array
C Syntax	<pre>#include "matrix.h" mwIndex *mxGetJc(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetJc(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to a sparse mxArray</p>
Returns	<p>Pointer to the first element in the jc array, if successful, and NULL in C (0 in Fortran) otherwise. Possible causes of failure include</p> <ul style="list-style-type: none">• Specifying a full (nonsparse) mxArray.• Specifying a value for pm that is NULL in C (0 in Fortran). This usually means that an earlier call to mxCreateSparse failed.
Description	<p>Use mxGetJc to obtain the starting address of the jc array. The jc array is an integer array having n+1 elements, where n is the number of columns in the sparse mxArray. The values in the jc array indirectly indicate columns containing nonzero elements. For a detailed explanation of the jc array, see mxSetJc.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• fulltosparse.c• fulltosparse.F <p>See the following examples in <i>matlabroot/extern/examples/mx</i>.</p> <ul style="list-style-type: none">• mxgetnzmax.c• mxsetdimensions.c• mxsetnzmax.c

mxGetJc (C and Fortran)

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

`mxGetIr`, `mxGetNzmax`, `mxSetIr`, `mxSetJc`, `mxSetNzmax`

Purpose Pointer to logical array data

C Syntax

```
#include "matrix.h"
mxLogical *mxGetLogicals(const mxArray *array_ptr);
```

Arguments `array_ptr`
Pointer to an mxArray

Returns Pointer to the first logical element in the mxArray. The result is unspecified if the mxArray is not a logical array.

Description Call `mxGetLogicals` to access the first logical element in the mxArray that `array_ptr` points to. Once you have the starting address, you can access any other element in the mxArray.

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxislogical.c`

See Also `mxCreateLogicalArray`, `mxCreateLogicalMatrix`,
`mxCreateLogicalScalar`, `mxIsLogical`, `mxIsLogicalScalar`,
`mxIsLogicalScalarTrue`

mxGetM (C and Fortran)

Purpose Number of rows in array

C Syntax

```
#include "matrix.h"
size_t mxGetM(const mxArray *pm);
```

Fortran Syntax

```
mwPointer mxGetM(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Number of rows in the mxArray to which pm 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 pm points to a four-dimensional array having dimensions 8-by-9-by-5-by-3, mxGetM returns 8.

Note Fortran does not have an equivalent of `size_t`. `mwPointer` is a preprocessor macro that provides the appropriate Fortran type. The value returned by this function, however, is not a pointer.

Examples See the following examples in `matlabroot/extern/examples/refbook`.

- `convec.c`
- `fulltoparse.c`
- `matrixDivide.c`
- `matrixDivideComplex.c`
- `revord.c`
- `timestwo.c`

- `xtimesy.c`

For Fortran examples, see:

- `convec.F`
- `dblmat.F`
- `fulltosparse.F`
- `matsq.F`
- `timestwo.F`
- `xtimesy.F`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxmalloc.c`
- `mxsetdimensions.c`
- `mxgetnzmax.c`
- `mxsetnzmax.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`
- `mexget.c`
- `mexlock.c`
- `yprime.c`

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matdemo2.F`

See Also

`mxGetN`, `mxSetM`, `mxSetN`

mxGetN (C and Fortran)

Purpose Number of columns in array

C Syntax

```
#include "matrix.h"
size_t mxGetN(const mxArray *pm);
```

Fortran Syntax

```
mwPointer mxGetN(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Number of columns in the mxArray.

Description Call `mxGetN` to determine the number of columns in the specified mxArray.

If `pm` is an N-dimensional mxArray, `mxGetN` is the product of dimensions 2 through N. For example, if `pm` points to a four-dimensional mxArray having dimensions 13-by-5-by-4-by-6, `mxGetN` returns the value 120 ($5 \times 4 \times 6$). If the specified mxArray has more than two dimensions and you need to know exactly how many elements are in each dimension, call `mxGetDimensions`.

If `pm` points to a sparse mxArray, `mxGetN` still returns the number of columns, not the number of occupied columns.

Note Fortran does not have an equivalent of `size_t`. `mwPointer` is a preprocessor macro that provides the appropriate Fortran type. The value returned by this function, however, is not a pointer.

Examples See the following examples in `matlabroot/extern/examples/refbook`.

- `convec.c`
- `fulltosparse.c`

- `revord.c`
- `timestwo.c`
- `xtimesy.c`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxmalloc.c`
- `mxsetdimensions.c`
- `mxgetnzmax.c`
- `mxsetnzmax.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`
- `mexget.c`
- `mexlock.c`
- `yprime.c`

See the following examples in *matlabroot/extern/examples/eng_mat*.

- `matdemo2.F`

See Also

`mxGetM`, `mxGetDimensions`, `mxSetM`, `mxSetN`

mxGetNaN (C and Fortran)

Purpose Value of NaN (Not-a-Number)

C Syntax

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

Fortran Syntax

```
real*8 mxGetNaN
```

Returns Value of NaN (Not-a-Number) on your system

Description Call `mxGetNaN` 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,

- `0.0/0.0`
- `Inf-Inf`

Your system specifies the value of Not-a-Number. You cannot modify it.

C Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxgetinf.c`

See Also `mxGetEps`, `mxGetInf`

mxGetNumberOfDimensions (C and Fortran)

Purpose	Number of dimensions in array
C Syntax	<pre>#include "matrix.h" mwSize mxGetNumberOfDimensions(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwSize mxGetNumberOfDimensions(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Number of dimensions in the specified mxArray. The returned value is always 2 or greater.
Description	Use <code>mxGetNumberOfDimensions</code> to determine how many dimensions are in the specified array. To determine how many elements are in each dimension, call <code>mxGetDimensions</code> .
Examples	See the following examples in <i>matlabroot/extern/examples/mex</i> . <ul style="list-style-type: none">• <code>explore.c</code> See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• <code>findnz.c</code>• <code>fulltosparse.c</code>• <code>phonebook.c</code> See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• <code>mxcalcsinglesubscript.c</code>• <code>mxgeteps.c</code>• <code>mxisfinite.c</code>

mxGetNumberOfDimensions (C and Fortran)

See Also `mxSetM`, `mxSetN`, `mxGetDimensions`

mxGetNumberOfElements (C and Fortran)

Purpose Number of elements in array

C Syntax

```
#include "matrix.h"
size_t mxGetNumberOfElements(const mxArray *pm);
```

Fortran Syntax

```
mwPointer mxGetNumberOfElements(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Number of elements in the specified mxArray

Description mxGetNumberOfElements tells you how many elements an array has. For example, if the dimensions of an array are 3-by-5-by-10, mxGetNumberOfElements returns the number 150.

Note Fortran does not have an equivalent of `size_t`. `mwPointer` is a preprocessor macro that provides the appropriate Fortran type. The value returned by this function, however, is not a pointer.

Examples See the following examples in *matlabroot/extern/examples/refbook*.

- `findnz.c`
- `phonebook.c`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcalcsinglesubscript.c`
- `mxgeteps.c`
- `mxgetepsf.F`
- `mxgetinf.c`

mxGetNumberOfElements (C and Fortran)

- `mxisfinite.c`
- `mxsetdimensions.c`
- `mxsetdimensionsf.F`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`

See Also

`mxGetDimensions`, `mxGetM`, `mxGetN`, `mxGetClassID`, `mxGetClassName`

mxGetNumberOfFields (C and Fortran)

Purpose	Number of fields in structure array
C Syntax	<pre>#include "matrix.h" int mxGetNumberOfFields(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxGetNumberOfFields(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to a structure mxArray</p>
Returns	Number of fields, on success. Returns 0 on failure. The most common cause of failure is that pm is not a structure mxArray. Call mxIsStruct to determine whether pm is a structure.
Description	<p>Call mxGetNumberOfFields to determine how many fields are in the specified structure mxArray.</p> <p>Once you know the number of fields in a structure, you can loop through every field in order to set or to get field values.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• <code>phonebook.c</code> <p>See the following examples in <i>matlabroot/extern/examples/mx</i>.</p> <ul style="list-style-type: none">• <code>mxisclass.c</code> <p>See the following examples in <i>matlabroot/extern/examples/mex</i>.</p> <ul style="list-style-type: none">• <code>explore.c</code>
See Also	<code>mxGetField</code> , <code>mxIsStruct</code> , <code>mxSetField</code>

mxGetNzmax (C and Fortran)

Purpose Number of elements in IR, PR, and PI arrays

C Syntax

```
#include "matrix.h"
mwSize mxGetNzmax(const mxArray *pm);
```

Fortran Syntax

```
mwSize mxGetNzmax(pm)
mwPointer pm
```

Arguments pm
Pointer to a sparse mxArray

Returns Number of elements allocated to hold nonzero entries in the specified sparse mxArray, on success. Returns an indeterminate value on error. The most likely cause of failure is that pm points to a full (nonsparse) mxArray.

Description Use mxGetNzmax to get the value of the nzmax field. The nzmax field holds an integer value that signifies the number of elements in the ir, pr, and, if it exists, the pi arrays. The value of nzmax is always greater than or equal to the number of nonzero elements in a sparse mxArray. In addition, the value of nzmax is always less than or equal to the number of rows times the number of columns.

As you adjust the number of nonzero elements in a sparse mxArray, MATLAB software often adjusts the value of the nzmax field. MATLAB adjusts nzmax in order to reduce the number of costly reallocations and in order to optimize its use of heap space.

Examples See the following examples in *matlabroot/extern/examples/mx*.

- mxgetnzmax.c
- mxsetnzmax.c

See Also mxSetNzmax

Purpose	Imaginary data elements in array of type DOUBLE
C Syntax	<pre>#include "matrix.h" double *mxGetPi(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetPi(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray of type double</p>
Returns	Pointer to the imaginary data elements of the specified mxArray, on success. Returns NULL in C (0 in Fortran) if there is no imaginary data or if there is an error.
Description	<p>Use <code>mxGetPi</code> on arrays of type <code>double</code> only. Use <code>mxIsDouble</code> to validate the mxArray type. For other mxArray types, use <code>mxGetImagData</code>.</p> <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 whether an mxArray is purely real is to call <code>mxIsComplex</code>.</p> <p>If any of the input matrices to a function are complex, MATLAB allocates the imaginary parts of all input matrices.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>convec.c</code>• <code>findnz.c</code>• <code>fulltosparse.c</code> <p>For Fortran examples, see:</p>

mxGetPi (C and Fortran)

- `convec.F`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcalcsinglesubscript.c`
- `mxgetinf.c`
- `mxisfinite.c`
- `mxsetnzmax.c`

See the following examples in *matlabroot/extern/examples/mex*.

- `explore.c`
- `mexcallmatlab.c`

See Also

`mxGetPr`, `mxSetPi`, `mxSetPr`, `mxGetImagData`, `mxIsDouble`

Purpose	Real data elements in array of type DOUBLE
C Syntax	<pre>#include "matrix.h" double *mxGetPr(const mxArray *pm);</pre>
Fortran Syntax	<pre>mwPointer mxGetPr(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray of type double</p>
Returns	Pointer to the first element of the real data. Returns NULL in C (0 in Fortran) if there is no real data.
Description	<p>Use mxGetPr on arrays of type double only. Use mxIsDouble to validate the mxArray type. For other mxArray types, use mxGetData.</p> <p>Call mxGetPr to access the real data in the mxArray that pm points to. Once you have the starting address, you can access any other element in the mxArray.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• arrayFillGetPrDynamicData.c• arrayFillGetPr.c• convec.c• doubleelement.c• findnz.c• fulltosparse.c• matrixDivide.c• matrixMultiply.c• sincall.c

mxGetPr (C and Fortran)

- `timestwo.c`
- `timestwoalt.c`
- `xtimesy.c`

For Fortran examples, see:

- `convec.F`
- `dblmat.F`
- `fulltosparse.F`
- `matsq.F`
- `sincall.F`
- `timestwo.F`
- `xtimesy.F`

See Also

`mxGetPi`, `mxSetPi`, `mxSetPr`, `mxGetData`, `mxIsDouble`

Purpose	Value of public property of MATLAB object
C Syntax	<pre>#include "matrix.h" mxArray *mxGetProperty(const mxArray *pa, mwIndex index, const char *proprname);</pre>
Fortran Syntax	<pre>mwPointer mxGetProperty(pa, index, proprname) mwPointer pa mwIndex index character*(*) proprname</pre>
Arguments	<p>pa Pointer to an mxArray which is an object.</p> <p>index Index of the desired element of the object array.</p> <p>In C, the first element of an mxArray has an index of 0. The index of the last element is N-1, where N is the number of elements in the array. In Fortran, the first element of an mxArray has an index of 1. The index of the last element is N, where N is the number of elements in the array.</p> <p>proprname Name of the property whose value you want to extract.</p>
Returns	<p>Pointer to the mxArray of the specified proprname on success. Returns NULL in C (0 in Fortran) if unsuccessful. Common causes of failure include:</p> <ul style="list-style-type: none">• Specifying a nonexistent proprname.• Specifying a nonpublic proprname.• Specifying an index to an element outside the bounds of the mxArray. Use <code>mxGetNumberOfElements</code> or <code>mxGetM</code> and <code>mxGetN</code> to test the index value.• Insufficient heap space.

mxGetProperty (C and Fortran)

Description

Call `mxGetProperty` to get the value held in the specified element. In pseudo-C terminology, `mxGetProperty` returns the value at:

```
pa[index].propname
```

`mxGetProperty` makes a copy of the value. This might be a concern if the property uses a large amount of memory. There must be sufficient memory (in the heap) to hold the copy of the value.

Examples

Display Property

This example displays the Name property of a timeseries object.

Create a MEX-file, `mxgetproperty.c`, in a folder on your MATLAB path.

```
/*=====
 * mxgetproperty.c - Display timeseries Name property
 * This is a MEX-file for MATLAB.
 * Copyright 2013 The MathWorks, Inc.
 * All rights reserved.
 *=====*/

#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[])
{
    /* Check for proper number of arguments. */
    if(nrhs!=1) {
        mexErrMsgIdAndTxt( "MATLAB:mxgetproperty:invalidNumInputs",
                          "One input required.");
    } else if(nlhs>1) {
        mexErrMsgIdAndTxt( "MATLAB:mxgetproperty:maxlhs",
                          "Too many output arguments.");
    }
    /* Check for timeseries object. */
    if (!mxIsClass(prhs[0], "timeseries")) {
```

```
        mexErrMsgIdAndTxt( "MATLAB:mxgetproperty:invalidClass",  
                          "Input must be timeseries object.");  
    }  
    plhs[0] = mxGetProperty(prhs[0],0,"Name");  
}
```

Build the MEX-file.

```
mex('-v','mxgetproperty.c')
```

Create a timeseries object.

```
tsobj = timeseries(rand(5, 4), 'Name', 'LaunchData');
```

Display name.

```
tsname = mxgetproperty(tsobj)
```

```
tsname =  
LaunchData
```

See Also

[mxSetProperty](#), [mxGetNumberOfElements](#), [mxGetM](#), [mxGetN](#)

mxGetScalar (C and Fortran)

Purpose Real component of first data element in array

C Syntax

```
#include "matrix.h"
double mxGetScalar(const mxArray *pm);
```

Fortran Syntax

```
real*8 mxGetScalar(pm)
mwPointer pm
```

Arguments `pm`
Pointer to an mxArray; cannot be a cell mxArray, a structure mxArray, or an empty mxArray.

Returns Pointer to the value of the first real (nonimaginary) element of the mxArray.

In C, mxGetScalar returns a double. If real elements in the mxArray are of a type other than double, mxGetScalar automatically converts the scalar value into a double. To preserve the original data representation of the scalar, cast the return value to the desired data type.

If pm points to a sparse mxArray, mxGetScalar returns the value of the first nonzero real element in the mxArray. If there are no nonzero elements, mxGetScalar returns 0.

Description Call mxGetScalar to get the value of the first real (nonimaginary) element of the mxArray.

In most cases, you call mxGetScalar when pm points to an mxArray containing only one element (a scalar). However, pm can point to an mxArray containing many elements. If pm points to an mxArray containing multiple elements, mxGetScalar returns the value of the first real element. For example, if pm points to a two-dimensional mxArray, mxGetScalar returns the value of the (1,1) element. If pm points to a three-dimensional mxArray, mxGetScalar returns the value of the (1,1,1) element; and so on.

Use `mxGetScalar` on a 32-bit, nonempty `mxArray` of type numeric, logical, or char only. To test for these conditions, use MX Matrix Library functions such as `mxIsEmpty`, `mxIsLogical`, `mxIsNumeric`, or `mxIsChar`.

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `timestwoalt.c`
- `xtimesy.c`

See the following examples in `matlabroot/extern/examples/mex`.

- `mexlock.c`
- `mexlockf.F`

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetdimensions.c`

See Also

`mxGetM`, `mxGetN`

mxGetString (C and Fortran)

Purpose	String array to C-style string
C Syntax	<pre>#include "matrix.h" int mxGetString(const mxArray *pm, char *str, mwSize strlen);</pre>
Fortran Syntax	<pre>integer*4 mxGetString(pm, str, strlen) mwPointer pm character*(*) str mwSize strlen</pre>
Arguments	<p>pm Pointer to a string mxArray; that is, a pointer to an mxArray having the mxCHAR_CLASS class.</p> <p>str Starting location for the string. mxGetString writes the character data into str and then, in C, terminates the string with a NULL character (in the manner of C strings). str can point to either dynamic or static memory.</p> <p>strlen Size in bytes of destination buffer pointed to by str. Typically, in C, you set strlen to 1 plus the number of elements in the string mxArray to which pm points. See the mxGetM and mxGetN reference pages to find out how to get the number of elements.</p> <p>Do not use with “Multibyte Encoded Characters” on page 1-195.</p>
Returns	<p>0 on success or if strlen == 0, and 1 on failure. Possible reasons for failure include</p> <ul style="list-style-type: none">• mxArray is not a string array.• strlen is not large enough to store the entire mxArray. If so, the function returns 1 and truncates the string.
Description	Call mxGetString to copy the character data of a string mxArray into a C-style string in C or a character array in Fortran. The copied string

starts at `str` and contains no more than `strlen-1` characters in C (no more than `strlen` characters in Fortran). In C, the C-style string is always terminated with a NULL character.

If the string array contains several rows, the function copies them into one long string array, one column at a time.

Multibyte Encoded Characters

Use this function only with strings represented in single-byte encoding schemes. For strings represented in multibyte encoding schemes, use the C function `mxArrayToString`. Fortran users must allocate sufficient space for the return string to avoid possible truncation.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxmalloc.c`

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `revord.F`

See Also

`mxArrayToString`, `mxCreateCharArray`,
`mxCreateCharMatrixFromStrings`, `mxCreateString`

mxIsCell (C and Fortran)

Purpose	Determine whether input is cell array
C Syntax	<pre>#include "matrix.h" bool mxIsCell(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsCell(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if pm points to an array having the class mxCELL_CLASS, and logical 0 (false) otherwise.
Description	Use mxIsCell to determine whether the specified array is a cell array. In C, calling mxIsCell is equivalent to calling: <pre>mxGetClassID(pm) == mxCELL_CLASS</pre> In Fortran, calling mxIsCell is equivalent to calling: <pre>mxGetClassName(pm) .eq. 'cell'</pre>
	<hr/> Note mxIsCell does not answer the question “Is this mxArray a cell of a cell array?” An individual cell of a cell array can be of any type. <hr/>
See Also	mxIsClass

Purpose	Determine whether input is string array
C Syntax	<pre>#include "matrix.h" bool mxIsChar(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsChar(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if pm points to an array having the class mxCHAR_CLASS, and logical 0 (false) otherwise.
Description	<p>Use mxIsChar to determine whether pm points to string mxArray.</p> <p>In C, calling mxIsChar is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxCHAR_CLASS</pre> <p>In Fortran, calling mxIsChar is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'char'</pre>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• phonebook.c• revord.c <p>See the following examples in <i>matlabroot/extern/examples/mx</i>.</p> <ul style="list-style-type: none">• mxcreatecharmatrixfromstr.c• mxislogical.c• mxmalloc.c

mxIsChar (C and Fortran)

See Also `mxIsClass`, `mxGetClassID`

Purpose Determine whether array is member of specified class

C Syntax

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

Fortran Syntax

```
integer*4 mxIsClass(pm, classname)
mwPointer pm
character*(*) classname
```

Arguments

pm
Pointer to an mxArray

classname
Array category you are testing. Specify classname as a string (not as an integer identifier). You can specify any one of the following predefined constants:

Value of classname	Corresponding Class
cell	mxCELL_CLASS
char	mxCHAR_CLASS
double	mxDOUBLE_CLASS
function_handle	mxFUNCTION_CLASS
int8	mxINT8_CLASS
int16	mxINT16_CLASS
int32	mxINT32_CLASS
int64	mxINT64_CLASS
logical	mxLOGICAL_CLASS
single	mxSINGLE_CLASS
struct	mxSTRUCT_CLASS
uint8	mxUINT8_CLASS

mxIsClass (C and Fortran)

Value of classname	Corresponding Class
uint16	mxUINT16_CLASS
uint32	mxUINT32_CLASS
uint64	mxUINT64_CLASS
<class_name>	<class_id>
unknown	mxUNKNOWN_CLASS

In the table, <class_name> represents the name of a specific MATLAB custom object. You can also specify one of your own class names.

Returns

Logical 1 (true) if pm points to an array having category classname, and logical 0 (false) otherwise.

Description

Each mxArray is tagged as being a certain type. Call mxIsClass to determine whether the specified mxArray has this type.

In C:

```
mxIsClass(pm, "double");
```

is equivalent to calling either of these forms:

```
mxIsDouble(pm);
```

```
strcmp(mxGetClassName(pm), "double");
```

In Fortran:

```
mxIsClass(pm, 'double')
```

is equivalent to calling either one of the following:

```
mxIsDouble(pm)
```

```
mxGetClassName(pm) .eq. 'double'
```

It is most efficient to use the `mxIsDouble` form.

Examples

See the following examples in *matlabroot/extern/examples/mx*.

- `mxisclass.c`

See Also

`mxClassID`, `mxGetClassID`, `mxIsEmpty`, `mxGetClassName`

mxIsComplex (C and Fortran)

Purpose	Determine whether data is complex
C Syntax	<pre>#include "matrix.h" bool mxIsComplex(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsComplex(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if pm is a numeric array containing complex data, and logical 0 (false) otherwise. If pm points to a cell array or a structure array, mxIsComplex returns false.
Description	Use mxIsComplex to determine whether an imaginary part is allocated for an mxArray. The imaginary pointer pi is NULL in C (0 in Fortran) 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.
Examples	See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• mxisfinite.c• mxgetinf.c See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• convec.c• convec.F• fulltoparse.F• phonebook.c See the following examples in <i>matlabroot/extern/examples/mex</i> .

- `explore.c`
- `yprime.c`
- `mexlock.c`

See Also

`mxIsNumeric`

mxIsDouble (C and Fortran)

Purpose	Determine whether mxArray represents data as double-precision, floating-point numbers
C Syntax	<pre>#include "matrix.h" bool mxIsDouble(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsDouble(pm) mwPointer pm</pre>
Arguments	<p>pm Pointer to an mxArray</p>
Returns	Logical 1 (true) if the mxArray stores its data as double-precision, floating-point numbers, and logical 0 (false) otherwise.
Description	<p>Call <code>mxIsDouble</code> to determine whether the specified mxArray represents its real and imaginary data as double-precision, floating-point numbers.</p> <p>Older versions of MATLAB software store all mxArray data as double-precision, floating-point numbers. However, starting with MATLAB Version 5 software, MATLAB can store real and imaginary data in various numerical formats.</p> <p>In C, calling <code>mxIsDouble</code> is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxDOUBLE_CLASS</pre> <p>In Fortran, calling <code>mxIsDouble</code> is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'double'</pre>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>fulltospase.c</code>• <code>fulltospase.F</code> <p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p>

- `mxgeteps.c`
- `mxgetepsf.F`

See the following examples in *matlabroot/extern/examples/mex*.

- `mexget.c`

See Also

`mxIsClass`, `mxGetClassID`

mxIsEmpty (C and Fortran)

Purpose	Determine whether array is empty
C Syntax	<pre>#include "matrix.h" bool mxIsEmpty(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsEmpty(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if the mxArray is empty, and logical 0 (false) otherwise.
Description	Use <code>mxIsEmpty</code> to determine whether an mxArray contains no data. An mxArray is empty if the size of any of its dimensions is 0.
Examples	See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• <code>mxisfinite.c</code>
See Also	<code>mxIsClass</code>

Purpose	Determine whether input is finite
C Syntax	<pre>#include "matrix.h" bool mxIsFinite(double value);</pre>
Fortran Syntax	<pre>integer*4 mxIsFinite(value) real*8 value</pre>
Arguments	value Double-precision, floating-point number you are testing
Returns	Logical 1 (true) if value is finite, and logical 0 (false) otherwise.
Description	Call <code>mxIsFinite</code> to determine whether value is finite. A number is finite if it is greater than <code>-Inf</code> and less than <code>Inf</code> .
Examples	See the following examples in <code>matlabroot/extern/examples/mx</code> . <ul style="list-style-type: none">• <code>mxisfinite.c</code>
See Also	<code>mxIsInf</code> , <code>mxIsNan</code>

mxIsFromGlobalWS (C and Fortran)

Purpose Determine whether array was copied from MATLAB global workspace

C Syntax

```
#include "matrix.h"
bool mxIsFromGlobalWS(const mxArray *pm);
```

Fortran Syntax

```
integer*4 mxIsFromGlobalWS(pm)
mwPointer pm
```

Arguments

pm
 Pointer to an mxArray

Returns Logical 1 (true) if the array was copied out of the global workspace, and logical 0 (false) otherwise.

Description mxIsFromGlobalWS is useful for standalone MAT-file programs.

Examples See the following examples in *matlabroot/extern/examples/eng_mat*.

- matcreat.c
- matdgn.c

See the following examples in *matlabroot/extern/examples/mx*.

- mxislogical.c

Purpose	Determine whether input is infinite
C Syntax	<pre>#include "matrix.h" bool mxIsInf(double value);</pre>
Fortran Syntax	<pre>integer*4 mxIsInf(value) real*8 value</pre>
Arguments	value Double-precision, floating-point number you are testing
Returns	Logical 1 (true) if value is infinite, and logical 0 (false) otherwise.
Description	<p>Call <code>mxIsInf</code> to determine whether <code>value</code> is equal to infinity or minus infinity. MATLAB software stores the value of infinity in a permanent variable named <code>Inf</code>, which represents IEEE arithmetic positive infinity. The value of the variable <code>Inf</code> is built into the system; you cannot modify it.</p> <p>Operations that return infinity include</p> <ul style="list-style-type: none">• Division by 0. For example, <code>5/0</code> returns infinity.• Operations resulting in overflow. For example, <code>exp(10000)</code> returns infinity because the result is too large to be represented on your machine. <p>If <code>value</code> equals NaN (Not-a-Number), <code>mxIsInf</code> returns false. In other words, NaN is not equal to infinity.</p>
Examples	See the following examples in <code>matlabroot/extern/examples/mx</code> . <ul style="list-style-type: none">• <code>mxisfinite.c</code>
See Also	<code>mxIsFinite</code> , <code>mxIsNaN</code>

mxIsInt16 (C and Fortran)

Purpose Determine whether array represents data as signed 16-bit integers

C Syntax

```
#include "matrix.h"
bool mxIsInt16(const mxArray *pm);
```

Fortran Syntax

```
integer*4 mxIsInt16(pm)
mwPointer pm
```

Arguments pm
Pointer to an mxArray

Returns Logical 1 (true) if the array stores its data as signed 16-bit integers, and logical 0 (false) otherwise.

Description Use mxIsInt16 to determine whether the specified array represents its real and imaginary data as 16-bit signed integers.

In C, calling mxIsInt16 is equivalent to calling:

```
mxGetClassID(pm) == mxINT16_CLASS
```

In Fortran, calling mxIsInt16 is equivalent to calling:

```
mxGetClassName(pm) == 'int16'
```

See Also mxIsClass, mxGetClassID, mxIsInt8, mxIsInt32, mxIsInt64, mxIsUInt8, mxIsUInt16, mxIsUInt32, mxIsUInt64

Purpose	Determine whether array represents data as signed 32-bit integers
C Syntax	<pre>#include "matrix.h" bool mxIsInt32(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsInt32(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray</p>
Returns	Logical 1 (true) if the array stores its data as signed 32-bit integers, and logical 0 (false) otherwise.
Description	<p>Use mxIsInt32 to determine whether the specified array represents its real and imaginary data as 32-bit signed integers.</p> <p>In C, calling mxIsInt32 is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxINT32_CLASS</pre> <p>In Fortran, calling mxIsInt32 is equivalent to calling:</p> <pre>mxGetClassName(pm) == 'int32'</pre>
See Also	<p>mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt64, mxIsUInt8, mxIsUInt16, mxIsUInt32, mxIsUInt64</p>

mxIsInt64 (C and Fortran)

Purpose Determine whether array represents data as signed 64-bit integers

C Syntax

```
#include "matrix.h"
bool mxIsInt64(const mxArray *pm);
```

Fortran Syntax

```
integer*4 mxIsInt64(pm)
mwPointer pm
```

Arguments

pm
 Pointer to an mxArray

Returns Logical 1 (true) if the array stores its data as signed 64-bit integers, and logical 0 (false) otherwise.

Description Use mxIsInt64 to determine whether the specified array represents its real and imaginary data as 64-bit signed integers.

In C, calling mxIsInt64 is equivalent to calling:

```
mxGetClassID(pm) == mxINT64_CLASS
```

In Fortran, calling mxIsInt64 is equivalent to calling:

```
mxGetClassName(pm) == 'int64'
```

See Also mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt32, mxIsUInt8, mxIsUInt16, mxIsUInt32, mxIsUInt64

Purpose	Determine whether array represents data as signed 8-bit integers
C Syntax	<pre>#include "matrix.h" bool mxIsInt8(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsInt8(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray</p>
Returns	Logical 1 (true) if the array stores its data as signed 8-bit integers, and logical 0 (false) otherwise.
Description	<p>Use mxIsInt8 to determine whether the specified array represents its real and imaginary data as 8-bit signed integers.</p> <p>In C, calling mxIsInt8 is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxINT8_CLASS</pre> <p>In Fortran, calling mxIsInt8 is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'int8'</pre>
See Also	<p>mxIsClass, mxGetClassID, mxIsInt16, mxIsInt32, mxIsInt64, mxIsUInt8, mxIsUInt16, mxIsUInt32, mxIsUInt64</p>

mxIsLogical (C and Fortran)

Purpose	Determine whether array is of type mxLogical
C Syntax	<pre>#include "matrix.h" bool mxIsLogical(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsLogical(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if pm points to a logical mxArray. Otherwise, it returns logical 0 (false).
Description	Use mxIsLogical to determine whether MATLAB software treats the data in the mxArray as Boolean (logical). If an mxArray is logical, MATLAB treats all zeros as meaning false and all nonzero values as meaning true.
Examples	See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• <code>mxislogical.c</code>
See Also	<code>mxIsClass</code>
Concepts	<ul style="list-style-type: none">• “Logical Operations”

Purpose	Determine whether scalar array is of type mxLogical
C Syntax	<pre>#include "matrix.h" bool mxIsLogicalScalar(const mxArray *array_ptr);</pre>
Arguments	array_ptr Pointer to an mxArray
Returns	Logical 1 (true) if the mxArray is of class mxLogical and has 1-by-1 dimensions. Otherwise, it returns logical 0 (false).
Description	Use mxIsLogicalScalar to determine whether MATLAB treats the scalar data in the mxArray as logical or numerical.
See Also	mxGetScalar mxGetLogicals mxIsLogicalScalarTrue mxIsLogical
Concepts	<ul style="list-style-type: none">“Logical Operations”

mxIsLogicalScalarTrue (C)

Purpose	Determine whether scalar array of type mxLogical is true
C Syntax	<pre>#include "matrix.h" bool mxIsLogicalScalarTrue(const mxArray *array_ptr);</pre>
Arguments	array_ptr Pointer to an mxArray
Returns	Logical 1 (true) if the value of the mxArray's logical, scalar element is true. Otherwise, it returns logical 0 (false).
Description	Use mxIsLogicalScalarTrue to determine whether the value of a scalar mxArray is true or false.
See Also	mxGetScalar mxGetLogicals mxIsLogicalScalar mxIsLogical
Concepts	<ul style="list-style-type: none">• “Logical Operations”

Purpose	Determine whether input is NaN (Not-a-Number)
C Syntax	<pre>#include "matrix.h" bool mxIsNaN(double value);</pre>
Fortran Syntax	<pre>integer*4 mxIsNaN(value) real*8 value</pre>
Arguments	value Double-precision, floating-point number you are testing
Returns	Logical 1 (true) if value is NaN (Not-a-Number), and logical 0 (false) otherwise.
Description	<p>Call <code>mxIsNaN</code> to determine whether <code>value</code> is 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">• <code>0.0/0.0</code>• <code>Inf - Inf</code> <p>The system understands a family of bit patterns as representing NaN. In other words, NaN is not a single value; rather, it is a family of numbers that MATLAB software (and other IEEE-compliant applications) uses to represent an error condition or missing data.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxisfinite.c</code> <p>See the following examples in <code>matlabroot/extern/examples/refbook</code>.</p> <ul style="list-style-type: none">• <code>findnz.c</code>• <code>fulltosparse.c</code>

mxIsNaN (C and Fortran)

See Also `mxIsFinite`, `mxIsInf`

Purpose Determine whether array is numeric

C Syntax

```
#include "matrix.h"
bool mxIsNumeric(const mxArray *pm);
```

Fortran Syntax

```
integer*4 mxIsNumeric(pm)
mwPointer pm
```

Arguments

pm
 Pointer to an mxArray

Returns Logical 1 (true) if the array can contain numeric data. The following class IDs represent storage types for arrays that can contain numeric data:

- mxDOUBLE_CLASS
- mxSINGLE_CLASS
- mxINT8_CLASS
- mxUINT8_CLASS
- mxINT16_CLASS
- mxUINT16_CLASS
- mxINT32_CLASS
- mxUINT32_CLASS
- mxINT64_CLASS
- mxUINT64_CLASS

Logical 0 (false) if the array cannot contain numeric data.

Description Call mxIsNumeric to determine whether the specified array contains numeric data. If the specified array has a storage type that represents

mxIsNumeric (C and Fortran)

numeric data, `mxIsNumeric` returns logical 1 (true). Otherwise, `mxIsNumeric` returns logical 0 (false).

Call `mxGetClassID` to determine the exact storage type.

Examples

See the following examples in `matlabroot/extern/examples/refbook`.

- `phonebook.c`

See the following examples in `matlabroot/extern/examples/eng_mat`.

- `matdemo1.F`

See Also

`mxGetClassID`

Purpose	Determine whether array represents data as single-precision, floating-point numbers
C Syntax	<pre>#include "matrix.h" bool mxIsSingle(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsSingle(pm) mwPointer pm</pre>
Arguments	<p>pm Pointer to an mxArray</p>
Returns	Logical 1 (true) if the array stores its data as single-precision, floating-point numbers, and logical 0 (false) otherwise.
Description	<p>Use <code>mxIsSingle</code> to determine whether the specified array represents its real and imaginary data as single-precision, floating-point numbers.</p> <p>In C, calling <code>mxIsSingle</code> is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxSINGLE_CLASS</pre> <p>In Fortran, calling <code>mxIsSingle</code> is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'single'</pre>
See Also	<code>mxIsClass</code> , <code>mxGetClassID</code>

mxIsSparse (C and Fortran)

Purpose	Determine whether input is sparse array
C Syntax	<pre>#include "matrix.h" bool mxIsSparse(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsSparse(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if pm points to a sparse mxArray, and logical 0 (false) otherwise. A false return value means that pm points to a full mxArray or that pm does not point to a valid mxArray.
Description	Use mxIsSparse to determine whether pm points to a sparse mxArray. Many routines (for example, mxGetIr and mxGetJc) require a sparse mxArray as input.
Examples	See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• phonebook.c See the following examples in <i>matlabroot/extern/examples/mx</i> . <ul style="list-style-type: none">• mxgetnzmax.c• mxsetdimensions.c• mxsetdimensionsf.F• mxsetnzmax.c
See Also	sparse, mxGetIr, mxGetJc, mxCreateSparse

Purpose	Determine whether input is structure array
C Syntax	<pre>#include "matrix.h" bool mxIsStruct(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsStruct(pm) mwPointer pm</pre>
Arguments	<p>pm Pointer to an mxArray</p>
Returns	Logical 1 (true) if pm points to a structure mxArray, and logical 0 (false) otherwise.
Description	Use mxIsStruct to determine whether pm points to a structure mxArray. Many routines (for example, mxGetFieldNameByNumber and mxSetField) require a structure mxArray as an argument.
Examples	See the following examples in <i>matlabroot/extern/examples/refbook</i> . <ul style="list-style-type: none">• phonebook.c
See Also	mxCreateStructArray, mxCreateStructMatrix, mxGetFieldNameByNumber, mxGetField, mxSetField

mxIsUint16 (C and Fortran)

Purpose	Determine whether array represents data as unsigned 16-bit integers
C Syntax	<pre>#include "matrix.h" bool mxIsUint16(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsUint16(pm) mwPointer pm</pre>
Arguments	<p>pm Pointer to an mxArray</p>
Returns	Logical 1 (true) if the mxArray stores its data as unsigned 16-bit integers, and logical 0 (false) otherwise.
Description	<p>Use mxIsUint16 to determine whether the specified mxArray represents its real and imaginary data as 16-bit unsigned integers.</p> <p>In C, calling mxIsUint16 is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxUINT16_CLASS</pre> <p>In Fortran, calling mxIsUint16 is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'uint16'</pre>
See Also	<pre>mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt32, mxIsInt64, mxIsUint8, mxIsUint32, mxIsUint64</pre>

Purpose Determine whether array represents data as unsigned 32-bit integers

C Syntax

```
#include "matrix.h"
bool mxIsUint32(const mxArray *pm);
```

Fortran Syntax

```
integer*4 mxIsUint32(pm)
mwPointer pm
```

Arguments

pm
 Pointer to an mxArray

Returns Logical 1 (true) if the mxArray stores its data as unsigned 32-bit integers, and logical 0 (false) otherwise.

Description Use mxIsUint32 to determine whether the specified mxArray represents its real and imaginary data as 32-bit unsigned integers.

In C, calling mxIsUint32 is equivalent to calling:

```
mxGetClassID(pm) == mxUINT32_CLASS
```

In Fortran, calling mxIsUint32 is equivalent to calling:

```
mxGetClassName(pm) .eq. 'uint32'
```

See Also mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt32, mxIsInt64, mxIsUint8, mxIsUint16, mxIsUint64

mxIsUint64 (C and Fortran)

Purpose	Determine whether array represents data as unsigned 64-bit integers
C Syntax	<pre>#include "matrix.h" bool mxIsUint64(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsUint64(pm) mwPointer pm</pre>
Arguments	pm Pointer to an mxArray
Returns	Logical 1 (true) if the mxArray stores its data as unsigned 64-bit integers, and logical 0 (false) otherwise.
Description	<p>Use mxIsUint64 to determine whether the specified mxArray represents its real and imaginary data as 64-bit unsigned integers.</p> <p>In C, calling mxIsUint64 is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxUINT64_CLASS</pre> <p>In Fortran, calling mxIsUint64 is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'uint64'</pre>
See Also	mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt32, mxIsInt64, mxIsUint8, mxIsUint16, mxIsUint32

Purpose	Determine whether array represents data as unsigned 8-bit integers
C Syntax	<pre>#include "matrix.h" bool mxIsUint8(const mxArray *pm);</pre>
Fortran Syntax	<pre>integer*4 mxIsUint8(pm) mwPointer pm</pre>
Arguments	<p>pm</p> <p>Pointer to an mxArray</p>
Returns	Logical 1 (true) if the mxArray stores its data as unsigned 8-bit integers, and logical 0 (false) otherwise.
Description	<p>Use mxIsUint8 to determine whether the specified mxArray represents its real and imaginary data as 8-bit unsigned integers.</p> <p>In C, calling mxIsUint8 is equivalent to calling:</p> <pre>mxGetClassID(pm) == mxUINT8_CLASS</pre> <p>In Fortran, calling mxIsUint8 is equivalent to calling:</p> <pre>mxGetClassName(pm) .eq. 'uint8'</pre>
See Also	<pre>mxIsClass, mxGetClassID, mxIsInt8, mxIsInt16, mxIsInt32, mxIsInt64, mxIsUint16, mxIsUint32, mxIsUint64</pre>

mxLogical (C)

Purpose Type for logical array

Description All logical mxArray's store their data elements as mxLogical rather than as bool.

The header file containing this type is:

```
#include "matrix.h"
```

Examples See the following examples in *matlabroot/extern/examples/mx*.

- `mxislogical.c`
- For information about data in MATLAB language scripts and functions, see “Data Types”.

See Also `mxCreateLogicalArray`

Purpose	Allocate uninitialized dynamic memory using MATLAB memory manager
C Syntax	<pre>#include "matrix.h" #include <stdlib.h> void *mxMalloc(mwSize n);</pre>
Fortran Syntax	<pre>mwPointer mxMalloc(n) mwSize n</pre>
Arguments	<p>n</p> <p>Number of bytes to allocate for n greater than 0</p>
Returns	<p>Pointer to the start of the allocated dynamic memory, if successful. If unsuccessful in a MAT or engine standalone application, <code>mxMalloc</code> returns NULL in C (0 in Fortran). 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> <p>If you call <code>mxMalloc</code> in C with value <code>n = 0</code>, MATLAB returns either NULL or a valid pointer.</p>
Description	<p><code>mxMalloc</code> allocates contiguous heap space sufficient to hold n bytes. Use <code>mxMalloc</code> instead of the ANSI C <code>malloc</code> function to allocate memory in MATLAB applications.</p> <p>In MEX-files, but not MAT or engine applications, <code>mxMalloc</code> registers the allocated memory with the MATLAB memory manager. When control returns to the MATLAB prompt, the memory manager then automatically frees, or <i>deallocates</i>, this memory.</p> <p>How you manage the memory created by this function depends on the purpose of the data assigned to it. If you assign it to an output argument in <code>plhs[]</code> using the <code>mxSetPr</code> function, MATLAB is responsible for freeing the memory.</p>

mxMalloc (C and Fortran)

If you use the data internally, the MATLAB memory manager maintains a list of all memory allocated by the function and automatically frees (deallocates) the memory when control returns to the MATLAB prompt. In general, we recommend that MEX-file functions destroy their own temporary arrays and free their own dynamically allocated memory. It is more efficient to perform this cleanup in the source MEX-file than to rely on the automatic mechanism. Therefore, when you finish using the memory allocated by this function, call `mxFree` to deallocate the memory.

If you do not assign this data to an output argument, and you want it to persist after the MEX-file completes, call `mexMakeMemoryPersistent` after calling this function. 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.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxmalloc.c`
- `mxsetdimensions.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillSetPr.c`

See Also

`mexAtExit`, `mexMakeArrayPersistent`, `mexMakeMemoryPersistent`, `mxMalloc`, `mxDestroyArray`, `mxFree`, `mxRealloc`

Purpose	Reallocate dynamic memory using MATLAB memory manager
C Syntax	<pre>#include "matrix.h" #include <stdlib.h> void *mxRealloc(void *ptr, mwSize size);</pre>
Fortran Syntax	<pre>mwPointer mxRealloc(ptr, size) mwPointer ptr mwSize size</pre>
Arguments	<p>ptr Pointer to a block of memory allocated by <code>mxMalloc</code>, <code>mxRealloc</code>, or <code>mxRealloc</code>.</p> <p>size New size of allocated memory, in bytes.</p>
Returns	<p>Pointer to the start of the reallocated block of memory, if successful. If unsuccessful in a MAT or engine standalone application, <code>mxRealloc</code> returns <code>NULL</code> in C (0 in Fortran) and leaves the original memory block unchanged. (You must use <code>mxFree</code> to free the original memory block). If unsuccessful in a MEX-file, the MEX-file terminates and control returns to the MATLAB prompt.</p> <p><code>mxRealloc</code> is unsuccessful when there is insufficient free heap space.</p>
Description	<p><code>mxRealloc</code> changes the size of a memory block that has been allocated with <code>mxMalloc</code>, <code>mxRealloc</code>, or <code>mxRealloc</code>. Use <code>mxRealloc</code> instead of the ANSI C <code>realloc</code> function to allocate memory in MATLAB applications.</p> <p><code>mxRealloc</code> changes the size of the memory block pointed to by <code>ptr</code> to <code>size</code> bytes. The contents of the reallocated memory are unchanged up to the smaller of the new and old sizes. The reallocated memory might be in a different location from the original memory, so the returned pointer can be different from <code>ptr</code>. If the memory location changes, <code>mxRealloc</code> frees the original memory block pointed to by <code>ptr</code>.</p>

mxRealloc (C and Fortran)

If `size` is greater than 0 and `ptr` is NULL in C (0 in Fortran), `mxRealloc` behaves like `mxMalloc`, allocating a new block of memory of `size` bytes and returning a pointer to the new block.

If `size` is 0 and `ptr` is not NULL in C (0 in Fortran), `mxRealloc` frees the memory pointed to by `ptr` and returns NULL in C (0 in Fortran).

In MEX-files, but not MAT or engine applications, `mxRealloc` registers the allocated memory with the MATLAB memory manager. When control returns to the MATLAB prompt, the memory manager then automatically frees, or *deallocates*, this memory.

How you manage the memory created by this function depends on the purpose of the data assigned to it. If you assign it to an output argument in `plhs[]` using the `mxSetPr` function, MATLAB is responsible for freeing the memory.

If you use the data internally, the MATLAB memory manager maintains a list of all memory allocated by the function and automatically frees (deallocates) the memory when control returns to the MATLAB prompt. In general, we recommend that MEX-file functions destroy their own temporary arrays and free their own dynamically allocated memory. It is more efficient to perform this cleanup in the source MEX-file than to rely on the automatic mechanism. Therefore, when you finish using the memory allocated by this function, call `mxFree` to deallocate the memory.

If you do not assign this data to an output argument, and you want it to persist after the MEX-file completes, call `mexMakeMemoryPersistent` after calling this function. 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.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetnzmax.c`

See Also

`mexAtExit`, `mexMakeArrayPersistent`, `mexMakeMemoryPersistent`, `mxCalloc`, `mxDestroyArray`, `mxFree`, `mxMalloc`

mxRemoveField (C and Fortran)

Purpose	Remove field from structure array
C Syntax	<pre>#include "matrix.h" void mxRemoveField(mxArray *pm, int fieldnumber);</pre>
Fortran Syntax	<pre>subroutine mxRemoveField(pm, fieldnumber) mwPointer pm integer*4 fieldnumber</pre>
Arguments	<p><code>pm</code> Pointer to a structure mxArray</p> <p><code>fieldnumber</code> Number of the field you want to remove. In C, to remove the first field, set <code>fieldnumber</code> to 0; to remove the second field, set <code>fieldnumber</code> to 1; and so on. In Fortran, to remove the first field, set <code>fieldnumber</code> to 1; to remove the second field, set <code>fieldnumber</code> to 2; and so on.</p>
Description	<p>Call <code>mxRemoveField</code> to remove a field from a structure array. If the field does not exist, nothing happens. This function does not destroy the field values. To destroy the actual field values, call <code>mxRemoveField</code> and then call <code>mxDestroyArray</code>.</p> <p>Consider a MATLAB structure initialized to:</p> <pre>patient.name = 'John Doe'; patient.billing = 127.00; patient.test = [79 75 73; 180 178 177.5; 220 210 205];</pre> <p>In C, the field number 0 represents the field name; field number 1 represents field <code>billing</code>; field number 2 represents field <code>test</code>. In Fortran, the field number 1 represents the field name; field number 2 represents field <code>billing</code>; field number 3 represents field <code>test</code>.</p>
See Also	<code>mxAddField</code> , <code>mxDestroyArray</code> , <code>mxGetFieldByNumber</code>

mxSetCell (C and Fortran)

Purpose Set contents of cell array

C Syntax

```
#include "matrix.h"
void mxSetCell(mxArray *pm, mwIndex index, mxArray *value);
```

Fortran Syntax

```
subroutine mxSetCell(pm, index, value)
mwPointer pm, value
mwIndex index
```

Arguments

pm
Pointer to a cell mxArray

index
Index from the beginning of the mxArray. Specify the number of elements between the first cell of the mxArray and the cell you want to set. The easiest way to calculate index in a multidimensional cell array is to call mxCalcSingleSubscript.

value
Pointer to new value for the cell. You can put an mxArray of any type into a cell. You can even put another cell mxArray into a cell.

Description Call mxSetCell to put the designated value into a particular cell of a cell mxArray.

Note Inputs to a MEX-file are constant read-only mxArrays. Do not modify the inputs. Using mxSetCell* or mxSetField* functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

This function does not free any memory allocated for existing data that it displaces. To free existing memory, call mxDestroyArray on the pointer returned by mxGetCell before you call mxSetCell.

Examples

See the following examples in *matlabroot/extern/examples/refbook*.

- `phonebook.c`

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcreatecellmatrix.c`
- `mxcreatecellmatrixf.F`

See Also

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

mxSetClassName (C)

Purpose	Structure array to MATLAB object array
C Syntax	<pre>#include "matrix.h" int mxSetClassName(mxArray *array_ptr, const char *classname);</pre>
Arguments	<p>array_ptr Pointer to an mxArray of class mxSTRUCT_CLASS</p> <p>classname Object class to which to convert array_ptr</p>
Returns	0 if successful, and nonzero otherwise. One cause of failure is that array_ptr is not a structure mxArray. Call mxIsStruct to determine whether array_ptr is a structure.
Description	mxSetClassName 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 software until it is loaded via the LOAD command. If the specified classname is an undefined class within MATLAB, LOAD converts the object back to a simple structure array.
See Also	mxIsClass, mxGetClassID

Purpose	Set pointer to real numeric data elements in array
C Syntax	<pre>#include "matrix.h" void mxSetData(mxArray *pm, void *pr);</pre>
Fortran Syntax	<pre>subroutine mxSetData(pm, pr) mwPointer pm, pr</pre>
Arguments	<p><code>pm</code> Pointer to an mxArray</p> <p><code>pr</code> Pointer to 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 memory. Do not use the ANSI C <code>calloc</code> function, which can cause memory alignment issues leading to program termination.</p>
Description	<p><code>mxSetData</code> is like <code>mxSetPr</code>, except that in C, its second argument is a <code>void *</code>. Use this function on numeric arrays with contents other than double.</p> <p>This function does not free any memory allocated for existing data that it displaces. To free existing memory, call <code>mxFree</code> on the pointer returned by <code>mxGetData</code> before you call <code>mxSetData</code>.</p>
Examples	<p>See the following examples in <i>matlabroot/extern/examples/refbook</i>.</p> <ul style="list-style-type: none">• <code>arrayFillSetData.c</code>
See Also	<code>mxMalloc</code> , <code>mxFree</code> , <code>mxGetData</code> , <code>mxSetPr</code>

mxSetDimensions (C and Fortran)

Purpose Modify number of dimensions and size of each dimension

C Syntax

```
#include "matrix.h"
int mxSetDimensions(mxArray *pm, const mwSize *dims,
    mwSize ndim);
```

Fortran Syntax

```
integer*4 mxSetDimensions(pm, dims, ndim)
mwPointer pm
mwSize ndim
mwSize dims(ndim)
```

Arguments

pm
Pointer to an mxArray

dims
Dimensions array. Each element in the dimensions array contains the size of the array in that dimension. For example, in C, setting `dims[0]` to 5 and `dims[1]` to 7 establishes a 5-by-7 mxArray. In Fortran, setting `dims(1)` to 5 and `dims(2)` to 7 establishes a 5-by-7 mxArray. In most cases, there are `ndim` elements in the `dims` array.

ndim
Number of dimensions

Returns 0 on success, and 1 on failure. `mxSetDimensions` allocates heap space to hold the input size array. So it is possible (though unlikely) that increasing the number of dimensions can cause the system to run out of heap space.

Description Call `mxSetDimensions` to reshape an existing mxArray. `mxSetDimensions` is like `mxSetM` and `mxSetN`; however, `mxSetDimensions` provides greater control for reshaping mxArrays that have more than two dimensions.

`mxSetDimensions` does not allocate or deallocate any space for the `pr` or `pi` arrays. Consequently, if your call to `mxSetDimensions` increases

the number of elements in the mxArray, enlarge the `pr` (and `pi`, if it exists) arrays accordingly.

If your call to `mxSetDimensions` reduces the number of elements in the mxArray, you can optionally reduce the size of the `pr` and `pi` arrays using `mxRealloc`.

MATLAB automatically removes any trailing singleton dimensions specified in the `dims` argument. For example, if `ndim` equals 5 and `dims` equals [4 1 7 1 1], the resulting array has the dimensions 4-by-1-by-7.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetdimensions.c`
- `mxsetdimensionsf.F`

See Also

`mxGetNumberOfDimensions`, `mxSetM`, `mxSetN`, `mxRealloc`

mxSetField (C and Fortran)

Purpose Set field value in structure array, given index and field name

C Syntax

```
#include "matrix.h"
void mxSetField(mxArray *pm, mwIndex index,
    const char *fieldname, mxArray *pvalue);
```

Fortran Syntax

```
subroutine mxSetField(pm, index, fieldname, pvalue)
mwPointer pm, pvalue
mwIndex index
character*(*) fieldname
```

Arguments

pm
Pointer to a structure mxArray. Call `mxIsStruct` to determine whether `pm` points to a structure mxArray.

index
Index of an element in the array.

In C, the first element of an mxArray has an index of 0. The index of the last element is $N - 1$, where N is the number of elements in the array. In Fortran, the first element of an mxArray has an index of 1. The index of the last element is N , where N is the number of elements in the array.

See `mxCalcSingleSubscript` for details on calculating an index.

fieldname
Name of a field in the structure. The field must exist in the structure. Call `mxGetFieldNameByNumber` or `mxGetFieldNumber` to determine existing field names.

pvalue
Pointer to an mxArray containing the data you want to assign to `fieldname`.

Description Use `mxSetField` to assign the contents of `pvalue` to the field `fieldname` of element `index`.

If you want to replace the contents of `fieldname`, you must first free the memory of the existing data. Use the `mxGetField` function to get a pointer to the field, call `mxDestroyArray` on the pointer, then call `mxSetField` to assign the new value.

You cannot assign `pvalue` to more than one field in a structure or to more than one element in the `mxAarray`. If you want to assign the contents of `pvalue` to multiple fields, use the `mxDuplicateArray` function to make copies of the data then call `mxSetField` on each copy.

To free memory for structures created using this function, call `mxDestroyArray` only on the structure array. Do not call `mxDestroyArray` on the array `pvalue` points to. If you do, MATLAB attempts to free the same memory twice, which can corrupt memory.

Note Inputs to a MEX-file are constant read-only `mxAarrays`. Do not modify the inputs. Using `mxSetCell*` or `mxSetField*` functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

Alternatives

C Language

In C, you can replace the statements:

```
field_num = mxGetFieldNumber(pa, "fieldname");  
mxSetFieldByNumber(pa, index, field_num, new_value_pa);
```

with a call to `mxSetField`:

```
mxSetField(pa, index, "fieldname", new_value_pa);
```

Fortran Language

In Fortran, you can replace the statements:

```
fieldnum = mxGetFieldNumber(pm, 'fieldname')  
mxSetFieldByNumber(pm, index, fieldnum, newvalue)
```

mxSetField (C and Fortran)

with a call to `mxSetField`:

```
mxSetField(pm, index, 'fieldname', newvalue)
```

Examples

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcreatestructarray.c`

See Also

`mxCreateStructArray`, `mxCreateStructMatrix`, `mxGetField`,
`mxGetFieldNameByNumber`, `mxGetFieldNumber`, `mxGetNumberOfFields`,
`mxIsStruct`, `mxSetFieldByNumber`, `mxDestroyArray`,
`mxCalcSingleSubscript`

mxSetFieldByNumber (C and Fortran)

Purpose

Set field value in structure array, given index and field number

C Syntax

```
#include "matrix.h"
void mxSetFieldByNumber(mxArray *pm, mwIndex index,
    int fieldnumber, mxArray *pvalue);
```

Fortran Syntax

```
subroutine mxSetFieldByNumber(pm, index, fieldnumber, pvalue)
mwPointer pm, pvalue
mwIndex index
integer*4 fieldnumber
```

Arguments

pm

Pointer to a structure mxArray. Call `mxIsStruct` to determine whether `pm` points to a structure mxArray.

index

Index of the desired element.

In C, the first element of an mxArray has an index of 0. The index of the last element is $N-1$, where N is the number of elements in the array. In Fortran, the first element of an mxArray has an index of 1. The index of the last element is N , where N is the number of elements in the array.

See `mxCalcSingleSubscript` for details on calculating an index.

fieldnumber

Position of the field in the structure. The field must exist in the structure.

In C, the first field within each element has a `fieldnumber` of 0. The `fieldnumber` of the last is $N-1$, where N is the number of fields.

In Fortran, the first field within each element has a `fieldnumber` of 1. The `fieldnumber` of the last is N , where N is the number of fields.

mxSetFieldByNumber (C and Fortran)

pvalue

Pointer to the mxArray containing the data you want to assign.

Description

Use `mxSetFieldByNumber` to assign the contents of `pvalue` to the field specified by `fieldnumber` of element `index`. `mxSetFieldByNumber` is like `mxSetField`; however, the function identifies the field by position number, not by name.

If you want to replace the contents at `fieldnumber`, you must first free the memory of the existing data. Use the `mxGetFieldByNumber` function to get a pointer to the field, call `mxDestroyArray` on the pointer, then call `mxSetFieldByNumber` to assign the new value.

You cannot assign `pvalue` to more than one field in a structure or to more than one element in the mxArray. If you want to assign the contents of `pvalue` to multiple fields, use the `mxDuplicateArray` function to make copies of the data then call `mxSetFieldByNumber` on each copy.

To free memory for structures created using this function, call `mxDestroyArray` only on the structure array. Do not call `mxDestroyArray` on the array `pvalue` points to. If you do, MATLAB attempts to free the same memory twice, which can corrupt memory.

Note Inputs to a MEX-file are constant read-only mxArrays. Do not modify the inputs. Using `mxSetCell*` or `mxSetField*` functions to modify the cells or fields of a MATLAB argument causes unpredictable results.

Alternatives

C Language

In C, calling:

```
mxSetField(pa, index, "field_name", new_value_pa);
```

is equivalent to calling:

mxSetFieldByNumber (C and Fortran)

```
field_num = mxGetFieldNumber(pa, "field_name");
mxSetFieldByNumber(pa, index, field_num, new_value_pa);
```

Fortran Language

In Fortran, calling:

```
mxSetField(pm, index, 'fieldname', newvalue)
```

is equivalent to calling:

```
fieldnum = mxGetFieldNumber(pm, 'fieldname')
mxSetFieldByNumber(pm, index, fieldnum, newvalue)
```

Examples

See the following examples in *matlabroot/extern/examples/mx*.

- `mxcreatestructarray.c`

See Also

`mxCreateStructArray`, `mxCreateStructMatrix`, `mxGetFieldByNumber`,
`mxGetFieldNameByNumber`, `mxGetFieldNumber`, `mxGetNumberOfFields`,
`mxIsStruct`, `mxSetField`, `mxDestroyArray`, `mxCalcSingleSubscript`

mxSetImagData (C and Fortran)

Purpose Set pointer to imaginary data elements in array

C Syntax

```
#include "matrix.h"
void mxSetImagData(mxArray *pm, void *pi);
```

Fortran Syntax

```
subroutine mxSetImagData(pm, pi)
mwPointer pm, pi
```

Arguments

`pm` 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 memory. Do not use the ANSI C `calloc` function, which can cause memory alignment issues leading to program termination. If `pi` points to static memory, memory errors will result when the array is destroyed.

Description `mxSetImagData` is like `mxSetPi`, except that in C, its `pi` argument is a `void *`. Use this function on numeric arrays with contents other than `double`.

This function does not free any memory allocated for existing data that it displaces. To free existing memory, call `mxFree` on the pointer returned by `mxGetImagData` before you call `mxSetImagData`.

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxisfinite.c`

See Also `mxMalloc`, `mxFree`, `mxGetImagData`, `mxSetPi`

Purpose IR array of sparse array

C Syntax

```
#include "matrix.h"
void mxSetIr(mxArray *pm, mwIndex *ir);
```

Fortran Syntax

```
subroutine mxSetIr(pm, ir)
mwPointer 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 equals 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`.)

For example, suppose you create a 7-by-3 sparse mxArray named `Sparrow` containing six nonzero elements by typing:

```
Sparrow = zeros(7,3);
Sparrow(2,1) = 1;
Sparrow(5,1) = 1;
Sparrow(3,2) = 1;
Sparrow(2,3) = 2;
Sparrow(5,3) = 1;
Sparrow(6,3) = 1;
Sparrow = sparse(Sparrow);
```

mxSetIr (C and Fortran)

The `pr` array holds the real data for the sparse matrix, which in Sparrow is the five 1s and the one 2. If there is any nonzero imaginary data, it is in a `pi` array.

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 before 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.

This function does not free any memory allocated for existing data that it displaces. To free existing memory, call `mxFree` on the pointer returned by `mxGetIr` before you call `mxSetIr`.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetnzmax.c`

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See Also

`mxCreateSparse`, `mxGetIr`, `mxGetJc`, `mxSetJc`, `mxFree`

mxSetJc (C and Fortran)

Purpose JC array of sparse array

C Syntax

```
#include "matrix.h"
void mxSetJc(mxArray *pm, mwIndex *jc);
```

Fortran Syntax

```
subroutine mxSetJc(pm, jc)
mwPointer pm, jc
```

Arguments

pm
 Pointer to a sparse mxArray

jc
 Pointer to the jc array

Description Use mxSetJc to specify a new jc array for a sparse mxArray. The jc array is an integer array having n+1 elements, where n is the number of columns in the sparse mxArray.

If the jth column of the sparse mxArray has any nonzero elements:

- jc[j] is the index in ir, pr, and pi (if it exists) of the first nonzero element in the jth column.
- jc[j+1]-1 is the index of the last nonzero element in the jth column.
- For the jth column of the sparse matrix, jc[j] is the total number of nonzero elements in all preceding columns.

The number of nonzero elements in the jth column of the sparse mxArray is:

```
jc[j+1] - jc[j];
```

For the jth column of the sparse mxArray, jc[j] is the total number of nonzero elements in all preceding columns. The last element of the jc array, jc[number of columns], is equal to nnz, which is the number of nonzero elements in the entire sparse mxArray.

For example, consider a 7-by-3 sparse mxArray named Sparrow containing six nonzero elements, created by typing:

```
Sparrow = zeros(7,3);  
Sparrow(2,1) = 1;  
Sparrow(5,1) = 1;  
Sparrow(3,2) = 1;  
Sparrow(2,3) = 2;  
Sparrow(5,3) = 1;  
Sparrow(6,3) = 1;  
Sparrow = sparse(Sparrow);
```

The following table lists the contents of the `ir`, `jc`, and `pr` arrays.

Subscript	ir	pr	jc	Comment
(2,1)	1	1	0	Column 1 contains two nonzero elements, with rows designated by <code>ir[0]</code> and <code>ir[1]</code>
(5,1)	4	1	2	Column 2 contains one nonzero element, with row designated by <code>ir[2]</code>
(3,2)	2	1	3	Column 3 contains three nonzero elements, with rows designated by <code>ir[3]</code> , <code>ir[4]</code> , and <code>ir[5]</code>
(2,3)	1	2	6	There are six nonzero elements in all.
(5,3)	4	1		
(6,3)	5	1		

As an example of a much sparser mxArray, consider a 1000-by-8 sparse mxArray named Spacious containing only three nonzero elements. The `ir`, `pr`, and `jc` arrays contain the values listed in this table.

mxSetJc (C and Fortran)

Subscript	ir	pr	jc	Comment
(73,2)	72	1	0	Column 1 contains no nonzero elements.
(50,3)	49	1	0	Column 2 contains one nonzero element, with row designated by ir[0].
(64,5)	63	1	1	Column 3 contains one nonzero element, with row designated by ir[1].
			2	Column 4 contains no nonzero elements.
			2	Column 5 contains one nonzero element, with row designated by ir[2].
			3	Column 6 contains no nonzero elements.
			3	Column 7 contains no nonzero elements.
			3	Column 8 contains no nonzero elements.
			3	There are three nonzero elements in all.

This function does not free any memory allocated for existing data that it displaces. To free existing memory, call `mxFree` on the pointer returned by `mxGetJc` before you call `mxSetJc`.

Examples

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetdimensions.c`

See the following examples in `matlabroot/extern/examples/mex`.

- `explore.c`

See Also

`mxCreateSparse`, `mxGetIr`, `mxGetJc`, `mxSetIr`, `mxFree`

mxSetM (C and Fortran)

Purpose Set number of rows in array

C Syntax

```
#include "matrix.h"
void mxSetM(mxArray *pm, mwSize m);
```

Fortran Syntax

```
subroutine mxSetM(pm, m)
mwPointer pm
mwSize m
```

Arguments

pm
 Pointer to an mxArray

m
 Number of rows

Description Call `mxSetM` 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 `mxSetN` to set the number of columns.

You typically use `mxSetM` to change the shape of an existing mxArray. The `mxSetM` function 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, enlarge the `pr`, `pi`, `ir`, and/or `jc` arrays. Call `mxRealloc` to enlarge them.

If your calls to `mxSetM` and `mxSetN` end up reducing the number of elements in the mxArray, you might want to reduce the sizes of the `pr`, `pi`, `ir`, and/or `jc` arrays in order to use heap space more efficiently. However, reducing the size is not mandatory.

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetdimensions.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `sincall.c`

- `sincall.F`

See Also

`mxGetM`, `mxGetN`, `mxSetN`

mxSetN (C and Fortran)

Purpose Set number of columns in array

C Syntax

```
#include "matrix.h"
void mxSetN(mxArray *pm, mwSize n);
```

Fortran Syntax

```
subroutine mxSetN(pm, n)
mwPointer pm
mwSize n
```

Arguments

pm
 Pointer to an mxArray

n
 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 `pm` 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. The `mxSetN` function 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, 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, you might want to reduce the sizes of the `pr`, `pi`, `ir`, and/or `jc` arrays in order to use heap space more efficiently. However, reducing the size is not mandatory.

Examples See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetdimensions.c`

See the following examples in `matlabroot/extern/examples/refbook`.

- `sincall.c`
- `sincall.F`

See Also

`mxGetM`, `mxGetN`, `mxSetM`

mxSetNzmax (C and Fortran)

Purpose Set storage space for nonzero elements

C Syntax

```
#include "matrix.h"
void mxSetNzmax(mxArray *pm, mwSize nzmax);
```

Fortran Syntax

```
subroutine mxSetNzmax(pm, nzmax)
mwPointer pm
mwSize nzmax
```

Arguments

pm
Pointer to a sparse mxArray.

nzmax
Number of elements mxCreateSparse should allocate to hold the arrays pointed to by ir, pr, and pi (if it exists). Set nzmax greater than or equal to the number of nonzero elements in the mxArray, but 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 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. To change the size of one of these arrays:

- 1 Call mxRealloc with a pointer to the array, setting the size to the new value of nzmax.
- 2 Call the appropriate mxSet routine (mxSetIr, mxSetPr, or mxSetPi) to establish the new memory area as the current one.

Ways to determine how large to make nzmax are:

- Set `nzmax` equal to or slightly greater than the number of nonzero elements in a sparse `mxArray`. This approach conserves precious heap space.
- Make `nzmax` equal to the total number of elements in an `mxArray`. This approach eliminates (or, at least reduces) expensive reallocations.

Examples

See the following examples in *matlabroot/extern/examples/mx*.

- `mxsetnzmax.c`

See Also

`mxGetNzmax`, `mxRealloc`

mxSetPi (C and Fortran)

Purpose	Set new imaginary data elements in array of type DOUBLE
C Syntax	<pre>#include "matrix.h" void mxSetPi(mxArray *pm, double *pi);</pre>
Fortran Syntax	<pre>subroutine mxSetPi(pm, pi) mwPointer pm, pi</pre>
Arguments	<p>pm Pointer to a full (nonsparse) mxArray</p> <p>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 <code>mxMalloc</code> to allocate this memory. Do not use the ANSI C <code>calloc</code> function, which can cause memory alignment issues leading to program termination. If <code>pi</code> points to static memory, memory leaks and other memory errors might result.</p>
Description	<p>Use <code>mxSetPi</code> to set the imaginary data of the specified mxArray.</p> <p>Most <code>mxCreate*</code> functions optionally allocate heap space to hold imaginary data. If you tell an <code>mxCreate*</code> function to allocate heap space—for example, by setting the <code>ComplexFlag</code> to <code>mxCOMPLEX</code> in C (1 in Fortran) or by setting <code>pi</code> to a non-NULL value in C (a nonzero value in Fortran)—you do not ordinarily use <code>mxSetPi</code> to initialize the created mxArray's imaginary elements. Rather, you call <code>mxSetPi</code> to replace the initial imaginary values with new ones.</p> <p>This function does not free any memory allocated for existing data that it displaces. To free existing memory, call <code>mxFree</code> on the pointer returned by <code>mxGetPi</code> before you call <code>mxSetPi</code>.</p>
Examples	<p>See the following examples in <code>matlabroot/extern/examples/mx</code>.</p> <ul style="list-style-type: none">• <code>mxisfinite.c</code>

- `mxsetnzmax.c`

See Also

`mxGetPi`, `mxGetPr`, `mxSetImagData`, `mxSetPr`, `mxFree`

mxSetPr (C and Fortran)

Purpose Set new real data elements in array of type DOUBLE

C Syntax

```
#include "matrix.h"
void mxSetPr(mxArray *pm, double *pr);
```

Fortran Syntax

```
subroutine mxSetPr(pm, pr)
mwPointer pm, pr
```

Arguments

pm
Pointer to a full (nonsparse) mxArray

pr
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 `mxMalloc` to allocate this memory. Do not use the ANSI C `calloc` function, which can cause memory alignment issues leading to program termination. If `pr` points to static memory, memory leaks and other memory errors can result.

Description Use `mxSetPr` to set the real data of the specified mxArray.

All `mxCreate*` calls allocate heap space to hold real data. Therefore, you do not ordinarily use `mxSetPr` to initialize the real elements of a freshly created mxArray. Rather, you call `mxSetPr` to replace the initial real values with new ones.

This function does not free any memory allocated for existing data that it displaces. To free existing memory, call `mxFree` on the pointer returned by `mxGetPr` before you call `mxSetPr`.

Examples See the following examples in `matlabroot/extern/examples/refbook`.

- `arrayFillSetPr.c`

See the following examples in `matlabroot/extern/examples/mx`.

- `mxsetnzmax.c`

See Also mxGetPi, mxGetPr, mxSetData, mxSetPi, mxFree

mxSetProperty (C and Fortran)

Purpose Set value of public property of MATLAB object

C Syntax

```
#include "matrix.h"
void mxSetProperty(mxArray *pa, mwIndex index,
    const char *propname, const mxArray *value);
```

Fortran Syntax

```
subroutine mxSetProperty(pa, index, propname, value)
mwPointer pa, value
mwIndex index
character*(*) propname
```

Arguments

pa
Pointer to an mxArray which is an object.

index
Index of the desired element of the object array.

In C, the first element of an mxArray has an index of 0. The index of the last element is N-1, where N is the number of elements in the array. In Fortran, the first element of an mxArray has an index of 1. The index of the last element is N, where N is the number of elements in the array.

propname
Name of the property whose value you are assigning.

value
Pointer to the mxArray you are assigning.

Description Use mxSetProperty to assign a value to the specified property. In pseudo-C terminology, mxSetProperty performs the assignment:

```
pa[index].propname = value;
```

Property propname must be an existing, public property and index must be within the bounds of the mxArray. Use mxGetNumberOfElements or mxGetM and mxGetN to test the index value.

`mxSetProperty` makes a copy of the value before assigning it as the new property value. This might be a concern if the property uses a large amount of memory. There must be sufficient memory (in the heap) to hold the copy of the value.

See Also

`mxGetProperty`