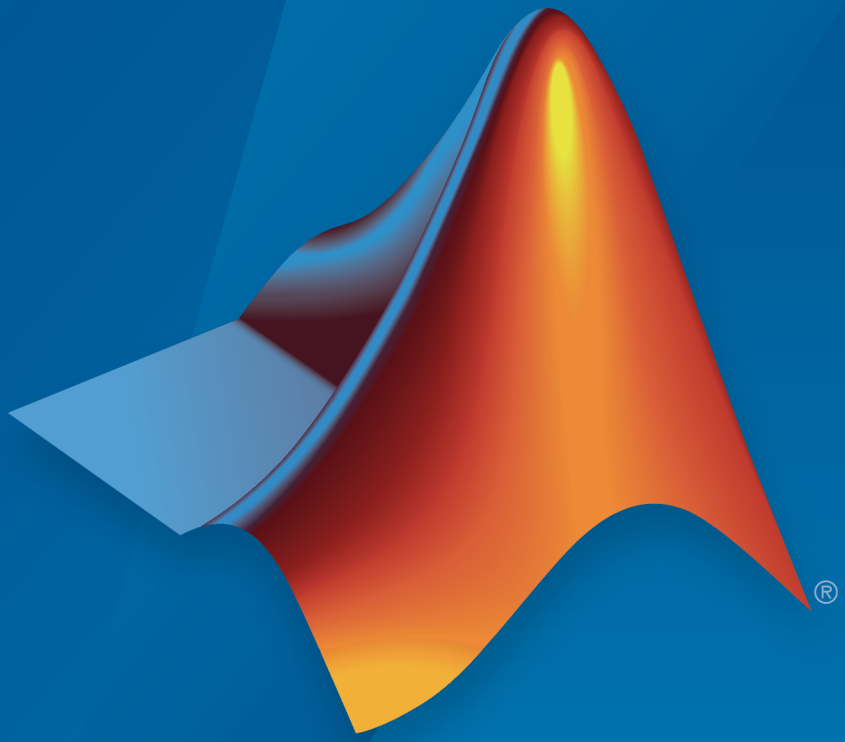
MATLAB®

MAT-File Format



MATLAB®

R2016a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

MAT-File Format

Revision History

June 1999	Online only	New for MATLAB® 5.3 (Release 11)
November 2000	PDF only	Revised for MATLAB® 6.0 (Release 12)
June 2001	PDF only	Revised for MATLAB® 6.1 (Release 12.1)
July 2002	PDF only	Revised for MATLAB® 6.5 (Release 13)
January 2003	PDF only	Revised for MATLAB® 6.5.1 (Release 13 SP1)
June 2004	PDF only	Revised for MATLAB® 7.0 (Release 14)
October 2004	PDF only	Revised for MATLAB® 7.0.1 (Release 14SP1)
September 2005	PDF only	Minor revision for MATLAB® 7.1 (Release 14SP3)
September 2007	PDF only	Rereleased for Version 7.5 (Release 2007b)
March 2008	PDF only	Revised for Version 7.6 (Release 2008a)
October 2008	PDF only	Rereleased for Version 7.7 (Release 2008b)
March 2009	PDF only	Rereleased for Version 7.8 (Release 2009a)
September 2009	PDF only	Rereleased for Version 7.9 (Release 2009b)
March 2010	PDF only	Rereleased for Version 7.10 (Release 2010a)
September 2010	PDF only	Rereleased for Version 7.11 (Release 2010b)
April 2011	PDF only	Rereleased for Version 7.12 (Release 2011a)
September 2011	PDF only	Rereleased for Version 7.13 (Release 2011b)
March 2012	PDF only	Rereleased for Version 7.14 (Release 2012a)
September 2012	PDF only	Rereleased for Version 8.0 (Release 2012b)
March 2013	PDF only	Rereleased for Version 8.1 (Release 2013a)
September 2013	PDF only	Rereleased for Version 8.2 (Release 2013b)
March 2014	PDF only	Rereleased for Version 8.3 (Release 2014a)
October 2014	PDF only	Minor revision for Version 8.4 (Release 2014b)
March 2015	PDF only	Rereleased for Version 8.5 (Release 2015a)
September 2015	PDF only	Rereleased for Version 8.6 (Release 2015b)
March 2016	PDF only	Rereleased for Version 9.0 (Release 2016a)

Contents

1 MAT-File Format

MAT-File Format

Introduction

This document describes the internal format of MATLAB Level 4 and Level 5 MAT-files. Level 4 MAT-files are compatible with versions of MATLAB up to Version 4. Level 5 MAT-files are compatible with MATLAB Versions 5 and up. You can read and write Level 4 MAT-files with the later versions of MATLAB, but when writing a MAT-file under these circumstances, you need to specify a switch in the `save` or `matOpen` command line to tell MATLAB that the MAT-file is at Level 4.

A MAT-file stores data in binary (not human-readable) form. In MATLAB, you create MAT-files using the `save` function, which writes the arrays currently in memory to a file as a continuous byte stream. By convention, this file has the filename extension `.mat`; thus the name MAT-file. The `load` function reads the arrays from a MAT-file into the MATLAB workspace.

Most MATLAB users do not need to know the internal format of a MAT-file. Even users who must read and write MAT-files from C and Fortran programs do not need to know the MAT-file format if they use the MAT-file interface. This interface shields users from dependence on the details of the MAT-file format.

Note See "Importing and Exporting Data" in the MATLAB External Interfaces documentation for information on the MAT-file interface. See "C [or Fortran] MAT-File Functions" in the MATLAB External Interfaces Reference documentation for information on the functions available with this interface.

However, if you need to read or write MAT-files on a system that does not support the MAT-file interface, you must write your own read and write routines. The MAT-file interface is only available for platforms on which MATLAB is supported. This document provides the details about the MAT-file format you will need to read and write MAT-files on these systems.

Note Whenever possible, MathWorks strongly advises you to use the MAT-file interface functions to read and write MAT-files. Any code you write that depends on the MAT-file format may need to be rewritten when the format changes in future releases.

This document describes MAT-files from a big-endian perspective. The associated figures also reflect MAT-files written by a big-endian system. In MAT-files written by a

little-endian system, the order of bytes within each instance of a MAT-file data type is reversed.

MAT-File Formats

This document describes both Level 5 and Level 4 MAT-file formats. The Level 5 MAT-file format supports all the array types supported in MATLAB Versions 5 and up, including multidimensional numeric arrays, character arrays, sparse arrays, cell arrays, structures, and objects. “Level 5 MAT-File Format” on page 1-4 describes this format.

The Level 4 MAT-file format is a simpler format, but it only supports two-dimensional matrices and character strings. “Level 4 MAT-File Format” on page 1-36 describes this format.

Level 5 MAT-File Format

Level 5 MAT-files are made up of a 128-byte *header* followed by one or more *data elements*. Each data element is composed of an 8-byte *tag* followed by the data in the element. The tag specifies the number of bytes in the data element and how these bytes should be interpreted; that is, should the bytes be read as 16-bit values, 32-bit values, floating-point values or some other data type.

By using tags, the Level 5 MAT-file format provides quick access to individual data elements within a MAT-file. You can move through a MAT-file by finding a tag, and then skipping ahead the specified number of bytes until the next tag.

MATLAB Level 5 MAT-File Format graphically illustrates this MAT-file format. The sections that follow provide more details about these MAT-file elements:

- "MAT-File Header Format" on page 1-6
- "Data Element Format" on page 1-7
- "Data Compression" on page 1-11

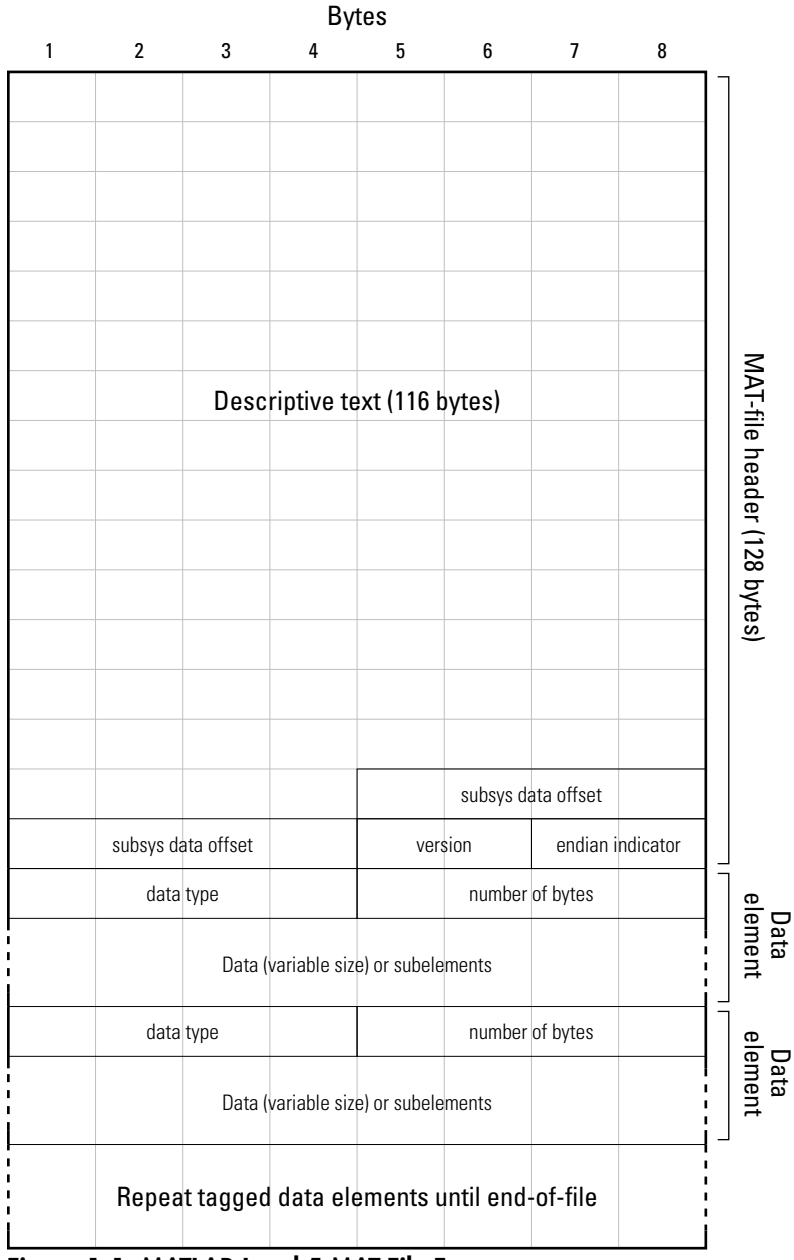


Figure 1-1. MATLAB Level 5 MAT-File Format

MAT-File Header Format

Level 5 MAT-files begin with a 128-byte header made up of a 124-byte text field and two, 16-bit flag fields.

This section covers the following topics:

Header Text Field

The first 116 bytes of the header can contain text data in human-readable form. This text typically provides information that describes how the MAT-file was created. For example, MAT-files created by MATLAB include the following information in their headers:

- Level of the MAT-file (value equals `1` for Level 5)
- Platform on which the file was created
- Date and time the file was created

You can view the text in a MAT-file header using the `cat` command on UNIX® systems, or the `type` command on a PC. The output displays the text in this part of the header. (The display of the header is followed by unreadable characters representing the binary data in the file.)

```
cat my_matfile.mat
MATLAB 5.0 MAT-file, Platform: SOL2, Created on: Thu Nov 13
10:10:27 1997
```

Note When creating a MAT-file, you *must* write data in the first 4 bytes of this header. MATLAB uses these bytes to determine if a MAT-file uses a Level 5 format or a Level 4 format. If any of these bytes contains a zero, MATLAB will incorrectly assume the file is a Level 4 MAT-file.

Header Subsystem Data Offset Field

Bytes 117 through 124 of the header contain an offset to subsystem-specific data in the MAT-file. All zeros or all spaces in this field indicate that there is no subsystem-specific data stored in the file.

Header Flag Fields

The last 4 bytes in the header are divided into two, 16-bit flag fields (`int16`).

Field	Value
Version	When creating a MAT-file, set this field to `0x0100`.
Endian Indicator	Contains the two characters, `M` and `I`, written to the MAT-file in this order, as a 16-bit value. If, when read from the MAT-file as a 16-bit value, the characters appear in reversed order (`IM` rather than `MI`), it indicates that the program reading the MAT-file must perform byte-swapping to interpret the data in the MAT-file correctly.

Note Programs that create MAT-files always write data in their native machine format. Programs that read MAT-files are responsible for byte-swapping.

Data Element Format

Each data element begins with an 8-byte tag followed immediately by the data in the element. Figure 1-2 shows this format. (MATLAB also supports a compressed data element format. See "Small Data Element Format" on page 1-10 for more information.)

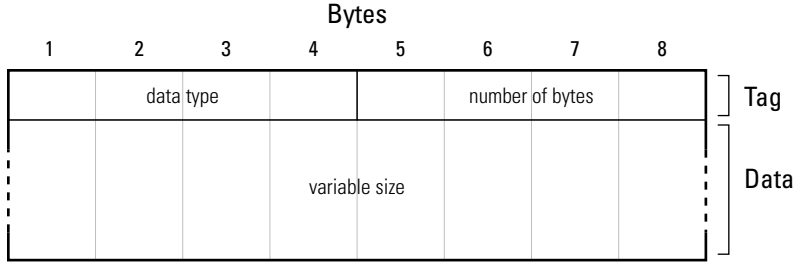


Figure 1-2. MAT-File Data Element Format

This section covers the following topics:

- "The Tag Field" on page 1-8

The Tag Field

The 8-byte data element tag is composed of two, 32-bit fields:

- Data Type
- Number of Bytes

Data Type

The Data Type field specifies how the data in the element should be interpreted, that is, its size and format. The MAT-file format supports many data types including signed and unsigned, 8-bit, 16-bit, 32-bit, and 64-bit data types, a special data type that represents MATLAB arrays, Unicode® encoded character data, and data stored in compressed format. The MAT-File Data Types table lists all these data types with the values used to specify them. The table also includes symbols that are used to represent these data types in the examples in this document.

Table 1-1. MAT-File Data Types

Value	Symbol	MAT-File Data Type
1	`miINT8`	8 bit, signed
2	`miUINT8`	8 bit, unsigned
3	`miINT16`	16-bit, signed
4	`miUINT16`	16-bit, unsigned
5	`miINT32`	32-bit, signed
6	`miUINT32`	32-bit, unsigned
7	`miSINGLE`	IEEE® 754 single format
8	`--`	Reserved
9	`miDOUBLE`	IEEE 754 double format
10	`--`	Reserved
11	`--`	Reserved
12	`miINT64`	64-bit, signed

Value	Symbol	MAT-File Data Type
		nn
13	miUINT64	64-bit, unsigned
14	miMATRIX	MATLAB array
15	miCOMPRESSED	Compressed Data
16	miUTF8	Unicode UTF-8 Encoded Character Data
17	miUTF16	Unicode UTF-16 Encoded Character Data
18	miUTF32	Unicode UTF-32 Encoded Character Data

The UTF-16 and UTF-32 encodings are in the byte order specified by the Endian Indicator (See “Header Flag Fields” on page 1-7). UTF-8 is byte order neutral.

For character data that is not Unicode encoded, the Data Type part of the Tag field should be set to miUINT16.

For more information about the miMATRIX data type, see “Level 5 MATLAB Array Data Element Formats” on page 1-13.

Number of Bytes

The Number of Bytes field is a 32-bit value that specifies the number of bytes of data in the element. This value does not include the 8 bytes of the data element's tag.

If Data Type is miCOMPRESSED, then the Number of Bytes field contains the compressed MATLAB array size in bytes. (See “Data Compression” on page 1-11.)

The Data Field

The data immediately follows the tag. All data that is uncompressed must be aligned on 64-bit boundaries. When writing a MAT-file, if the amount of data in a data element falls short of a 64-bit boundary, you must add bytes of padding to make sure the tag of the next data element falls on a 64-bit boundary. Likewise, when reading data from a MAT-file, be sure to account for these padding bytes.

Note For data elements representing MATLAB arrays, (type miMATRIX), the value of the Number of Bytes field includes padding bytes in the total. For all other MAT-file data types, the value of the Number of Bytes field does *not* include padding bytes.

Small Data Element Format

If a data element takes up only 1 to 4 bytes, MATLAB saves storage space by storing the data in an 8-byte format. In this format, the Data Type and Number of Bytes fields are stored as 16-bit values, freeing 4 bytes in the tag in which to store the data. Figure 4 illustrates this format.

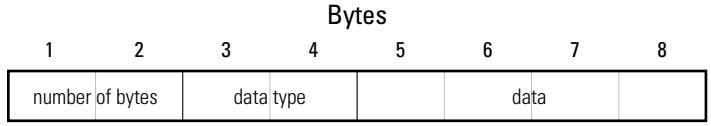


Figure 1-3. Small Data Element Format

Note When reading a MAT-file, you can tell if you are processing a small data element by comparing the value of the first 2 bytes of the tag with the value zero (0). If these 2 bytes are not zero, the tag uses the small data element format. When writing to a MAT-file, use of the small data element format is optional.

Example Data Element

Figure 1-4 illustrates a data element representing an array of six 32-bit, unsigned integers: 1, 2, 3, 4, 5, 6. In the figure, the Data Type field contains the value from MAT-File Data Types that specifies unsigned, 32-bit integers (`miUINT32`). The Number of Bytes field in the data element tag contains the number of data values multiplied by the number of bytes used to represent each value. Note that this value does not include the 8 bytes in the data element tag.

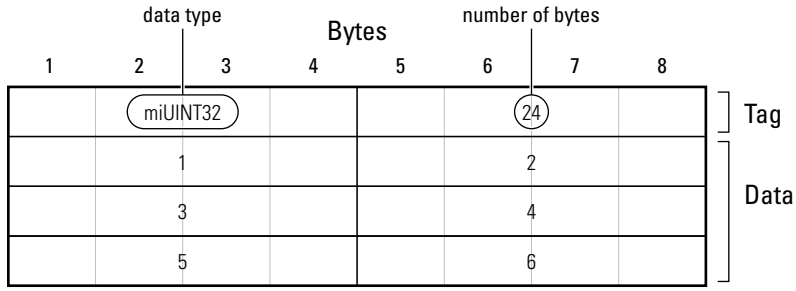


Figure 1-4. Example MAT-File Data Element

Data Compression

MATLAB compresses the data it saves to a MAT-file using buffered in-memory gzip compression. It compresses MATLAB variables transparently as they are written out to disk. This technique uses less memory than systems that compress an entire variable at once before writing it out to disk. Also, no temporary files are required to read or write compressed data.

Because it compresses each variable individually, MATLAB can read a compressed MAT-file like any other MAT-file. No code changes are required at either the C or M level to read a compressed MAT-file.

MATLAB compresses data for MAT-files (file I/O) only, not for sequential streams. The reason for this is that the size of the compressed variable is known only after it is compressed, but it must be written in the tag at the beginning of the variable. In a file it is possible to seek back and write the size, while in a stream this cannot be done.

MAT-files containing compressed variables are nonplatform specific like any MAT-file, and such a file saved on any MATLAB supported platform can be loaded on any other supported platform.

To decompress the contents of a compressed variable in a MAT-file, you can use the `uncompress` function from the freeware `zlib-1.1.4` library available at the `gzip` web site, http://www.gzip.org/zlib/. Once a MATLAB array has been decompressed, you can ignore the `miCOMPRESSED` tag and process the data normally, as if it had not been compressed.

Storing Compressed Data

MATLAB stores compressed data in gzip-compressed MATLAB arrays. Each compressed variable is stored complete with its tags and data field in the same format as uncompressed variables, as described in “Data Element Format” on page 1-7. The difference is that the entire variable is compressed into a data buffer, and this buffer is preceded by an 8-byte tag named `miCOMPRESSED`. The tag contains the length of the compressed buffer.

Each variable in a MAT-file has a 56-byte header. Even if the data is stored in the header itself, as can be the case for variables containing 1 to 4 bytes of data, no variable in a MAT-file can be less than 56 bytes. Thus, regardless of how random the data in a variable may be, it is unlikely (but not impossible) that a compressed variable will take more space than its uncompressed counterpart. This is because the 56-byte header

always compresses to a smaller size. Note that compression works best on nonrandom data. The more random the data, the less it will compress.

Level 5 MATLAB Array Data Element Formats

The MAT-file data type `miMATRIX` (`14`) is used to represent all types of MATLAB arrays, including:

- Numeric arrays
- Character arrays
- Sparse arrays
- Cell arrays
- Structures
- Objects

The `miMATRIX` data type is a compound data type. MAT-file data elements of this type are composed of multiple *subelements*. The subelements can be of any other MAT-file data type, including other `miMATRIX` data types.

Figure 1-5 shows a `miMATRIX` data element composed of three subelements. Note how each subelement is a data element with its own tag. The value of the Number of Bytes field (`96` in the figure) in the data element tag includes all the subelements.

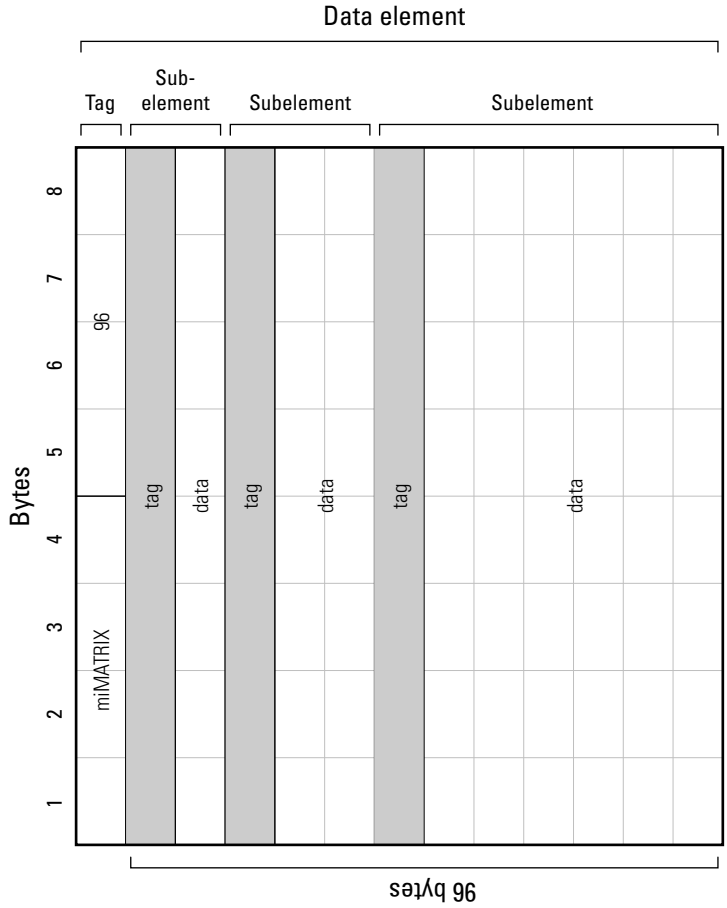


Figure 1-5. MATLAB Array Data Element with Subelements

Each `miMATRIX` data element representing the different types of MATLAB arrays each has a specific set of subelements. Some of these subelements are common to all MATLAB arrays. Others subelements are unique to a particular type of array. The following sections detail the subelements for each MATLAB array type.

Numeric Array and Character Array Data Element Formats

A MAT-file data element representing a MATLAB numeric array or character array is composed of four subelements and one optional subelement. Table 1-2 lists the

subelements in the order in which they appear in the data element. The table also includes the values of the Data Type and Number of Bytes fields you would use in the tag of each subelement. For an example, see "Examples of Numeric Array Data Elements" on page 1-19.

Table 1-2. Numeric and Character Array Subelements with Tag Data

Subelement	Data Type	Number of Bytes
Array Flags	`miUINT32`	`2 * sizeOfDataType (8 bytes)`
Dimensions Array	`miINT32`	`numberOfDimensions * sizeOfDataType` (To learn how to determine the number of dimensions, see "Dimensions Array Subelement" on page 1-17.)
Array Name	`miINT8`	`numberOfCharacters * sizeOfDataType`
Real part (`pr`)	Any of the numeric data types.	`numberOfValues * sizeOfDataType`
Imaginary part (`pi`) (Optional)	Any of the numeric data types.	`numberOfvalues * sizeOfDataType`

Array Flags Subelement

This subelement identifies the MATLAB array type (class) represented by the data element and provides other information about the array. The Array Flags subelement is common to all array types.

Figure 1-6 illustrates the format of the Array Flags subelement. (For sparse matrices, bytes 5 through 8 are used to store the maximum number of nonzero elements in the matrix. See "Sparse Array Data Element Format" on page 1-21 for more information.)

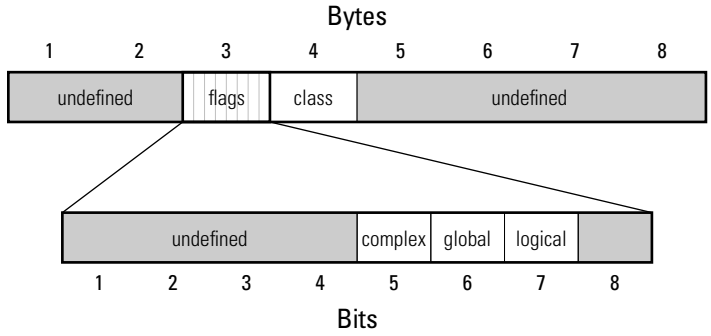


Figure 1-6. Array Flags Format

Flags

This field contains three, single-bit flags that indicate whether the numeric data is complex, global, or logical. If the complex bit is set, the data element includes an imaginary part (`pi`). If the global bit is set, MATLAB loads the data element as a global variable in the base workspace. If the logical bit is set, it indicates the array is used for logical indexing.

Class

This field contains a value that identifies the MATLAB array type (class) represented by the data element. Table 1-3 lists the MATLAB array types with the values you use to specify them. The table also includes symbols that are used to represent the MATLAB array type in the examples in this document.

Note The value of the Class field identifies the MATLAB data type. The value of the Data Type field in the data element tag identifies the data type used to store the data *in the MAT-file*. The MAT-file data types are listed in Table 1-1. The value of the Class and the Data Type fields do not need to be the same; for more information, see "Automatic Compression of Numeric Data" on page 1-18.

Table 1-3. MATLAB Array Types (Classes)

MATLAB Array Type (Class)	Value	Symbol
Cell array	1	`mxCELL_CLASS`

MATLAB Array Type (Class)	Value	Symbol
Structure	2	mxSTRUCT_CLASS
Object	3	mxOBJECT_CLASS
Character array	4	mxCHAR_CLASS
Sparse array	5	mxSPARSE_CLASS
Double precision array	6	mxDOUBLE_CLASS
Single precision array	7	mxSINGLE_CLASS
8-bit, signed integer	8	mxINT8_CLASS
8-bit, unsigned integer	9	mxUINT8_CLASS
16-bit, signed integer	10	mxINT16_CLASS
16-bit, unsigned integer	11	mxUINT16_CLASS
32-bit, signed integer	12	mxINT32_CLASS
32-bit, unsigned integer	13	mxUINT32_CLASS
64-bit, signed integer	14	mxINT64_CLASS
64-bit, unsigned integer	15	mxUINT64_CLASS

For numeric arrays, Class can contain any of the numeric array types: mxDOUBLE_CLASS, mxSINGLE_CLASS, mxINT8_CLASS, mxUINT8_CLASS, mxINT16_CLASS, mxUINT16_CLASS, mxINT32_CLASS, or mxUINT32_CLASS.

For character arrays, Class contains mxCHAR_CLASS.

Dimensions Array Subelement

This subelement specifies the size of each dimension of an n-dimensional array in an n-sized array of 32-bit values (miINT32). All numeric arrays have at least two dimensions. The Dimensions Array subelement is common to all MATLAB array types.

For example, if a data element represents a 2-by-3-by-2 MATLAB array, the Dimensions Array subelement would contain three values: 2, 3, and 2.

Note To calculate the number of dimensions in an array, divide the value stored in the Number of Bytes field in the Dimensions Array subelement tag by 4, the number of bytes in the data type (miINT32) used in the subelement.

Array Name Subelement

This subelement specifies the name assigned to the array, as an array of signed, 8-bit values (`miINT8`). This subelement is common to all array types.

Real Part (pr) Subelement

This subelement contains the numeric data in the MATLAB array. If the array contains complex numbers (the complex bit in the Array Flags is set), this is the real part of the number.

The data type of the values can be any of the numeric data types listed in MAT-File Data Types.

For character data that is not Unicode encoded, the Data Type part of the Tag field should be set to `miUINT16`.

Imaginary Part (pi) Subelement

This subelement contains the imaginary part of the numeric data in the MATLAB array. This subelement is only present if one or more of the numeric values in the MATLAB array is a complex number (if the complex bit is set in Array Flags). The data type of the values can be any of the numeric data types listed in MAT-File Data Types.

Note When reading a MAT-file, check the value of the Data Type field in the tag of Real Part and Imaginary Part subelements to identify the data type used to store data. Also note that MATLAB reads and writes these values in column-major order.

Automatic Compression of Numeric Data

MATLAB stores the numeric data in an array in double-precision format. When MATLAB writes a numeric (or sparse) array to a MAT-file, ````it uses the smallest one of `miINT32`, `miUINT16`, `miINT16`, or `miUINT8` to store the data, both the real and imaginary parts.

For example, if MATLAB determines that the data stored in double-precision format can actually be stored in an 8-bit format, it will use an `miUINT8` to store it in a MAT-file. Note, however, that if any of the numeric values in the array requires a 64-bit representation, MATLAB stores all of the data in a 64-bit data type. See “Compressed Data Element” on page 1-20 for an example.

When you create a MAT-file, compressing data is optional.

Note When MATLAB uses a smaller data type to store data in a MAT-file, the value of the Class field in the Array Flags subelement identifies the original MATLAB data type.

Examples of Numeric Array Data Elements

This section uses examples to illustrate both the compressed and uncompressed numeric array data element formats.

Uncompressed Data Element

Figure 1-7 shows how this 2-by-2 numeric array, `my_array`, is represented in a MAT-file.

```
my_array = [ 1.1+1.1i 2 ; 3 4 ]

my_array =

     1.1000 + 1.1000i     2.0000
     3.0000               4.0000
```

In the figure, note:

- The data element includes five subelements. Because one of the numeric values in the array is a complex number, the complex bit flag in the Array Flags subelement is set and the Imaginary Part (`pi`) subelement is included.
- The value of the Number of Bytes field in the data element tag includes all the subelements, but not the 8 bytes of the tag itself.

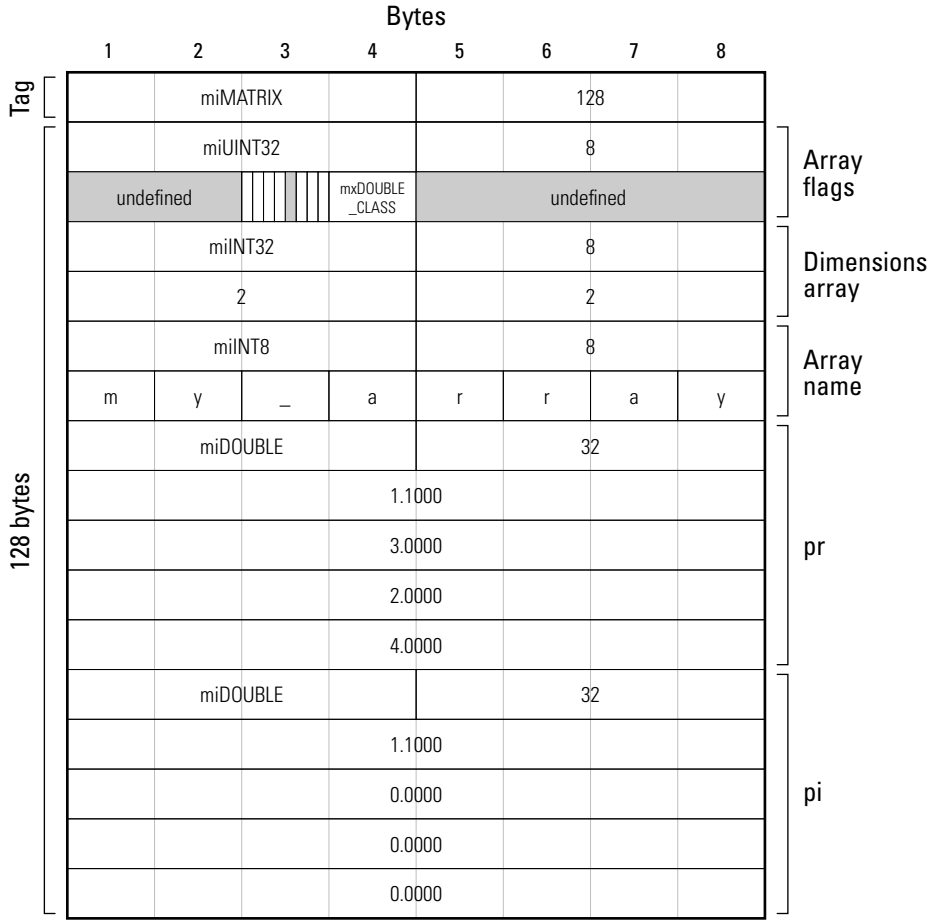


Figure 1-7. Example Numeric Array MAT-File Data Element

Compressed Data Element

Figure 1-8 shows how the three-dimensional numeric array in this example, `arr`, is represented in a MAT-file when compression is used to conserve storage space.

```
A = [ 1 2 3 ; 4 5 6 ];
B = [ 7 8 9 ; 10 11 12];
arr = cat(3,A,B)
arr(:,:,1) =
```

```
     1     2     3
     4     5     6

arr(:,:,2) =
     7     8     9
    10    11    12
```

In the figure, note:

- The Array Name subelement uses the compressed data element format.
- The numeric data in the array, stored in double-precision format in MATLAB, is stored as 8-bit, unsigned values in the `pr` subelement. The Class field in the Array Flags subelement identifies the original MATLAB data type.

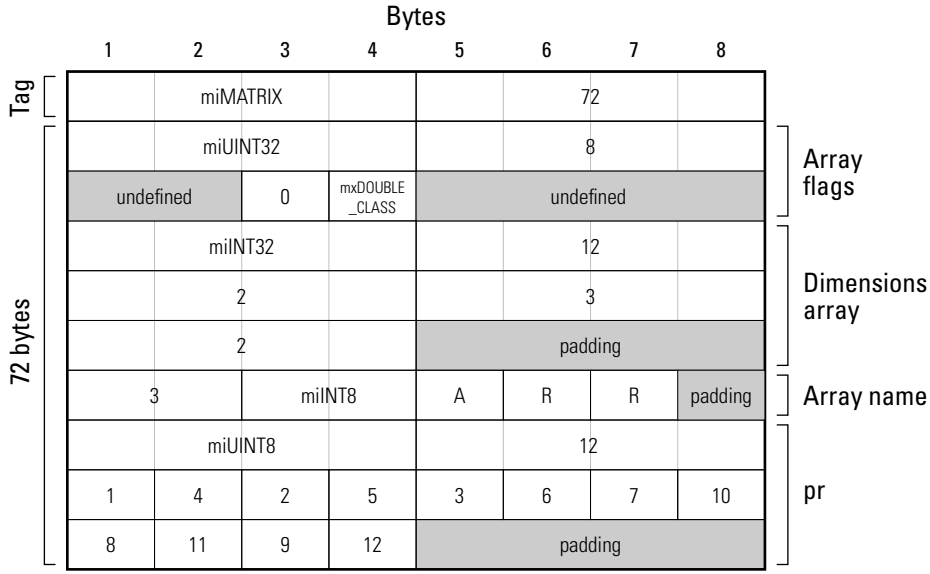


Figure 1-8. Example Numeric Array MAT-file Data Element (Compressed)

Sparse Array Data Element Format

A MAT-file data element representing a MATLAB sparse array is composed of six subelements and one optional subelement. Table 1-4 lists the subelements in the order in which they appear in the data element. The table lists the values of the Data Type and Number of Bytes fields of the tag for each subelement.

Table 1-4. Sparse Array Subelements with Tag Data

Subelement	Data Type	Number of Bytes
Array Flags	`miUINT32`	`2 * sizeOfDataType (8 bytes)`
Dimensions Array	`miINT32`	`numberOfDimensions * sizeOfDataType` where `numberOfdimensions` can be `0`, `1` or `2`.
Array Name	`miINT8`	`numberOfcharacters * sizeOfDataType`
Row Index (`ir`)	`miINT32`	`nzmax * sizeOfDataType` (The `nzmax` value is stored in Array Flags.)
Column Index (`jc`)	`miINT32`	`(N+1) * sizeof(int32)` where `N` is the second element of the Dimensions array subelement.
Real part (`pr`)	Any numeric data type	`numberOfNonzeroValues * sizeOfDataType`
Imaginary part (`pi`) (Optional)	Any numeric data type	`numberOfNonzeroValues * sizeOfDataType`

Array Flags Subelement

This subelement identifies the MATLAB array type (class) represented by the data element and provides other information about the array. The Array Flags subelement is common to all array types.

Figure 1-9 shows the Array Flags format. For sparse arrays, this value also contains the maximum number of nonzero elements in the array (`nzmax`).

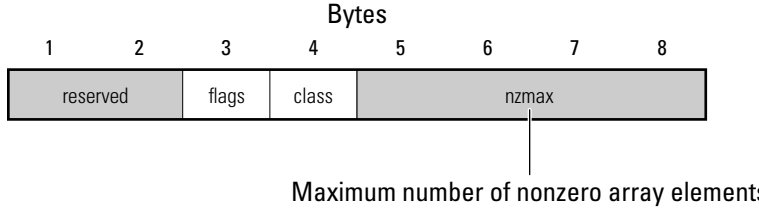


Figure 1-9. Array Flags Format for Sparse Arrays

Flag

For more information, see “Flags” on page 1-16.

Class

This field contains a value that identifies the MATLAB data type represented by the data element. For sparse arrays, Class contains the value 5 (`mxSPARSE_CLASS`). See "Class" on page 1-16 for more information.

Dimensions Array Subelement

This subelement specifies the size of each dimension of the array. This subelement is common to all array types. For more information, see "Dimensions Array Subelement" on page 1-17.

Note that MATLAB only supports two-dimensional sparse arrays.

Array Name Subelement

This subelement specifies the name assigned to the array. This subelement is common to all array types. For more information, see "Array Name Subelement" on page 1-18.

Row Index for Nonzero Values (ir) Subelement

This subelement specifies the row indices of the nonzero elements in the real part (`pr`) of the matrix data and the imaginary part (`pi`) of the matrix data, if present. This subelement is a series of 32-bit (`miINT32`) values.

Column Index for Nonzero Values (jc) Subelement

This subelement contains column index information as a series of 32-bit (`miINT32`) values. For more information about what this subelement contains, see the *MATLAB Application Program Interface Guide*.

Real Part (pr) Subelement

This subelement contains the numeric data in the MATLAB array. If the array contains complex numbers (the complex bit in the Array Flags is set), this is the real part of the number.

Because MATLAB uses data compression to save storage space, the data type of the values can be any of the numeric data types listed in MAT-File Data Types. For more information, see "Automatic Compression of Numeric Data" on page 1-18.

Imaginary Part (pi) Subelement

This subelement contains the imaginary data in the array, if one or more of the numeric values in the MATLAB array is a complex number (if the complex bit is set in Array Flags).

Because MATLAB uses data compression to save storage space, the data type of the values can be any of the numeric data types listed in MAT-File Data Types. For more information, see "Automatic Compression of Numeric Data" on page 1-18.

Note You must check the value of the Data Type field in the tag of Real Part and Imaginary Part subelements to identify the type of the data. Also note that MATLAB reads and writes these values in column-major order.

Example Sparse Array

Figure 1-10 illustrates the MAT-file data element format of this 3-by-3 sparse matrix:

```
a = [ 1 2 3 ];
S = sparse(a,a,a+.5)

S =

   (1,1)        1.5000
   (2,2)        2.5000
   (3,3)        3.5000
```

In the figure, note:

- The data element contains six subelements.
- The value of the Number of Bytes field in the data element tag includes all the subelements, but not the 8 bytes of the tag itself.
- Bytes 5 through 8 of the Array Flags subelement contain the maximum number of nonzero elements (`nzmax`) in the sparse array.
- The Array Name subelement uses the compressed data element format.

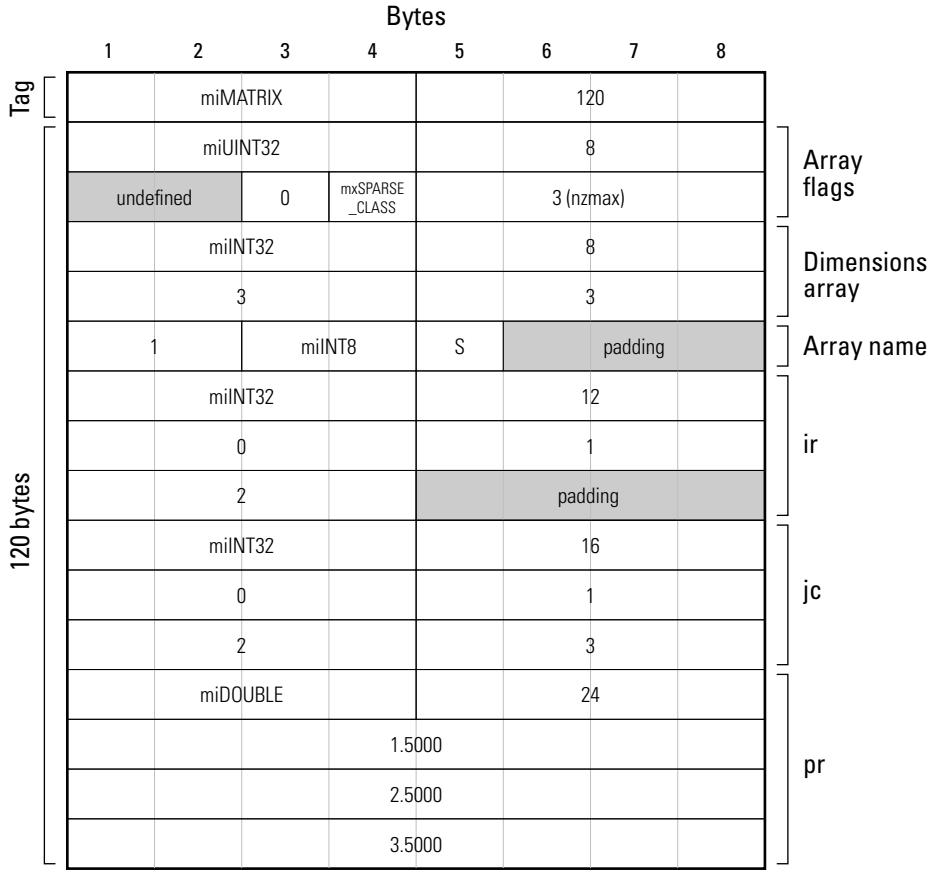


Figure 1-10. Example Sparse Array MAT-file Data Element

Cell Array Data Element Format

A MAT-file data element representing a MATLAB cell array is composed of four subelements. Table 1-5 lists the subelements in the order in which they appear in the data element. The table lists the values of the Data Type and Number of Bytes fields of the tag for each subelement.

Table 1-5. Cell Array Subelements with Tag Data

Subelement	Data Type	Number of Bytes
Array Flags	`miUINT32`	`2 * sizeOfDataType (8 bytes)`
Dimensions Array	`miINT32`	`numberOfDimensions * sizeOfDataType`
Array Name	`miINT8`	`numberOfcharacters * sizeOfDataType`
Cells	Each cell is written in place as an `miMATRIX` element.	

Array Flags Subelement

This subelement identifies the MATLAB array type (class) represented by the data element and provides other information about the array. Figure 1-11 shows the Array Flags format. The Array Flags subelement is common to all array types.

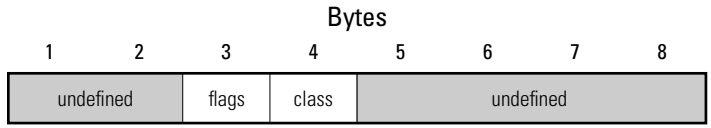


Figure 1-11. Array Flags Format

Flags

See “Flags” on page 1-16 for more information.

Class

This field contains a value that identifies the MATLAB data type represented by the data element. For cell arrays, Class contains the value 1 (`mxCELL_CLASS`). For more information, see “Class” on page 1-16.

Dimensions Array Subelement

This subelement specifies the size of each dimension of the array. This subelement is common to all array types. For more information, see “Dimensions Array Subelement” on page 1-17.

Array Name Subelement

This subelement specifies the name assigned to the array. This subelement is common to all array types. For more information, see “Array Name Subelement” on page 1-18.

Cells Subelement

This subelement contains the value stored in a cell. These values are MATLAB arrays, represented using the `miMATRIX` format specific to the array type: numeric array, sparse array, structure, object or other cell array. See the appropriate section in this document for details about the MAT-file representation of a each of these array types. Cells are written in column-major order.

Example Cell Array

Figure 1-12 illustrates the MAT-file data element format of this cell array:

```
A = [ 1 2 3 ; 4 5 6 ]
A =
     1     2     3
     4     5     6

B = [ 7 8 9 ; 10 11 12 ]
B =
     7     8     9
    10    11    12

C = { A, B }
C =
   [2x3 double]     [2x3 double]
```

In the figure, note:

- The data element contains five subelements, the three common subelements; Array Flags, Dimensions and Array Name; and two cell subelements.
- The value of the Number of Bytes field in the data element tag includes all the subelements, but not the 8 bytes of the tag itself.
- Each cell subelement is an `miMATRIX` type. In the example, each cell contains a numeric array. For more information about the format of these elements, see “Numeric Array and Character Array Data Element Formats” on page 1-14.

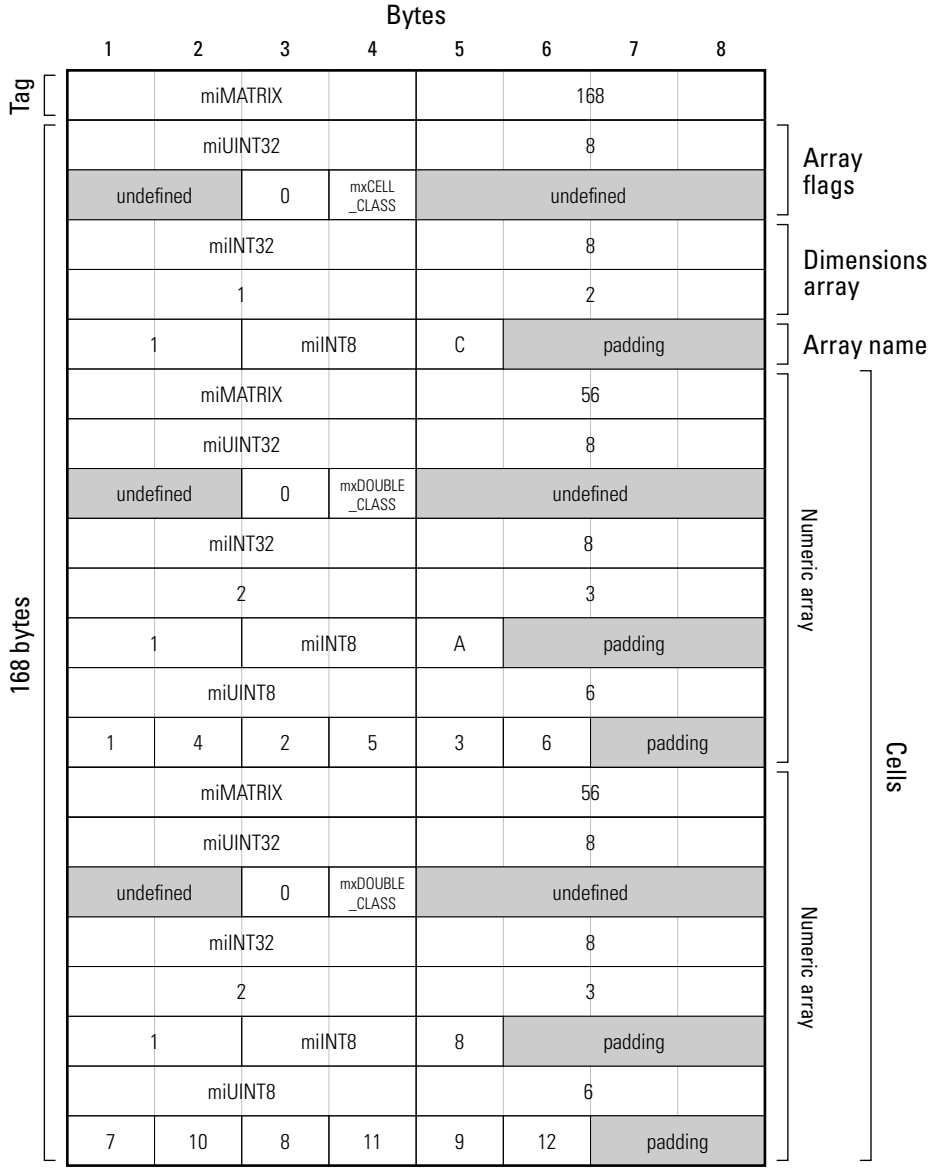


Figure 1-12. Example Cell Array Data Element

Structure MAT-File Data Element Format

A MAT-file data element representing a MATLAB structure is composed of six subelements. Table 1-6 lists the subelements in the order in which they appear in the data element. The table lists the values of the Data Type and Number of Bytes fields of the tag for each subelement.

Table 1-6. Structure Subelements with Tag Data

Subelements	Data Type	Number of Bytes
Array Flags	`miUINT32`	`2*sizeOfDataType (8 bytes)`
Dimensions Array	`miINT32`	`numberOfDimensions * sizeOfDataType`
Array Name	`miINT8`	`numberOfCharacters * sizeOfDataType`
Field Name Length	`miINT32`	`sizeOfDataType (4 bytes)`
Field Names	`miINT8`	`numberOfFields * FieldNameLength`
Fields	Each field is written in place as an array. Fields are written in column order.	

Array Flags Subelement

This subelement identifies the MATLAB array type (class) represented by the data element and provides other information about the array. Figure 1-13 shows the Array Flags format. The Array Flags subelement is common to all array types.

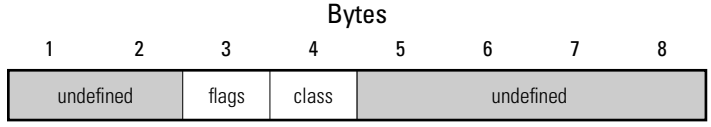


Figure 1-13. Array Flags Format

Flags

See “Flags” on page 1-16 for more information.

Class

This field contains a value that identifies the MATLAB data type represented by the data element. For structures, Class contains the value 2 (`mxSTRUCT_CLASS`). For more information, see “Class” on page 1-16.

Dimensions Array Subelement

This subelements Specifies the size of each dimension of the array. This subelement is common to all array types. For more information, see “Dimensions Array Subelement” on page 1-17.

Array Name Subelement

This subelement specifies the name assigned to the structure. This subelement is common to all array types. For more information, see “Array Name Subelement” on page 1-18.

Field Name Length Subelement

This subelement specifies the maximum length of a Field Name. MATLAB sets this limit to 32 (31 characters and a `NULL` terminator). In a MAT-file created by MATLAB, this subelement always uses the compressed data element format.

Field Names Subelement

This subelement specifies the name of each field in the structure as a series of 8-bit (`miINT8`) character arrays. The value of the Field Name Length subelement determines the length of each field name array (32 bytes). Field names must be `NULL`-terminated.

Fields Subelement

This subelement contains the value stored in a field. These values are MATLAB arrays, represented using the `miMATRIX` format specific to the array type: numeric array, sparse array, cell, object or other structure. See the appropriate section of this document for details about the MAT-file format of each of these array type. MATLAB reads and writes these fields in column-major order.

Example

Figure 1-14 illustrates the MAT-file data element format for this MATLAB structure:

```
X.w = [1];
X.y = [2];
X.z = [3];
X
   X =
```

```
w: 1
y: 2
z: 3
```

In the figure, note:

- The data element contains eight subelements: the three common subelements (Array Flags, Dimensions and Array Name) and five structure-specific subelements (Field Name Length, Field Names, and three Field subelements).
- The value of the Number of Bytes field in the data element tag includes all the subelements, but not the 8 bytes of the tag itself.
- The Field Names subelement allocates 32 bytes of storage for each field name. A `NULL` terminator indicates the end of each field name.
- Each Field subelement is an `miMATRIX` data type. In the example, each field contains a numeric array. For more information about the format of these elements, see “Numeric Array and Character Array Data Element Formats” on page 1-14.
- Each of the numeric arrays contain zero-length Array Name subelements. The Field Names subelement contains the names of the numeric arrays.

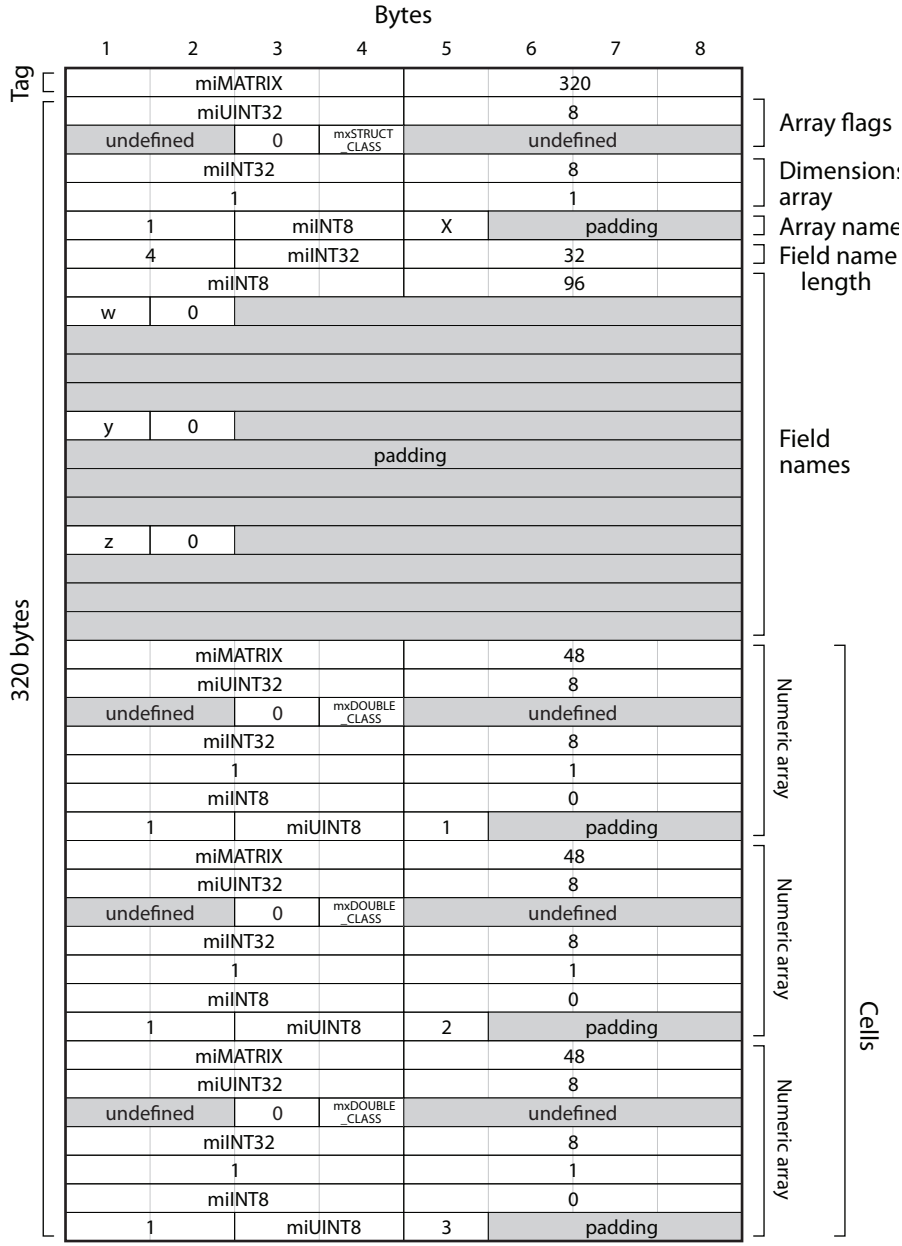


Figure 1-14. Example Structure MAT-File Data Element

MATLAB Object MAT-File Data Element Format

A MAT-file data element representing a MATLAB object is composed of seven subelements. Table 1-7 lists the subelements in the order in which they appear in the data element. An object data element has the same subelements as a structure with the addition of the Class Name subelement. The table lists the values of the Data Type and Number of Bytes fields of the tag for each subelement.

Table 1-7. MATLAB Object Subelements with Tag Data

Subelement	Data Type	Number of Bytes
Array Flags	`miUINT32`	`2 * sizeOfDataType (8 bytes)`
Dimensions Array	`miINT32`	`numberOfDimensions * sizeOfDataType`
Array Name	`miINT8`	`numberOfCharacters * sizeOfDataType`
Class Name	`miINT8`	`numberOfCharacters * sizeOfDataType`
Field Name Length	`miINT32`	`sizeOfDataType (4 bytes)`
Field Names	`miINT8`	`numberOfFields * FieldNameLength`
Fields	Each field is written in place as an array.	

Array Flags Subelement

This subelement identifies the MATLAB array type (class) represented by the data element and provides other information about the array. Figure 1-15 shows the Array Flags format. The Array Flags subelement is common to all array types.

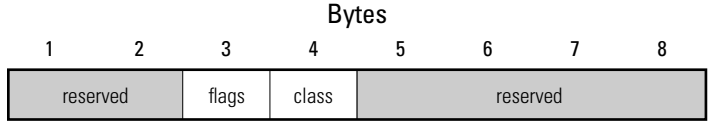


Figure 1-15. Array Flags Format

Flags

See “Flags” on page 1-16 for more information.

Class

This field contains a value that identifies the MATLAB data type represented by the data element. For objects, the Class byte has the value 3 (`mxOBJECT_CLASS`). For more information, see “Class” on page 1-16.

Dimensions Array Subelement

This subelement specifies the size of each dimension of the array. This subelement is common to all array types. For more information, see “Dimensions Array Subelement” on page 1-17.

Array Name Subelement

This subelement specifies the name assigned to the array. This subelement is common to all array types. For more information, see “Array Name Subelement” on page 1-18.

Class Name Subelement

This subelement specifies the name assigned to the object class. This subelement is an array of 8-bit characters (`miINT8`).

Field Name Length Subelement

This subelement specifies the maximum length of a Field Name. See “Field Name Length Subelement” on page 1-30 for more information.

Field Names Subelement

This subelement specifies the name of each field in the structure. See “Field Names Subelement” on page 1-30 for more information.

Fields Subelement

This subelement contains the value stored in a field. See “Fields Subelement” on page 1-30 for more information.

Example

Figure 1-16 illustrates how the MATLAB object in this example is represented in a MAT-file.

```
X = inline(`t^2');
```

The figure only shows the first four subelements of the object. For an example that shows the remaining subelements, see “Example” on page 1-30.

In the figure, note:

- The Array Flag Class byte is set to `mxOBJECT_CLASS`.

- The data element includes the Class Name subelement.

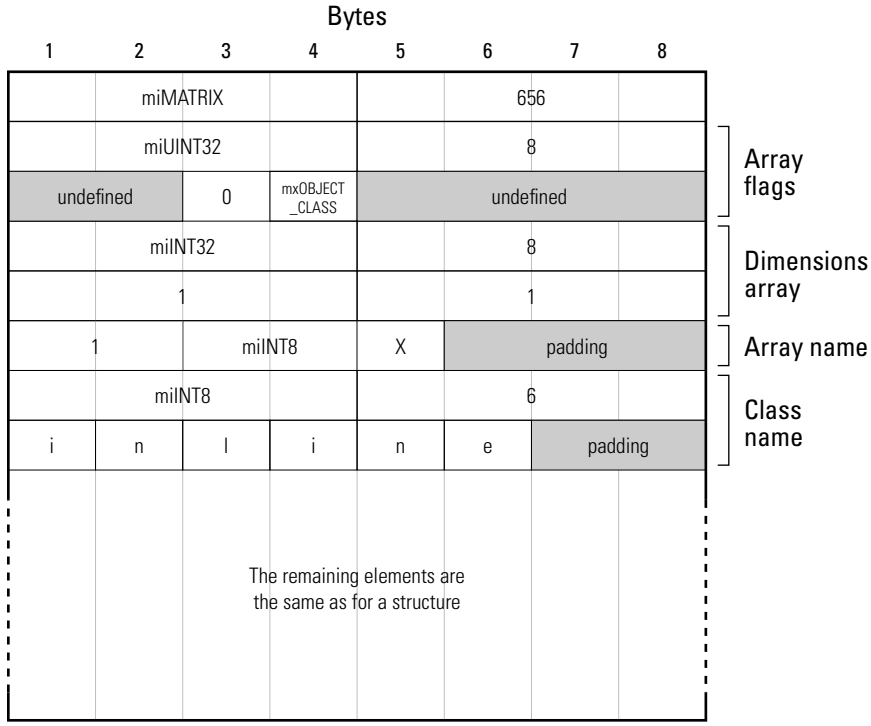


Figure 1-16. Example Object MAT-file Data Element

Level 4 MAT-File Format

Note This section is taken from the MATLAB V4.2 *External Interface Guide*, which is no longer available in printed form.

This section presents the internal structure of Level 4 MAT-files. This information is provided to enable users to read and write MAT-files on machines for which the MAT-file access routine library is not available. It is not needed when using the MAT-file subroutine library to read and write MAT-files, and we strongly advise that you do use the External Interface Library if it is available for all of the machines that you are working with.

A MAT-file may contain one or more matrices. The matrices are written sequentially on disk, with the bytes forming a continuous stream. Each matrix starts with a fixed-length 20-byte header that contains information describing certain attributes of the Matrix. The 20-byte header consists of five long (4-byte) integers:

Table 1-8. Level 4 MAT-File Matrix Header Format

Field	Description	
`type`	The `type` flag contains an integer whose decimal digits encode storage information. If the integer is represented as `MOPT` where `M` is the thousands digit, `O` is the hundreds digit, `P` is the tens digit, and `T` is the ones digit, then:	
	`M` indicates the numeric format of binary numbers on the machine that wrote the file. Use this table to determine the number to use for your machine:	
	`0`	IEEE Little Endian (PC, 386, 486, DEC Risc)
	`1`	IEEE Big Endian (Macintosh, SPARC®, Apollo, SGI, HP 9000/300, other Motorola® systems)
	`2`	VAX D-float
	`3`	VAX G-float
	`4`	Cray
	`O` is always 0 (zero) and is reserved for future use.	
	`P` indicates which format the data is stored in according to the following table:	
	`0`	double-precision (64-bit) floating-point numbers

Field	Description	
	1	single-precision (32-bit) floating-point numbers
	2	32-bit signed integers
	3	16-bit signed integers
	4	16-bit unsigned integers
	5	8-bit unsigned integers
	The precision used by the `save` command depends on the size and type of each matrix. Matrices with any noninteger entries and matrices with 10,000 or fewer elements are saved in floating-point formats requiring 8 bytes per real element. Matrices with all integer entries and more than 10,000 elements are saved in the following formats, requiring fewer bytes per element.	
	Element range	Bytes per element
	[0:255]	1
	[0:65535]	2
	[-32767:32767]	2
	[-2^31+1:2^31-1]	4
	other	8
	`T` indicates the matrix type according to the following table:	
	0	Numeric (Full) matrix
	1	Text matrix
	2	Sparse matrix
	Note that the elements of a text matrix are stored as floating-point numbers between 0 and 255 representing ASCII-encoded characters.	
`mrows`	The row dimension contains an integer with the number of rows in the matrix.	
`ncols`	The column dimension contains an integer with the number of columns in the matrix.	
`imagf`	The imaginary flag is an integer whose value is either 0 or 1. If 1, then the matrix has an imaginary part. If 0, there is only real data.	
`namlen`	The name length contains an integer with 1 plus the length of the matrix name.	

Immediately following the fixed length header is the data whose length is dependent on the variables in the fixed length header:

Table 1-9. Level 4 MAT-File Matrix Data Format

Field	Description
`name`	The matrix name consists of `namlen` ASCII bytes, the last one of which must be a `null` character (`'\0'`).
`real`	Real part of the matrix consists of `mrows * ncols` numbers in the format specified by the `P` element of the type flag. The data is stored column-wise such that the second column follows the first column, etc.
`imag`	Imaginary part of the matrix, if any. If the imaginary flag `imagf` is nonzero, the imaginary part of a matrix is placed here. It is stored in the same manner as the real data.

This structure is repeated for each matrix stored in the file.

The following C language code demonstrates how to write a single matrix to disk in Level 1.0 MAT-file format.

```
#include <stdio.h>

main() {
typedef struct {
   long type;
   long mrows;
   long ncols;
   long imagf;
   long namelen;
   } Fmatrix;

char *pname;
double *pr;
double *pi;
Fmatrix x;
int mn;
FILE *fp;

double real_data = 1.0;
double imag_data = 2.0;

fp = fopen("mymatfile.mat", "wb");
if (fp != NULL) {
   pname = "x";
```

```
   x.type = 1000;
   x.mrows = 1;
   x.ncols = 1;
   x.imagf = 1;
   x.namelen = 2;

   pr = &real_data;
   pi = &imag_data;

   fwrite(&x, sizeof(Fmatrix), 1, fp);
   fwrite(pname, sizeof(char), x.namelen, fp);

   mn = x.mrows *x.ncols;
   fwrite(pr, sizeof(double), mn, fp);

   if(x.imagf)
      fwrite(pi, sizeof(double), mn, fp);
   }

else
   printf("File could not be opened.\n");

fclose(fp);
}
```

Again, we strongly advise against using this approach, and recommend that you instead use the MAT-file access routines provided in the External Interface Library. You will need to write your own C code as shown above only if you do not have the MAT-file access routines for the particular platform on which you need to read and write MAT-files.