LIBMATIO API 1.3.1

Christopher Hulbert

7 Sep 2006

Contents

1 LIBMATIO API Library Documentation 3

1.1 Matlab MAT File I/O Library . . . 3

1.2 Internal Functions . . . 19

2 LIBMATIO API Data Structure Documentation 35

2.1 ComplexSplit Struct Reference . . . 35

2.2 mat_t Struct Reference . . . 36

2.3 matvar_t Struct Reference . . . 38

2.4 sparse_t Struct Reference . . . 40

Chapter 1

LIBMATIO API Library Documentation

1.1 Matlab MAT File I/O Library

Data Structures

- struct ComplexSplit

  Complex data type using split storage.

- struct mat_t

  Matlab MAT File information.

- struct matvar_t

  Matlab variable information.

- struct sparse_t

  sparse data information

Enumerations

- enum { BY_NAME = 1, BY_INDEX = 2 }

  matio lookup type

- enum mat_acc { MAT_ACC_RDONLY = 1, MAT_ACC_RDWR = 2 }

  MAT file access types.

- enum mat_ft { MAT_FT_MAT5 = 1, MAT_FT_MAT4 = 1 << 16 }

  MAT file versions.

- enum matio_classes {
  MAT_C_CELL = 1, MAT_C_STRUCT = 2, MAT_C_OBJECT = 3, MAT_C_CHAR = 4,
  MAT_C_SPARSE = 5, MAT_C_DOUBLE = 6, MAT_C_SINGLE = 7, MAT_C_INT8 = 8,

MAT_C_UINT8 = 9, MAT_C_INT16 = 10, MAT_C_UINT16 = 11, MAT_C_INT32 = 12, MAT_C_UINT32 = 13, MAT_C_INT64 = 14, MAT_C_UINT64 = 15, MAT_C_FUNCTION = 16 }

Matlab variable classes.

- enum matio_compression { COMPRESSION_NONE = 0, COMPRESSION_ZLIB = 1 }

  Matlab compression options.

- enum matio_flags { MAT_F_COMPLEX = 0x0800, MAT_F_GLOBAL = 0x0400, MAT_F_LOGICAL = 0x0200, MAT_F_CLASS_T = 0x00ff }

  Matlab array flags.

- enum matio_types {
  MAT_T_UNKNOWN = 0, MAT_T_INT8 = 1, MAT_T_UINT8 = 2, MAT_T_INT16 = 3,
  MAT_T_UINT16 = 4, MAT_T_INT32 = 5, MAT_T_UINT32 = 6, MAT_T_SINGLE = 7,
  MAT_T_DOUBLE = 9, MAT_T_INT64 = 12, MAT_T_UINT64 = 13, MAT_T_MATRIX = 14,
  MAT_T_COMPRESSED = 15, MAT_T_UTF8 = 16, MAT_T_UTF16 = 17, MAT_T_UTF32 = 18,
  MAT_T_STRING = 20, MAT_T_CELL = 21, MAT_T_STRUCT = 22, MAT_T_ARRAY = 23,
  MAT_T_FUNCTION = 24 }

  Matlab data types.

Functions

- int Mat_CalcSingleSubscript (int rank, int ∗dims, int ∗subs)

  Calculate a single subscript from a set of subscript values.

- int ∗ Mat_CalcSubscripts (int rank, int ∗dims, int index)

  Calculate a set of subscript values from a single(linear) subscript.

- int Mat_Close (mat_t ∗mat)

  Closes an open Matlab MAT file.

- mat_t ∗ Mat_Create (const char ∗matname, const char ∗hdr_str)

  Creates a new Matlab MAT file.

- mat_t ∗ Mat_Open (const char ∗matname, int mode)

  Opens an existing Matlab MAT file.

- int Mat_Rewind (mat_t ∗mat)

  Rewinds a Matlab MAT file to the first variable.

- size_t Mat_SizeOfClass (int class_type)

  Returns the size of a Matlab Class.

- int Mat_VarAddStructField (matvar_t ∗matvar, matvar_t ∗∗fields)

  Adds a field to a structure.

- matvar_t ∗ Mat_VarCalloc (void)

  Allocates memory for a new matvar_t and initializes all the fields.

- matvar_t ∗ Mat_VarCreate (const char ∗name, int class_type, int data_type, int rank, int ∗dims, void ∗data, int opt)

  Creates a MAT Variable with the given name and (optionally) data.

- int Mat_VarDelete (mat_t ∗mat, char ∗name)

  Deletes a variable from a file.

- matvar_t ∗ Mat_VarDuplicate (const matvar_t ∗in, int opt)

  Duplicates a matvar_t structure.

- void Mat_VarFree (matvar_t ∗matvar)

  Frees all the allocated memory associated with the structure.

- matvar_t ∗ Mat_VarGetCell (matvar_t ∗matvar, int index)

  Returns a pointer to the Cell array at a specific index.

- matvar_t ∗∗ Mat_VarGetCells (matvar_t ∗matvar, int ∗start, int ∗stride, int ∗edge)

  Indexes a cell array.

- matvar_t ∗∗ Mat_VarGetCellsLinear (matvar_t ∗matvar, int start, int stride, int edge)

  Indexes a cell array.

- int Mat_VarGetNumberOfFields (matvar_t ∗matvar)

  Returns the number of fields in a structure variable.

- size_t Mat_VarGetSize (matvar_t ∗matvar)

  Calculates the size of a matlab variable in bytes.

- matvar_t ∗ Mat_VarGetStructField (matvar_t ∗matvar, void ∗name_or_index, int opt, int index)

  Finds a field of a structure.

- matvar_t ∗ Mat_VarGetStructs (matvar_t ∗matvar, int ∗start, int ∗stride, int ∗edge, int copy_fields)

  Indexes a structure.

- matvar_t ∗ Mat_VarGetStructsLinear (matvar_t ∗matvar, int start, int stride, int edge, int copy_fields)

  Indexes a structure.

- void Mat_VarPrint (matvar_t ∗matvar, int printdata)

  Prints the variable information.

- matvar_t ∗ Mat_VarRead (mat_t ∗mat, char ∗name)

  Reads the variable with the given name from a MAT file.

- int Mat_VarReadData (mat_t ∗mat, matvar_t ∗matvar, void ∗data, int ∗start, int ∗stride, int ∗edge)

  Reads MAT variable data from a file.

- int Mat_VarReadDataAll (mat_t ∗mat, matvar_t ∗matvar)

  Reads all the data for a matlab variable.

- int Mat_VarReadDataLinear (mat_t ∗mat, matvar_t ∗matvar, void ∗data, int start, int stride, int edge)

  Reads MAT variable data from a file.

- matvar_t ∗ Mat_VarReadInfo (mat_t ∗mat, char ∗name)

  Reads the information of a variable with the given name from a MAT file.

- matvar_t ∗ Mat_VarReadNext (mat_t ∗mat)

  Reads the next variable in a MAT file.

- matvar_t ∗ Mat_VarReadNextInfo (mat_t ∗mat)

  Reads the information of the next variable in a MAT file.

- int Mat_VarWrite (mat_t ∗mat, matvar_t ∗matvar, int compress)

  Writes the given MAT variable to a MAT file.

- int Mat_VarWriteData (mat_t ∗mat, matvar_t ∗matvar, void ∗data, int ∗start, int ∗stride, int ∗edge)

  Writes the given data to the MAT variable.

- int Mat_VarWriteInfo (mat_t ∗mat, matvar_t ∗matvar)

  Writes the given MAT variable to a MAT file.

1.1.1 Enumeration Type Documentation

1.1.1.1 anonymous enum

matio lookup type

Enumerator:

BY_NAME Lookup by name

BY_INDEX Lookup by index

1.1.1.2 enum mat_acc

MAT file access types

Enumerator:

MAT_ACC_RDONLY Read only file access.

MAT_ACC_RDWR Read/Write file access.

1.1.1.3 enum mat_ft

MAT file versions

Enumerator:

MAT_FT_MAT5 Matlab level-5 file.

MAT_FT_MAT4 Version 4 file.

1.1.1.4 enum matio_classes

Matlab variable classes

Enumerator:

MAT_C_CELL Matlab cell array class.

MAT_C_STRUCT Matlab structure class.

MAT_C_OBJECT Matlab object class.

MAT_C_CHAR Matlab character array class.

MAT_C_SPARSE Matlab sparse array class.

MAT_C_DOUBLE Matlab double-precision class.

MAT_C_SINGLE Matlab single-precision class.

MAT_C_INT8 Matlab signed 8-bit integer class.

MAT_C_UINT8 Matlab unsigned 8-bit integer class.

MAT_C_INT16 Matlab signed 16-bit integer class.

MAT_C_UINT16 Matlab unsigned 16-bit integer class.

MAT_C_INT32 Matlab signed 32-bit integer class.

MAT_C_UINT32 Matlab unsigned 32-bit integer class.

MAT_C_INT64 Matlab unsigned 32-bit integer class.

MAT_C_UINT64 Matlab unsigned 32-bit integer class.

MAT_C_FUNCTION Matlab unsigned 32-bit integer class.

1.1.1.5 enum matio_compression

Matlab compression options

Enumerator:

COMPRESSION_NONE No compression.

COMPRESSION_ZLIB zlib compression

1.1.1.6 enum matio_flags

Matlab array flags

Enumerator:

MAT_F_COMPLEX Complex bit flag.

MAT_F_GLOBAL Global bit flag.

MAT_F_LOGICAL Logical bit flag.

MAT_F_CLASS_T Class-Type bits flag.

1.1.1.7 enum matio_types

Matlab data types

Enumerator:

MAT_T_UNKNOWN UNKOWN data type.

MAT_T_INT8 8-bit signed integer data type

MAT_T_UINT8 8-bit unsigned integer data type

MAT_T_INT16 16-bit signed integer data type

MAT_T_UINT16 16-bit unsigned integer data type

MAT_T_INT32 32-bit signed integer data type

MAT_T_UINT32 32-bit unsigned integer data type

MAT_T_SINGLE IEEE 754 single precision data type.

MAT_T_DOUBLE IEEE 754 double precision data type.

MAT_T_INT64 64-bit signed integer data type

MAT_T_UINT64 64-bit unsigned integer data type

MAT_T_MATRIX matrix data type

MAT_T_COMPRESSED compressed data type

MAT_T_UTF8 8-bit unicode text data type

MAT_T_UTF16 16-bit unicode text data type

MAT_T_UTF32 32-bit unicode text data type

MAT_T_STRING String data type.

MAT_T_CELL Cell array data type.

MAT_T_STRUCT Structure data type.

MAT_T_ARRAY Array data type.

MAT_T_FUNCTION Function data type.

1.1.2 Function Documentation

1.1.2.1 int Mat_CalcSingleSubscript (int *rank*, int ∗ *dims*, int ∗ *subs*)

Calculates a single linear subscript (0-relative) given a 1-relative subscript for each dimension. The calculation uses the formula below where index is the linear index, s is an array of length RANK where each element is the subscript for the correspondind dimension, D is an array whose elements are the dimensions of the variable.

$$index = \sum_{k=0}^{RANK-1} [(s_k - 1) \prod_{l=0}^{k} D_l]$$

Parameters:

rank Rank of the variable

dims dimensions of the variable

subs Dimension subscripts

Returns:

Single (linear) subscript

1.1.2.2 int∗ Mat_CalcSubscripts (int *rank*, int ∗ *dims*, int *index*)

Calculates 1-relative subscripts for each dimension given a 0-relative linear index. Subscripts are calculated as follows where s is the array of dimension subscripts, D is the array of dimensions, and index is the linear index.

$$s_k = \lfloor \frac{1}{L} \prod_{l=0}^{k} D_l \rfloor + 1$$

$$L = index - \sum_{l=k}^{RANK-1} s_k \prod_{m=0}^{k} D_m$$

Parameters:

rank Rank of the variable

dims dimensions of the variable

index linear index

Returns:

Array of dimension subscripts

1.1.2.3 int Mat_Close (mat_t ∗ *mat*)

Closes the given Matlab MAT file and frees any memory with it.

Parameters:

mat Pointer to the MAT file

Return values:

0

1.1.2.4 mat_t∗ Mat_Create (const char ∗ *matname*, const char ∗ *hdr_str*)

Tries to create a new Matlab MAT file with the given name and optional header string. If no header string is given, the default string is used containing the software, version, and date in it. If a header string is given, at most the first 116 characters is written to the file. The given header string need not be the full 116 characters, but MUST be NULL terminated.

Parameters:

matname Name of MAT file to create

hdr_str Optional header string, NULL to use default

Returns:

A pointer to the MAT file or NULL if it failed. This is not a simple FILE ∗ and should not be used as one.

1.1.2.5 mat_t∗ Mat_Open (const char ∗ *matname*, int *mode*)

Tries to open a Matlab MAT file with the given name

Parameters:

matname Name of MAT file to open

mode File access mode (MAT_ACC_RDONLY,MAT_ACC_RDWR,etc).

Returns:

A pointer to the MAT file or NULL if it failed. This is not a simple FILE ∗ and should not be used as one.

1.1.2.6 int Mat_Rewind (mat_t ∗ *mat*)

Rewinds a Matlab MAT file to the first variable

Parameters:

mat Pointer to the MAT file

Return values:

0 on success

1.1.2.7 size_t Mat_SizeOfClass (int *class_type*)

Returns the size (in bytes) of the matlab class class_type

Parameters:

class_type Matlab class type (MAT_C_∗)

Returns:

Size of the class

1.1.2.8 int Mat_VarAddStructField (matvar_t ∗ *matvar*, matvar_t ∗∗ *fields*)

Adds the given field to the structure. fields should be an array of matvar_t pointers of the same size as the structure (i.e. 1 field per structure element).

Parameters:

matvar Pointer to the Structure MAT variable

fields Array of fields to be added

Return values:

0 on success

1.1.2.9 matvar_t∗ Mat_VarCalloc (void)

Returns:

A newly allocated matvar_t

1.1.2.10 matvar_t∗ Mat_VarCreate (const char ∗ *name*, int *class_type*, int *data_type*, int *rank*, int ∗ *dims*, void ∗ *data*, int *opt*)

Creates a MAT variable that can be written to a Matlab MAT file with the given name, data type, dimensions and data. Rank should always be 2 or more. i.e. Scalar values would have rank=2 and dims[2] = {1,1}. Data type is one of the MAT_T types. MAT adds MAT_T_STRUCT and MAT_T_CELL to create Structures and Cell Arrays respectively. For MAT_T_STRUCT, data should be a NULL terminated array of matvar_t ∗ variables (i.e. for a 3x2 structure with 10 fields, there should be 61 matvar_t ∗ variables where the last one is NULL). For cell arrays, the NULL termination isn't necessary. So to create a cell array of size 3x2, data would be the address of an array of 6 matvar_t ∗ variables.

EXAMPLE: To create a struct of size 3x2 with 3 fields:

```
int rank=2, dims[2] = {3,2}, nfields = 3;
matvar_t **vars;

vars = malloc((3*2*nfields+1)*sizeof(matvar_t *));
vars[0]             = Mat_VarCreate(...);
   :
vars[3*2*nfields-1] = Mat_VarCreate(...);
vars[3*2*nfields]   = NULL;
```

EXAMPLE: To create a cell array of size 3x2:

```
int rank=2, dims[2] = {3,2};
matvar_t **vars;

vars = malloc(3*2*sizeof(matvar_t *));
vars[0]             = Mat_VarCreate(...);
   :
vars[5] = Mat_VarCreate(...);
```

Parameters:

name Name of the variable to create

class_type class type of the variable in Matlab(one of the mx Classes)

data_type data type of the variable (one of the MAT_T_ Types)

rank Rank of the variable

dims array of dimensions of the variable of size rank

data pointer to the data

opt 0, or bitwise or of the following options:

- MEM_CONSERVE to just use the pointer to the data and not copy the data itself. Note that the pointer should not be freed until you are done with the mat variable. The Mat_VarFree function will NOT free data that was created with MEM_CONSERVE, so free it yourself.
- MAT_F_COMPLEX to specify that the data is complex. The data variable should be a contigouse piece of memory with the real part written first and the imaginary second
- MAT_F_GLOBAL to assign the variable as a global variable
- MAT_F_LOGICAL to specify that it is a logical variable

Returns:

A MAT variable that can be written to a file or otherwise used

1.1.2.11 int Mat_VarDelete (mat_t ∗ *mat*, char ∗ *name*)

Parameters:

mat Pointer to the mat_t file structure

name Name of the variable to delete

Returns:

0 on success

1.1.2.12 matvar_t∗ Mat_VarDuplicate (const matvar_t ∗ *in*, int *opt*)

Provides a clean function for duplicating a matvar_t structure.

Parameters:

in pointer to the matvar_t structure to be duplicated

opt 0 does a shallow duplicate and only assigns the data pointer to the duplicated array. 1 will do a deep duplicate and actually duplicate the contents of the data. Warning: If you do a shallow copy and free both structures, the data will be freed twice and memory will be corrupted. This may be fixed in a later release.

Returns:

Pointer to the duplicated matvar_t structure.

1.1.2.13 void Mat_VarFree (matvar_t ∗ *matvar*)

Frees memory used by a MAT variable. Frees the data associated with a MAT variable if it's non-NULL and MEM_CONSERVE was not used.

Parameters:

matvar Pointer to the matvar_t structure

1.1.2.14 matvar_t* Mat_VarGetCell (matvar_t * *matvar*, int *index*)

Returns a pointer to the Cell Array Field at the given 1-relative index. MAT file must be a version 5 matlab file.

Parameters:

matvar Pointer to the Cell Array MAT variable

index linear index of cell to return

Returns:

Pointer to the Cell Array Field on success, NULL on error

1.1.2.15 matvar_t** Mat_VarGetCells (matvar_t * *matvar*, int * *start*, int * *stride*, int * *edge*)

Finds cells of a cell array given a start, stride, and edge for each. dimension. The cells are placed in a pointer array. The cells should not be freed, but the array of pointers should be. If copies are needed, use Mat_VarDuplicate on each cell. MAT File version must be 5.

Parameters:

matvar Cell Array matlab variable

start vector of length rank with 0-relative starting coordinates for each diemnsion.

stride vector of length rank with strides for each diemnsion.

edge vector of length rank with the number of elements to read in each diemnsion.

Returns:

an array of pointers to the cells

1.1.2.16 matvar_t** Mat_VarGetCellsLinear (matvar_t * *matvar*, int *start*, int *stride*, int *edge*)

Finds cells of a cell array given a linear indexed start, stride, and edge. The cells are placed in a pointer array. The cells themself should not be freed as they are part of the original cell array, but the pointer array should be. If copies are needed, use Mat_VarDuplicate on each of the cells. MAT file version must be 5.

Parameters:

matvar Cell Array matlab variable

start starting index

stride stride

edge Number of cells to get

Returns:

an array of pointers to the cells

1.1.2.17 int Mat_VarGetNumberOfFields (matvar_t ∗ *matvar*)

Returns the number of fields in the given structure. MAT file version must be 5.

Parameters:

matvar Structure matlab variable

Returns:

Number of fields, or a negative number on error

1.1.2.18 size_t Mat_VarGetSize (matvar_t ∗ *matvar*)

Parameters:

matvar matlab variable

Returns:

size of the variable in bytes

1.1.2.19 matvar_t∗ Mat_VarGetStructField (matvar_t ∗ *matvar*, void ∗ *name_or_index*, int *opt*, int *index*)

Returns a pointer to the structure field at the given 0-relative index. MAT file version must be 5.

Parameters:

matvar Pointer to the Structure MAT variable

name_or_index Name of the field, or the 1-relative index of the field. If the index is used, it should be the address of an integer variable whose value is the index number.

opt BY_NAME if the name_or_index is the name or BY_INDEX if the index was passed.

index linear index of the structure to find the field of

Returns:

Pointer to the Structure Field on success, NULL on error

1.1.2.20 matvar_t∗ Mat_VarGetStructs (matvar_t ∗ *matvar*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*, int *copy_fields*)

Finds structures of a structure array given a start, stride, and edge for each dimension. The structures are placed in a new structure array. If copy_fields is non-zero, the indexed structures are copied and should be freed, but if copy_fields is zero, the indexed structures are pointers to the original, but should still be freed since the mem_conserve flag is set so that the structures are not freed. MAT File version must be 5.

Parameters:

matvar Structure matlab variable

start vector of length rank with 0-relative starting coordinates for each diemnsion.

stride vector of length rank with strides for each diemnsion.

edge vector of length rank with the number of elements to read in each diemnsion.

copy_fields 1 to copy the fields, 0 to just set pointers to them. If 0 is used, the fields should not be freed themselves.

Returns:

A new structure with fields indexed from matvar.

1.1.2.21 matvar_t∗ Mat_VarGetStructsLinear (matvar_t ∗ *matvar*, int *start*, int *stride*, int *edge*, int *copy_fields*)

Finds structures of a structure array given a single (linear)start, stride, and edge. The structures are placed in a new structure array. If copy_fields is non-zero, the indexed structures are copied and should be freed, but if copy_fields is zero, the indexed structures are pointers to the original, but should still be freed since the mem_conserve flag is set so that the structures are not freed. MAT File version must be 5.

Parameters:

matvar Structure matlab variable

start starting index

stride stride

edge Number of structures to get

copy_fields 1 to copy the fields, 0 to just set pointers to them. If 0 is used, the fields should not be freed themselves.

Returns:

A new structure with fields indexed from matvar

1.1.2.22 void Mat_VarPrint (matvar_t ∗ *matvar*, int *printdata*)

Prints to stdout the values of the matvar_t structure

Parameters:

matvar Pointer to the matvar_t structure

printdata set to 1 if the Variables data should be printed, else 0

1.1.2.23 matvar_t∗ Mat_VarRead (mat_t ∗ *mat*, char ∗ *name*)

Reads the next variable in the Matlab MAT file

Parameters:

mat Pointer to the MAT file

name Name of the variable to read

Returns:

Pointer to the matvar_t structure containing the MAT variable information

1.1.2.24 int Mat_VarReadData (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *data*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Reads data from a MAT variable. The variable must have been read by Mat_VarReadInfo.

Parameters:

mat MAT file to read data from

matvar MAT variable information

data pointer to store data in (must be pre-allocated)

start array of starting indeces

stride stride of data

edge array specifying the number to read in each direction

Return values:

0 on success

1.1.2.25 int Mat_VarReadDataAll (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Allocates memory for an reads the data for a given matlab variable.

Parameters:

mat Matlab MAT file structure pointer

matvar Variable whose data is to be read

Returns:

non-zero on error

1.1.2.26 int Mat_VarReadDataLinear (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *data*, int *start*, int *stride*, int *edge*)

Reads data from a MAT variable using a linear indexingmode. The variable must have been read by Mat_VarReadInfo.

Parameters:

mat MAT file to read data from

matvar MAT variable information

data pointer to store data in (must be pre-allocated)

start starting index

stride stride of data

edge number of elements to read

Return values:

0 on success

1.1.2.27 matvar_t* Mat_VarReadInfo (mat_t * *mat*, char * *name*)

Reads the named variable (or the next variable if name is NULL) information (class,flags-complex/global/logical,rank,dimensions,and name) from the Matlab MAT file

Parameters:

mat Pointer to the MAT file

name Name of the variable to read

Returns:

Pointer to the matvar_t structure containing the MAT variable information

1.1.2.28 matvar_t* Mat_VarReadNext (mat_t * *mat*)

Reads the next variable in the Matlab MAT file

Parameters:

mat Pointer to the MAT file

Returns:

Pointer to the matvar_t structure containing the MAT variable information

1.1.2.29 matvar_t* Mat_VarReadNextInfo (mat_t * *mat*)

Reads the next variable's information (class,flags-complex/global/logical, rank,dimensions, name, etc) from the Matlab MAT file. After reading, the MAT file is positioned past the current variable.

Parameters:

mat Pointer to the MAT file

Returns:

Pointer to the matvar_t structure containing the MAT variable information

1.1.2.30 int Mat_VarWrite (mat_t * *mat*, matvar_t * *matvar*, int *compress*)

Writes the MAT variable information stored in matvar to the given MAT file. The variable will be written to the end of the file.

Parameters:

mat MAT file to write to

matvar MAT variable information to write

compress Whether or not to compress the data (Only valid for version 5 MAT files and variables with numeric data)

Return values:

0 on success

1.1.2.31 int Mat_VarWriteData (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *data*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Writes data to a MAT variable. The variable must have previously been written with Mat_VarWriteInfo.

Parameters:

mat MAT file to write to

matvar MAT variable information to write

data pointer to the data to write

start array of starting indeces

stride stride of data

edge array specifying the number to read in each direction

Return values:

0 on success

1.1.2.32 int Mat_VarWriteInfo (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Writes the MAT variable information stored in matvar to the given MAT file. The variable will be written to the end of the file.

Parameters:

mat MAT file to write to

matvar MAT variable information to write

Return values:

0 on success

1.2 Internal Functions

Defines

- #define swap(a, b) a^=b;b^=a;a^=b

  swap the bytes `a` *and* `b`

Functions

- double doubleSwap (double ∗a)

  swap the bytes of a 4 or 8 byte double-precision float

- float floatSwap (float ∗a)

  swap the bytes of a 4 byte single-precision float

- int InflateArrayFlags (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the Array Flags Tag and the Array Flags data.

- int InflateData (mat_t ∗mat, z_stream ∗z, void ∗buf, int nBytes)

  Inflates the data.

- int InflateDataTag (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the data's tag.

- int InflateDataType (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the data's type.

- int InflateDimensions (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the dimensions tag and the dimensions data.

- int InflateFieldNameLength (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the structure's fieldname length.

- int InflateFieldNames (mat_t ∗mat, matvar_t ∗matvar, void ∗buf, int nfields, int fieldname_length, int padding)

  Inflates the structure's fieldnames.

- int InflateFieldNamesTag (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the structure's fieldname tag.

- int InflateSkip (mat_t ∗mat, z_stream ∗z, int nbytes)

  Inflate the data until `nbytes` *of uncompressed data has been inflated.*

- int InflateSkip2 (mat_t ∗mat, matvar_t ∗matvar, int nbytes)

  Inflate the data until `nbytes` *of compressed data has been inflated.*

- int InflateSkipData (mat_t ∗mat, z_stream ∗z, int data_type, int len)

  Inflate the data until `len` *elements of compressed data with data type* `data_type` *has been inflated.*

- int InflateVarName (mat_t ∗mat, matvar_t ∗matvar, void ∗buf, int N)

  Inflates the variable name.

- int InflateVarNameTag (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the variable name tag.

- int InflateVarTag (mat_t ∗mat, matvar_t ∗matvar, void ∗buf)

  Inflates the variable's tag.

- mat_int16_t int16Swap (mat_int16_t ∗a)

  swap the bytes of a 16-bit signed integer

- mat_int32_t int32Swap (mat_int32_t ∗a)

  swap the bytes of a 32-bit signed integer

- void Mat_VarPrint5 (matvar_t ∗matvar, int printdata)

  Prints the mat variable.

- matvar_t ∗ Mat_VarReadNextInfo5 (mat_t ∗mat)

  Reads the header information for the next MAT variable.

- void Read5 (mat_t ∗mat, matvar_t ∗matvar)

  Reads the data of a version 5 MAT variable.

- int ReadData5 (mat_t ∗mat, matvar_t ∗matvar, void ∗data, int ∗start, int ∗stride, int ∗edge)

  Reads a slab of data from the mat variable `matvar`*.*

- int ReadDataSlab2 (mat_t ∗mat, void ∗data, int class_type, int data_type, int ∗dims, int ∗start, int ∗stride, int ∗edge)

  Reads data of type `data_type` *by user-defined dimensions for 2-D data.*

- int ReadDataSlabN (mat_t ∗mat, void ∗data, int class_type, int data_type, int rank, int ∗dims, int ∗start, int ∗stride, int ∗edge)

  Reads data of type `data_type` *by user-defined dimensions.*

- int ReadDoubleData (mat_t ∗mat, double ∗data, int data_type, int len)

  Reads data of type `data_type` *into a double type.*

- int ReadInt16Data (mat_t ∗mat, mat_int16_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into a signed 16-bit integer type.*

- int ReadInt32Data (mat_t ∗mat, mat_int32_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into a signed 32-bit integer type.*

- int ReadInt8Data (mat_t ∗mat, mat_int8_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into a signed 8-bit integer type.*

- int ReadNextCell (mat_t ∗mat, matvar_t ∗matvar)

  Reads the next cell of the cell array in `matvar`*.*

- int ReadNextFunctionHandle (mat_t ∗mat, matvar_t ∗matvar)

  Reads the function handle data of the function handle in `matvar`*.*

- int ReadNextStructField (mat_t ∗mat, matvar_t ∗matvar)

  Reads the next struct field of the structure in `matvar`*.*

- int ReadSingleData (mat_t ∗mat, float ∗data, int data_type, int len)

  Reads data of type `data_type` *into a float type.*

- int ReadUInt16Data (mat_t ∗mat, mat_uint16_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into an unsigned 16-bit integer type.*

- int ReadUInt32Data (mat_t ∗mat, mat_uint32_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into an unsigned 32-bit integer type.*

- int ReadUInt8Data (mat_t ∗mat, mat_uint8_t ∗data, int data_type, int len)

  Reads data of type `data_type` *into an unsigned 8-bit integer type.*

- mat_uint16_t uint16Swap (mat_uint16_t ∗a)

  swap the bytes of a 16-bit unsigned integer

- mat_uint32_t uint32Swap (mat_uint32_t ∗a)

  swap the bytes of a 32-bit unsigned integer

- int Write5 (mat_t ∗mat, matvar_t ∗matvar, int compress)

  Writes a matlab variable to a version 5 matlab file.

- int WriteCellArrayField (mat_t ∗mat, matvar_t ∗matvar, int compress)

  Writes the header and data for an element of a cell array.

- int WriteCellArrayFieldInfo (mat_t ∗mat, matvar_t ∗matvar, int compress)

  Writes the header and blank data for a cell array.

- int WriteCharData (mat_t ∗mat, void ∗data, int N, int data_type)

  Writes `data` *as character data.*

- int WriteCharDataSlab2 (mat_t ∗mat, void ∗data, int data_type, int ∗dims, int ∗start, int ∗stride, int ∗edge)
- int WriteDataSlab2 (mat_t ∗mat, void ∗data, int data_type, int ∗dims, int ∗start, int ∗stride, int ∗edge)
- int WriteEmptyCharData (mat_t ∗mat, int N, int data_type)

  Writes empty characters to the MAT file.

- void WriteInfo5 (mat_t ∗mat, matvar_t ∗matvar)

  Writes the variable information and empty data.

- int WriteStructField (mat_t ∗mat, matvar_t ∗matvar)

  Writes the header and data for a field of a struct array.

1.2.1 Function Documentation

1.2.1.1 double doubleSwap (double ∗ *a*)

Parameters:

a pointer to integer to swap

Returns:

the swapped integer

1.2.1.2 float floatSwap (float ∗ *a*)

Parameters:

a pointer to integer to swap

Returns:

the swapped integer

1.2.1.3 int InflateArrayFlags (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

`buf` must hold at least 16 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the 16-byte array flags tag and data

Returns:

Number of bytes read from the file

1.2.1.4 int InflateData (mat_t ∗ *mat*, z_stream ∗ *z*, void ∗ *buf*, int *nBytes*)

buf must hold at least `nBytes` bytes

Parameters:

mat Pointer to the MAT file

z zlib compression stream

buf Pointer to store the data type

nBytes Number of bytes to inflate

Returns:

Number of bytes read from the file

1.2.1.5 int InflateDataTag (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

buf must hold at least 8 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the data tag

Returns:

Number of bytes read from the file

1.2.1.6 int InflateDataType (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

buf must hold at least 4 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the data type

Returns:

Number of bytes read from the file

1.2.1.7 int InflateDimensions (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

`buf` must hold at least (8+4∗rank) bytes where rank is the number of dimensions. If the end of the dimensions data is not aligned on an 8-byte boundary, this function eats up those bytes and stores then in `buf`.

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the dimensions flag and data

Returns:

Number of bytes read from the file

1.2.1.8 int InflateFieldNameLength (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

buf must hold at least 8 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the fieldname length

Returns:

Number of bytes read from the file

1.2.1.9 int InflateFieldNames (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*, int *nfields*, int *fieldname_length*, int *padding*)

buf must hold at least `nfields` ∗ `fieldname_length` bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the fieldnames

nfields Number of fields

fieldname_length Maximum length in bytes of each field

padding Number of padding bytes

Returns:

Number of bytes read from the file

1.2.1.10 int InflateFieldNamesTag (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

buf must hold at least 8 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the fieldname tag

Returns:

Number of bytes read from the file

1.2.1.11 int InflateSkip (mat_t ∗ *mat*, z_stream ∗ *z*, int *nbytes*)

Parameters:

mat Pointer to the MAT file

z zlib compression stream

nbytes Number of uncompressed bytes to skip

Returns:

Number of bytes read from the file

1.2.1.12 int InflateSkip2 (mat_t ∗ *mat*, matvar_t ∗ *matvar*, int *nbytes*)

Parameters:

mat Pointer to the MAT file

z zlib compression stream

nbytes Number of uncompressed bytes to skip

Returns:

Number of bytes read from the file

1.2.1.13 int InflateSkipData (mat_t ∗ *mat*, z_stream ∗ *z*, int *data_type*, int *len*)

Parameters:

mat Pointer to the MAT file

z zlib compression stream

data_type Data type (matio_types enumerations)

len Number of elements of datatype `data_type` to skip

Returns:

Number of bytes read from the file

1.2.1.14 int InflateVarName (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*, int *N*)

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the variables name

N Number of characters in the name

Returns:

Number of bytes read from the file

1.2.1.15 int InflateVarNameTag (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *buf*)

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the variables name tag

Returns:

Number of bytes read from the file

1.2.1.16 int InflateVarTag (mat_t * *mat*, matvar_t * *matvar*, void * *buf*)

`buf` must hold at least 8 bytes

Parameters:

mat Pointer to the MAT file

matvar Pointer to the MAT variable

buf Pointer to store the 8-byte variable tag

Returns:

Number of bytes read from the file

1.2.1.17 mat_int16_t int16Swap (mat_int16_t * *a*)

Parameters:

a pointer to integer to swap

Returns:

the swapped integer

1.2.1.18 mat_int32_t int32Swap (mat_int32_t * *a*)

Parameters:

a pointer to integer to swap

Returns:

the swapped integer

1.2.1.19 void Mat_VarPrint5 (matvar_t * *matvar*, int *printdata*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

1.2.1.20 matvar_t* Mat_VarReadNextInfo5 (mat_t * *mat*)

Parameters:

mat MAT file pointer pointer to the MAT variable or NULL

1.2.1.21 void Read5 (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Parameters:

mat MAT file pointer

matvar MAT variable pointer to read the data

1.2.1.22 int ReadData5 (mat_t ∗ *mat*, matvar_t ∗ *matvar*, void ∗ *data*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

data pointer to store the read data in (must be of size edge[0]∗...edge[rank-1]∗Mat_SizeOfClass(matvar->class_type))

start index to start reading data in each dimension

stride write data every `stride` elements in each dimension

edge number of elements to read in each dimension

Return values:

0 on success

1.2.1.23 int ReadDataSlab2 (mat_t ∗ *mat*, void ∗ *data*, int *class_type*, int *data_type*, int ∗ *dims*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Parameters:

mat MAT file pointer

data Pointer to store the output data

class_type Type of data class (matio_classes enumerations)

data_type Datatype of the stored data (matio_types enumerations)

dims Dimensions of the data

start Index to start reading data in each dimension

stride Read every `stride` elements in each dimension

edge Number of elements to read in each dimension

Return values:

Number of bytes read from the file, or -1 on error

1.2.1.24 int ReadDataSlabN (mat_t ∗ *mat*, void ∗ *data*, int *class_type*, int *data_type*, int *rank*, int ∗ *dims*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Parameters:

mat MAT file pointer

data Pointer to store the output data

class_type Type of data class (matio_classes enumerations)

data_type Datatype of the stored data (matio_types enumerations)

rank Number of dimensions in the data

dims Dimensions of the data

start Index to start reading data in each dimension

stride Read every `stride` elements in each dimension

edge Number of elements to read in each dimension

Return values:

Number of bytes read from the file, or -1 on error

1.2.1.25 int ReadDoubleData (mat_t ∗ *mat*, double ∗ *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as double's in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output double values (len∗sizeof(double))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.26 int ReadInt16Data (mat_t ∗ *mat*, mat_int16_t ∗ *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as signed 16-bit integers in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output signed 16-bit integer values (len∗sizeof(mat_int16_t))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.27 int ReadInt32Data (mat_t ∗ *mat*, mat_int32_t ∗ *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as signed 32-bit integers in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output signed 32-bit integer values (len∗sizeof(mat_int32_t))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.28 int ReadInt8Data (mat_t ∗ *mat*, mat_int8_t ∗ *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as signed 8-bit integers in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output signed 8-bit integer values (len∗sizeof(mat_int8_t))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.29 int ReadNextCell (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Parameters:

mat MAT file pointer

matvar MAT variable pointer

Returns:

Number of bytes read

1.2.1.30 int ReadNextFunctionHandle (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Parameters:

mat MAT file pointer

matvar MAT variable pointer

Returns:

Number of bytes read

1.2.1.31 int ReadNextStructField (mat_t * *mat*, matvar_t * *matvar*)

Reads the next struct fields (fieldname length,names,data headers for all the fields

Parameters:

mat MAT file pointer

matvar MAT variable pointer

Returns:

Number of bytes read

1.2.1.32 int ReadSingleData (mat_t * *mat*, float * *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as float's in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output float values (len*sizeof(float))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.33 int ReadUInt16Data (mat_t * *mat*, mat_uint16_t * *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as unsigned 16-bit integers in `data`.

Parameters:

mat MAT file pointer

data Pointer to store the output unsigned 16-bit integer values (len*sizeof(mat_uint16_t))

data_type one of the `matio_types` enumerations which is the source data type in the file

len Number of elements of type `data_type` to read from the file

Return values:

Number of bytes read from the file

1.2.1.34 int ReadUInt32Data (mat_t * *mat*, mat_uint32_t * *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as unsigned 32-bit integers in `data`.

Parameters:

- ***mat*** MAT file pointer
- ***data*** Pointer to store the output unsigned 32-bit integer values (len∗sizeof(mat_uint32_t))
- ***data_type*** one of the `matio_types` enumerations which is the source data type in the file
- ***len*** Number of elements of type `data_type` to read from the file

Return values:

- ***Number*** of bytes read from the file

1.2.1.35 int ReadUInt8Data (mat_t ∗ *mat*, mat_uint8_t ∗ *data*, int *data_type*, int *len*)

Reads from the MAT file `len` elements of data type `data_type` storing them as unsigned 8-bit integers in `data`.

Parameters:

- ***mat*** MAT file pointer
- ***data*** Pointer to store the output unsigned 8-bit integer values (len∗sizeof(mat_uint8_t))
- ***data_type*** one of the `matio_types` enumerations which is the source data type in the file
- ***len*** Number of elements of type `data_type` to read from the file

Return values:

- ***Number*** of bytes read from the file

1.2.1.36 mat_uint16_t uint16Swap (mat_uint16_t ∗ *a*)

Parameters:

- ***a*** pointer to integer to swap

Returns:

the swapped integer

1.2.1.37 mat_uint32_t uint32Swap (mat_uint32_t ∗ *a*)

Parameters:

- ***a*** pointer to integer to swap

Returns:

the swapped integer

1.2.1.38 int Write5 (mat_t * *mat*, matvar_t * *matvar*, int *compress*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

compress option to compress the variable (only works for numeric types)

Return values:

0 on success

1.2.1.39 int WriteCellArrayField (mat_t * *mat*, matvar_t * *matvar*, int *compress*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

compress option to write the data compressed (not used)

Return values:

0 on success

1.2.1.40 int WriteCellArrayFieldInfo (mat_t * *mat*, matvar_t * *matvar*, int *compress*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

compress option to write the data compressed (not used)

Returns:

number of bytes written

1.2.1.41 int WriteCharData (mat_t * *mat*, void * *data*, int *N*, int *data_type*)

This function uses the knowledge that the data is part of a character class to avoid some pitfalls with Matlab listed below.

- Matlab character data cannot be unsigned 8-bit integers, it needs at least unsigned 16-bit integers

Parameters:

mat MAT file pointer

data character data to write

N Number of elements to write

data_type character data type (enum matio_types)

Returns:

number of bytes written

1.2.1.42 int WriteCharDataSlab2 (mat_t ∗ *mat*, void ∗ *data*, int *data_type*, int ∗ *dims*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Parameters:

Writes a 2-D slab of character data to the MAT file

This function uses the knowledge that the data is part of a character class to avoid some pitfalls with Matlab listed below.

- Matlab character data cannot be unsigned 8-bit integers, it needs at least unsigned 16-bit integers

should return the number of bytes written, but currently returns 0

Parameters:

mat MAT file pointer

data pointer to the slab of data

data_type data type of the data (enum matio_types)

dims dimensions of the dataset

start index to start writing the data in each dimension

stride write data every `stride` elements

edge number of elements to write in each dimension

Returns:

number of byteswritten

1.2.1.43 int WriteDataSlab2 (mat_t ∗ *mat*, void ∗ *data*, int *data_type*, int ∗ *dims*, int ∗ *start*, int ∗ *stride*, int ∗ *edge*)

Parameters:

Writes a 2-D slab of data to the MAT file

should return the number of bytes written, but currently returns 0

Parameters:

mat MAT file pointer

data pointer to the slab of data

data_type data type of the data (enum matio_types)

dims dimensions of the dataset

start index to start writing the data in each dimension

stride write data every `stride` elements

edge number of elements to write in each dimension

Returns:

number of byteswritten

1.2.1.44 int WriteEmptyCharData (mat_t ∗ *mat*, int *N*, int *data_type*)

This function uses the knowledge that the data is part of a character class to avoid some pitfalls with Matlab listed below.

- Matlab character data cannot be unsigned 8-bit integers, it needs at least unsigned 16-bit integers

Parameters:

mat MAT file pointer

data character data to write

N Number of elements to write

data_type character data type (enum matio_types)

Returns:

number of bytes written

1.2.1.45 void WriteInfo5 (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

1.2.1.46 int WriteStructField (mat_t ∗ *mat*, matvar_t ∗ *matvar*)

Parameters:

mat MAT file pointer

matvar pointer to the mat variable

Return values:

0 on success

Chapter 2

LIBMATIO API Data Structure Documentation

2.1 ComplexSplit Struct Reference

Complex data type using split storage.

Data Fields

- void ∗ Im
- void ∗ Re

2.1.1 Detailed Description

Complex data type using split real/imaginary pointers

2.1.2 Field Documentation

2.1.2.1 void∗ ComplexSplit::Im

Pointer to the imaginary part

2.1.2.2 void∗ ComplexSplit::Re

Pointer to the real part

2.2 mat_t Struct Reference

Matlab MAT File information.

Data Fields

- long bof
- int byteswap
- char ∗ filename
- FILE ∗ fp
- char ∗ header
- int mode
- char ∗ subsys_offset
- int version

2.2.1 Detailed Description

Contains information about a Matlab MAT file

2.2.2 Field Documentation

2.2.2.1 long mat_t::bof

Beginning of file not including header

2.2.2.2 int mat_t::byteswap

1 if byte swapping is required, 0 else

2.2.2.3 char∗ mat_t::filename

Name of the file that fp points to

2.2.2.4 FILE∗ mat_t::fp

Pointer to the MAT file

2.2.2.5 char∗ mat_t::header

MAT File header string

2.2.2.6 int mat_t::mode

Access mode

2.2.2.7 char∗ mat_t::subsys_offset

offset

2.2.2.8 int mat_t::version

MAT File version

2.3 matvar_t Struct Reference

Matlab variable information.

Data Fields

- int class_type
- int compression
- void ∗ data
- int data_size
- int data_type
- long datapos
- int ∗ dims
- mat_t ∗ fp
- long fpos
- int isComplex
- int isGlobal
- int isLogical
- int mem_conserve
- char ∗ name
- int nbytes
- int rank

2.3.1 Detailed Description

Contains information about a Matlab variable

2.3.2 Field Documentation

2.3.2.1 int matvar_t::class_type

Class type in Matlab(mxDOUBLE_CLASS, etc)

2.3.2.2 int matvar_t::compression

Compression (0=>None,1=>ZLIB)

2.3.2.3 void∗ matvar_t::data

Pointer to the data

2.3.2.4 int matvar_t::data_size

Bytes / element for the data

2.3.2.5 int matvar_t::data_type

Data type(MAT_T_∗)

2.3.2.6 long matvar_t::datapos

Offset from the beginning of the MAT file to the data

2.3.2.7 int* matvar_t::dims

Array of lengths for each dimension

2.3.2.8 mat_t* matvar_t::fp

Pointer to the MAT file structure (mat_t)

2.3.2.9 long matvar_t::fpos

Offset from the beginning of the MAT file to the variable

2.3.2.10 int matvar_t::isComplex

non-zero if the data is complex, 0 if real

2.3.2.11 int matvar_t::isGlobal

non-zero if the variable is global

2.3.2.12 int matvar_t::isLogical

non-zero if the variable is logical

2.3.2.13 int matvar_t::mem_conserve

1 if Memory was conserved with data

2.3.2.14 char* matvar_t::name

Name of the variable

2.3.2.15 int matvar_t::nbytes

Number of bytes for the MAT variable

2.3.2.16 int matvar_t::rank

Rank (Number of dimensions) of the data

2.4 sparse_t Struct Reference

sparse data information

Data Fields

- void ∗ data
- int ∗ ir
- int ∗ jc
- int ndata
- int nir
- int njc
- int nzmax

2.4.1 Detailed Description

Contains information and data for a sparse matrix

2.4.2 Field Documentation

2.4.2.1 void∗ sparse_t::data

Array of data elements

2.4.2.2 int∗ sparse_t::ir

Array of size nzmax where ir[k] is the row of data[k]. 0 <= k <= nzmax

2.4.2.3 int∗ sparse_t::jc

Array size N+1 (N is number of columsn) with jc[k] being the index into ir/data of the first non-zero element for row k.

2.4.2.4 int sparse_t::ndata

Number of complex/real data values

2.4.2.5 int sparse_t::nir

number of elements in ir

2.4.2.6 int sparse_t::njc

Number of elements in jc

2.4.2.7 int sparse_t::nzmax

Maximum number of non-zero elements

Index

bof
  mat_t, 36
BY_INDEX
  MAT, 6
BY_NAME
  MAT, 6
byteswap
  mat_t, 36

class_type
  matvar_t, 38
ComplexSplit, 35
ComplexSplit
  Im, 35
  Re, 35
compression
  matvar_t, 38
COMPRESSION_NONE
  MAT, 7
COMPRESSION_ZLIB
  MAT, 7

data
  matvar_t, 38
  sparse_t, 40
data_size
  matvar_t, 38
data_type
  matvar_t, 38
datapos
  matvar_t, 38
dims
  matvar_t, 39
doubleSwap
  mat_internal, 22

filename
  mat_t, 36
floatSwap
  mat_internal, 22
fp
  mat_t, 36
  matvar_t, 39
fpos
  matvar_t, 39

header
  mat_t, 36

Im
  ComplexSplit, 35
InflateArrayFlags
  mat_internal, 22
InflateData
  mat_internal, 22
InflateDataTag
  mat_internal, 22
InflateDataType
  mat_internal, 23
InflateDimensions
  mat_internal, 23
InflateFieldNameLength
  mat_internal, 23
InflateFieldNames
  mat_internal, 24
InflateFieldNamesTag
  mat_internal, 24
InflateSkip
  mat_internal, 24
InflateSkip2
  mat_internal, 24
InflateSkipData
  mat_internal, 25
InflateVarName
  mat_internal, 25
InflateVarNameTag
  mat_internal, 25
InflateVarTag
  mat_internal, 25
int16Swap
  mat_internal, 26
int32Swap
  mat_internal, 26
Internal Functions, 19
ir
  sparse_t, 40
isComplex
  matvar_t, 39
isGlobal
  matvar_t, 39
isLogical

matvar_t, 39

jc
sparse_t, 40

MAT
BY_INDEX, 6
BY_NAME, 6
COMPRESSION_NONE, 7
COMPRESSION_ZLIB, 7
mat_acc, 6
MAT_ACC_RDONLY, 6
MAT_ACC_RDWR, 6
MAT_C_CELL, 7
MAT_C_CHAR, 7
MAT_C_DOUBLE, 7
MAT_C_FUNCTION, 7
MAT_C_INT16, 7
MAT_C_INT32, 7
MAT_C_INT64, 7
MAT_C_INT8, 7
MAT_C_OBJECT, 7
MAT_C_SINGLE, 7
MAT_C_SPARSE, 7
MAT_C_STRUCT, 7
MAT_C_UINT16, 7
MAT_C_UINT32, 7
MAT_C_UINT64, 7
MAT_C_UINT8, 7
Mat_CalcSingleSubscript, 8
Mat_CalcSubscripts, 9
Mat_Close, 9
Mat_Create, 9
MAT_F_CLASS_T, 8
MAT_F_COMPLEX, 8
MAT_F_GLOBAL, 8
MAT_F_LOGICAL, 8
mat_ft, 6
MAT_FT_MAT4, 7
MAT_FT_MAT5, 7
Mat_Open, 10
Mat_Rewind, 10
Mat_SizeOfClass, 10
MAT_T_ARRAY, 8
MAT_T_CELL, 8
MAT_T_COMPRESSED, 8
MAT_T_DOUBLE, 8
MAT_T_FUNCTION, 8
MAT_T_INT16, 8
MAT_T_INT32, 8
MAT_T_INT64, 8
MAT_T_INT8, 8
MAT_T_MATRIX, 8
MAT_T_SINGLE, 8
MAT_T_STRING, 8
MAT_T_STRUCT, 8
MAT_T_UINT16, 8
MAT_T_UINT32, 8
MAT_T_UINT64, 8
MAT_T_UINT8, 8
MAT_T_UNKNOWN, 8
MAT_T_UTF16, 8
MAT_T_UTF32, 8
MAT_T_UTF8, 8
Mat_VarAddStructField, 10
Mat_VarCalloc, 11
Mat_VarCreate, 11
Mat_VarDelete, 12
Mat_VarDuplicate, 12
Mat_VarFree, 12
Mat_VarGetCell, 12
Mat_VarGetCells, 13
Mat_VarGetCellsLinear, 13
Mat_VarGetNumberOfFields, 13
Mat_VarGetSize, 14
Mat_VarGetStructField, 14
Mat_VarGetStructs, 14
Mat_VarGetStructsLinear, 15
Mat_VarPrint, 15
Mat_VarRead, 15
Mat_VarReadData, 15
Mat_VarReadDataAll, 16
Mat_VarReadDataLinear, 16
Mat_VarReadInfo, 16
Mat_VarReadNext, 17
Mat_VarReadNextInfo, 17
Mat_VarWrite, 17
Mat_VarWriteData, 17
Mat_VarWriteInfo, 18
matio_classes, 7
matio_compression, 7
matio_flags, 7
matio_types, 8
mat_acc
MAT, 6
MAT_ACC_RDONLY
MAT, 6
MAT_ACC_RDWR
MAT, 6
MAT_C_CELL
MAT, 7
MAT_C_CHAR
MAT, 7
MAT_C_DOUBLE
MAT, 7
MAT_C_FUNCTION
MAT, 7
MAT_C_INT16

MAT, 7
MAT_C_INT32
MAT, 7
MAT_C_INT64
MAT, 7
MAT_C_INT8
MAT, 7
MAT_C_OBJECT
MAT, 7
MAT_C_SINGLE
MAT, 7
MAT_C_SPARSE
MAT, 7
MAT_C_STRUCT
MAT, 7
MAT_C_UINT16
MAT, 7
MAT_C_UINT32
MAT, 7
MAT_C_UINT64
MAT, 7
MAT_C_UINT8
MAT, 7
Mat_CalcSingleSubscript
MAT, 8
Mat_CalcSubscripts
MAT, 9
Mat_Close
MAT, 9
Mat_Create
MAT, 9
MAT_F_CLASS_T
MAT, 8
MAT_F_COMPLEX
MAT, 8
MAT_F_GLOBAL
MAT, 8
MAT_F_LOGICAL
MAT, 8
mat_ft
MAT, 6
MAT_FT_MAT4
MAT, 7
MAT_FT_MAT5
MAT, 7
mat_internal
doubleSwap, 22
floatSwap, 22
InflateArrayFlags, 22
InflateData, 22
InflateDataTag, 22
InflateDataType, 23
InflateDimensions, 23
InflateFieldNameLength, 23
InflateFieldNames, 24
InflateFieldNamesTag, 24
InflateSkip, 24
InflateSkip2, 24
InflateSkipData, 25
InflateVarName, 25
InflateVarNameTag, 25
InflateVarTag, 25
int16Swap, 26
int32Swap, 26
Mat_VarPrint5, 26
Mat_VarReadNextInfo5, 26
Read5, 26
ReadData5, 27
ReadDataSlab2, 27
ReadDataSlabN, 27
ReadDoubleData, 28
ReadInt16Data, 28
ReadInt32Data, 28
ReadInt8Data, 29
ReadNextCell, 29
ReadNextFunctionHandle, 29
ReadNextStructField, 29
ReadSingleData, 30
ReadUInt16Data, 30
ReadUInt32Data, 30
ReadUInt8Data, 31
uint16Swap, 31
uint32Swap, 31
Write5, 31
WriteCellArrayField, 32
WriteCellArrayFieldInfo, 32
WriteCharData, 32
WriteCharDataSlab2, 32
WriteDataSlab2, 33
WriteEmptyCharData, 33
WriteInfo5, 34
WriteStructField, 34
Mat_Open
MAT, 10
Mat_Rewind
MAT, 10
Mat_SizeOfClass
MAT, 10
mat_t, 36
bof, 36
byteswap, 36
filename, 36
fp, 36
header, 36
mode, 36
subsys_offset, 36
version, 37
MAT_T_ARRAY

MAT, 8
MAT_T_CELL
MAT, 8
MAT_T_COMPRESSED
MAT, 8
MAT_T_DOUBLE
MAT, 8
MAT_T_FUNCTION
MAT, 8
MAT_T_INT16
MAT, 8
MAT_T_INT32
MAT, 8
MAT_T_INT64
MAT, 8
MAT_T_INT8
MAT, 8
MAT_T_MATRIX
MAT, 8
MAT_T_SINGLE
MAT, 8
MAT_T_STRING
MAT, 8
MAT_T_STRUCT
MAT, 8
MAT_T_UINT16
MAT, 8
MAT_T_UINT32
MAT, 8
MAT_T_UINT64
MAT, 8
MAT_T_UINT8
MAT, 8
MAT_T_UNKNOWN
MAT, 8
MAT_T_UTF16
MAT, 8
MAT_T_UTF32
MAT, 8
MAT_T_UTF8
MAT, 8
Mat_VarAddStructField
MAT, 10
Mat_VarCalloc
MAT, 11
Mat_VarCreate
MAT, 11
Mat_VarDelete
MAT, 12
Mat_VarDuplicate
MAT, 12
Mat_VarFree
MAT, 12
Mat_VarGetCell
MAT, 12
Mat_VarGetCells
MAT, 13
Mat_VarGetCellsLinear
MAT, 13
Mat_VarGetNumberOfFields
MAT, 13
Mat_VarGetSize
MAT, 14
Mat_VarGetStructField
MAT, 14
Mat_VarGetStructs
MAT, 14
Mat_VarGetStructsLinear
MAT, 15
Mat_VarPrint
MAT, 15
Mat_VarPrint5
mat_internal, 26
Mat_VarRead
MAT, 15
Mat_VarReadData
MAT, 15
Mat_VarReadDataAll
MAT, 16
Mat_VarReadDataLinear
MAT, 16
Mat_VarReadInfo
MAT, 16
Mat_VarReadNext
MAT, 17
Mat_VarReadNextInfo
MAT, 17
Mat_VarReadNextInfo5
mat_internal, 26
Mat_VarWrite
MAT, 17
Mat_VarWriteData
MAT, 17
Mat_VarWriteInfo
MAT, 18
matio_classes
MAT, 7
matio_compression
MAT, 7
matio_flags
MAT, 7
matio_types
MAT, 8
Matlab MAT File I/O Library, 3
matvar_t, 38
class_type, 38
compression, 38
data, 38

data_size, 38
data_type, 38
datapos, 38
dims, 39
fp, 39
fpos, 39
isComplex, 39
isGlobal, 39
isLogical, 39
mem_conserve, 39
name, 39
nbytes, 39
rank, 39
mem_conserve
matvar_t, 39
mode
mat_t, 36

name
matvar_t, 39
nbytes
matvar_t, 39
ndata
sparse_t, 40
nir
sparse_t, 40
njc
sparse_t, 40
nzmax
sparse_t, 40

rank
matvar_t, 39
Re
ComplexSplit, 35
Read5
mat_internal, 26
ReadData5
mat_internal, 27
ReadDataSlab2
mat_internal, 27
ReadDataSlabN
mat_internal, 27
ReadDoubleData
mat_internal, 28
ReadInt16Data
mat_internal, 28
ReadInt32Data
mat_internal, 28
ReadInt8Data
mat_internal, 29
ReadNextCell
mat_internal, 29
ReadNextFunctionHandle
mat_internal, 29
ReadNextStructField
mat_internal, 29
ReadSingleData
mat_internal, 30
ReadUInt16Data
mat_internal, 30
ReadUInt32Data
mat_internal, 30
ReadUInt8Data
mat_internal, 31

sparse_t, 40
data, 40
ir, 40
jc, 40
ndata, 40
nir, 40
njc, 40
nzmax, 40
subsys_offset
mat_t, 36

uint16Swap
mat_internal, 31
uint32Swap
mat_internal, 31

version
mat_t, 37

Write5
mat_internal, 31
WriteCellArrayField
mat_internal, 32
WriteCellArrayFieldInfo
mat_internal, 32
WriteCharData
mat_internal, 32
WriteCharDataSlab2
mat_internal, 32
WriteDataSlab2
mat_internal, 33
WriteEmptyCharData
mat_internal, 33
WriteInfo5
mat_internal, 34
WriteStructField
mat_internal, 34