Version 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 1.0 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 8: MATLAB Version 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)
	2	VAX D-float
	3	VAX G-float
	4	Cray

Table 8: MATLAB Version 4 MAT-File Matrix Header Format

	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
	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.	

Table 8: MATLAB Version 4 MAT-File Matrix Header Format

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 9: MATLAB Version 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)
   printf("File could not be opened.\n");
else
{
   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);
        }

        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.