Application Program Interface(API)

By

Dr. R. R. Manza

Objectives

- Introduction
- Import and Export Data
- Low-level File Input/Output Functions
- Calling C-programs from MATLAB
- Calling JAVA from MATLAB
- Summary

Introduction

This chapter introduces major areas of MATLAB used to exchange data with external environments. The process used to load data into the workspace is called importing data and the way used to save workspace variable into files or disk files is called exporting data. To do this MATLAB gives different functions, which are covered in this chapter. Export and import process allows text, ASCII, numeric, mixed alphabetic, delimited ASCII data, binary data and so on. In this chapter the different types of files and their I/O functions are discussed in detail. These types of interfaces allow to call, compile and run programs of other languages like C, Java, FORTRAN etc from MATLAB environment. At the end of this chapter the user can import and export data values to or from MATLAB environment and other languages.

Import and Export Data

In MATLAB you can import data in many ways and load data into the workspace; this process is called importing data and to save workspace variable to a file or disk file, this process is called as exporting data. The easiest way to import text data into the workspace is to use the MATLAB import wizard. To use the import wizard to access text data, perform the following steps:

1. Start the import wizard by selecting the import data option of the file menu; MATLAB displays the selection dialog box. You can also start the import wizard by using the function or command uiimport on command line.

Cont..

2. Specify the file you want to import in the selection dialog box and click open.
3. Specify the character used to separate the individual data items. This character is called the delimiter or column-separator.
4. Select the variable that you want to import, by default the import wizard puts all the numeric data in one variable and all the text data in the other variable but you choose the other option.
5. Click finish to import the data into the workspace.

Cont..

When the Import opens the text file or copies data from the clipboard, it displays a portion of the raw data in the preview pane of the dialog box. Import and export function wizard can handle data values by three variables. First data variable is used to hold numeric data, the second textdata variable is used to store text found in the file and the third rowheader variable is used to assign the names in the left most column of data.

Cont..

Changing the variable selection; by default, the import wizard puts the entire numeric data file into one variable. If the file contains text data, the import wizard puts in a separate variable. If the file contains row or column-headers, the import wizard puts them in a separate variable called rowheader or colheader respectively. In some cases, it might be more convenient to create variables out of each row or column of data and use the row headers or column header text as the name of each variable. To do this click on the created variable from each column using rows name button.

Cont..

Import Text Data using import function:

To import text data from the command line or in M-file, use the import function. Your choice of function depends on how the data in the text file is formatted. The text data must be formatted in a uniform pattern of rows and columns, using a text character called delimiter or column separator to separate each data item. The individual data items can be alphabetic or numeric character or a mix of both. The text file may also contain one or more lines of text called header lines or may use text headers to label each column or row.

Cont..

ASCII – Data Import Functions:

1. *csvread*: Read comma separated value

Syntax:

```
M=csvread('filename');
M=csvread(‘filename', row, col);
```

The file contains only numeric data, and the result is returned in M. This command reads the file starting with specified row and column. *csvread* fills empty delimited fields with zero. Data files having lines that end with a nonspace delimiter, such as a semicolon, produce a result that has an additional last column of zeros.

Cont..

2. *dlmread*: Delimited file.

Syntax:

```
M=dlmread ('filename', 'delimiter');
```

This command reads numeric data from ASCII delimited file using the specified delimiter. Comma (,) used as a default delimiter, you can use '\t' to specify tab. This command is flexible and easy to use.

Cont..

3. *textread*: Read formatted data from text file:

Syntax:

```
[A,B,C…]=textread ('filename',' format')
```

Reads data from the file into the variables A, B, C, and so on, using the specified format, until the entire file is read. *textread* is useful for reading text files with a known format. Both fixed and free format files can be handled. Following are the formats used to read text from the files:

%d: - Reads the signed integer value.

%s: - Read the white space or delimited string.

%u: - Read the integer value

%f: - Read floating point value.

Cont..

4. fscanf: Reads formatted data from file.

Syntax:

```
A=fscanf(fid, format)
```

Reads all the data from the file specified by fid, converts it according to the specified format and returns it in matrix A. Argument fid is an integer file identifier obtained for *fopen*.

Cont..

Importing Numeric text Data:

If your data file contains only numeric data, you can use many of the MATLAB functions for importing the data. If the data is rectangular i.e. each row has the same number of elements. The simplest command used to do this is the load command.

Example: File named data.txt contains the two rows delimited by space

```
1  2  3  4  5
6  7  8  9 10
```

Cont..

```
>> load data.txt
```

When you use the load command, it imports the data and creates a variable in the workspace with the same name as the filename.

>>A=load ('data.txt') stores all values in data.txt in the variable A.

Example: data.dat contains alphabetic and numeric data like:

Ravi	class1	12	14	yes
Satish	class2	12	12	no
Ashok	class1	12	10	yes

Cont..

To read the entire contents of the file data.dat into the workspace specify the name of the data file and the format string as argument to *textread*. For each conversion specifier in your format string, you must specify a separate output variable. *textread* process read each data item in the file as specified in the formatted string and puts the value in the output variable. The number of output variables must match the number of conversion specifier in the format string.

Cont..

Output:

```
name =
    'Ravi'
    'Satish'
    'Ashok'
type =
    'class1'
    'class2'
    'class1'
x =
    12
    12
    12
y =
    14
    12
    10
answer =
    'yes'
    'no'
    'yes'
```

Cont..

If your data uses a character other than a space delimiter you must use the textread parameter 'delimiter' to specify the delimiter.

Example:

```
[grade1, grade2, grade3]=textread('data.dat',%d%d%d','delimeter',';');
```

Cont..

Importing Binary Data:

The easiest way to import binary data is by using the import wizard. If you need to work from the MATLAB command line or perform import operation as part of an M-file, you must use one of the MATLAB import function. MATLAB supports many functions to import data in different binary format, such as image file or spreadsheet data files. Using the import wizard import the binary data file. Viewing the variables when the import wizard opens a binary data file, it attempts to process the data in the file, creating variables in the data it finds to data file.

Cont..

Example:

```
save('mydat.mat','x','y');
```

It creates two variables; you can select the variables you want to import by clicking in the checkbox next to its name. All variables are pre-selected by default.

Cont..

Using Import functions with binary data:

To import binary data from the command line or in an M-file you must use one of the MATLAB import functions. Your choice of function depends upon how the data in the text file is formatted. Following are some of binary data import functions:

auread: Import sound file on Sun Microsystems platform.

Syntax: y=auread('filename');

Loads a sound file specified by the strong aufile(filename) returning a sampled data in y.

Cont..

aviread: To read/import an audio-video interleaved (AVI) file.

Syntax: mov=aviread('filename');

```
mov = aviread(filename,index)
```

To read the avi movie, filename into the MATLAB movie structure mov. If filename does not include an extension, then .avi is used. The index option read only the frame(s) specified by index.

Hierarchical Data Format: Importing data in hierarchical data format (hdf) for HDF image file format, use imread.

Cont..

imread: Importing or reading image file from graphics file.

Syntax: I= imread ('filename');

This command reads the grayscale or true color image named filename into the variable I If file contains a grayscale image I is a two-dimensional array. If file contains true color RGB image I is a three dimensional i.e. m-by-n-by-3 array.

Cont..

load: Load workspace variables from disk

Syntax: load or load filename

This command loads all the variables from MAT-file matlab.mat, if it is exists and returns an error if it does not exist. load filename command loads all the variables from specified string filename given a full pathname or MATLAB path relative partial pathname. If filename has no extension, load looks for file named filename or filename.mat and treats it as a binary mat file; other extension load treats the file ASCII data.

Cont..

wavread: Read Microsoft wave(.wav) sound file.

Syntax: Y= wavread('filename');

Loads a wave file specified by the string filename, returning the sampled data in Y. It is a must to give the .wav extension after filename. Amplitude values are in the range [-1, +1]

Cont..

wk1read: Read lotus123 spreadsheet file (.wk1)

Syntax: M = wk1read(filename)

M = wk1read(filename,r,c)

This command read a Lotus123 WK1 spreadsheet file into the matrix M. The parameter r and c starts reading at the row-column cell offset specified by (r,c) and these values are zero based. You can also provide third parameter range in the following form.

range = [upper_left_row upper_left_col lower_right_row lower_right_col]

Cont..

xlsread: Read Microsoft Excel spreadsheet file (.xls).

Syntax: A = xlsread('filename')

[A, B] = xlsread('filename')

This command returns numeric data in array A from the first sheet in Microsoft Excel spreadsheet file named specified by user filename. *xlsread* ignores leading rows or columns of text. In this second syntax the function returns two arrays; A contains the numeric data and B is cell array, which contain text data.

Cont..

Export ASCII Data:

MATLAB supports several ways to export data in many different ASCII formats. For example, you may want to export a MATLAB matrix as text file where the rows and columns are represented as space-separated, numeric value. The function you use depends on the amount of data you want to export and its format. Following are some ASCII data export functions: -

Cont..

1. diary: save MATLAB session to a file.

Syntax: diary ('filename');

The diary function creates a log of keyboard input and resulting output. The output of diary is an ASCII file. But diary command does not include Graphics. *diary* toggles diary mode on and off.

Cont..

2. dlmwrite: Write a matrix to an ASCII delimited file.

Syntax:-dlmwrite(filename,M,delimiter)

dlmwrite(filename,M,delimiter,R,C)

Write matrix M in an ASCII file format file using the delimiter to separate matrix element. A comma (,) is a default delimiter. You can use '\t' to produce tab delimiter. The resulting file is readable by spreadsheet program. The data is written to the spreadsheet filename, starting at spreadsheet cell R and C, where R is the row offset and C is the column offset.

Cont..

3. fprintf: write formatted data to a file.

Syntax: count= fprintf(fid, format,A,…)

Format the data in the real part of matrix A under control of the specified format string, and write into the file associated with file identifier fid, *fprintf* returns a count of the number. Argument fid is an integer file identifier obtained from fopen. Omitting fid causes output to appear on the screen.

Cont..

4. save: save workspace variable on disk.

Syntax: save filename;

    save filename var1 var2. . .

    save ('filename',…)

Save filename stores all workspace variables in the current directory and specified filename .mat. To save them into another directory give the full pathname with the filename. Save filename var1, var2 stores or save only the specified workspace variable in filename.mat. When saving in ASCII format consider the following conditions:

Cont..

- Each variable must save either a two-dimensional double array or two-dimensional character array.
- In order to read the file in the MATLAB use load function, all the variables of file must have the same number of columns.
- The values of all saved variables merge into a single variable that takes the name of the ASCII file.

Cont..

Example: To save all variables from workspace

```
save test.mat
```

To save p and q in binary mat file test.mat

```
savefile='test.mat';
p=1:1000;
q=magic(4);
save(savefile,'p','q');
```

Cont..

Export Binary Data:

To export binary data in one of the standard binary formats, you can use the following MATLAB high-level function designed to work with that format:

1. auwrite: Export sound data in Sun Microsystems format.

Syntax: auwrite(y,'filename')

This function writes a sound file specified by string value into the filename. The data sound is arranged with one channel per column.

Cont..

2. imwrite: Write image to graphics file.

Syntax: imwrite(I, filename, fmt)

This function writes the image in I to filename in the specified format, which is indicated by fmt. I can be either a grayscale image (M-by-N) or true color image (M-by-N-by-3). If I is of class uint8 or uint16, imwrite writes the actual values in the array to the file.

Cont..

3. wavwrite: Write Microsoft wave (.wav) sound file.

Syntax: wavwrite(Y, 'filename')

This supports to write multichannel 8 or 16-bit wave data. This function writes a wave file specified by the string filename. The data should be arranged with one channel per column.

Cont..

4. wk1write: Write a matrix to a lotus123 spreadsheet file.

Syntax: wk1write(filename, M)

This function writes the matrix M into a lotus 123wk1 spreadsheet file in the given filename.

Low-level File Input/Output Functions

MATLAB includes a set of low-level file input/output functions that are based on I/O function of the ANSI standard C library. MATLAB file I/O functions use the same programming model as the C-programming routines.

To read or write data using I/O functions:

To read data or write data using I/O function perform the following steps:

1. Open the file using fopen. fopen returns a file identifier which you use with all the other low-level file I/O routines.

Cont..

2. Operate on the files:
    a) Read binary data using fread.
    b) Write binary data using fwrite
    c) Read text strings from a file line-by-line using fgets or fgetl.
    d) Read formatted ASCII data using fscanf.
    e) Write formatted ASCII data using fprintf.
3. Close the file, using fclose.

Cont..

Opening files:

Before reading or writing a text or binary file you need to open it with fopen command.

Syntax: fid=fopen ('filename', 'permission')

This function returns the file identifier fid. The permission string specifies the mode of access to the file you require. Following are the possible permission strings:

Cont..

r - Reading only

w - Writing only

a - appending only

r+ - Both reading and writing

rb - Open binary file for reading.

If successful, open returns a non-negative integer, called a file identifier. You pass this value as an argument to the other I/O function to access the open file.

Cont..

Example:

```
fid=fopen('data.dat','r');
```

For example if you try to open a file which does not exist, fopen assign -1 to the file identifier and also assign an error message to an optional second output argument.

```
Ex: fid=0;
filename=input('open file’:'s');
[fid,message]=fopen(filename,'r');
if fid=-1
disp(message)
end
```

Cont..

Reading binary file:

The fread function reads all or part of a binary file and stores it in a matrix. It reads an entire file and interprets each byte of input as the next element of the matrix.

Example:

```
fid=fopen('data.dat','r');
A=fread(fid);
```

To echo the data to the screen after reading it, use char to display the contents of A as character, transposing the data so it displays horizontally.

Cont..

disp(char(A')); Controlling the Number of values read: fread accepts an optional second argument that controls the number of values read. A=fread(fid,100);

This command or statement reads, the first 100 data values of the file specified by fid into the column vector A.

Controlling data type of each value: An optional third argument to fread controls the data type of the input. The data type argument controls both the number of bits read for each value and interpretation of those bits as character, integer or float.

Cont..

Writing Binary File:

The fwrite function writes the element of a matrix to a file in a specified numeric precision, returning the number of values written.

Example:

```
fwid = fopen('data.dat','w');
count = fwrite(fwid,zeros(5),'int32');
status = fclose(fwid);
```

Controlling position in a file: Once you open a file with fopen, MATLAB maintains a file position indicator that specifies a particular location within a file.

Cont..

Determining end of file: The fseek and ftell function lets you set and query the position in the file at which the next input or output operation takes place. The fseek function repositions the file position indicator, letting you skip over data or backup to an earlier part of the file. The ftell function gives the offset in bytes of the file position indicator for specified file.

Syntax of fseek and ftell:

status=fseek(fid, offset, origion);

position=ftell(fid);

Cont..

fid is the identifier of the file offset which is a positive or negative offset value specified in bytes. Origin is an origin forms which is used to calculate the move specified as a string;

'bof'- Beginning of file

'cof'- Current position in file.

'eof'- End of file.

Cont..

Example:

```
I=1:4

fwid = fopen('data.dat','w');

count = fwrite(fwid,I,'int32');

status = fclose(fwid);
```

This code writes number 1 to 4 in the file data.dat. Reopen the file

```
>> fid = fopen('data.dat','r');

>> status = fseek(fid,3,'bof');

>>position=ftell(fid);
```

Cont..

Reading String line-by-line from text file:

MATLAB provides two functions, fgetl and fgets that read lines from formatted data/text file and store them in string vectors. The two functions are almost identical; the only difference is that fgets copies the new line character to the string vector but fgetl does not.

Example:

```
filename=input('open file:'s');
fid=fopen('filename','r+');
        y=0;
        while feof(fid)==0
                line=fgetl(fid);
        end
        fclose(fid);
```

Cont..

Reading formatted ASCII data:

The fscanf function is like the fscanf function in C-language, reading data from a file and assigning it to one or more variables. The conversion specifiers for fscanf being with a percent sign (%), following are the common conversion specifiers:

%s	Match a string.
%d	Match an integer in base 10 format.
%g	Match a double-precision floating-point value.

Cont..

Example:

test.dat contains values 2.33948373,3.48576304, 6.49585787 etc.

```
fid = fopen('test.dat','r');
MyData = fscanf(fid,'%g');
status = fclose(fid);
```

This code does not use any loops instated, the fscanf function continues to read in text as long as the input format is compatible with the format specifier. An optional size argument controls the number of matrix element read.

Cont..

Writing formatted text file:

The fprintf function converts data to character strings and outputs them to the screen or a file. A format control string containing conversion specifier and any optional text specify the output format. The conversion specifiers control the output of array elements, fprintf copies text directly. Optional fields in the format specifier control the minimum field width and precision.

Following are the common conversion specifier:

%e	Exponential notation
%f	Fixed point notation
%g	automatically selects the shorter of %e and %f

Cont..

Closing of file:

When you finish reading or writing use fclose function to close the file. This function closes the file associated with file identifier.

Syntax: status=fclose(fid) or status= fclose(all);

The all parameter close all open file. Both forms return zero if the file or files were successfully closed or -1 if the attempt was unsuccessful. fclose does not clear the file identifier.

Calling C-programs from MATLAB

MATLAB is a complete, self-contained environment for programming and manipulating data; it is often useful to interact with data and programs external to the MATLAB environment. MATLAB provides an external interface program written in C and FORTRAN language.

MEX-File: You can call your own C subroutines from MATLAB as if they were built-in function. MATLAB callable C programs are referred to as MEX-File. MEX files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute.

Cont..

MEX files are not appropriate for all applications. MATLAB is a high productivity system that especially helps to save time in low level programming languages like FORTRAN or C. In general, most programming should be done in MATLAB. Don't use the MEX facility unless your application requires it.

Cont..

Using MEX file:

MEX-file are subroutines produced from C source code. They behave just like M-files and built-in functions. MATLAB identifies MEX files by platform specific extension. MEX-file Extensions are as follows:

Alpha	Mexaxp
HP, Version 10.20	mexhpy
HP, version 11.x	mexhpux
IBM RS/6000	mexrs6
Linux	mexglx
SGI, SGI64	mexsg
solaris	mexsol
windows	dll

You can call MEX-file exactly as you would call any M-function. MEX-file takes precedence over M-file when like-named files exist in the same directory.

Cont..

The MATLAB Array:

The MATLAB language works with only a single object type i.e. the MATLAB array. All MATLAB variables, including scalars, vectors, matrices, strings, cell array, structures and objects are stored as MATLAB array. In C the MATLAB array is declared to be of type mxArray. The mxArray structure contains, among other things, its type, dimensions, the data associated with array, whether the variable is real or complex, if structure or object, the number of fields and field names. The data is stored as two vectors of double-precision numbers- one contains the imaginary data. The pointers to this data are referred to as pr (pointers to real data) and pi (pointers to imaginary data). You can write C program or MEX file in C that accept any data type supported by MATLAB.

Cont..

Compiler Requirement: MATLAB contains all the tools you need to work with the API (Application Program Interface). MATLAB includes a compiler for the PC called Lcc but does not include in the FORTRAN compiler. Also, if you are working on Microsoft windows platform your compiler must be able to create 32-bit windows dynamically linked libraries (DLLs). MATLAB supports many compilers and provides preconfigured files called option files, designed specifically for these compilers.

Cont..

Testing your configuration on windows: Before you can create MEX-file on the windows platform you must configure the default file option (mexopt.bat). The switch setup provides an easy way for you to configure the default file option:

```
>>mex –setup
```

To select a compiler or change to existing default compiler, use mex –setup.

Building MEX-file:

To compile and link the example source files on windows at the MATLAB prompt type:

```
mex yprime.c
```

Cont..

This should create the MEX-file called yprime with the .dll extension which corresponds to the windows platform; you can call this function or file like the M-function.

Cont..

Specifying an options file:

You can use the –f option to specify an options file on either UNIX or windows.

MEX filename –f <optionsfile> and specify the name of the option file along with its pathname. There are several situations when it may be necessary to specify an options file every time you use the mex script to use different compiler or to compile MAT or engine stand-alone programs.

Cont..

Defaults file option:

The default mex options file is placed in your user profile directory after you configure your system by running mex –setup. The mex script searches for an options file called mexopts.bat. If no options file is found, mex searches your machine for supported C compiler and automatically configures itself to use that compiler. Also during the configuration process, it copies the compilers default option file to the user profile directory. If multiple compilers are found, you are prompted to select one. The user profile directory is a directory that contains user specified information such as desktop appearance, recently used files and so on.

Cont..

MEX file problems: It contains information regarding common problems that may occur during creation of MEX file. Few of the problems are discussed below:

1. Mathworks program fails during compilation: This type of problem occurrs if the header files of Mathworks generated a sting of error when we tried to compile the program code.
2. Our own program fails during compilation: It generates a string of syntax error if we attempt to mix C code. Other common problems that can occur in C program are neglecting to include all necessary header files or neglecting to link against all required libraries.

Cont..

3. MEX file load error: This type of error occurs if MATLAB is unable to recognize if the given MEX file is valid or not. To recognize MEX file MATLAB looks the gateway routine and MEX function. If function name is misspelled, MATLAB is unable to load the MEX file and hence generates this type of error. On windows MATLAB will fail to load MEX files if it cannot find all the reference DLLs of MEX file. All DLLs must be on the path where MEX file is present.

4. Segmentation fault or Bur error: If MEX file causes a segmentation violation or Bus error, it means the MEX file has attempted to access protected, read only or unallocated memory. This error occurs after the MEX file finishes the execution hence it is much more difficult to trace out such error.

Cont..

5. Program generates incorrect results: There are several possible causes behind wrong results of a program. The first cause is something wrong in computational logic. The second is the program can read information from an un-initialized section of memory. Another possibility behind generating wrong results can be overwriting of valid data due to memory mishandling.

Cont..

Files and Directories of Windows System:

The following figure gives a detailed idea of directories in which the MATLAB API files are located.

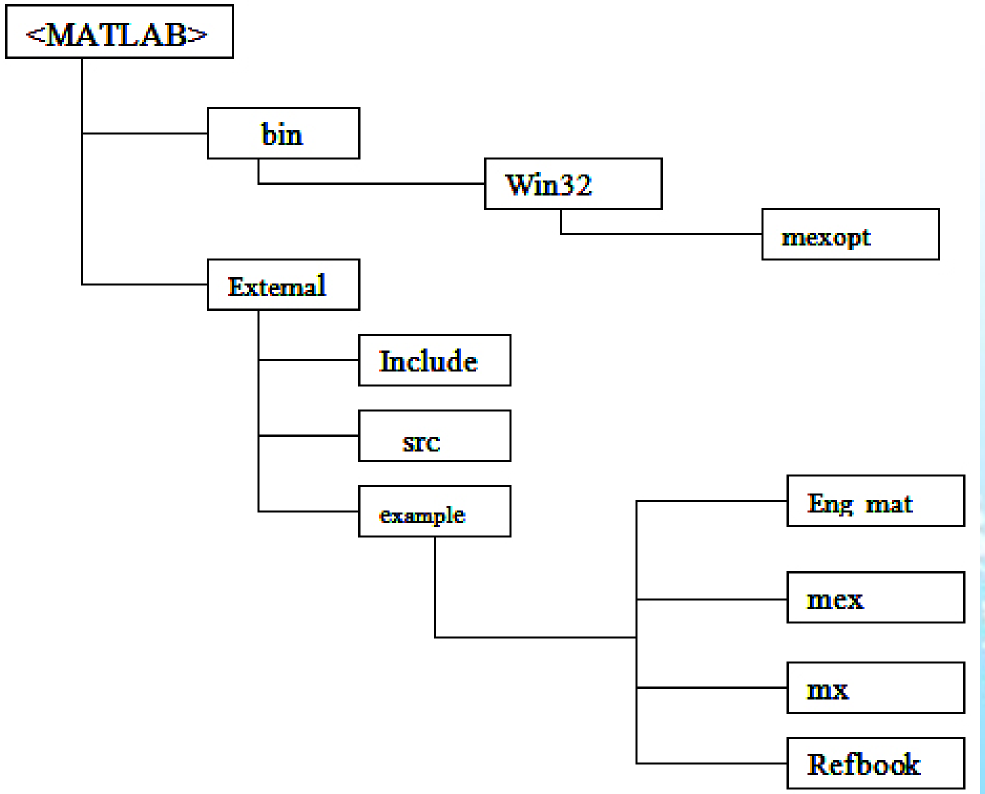

Fig 1: MATLAB API files and directories on windows system

Cont..

<matlab> \bin\win32: This directory consists of mex.bat batch file which is used to build C and Fortran files into mex files.

<matlab>\bin\win32\mexopts: This directory holds the pre-configured option file which makes script uses with particular compiler.

<matlab> \extern\include: This directory contains the header files for developing C and C++ application that interface with MATLAB. The relevant header files for the MATLAB API are:

1. engine.h: This header file is used for MATLAB engine programs. It contains function prototype for engine routines.

Cont..

2. mat.h: It is a header file for programs accessing MAT files and contains function prototype for mat routine.

3. mex.h: This header file is used for building MEX file. It contains function prototype for MEX routines.

< matlab>\extern\scr: This directory contains files that are used for debugging MEX files.

Cont..

C-MEX files and its components:

These files are built by using the MEX script to compile your C source code with additional call to API routines. The source code of MEX file consists of two routines:

A *computational routine* contains the code for performing the computations to implement in the MEX file. Computations can be numerical as well as input and output of data. A *gateway routine* that contains interfaces of computational routine of MATLAB by entry point makes function and its parameters like prhs, nrhs, plhs, nlhs. Where prhs is an array of right hand input arguments, nrhs is the no. of right hand input argument, plhs is an array left hand output argument and nlhs is the number of left to output argument.

Cont..

The gateway calls the computational routine as a subroutine. In the gateway routine, you can access the data in the mxArray structure and then manipulate this data in your C computational subroutine. For example, the expression mxGetpr(prhs[0] returns a pointer of type double to the real data in the mxArray pointed to by prhs[0]. You can then use this pointer like any other pointer of type double in C. After calling your C computational routine from the gateway you can set a pointer of type mxArray to the data it returns. MATLAB is then able to recognize the output from your computational routine as the output from the MEX-file.

Cont..

The following C MEX cycle figure shows how inputs enter a MEX file, what function the gateway routine performs and how output returns to MATLAB.

Required Argument to a MEX-file:

The two components of the MEX file may be separate or combined. In either case the files must contain the include “mex.h” header so that the entry point and interface routines are declared properly. The name of the gateway routine must always be mex function and must contain these parameters.

Cont..

Void mexfunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])

The parameter nlhs and nrhs contain the number of left and right hand arguments with which the MEX file is invoked. In the syntax of the MATLAB language function have the general form

[a,b,c…]=fun(d,e,f…)

Where, the ellipsis(…) denotes additional terms of the same format. The a,b,c… are left hand argument and d,e,f… are right hand arguments.

Cont..

The parameter plhs and prhs are vectors that contain pointers to the left and right hand argument of the MEX file. Note that both are declared as containing type mxArray* which mean that the variables pointed at are MATLAB array. prhs is a length nrhs array of pointers to the right-hand side inputs to the MEX file and plhs is a length nlhs array that will contain pointers to the left hand side output that your function generates.

Calling JAVA from MATLAB

MATLAB capability enables you to conveniently bring Java classes into the MATLAB environment, to construct objects from those classes to call methods on the Java object and save Java objects for reloading all accomplished with MATLAB functions and command.

Java interface is integral to MATLAB: Every installation of MATLAB includes a Java Virtual Machine (JVM) so that you can use the Java interpreter via MATLAB commands and you can create and run programs that create and access Java objects.

Cont..

Benefits of the MATLAB-JAVA Interface:

- Access Java API class package that support essential activities such as I/O and networking.
- Access third-party Java class.
- Easily construct Java object in MATLAB
- Call Java object method, using either Java or MATLAB syntax.
- Pass data between MATLAB variable and java objects.

Cont..

Bringing Java Classes into MATLAB: You can draw from an extensive collection of existing Java classes or create your own class definitions to use with MATLAB. Once you have the class you need defined in individual .class files, package or Java Archive files.

Sources of Java Classes: There are three main sources of Java classes that you can use in MATLAB.

1. Build in classes: Java language includes general purpose class packages such as java.awt.
2. Third party classes: There are a number of packages of special purpose java classes that you can use.
3. User –defined classes: You can define new Java classes or sub-classes. You need to use Java development to do this.

Cont..

Defining new java Classes: You can define new Java classes and sub-classes of existing classes; you must use a Java Development Environment external to MATLAB that support Java version 1.1. After you create class definitions in Java files, use your Java compiler to produce class files from them.

Making Java Classes available to MATLAB: To make your third-party and user defined java classes available in MATLAB, place them on your MATLAB path by adding their locations to the file classpath.txt. To make individual classes available in MATLAB, add to classpath.txt an entry containing the full path to the directory you want to use for the .class files.

Cont..

Example: To make available, your compiled Java classes in the file d:\work\Jclass\test.class and the following entry to your classpath.txt file:

D:\work\Jclass

To make entire package available by adding package name to classpath.txt the full path to the parent directory of the highest-level directory of the package path. This directory is the first component in the package name. You can also use the jar (Java Archive) tool to create JAR file, containing multiple Java classes and package in a compressed ZIP format.

Cont..

To make the contents of a JAR file available for use in MATLAB, add to classpath.txt, the full path (filename) for the jar file. The classpath.txt requirement for JAR files are different than that for .class files and packages for which you do not specify any filename.

Cont..

Example: To make available the JAR file d:\work\classes\utilpkg.jar add the following to your classpath.txt:

d:\work\classes\utilpkg.jar

Finding and editing classpath.txt: To make your java classes available, you can edit the default classpath.txt, which is in the directory toolbox/local or you can copy the default classpath.txt from this directory to your own startup directory where the changes you make to it do not affect anyone else. To find the classpath.txt that is used by your MATLAB environment, use the MATLAB which function.

Cont..

```
>>which classpath.txt
```

To edit either the default file or the copy you made in your own directory.

```
>>edit classpath.txt
```

MATLAB reads classpath.txt only upon startup. So if you edit classpath.txt or change your .class files while MATLAB is running, you must restart MATLAB to put those changes into effect.

Cont..

Loading Java class Definition: Normally MATLAB loads a Java Class automatically when your code first uses it (for ex. When you call its constructor). When you use the which function on methods defined by java classes, the function only acts on the classes currently loaded into the MATLAB working environment which operates on MATLAB classes, whether or not they are loaded.

Determining which classes are loaded; At any time during a MATLAB session you can obtain a listing of all the java classes that are currently loaded. To do so, you use the inmem function in the following form:

Cont..

```
>>[M,x,j]=inmem
```

This function returns the list of Java classes in output argument j. It also returns in M the names of all currently loaded M-files and in x in the names of all currently loaded mex files.

Simplifying Java class names: Your MATLAB commands can refer to any Java class by its fully qualified name, which includes its package name.

Cont..

Example: java.lang.String

java.util.Enumeration

A fully qualified name can be rather long making command and functions. You can refer to classes by the class name alone if you, first, import the fully qualified name into MATLAB. The import command has the following forms:

Cont..

```
>>import package_name.*

>>import plg_q.* pkg_2.*

>>import class_neme            %import one class

>>import                       % displays current import list

>>L=import                     %  returns current import list.
```

Cont..

MATLAB adds all classes that you import to a list called import list. You can see what classes are on that list by typing import.

Example: Suppose a function contains the following statement.

```
>>import java.lang.String

>>import java.util.* java.awt.*

>>import java.util.Enumeration
```

Cont..

Code that follows the import statement above can now refer to the Sting.Frame and Enumeration classes without using the package names.

```
>>str=String('Hello');                %Create java.lang.String object
>>frm=Frame                           % Create java.awt.Frame object
>>method =Enumeration                 % listof java.util.Wnumeration method
```

To clear the list of imported Java classes, invoke the command:

```
>>clear import
```

Cont..

Creating and using Java objects and functions: In MATLAB you create a Java object by calling one of the constructors of that class. You then use commands and programming statement to perform operation on these objects. You can construct java objects in MATLAB by calling the java class constructor, which has the same name as the class.

Example; Constructor creates a Frame object with the title ‘Frame A’ and the other properties with their default values.

Cont..

```
>>Frame=java.awt.Frame('Frame A');
```

Under certain circumstances you may need to use the *javaObject* function to construct a java object. The following syntax invokes the Java constructor for class, class_name with argument list that matches X1,……Xn and return a new object J.

```
>>J=javaObject('classname', X1…Xn);
```

Cont..

Example: strobj=javaObject (‘java.lang.String’, ‘Hello’);

Using the *javaObject* function enables you to use classes having names with more than 31 consecutive characters and specify the class for an object at run-time. The *javaObject* function also allows you to specify the java class for the object being constructed at run-time. In this situation you call *javaObject* with the string variable in place of the class name argument.

Cont..

```
>>class= ‘java.lang.String’;
>>text= ‘hello’ ;
>>strobj=javaObject(class,text);
```

In the usual case when the class to instantiate is known at development time, it is more convenient to use the MATLAB constructor syntax.

Cont..

Example:

```
>>strobj=java.lang.String('Hello');
```

In MATLAB the Java objects are references and do not adhere to MATLAB copy-on-assignment and pass-by-value rules.

Cont..

Example:

```
>>origfrm=java.awt.Frame;
>>setsize(origfrm,800,400);
>>newframe=origfrm;
```

In the last statement above the variable newframe is a second reference to the origfrm not a copy of object.

```
>>setsize (newframe,100,800);
getsize (origfrm)
ans=
        java.awt.dimension[width=1000, height=800]
```

Cont..

Any change to the object at newframe will also change the object at origfrm. You can also concatenate objects by using cat command. Use the MATLAB save function to save a Java object to a MAT-file. Use the load function to load it back into MATLAB from MAT-file. To save a Java object, to a MAT-file and to load the object from, the MAT-file. Make sure that the object and its class meets the following criteria:

- This class implements the serializable interface, either directly or by inheriting it from a parent class. Any embedded or otherwise referenced objects must also implement serializable.

Cont..

- Either the class does not have any transient data fields or the values in transient data fields of the object to be saved are not significant.
- If you define your Java classes or sub-classes of existing classes, you can follow the criteria above to enable object of the class to be saved and loaded in MATLAB.

Determining the class of an object: To find the class of a Java object, use the query from the MATLAB function class. After execution of the following example, frameclass contains the name of the package and class that Java object frame instantiates.

Cont..

```
>>frameclass=class(frame)

frameclass=

            java.awt.Frame
```

Because this form of class also works on MATLAB objects, it does not, in itself tell you whether it is a Java class. To determine the type of class use the isjava function, which has the form:

```
X=isjava(obj)
```

isjava returns 1 if obj is java and 0 if it is not.

Cont..

To find out whether or not an object is an instance of a specified class, use the isa function.

X=isa(obj, ‘classname’)

This function returns 1 if obj is an instance of the class named ‘classname’, and 0 if it is not.

Cont..

Invoking methods on Java objects: This section explains how to invoke an objects method in MATLAB. It also covers how to obtain information related to the method that you are using java and MATLAB calling syntax and also obtain information about method. To call methods on Java objects you can use the Java syntax:

object.method(arg1,…argn)

Cont..

Example:

```
>>frame.setTitle(‘Sample Frame’)

>>title=frame.getTitle

title=

        Sample Frame
```

Alternatively, you can call Java object method with MATLAB syntax;

method (object, arg1…argn)

Cont..

Example:

```
>>setTitle (frame, ‘Sample Frame’)

>>title=getTitle(frame)

title=

        Sample Frame
```

Under certain circumstances, you may need to use the javaMethod function to call Java method.

X=javaMethod(‘method_name’,j,X1…Xn);

j is Java object with the argument list that matches X1…Xn.

Cont..

- MATLAB offers several functions to help obtain information related to the java methods. The methodsview function is used to display the listing of Java methods. If you want to know what methods are implemented by particular Java classes, use the methodsview function in MATLAB. Specify the class name in the command line. If you have imported in package that defines this class, then the class name alone will suffice. The following command line lists information on all methods in the java.awt.MenuItem. A new window appears, listing one row of information for each method in the class. Methods function returns information on methods of MATLAB classes will also work on Java classes.

Cont..

Syntax: methods class_name

methods class_name –full

n=methods(‘class_neme’)

n=methods(‘class_name’, ‘-full’)

With the ‘-full’ qualifier to return the listing of the method names along with attributes, argument list and inheritance information on each. You can use the *which* function to display the fully qualified name of a method implemented by loaded java class with the –all qualifier the which function finds all classes with a method of the name specified.

Cont..

Example:

```
which –all equals
java.lang.Sting.equals
java.awt.Frame.equals
```

The *which* function operates differently on java classes than it does on MATLAB classes. MATLAB classes are always displayed by whether or not they are loaded. This is not true for Java classes.

MATLAB commands that operate on Java objects and arrays make use of the methods that are implemented within or inherited by, these objects classes. You can use the disp function to display the value of a variable or an expression in MATLAB. You can also use *disp* to display a java object in MATLAB.

Cont..

When *disp* operates on a java object, MATLAB formats the output using the to String method of the class to which the object belongs. If the class does not implement this method then an inherited to String method is used. If no intermediate ancestor classes define this method, it uses to String method defined by the java.lang.object class. In this way you can change the way MATLAB displays information regarding the object of the class.

Changing the effect is equal, double and char:

Cont..

The MATLAB *isequal* function compares two or more arrays for equality in type, size and contents. This function can also be used to test Java objects for equality. When you compare two-java objects *isequal*, MATLAB performs the comparison using the Java method equals.

You can also define your own Java methods to double and to char to change the output of the MATLAB double and char functions. If you are using a class of your own design, you can write own to double method to perform conversion on objects to that class to a MATLAB double the syntax of double command as follows:

double(obj) similarly *char*(obj)

Summary

In MATLAB you can import data in many ways and load data into the workspace and export it from the workspace. Includes import text data, ASCII data, Numeric data and binary data and similarly export binary data, text data and ASCII data. Also MATLAB includes a set of low-level file input/output functions that are based on I/O function of the ANSI standard C library. MATLAB file I/O functions use the same programming model as the C-programming routines. In Application program interfaces (API) or external interface MATLAB can call programs from C, FORTRAN or Java. The API concept for C and Java programs are described in detail in this chapter.

Thank You