72 DPI | M08_SMIT8708_03_SE_C08.indd Page 167 02/02/12 4:38 PM user-f396 F-396 EQA

CHAPTER 8

File Input and Output

Chapter Objectives

This chapter discusses three levels of capability for reading and writing files in MATLAB, each including a discussion of the circumstances under which they are appropriate:

- Saving and restoring the workspace
- High-level functions for accessing files in specific formats
- Low-level file access programs for general-purpose file processing

Reading and writing data in data files are fundamental to the utility of programming languages in general, and MATLAB in particular. In addition to the obvious need to save and restore scripts and functions (covered in Chapter 2), here we consider three types of activities that read and write data files.

- The MATLAB language provides for the basic ability to save your workspace (or parts of your workspace) to a file and restore it later for further processing.
- There are high-level functions that consume the name of a file whose contents are in any one of a number of popular formats and produce an internal representation of the data from that file in a form ready for processing.
- Almost all these functions have an equivalent write function that will write a new file in the same format after you have manipulated the data.
- However, we also need to deal with lower-level capabilities for manipulating text files that do not contain recognizable structures.

Introduction

This chapter discusses files that contain workspace variables, spreadsheet data, and text files containing delimited numbers and plain text. Subsequent chapters will discuss image files and sound files. For information on the other file formats, consult the help documentation for details of their usage.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 168 02/02/12 4:38 PM user-f396 F-396 EQA

The MATLAB language also provides the ability to access binary files—files whose data are not in text form—but the interpretation of binary data is beyond the scope of this text, and we will not consider binary files here. Refer to MATLAB documentation for information about binary files.

8.1 Concept: Serial Input and Output (I/O)

We frequently refer to the process of reading and writing data files as Input/Output (I/O). We have already seen and used examples of file I/O to store and retrieve data and programs. Your script and function files are stored in your current directory and could be invoked from there by name from the Command window. In general, any computer file system saves and retrieves data as a sequential (serial) stream of characters, as shown in Figure 8.1. Mixed in with the characters that represent the data are special characters ("delimiters") that specify the organization of the data.

When a program opens a file by name for reading, it continually requests blocks of data from the file data stream until the end of the file is reached. As the data are received, the program must identify the delimiting characters and reformat the data to reconstruct the organization of the data as represented in the file. Similarly, when writing data to a file, the program must serialize the data, as shown in Figure 8.2. To preserve the organization of the data, the appropriate delimiting characters must be inserted into the serial character stream.

The purpose of the file I/O functions discussed in this chapter is to encapsulate these fundamental operations into a single system function, or at least into a manageable collection of functions.

8.2 Workspace I/O

The MATLAB language defines the tools to save your complete workspace to a file with the `save` command and reload it with the `load` command. If you provide a file name with the save command, a file of that name will be

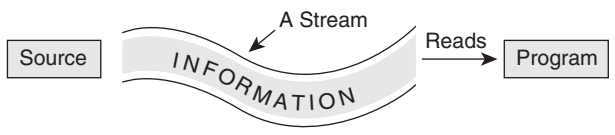

Figure 8.1 *An input stream*

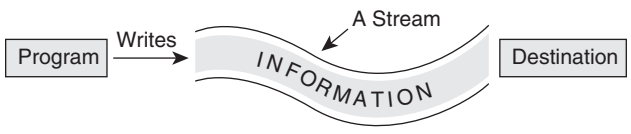

Figure 8.2 *An output stream*

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 169 02/02/12 4:38 PM user-f396 F-396 EQA

written into your current directory in such a form that a subsequent load command with that file name will restore the saved workspace. By convention, these files will have a `.mat` extension. If you do not provide a file name, the workspace is saved as `matlab.mat`.

If you are using MATLAB, you can also identify specific variables that you want to save—either by listing them explicitly or by providing logical expressions to indicate the variable names. For example:

```
>> save mydata.mat a b c*
```

would save the variables `a` and `b` and any variable beginning with the letter `c`. For more details, consult the MATLAB help documentation.

8.3 High-Level I/O Functions

We turn to the general case of file I/O in which we expect to load data from external sources, process that data, and perhaps save data back to the file system with enhancements created by your program. When you try to process data from some unknown source, it is difficult to write code to process the data without some initial exploration of the nature and organization of that data. So a good habit is to explore the data in a file by whatever means you have available and then decide how to process the data according to your observations.

Style Points 8.1

In a practical sense, saving workspace data is very rarely an appropriate approach to saving work because it saves the results but not the code that generated the results. It is almost always better to save the scripts and raw data that created the workspace. For example, this is a good idea when you have a lengthy computation (perhaps one run overnight) to prepare data for a display. You could split that script into two halves. The first half would do the overnight calculation and save the workspace. The second part can then read the workspace quickly, and you can develop sophisticated ways to display the data without having to re-run the lengthy calculations.

Most programming languages require the programmer to write detailed programs to read and write files, especially those produced by other application programs or data acquisition packages. Fortunately for MATLAB programmers, much of this messy work has been built into special file readers and writers. Table 8.1 identifies the type of data, the name of the appropriate reader and writer, and the internal form in which MATLAB returns the data.

8.3.1 Exploration

The types of data of immediate interest are text files and spreadsheets. In Table 8.1 notice that the delimited text files are presumed to contain numerical values, whereas the spreadsheet data may be either numerical data stored as doubles or string data stored in cell arrays. Typically, text files are delimited by a special character (comma, tab, or anything else) to designate the column divider and a new-line character to designate the

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 170 02/02/12 4:38 PM user-f396 F-396 EQA

Table 8.1 File reading and writing

File	Content	Reader	Writer	Data Format	Extension
Plain text	Any	`textscan`	`fprintf`	Specified in the function calls	.txt usually
CSV	Comma separated values	`csvread`	`csvwrite`	double array	.csv
Delimited	Numbers separated by delimiters	`dlmread`	`dlmwrite`	double array	.txt usually
Excel worksheet	Microsoft specific	`xlsread`	`xlswrite`	Double array + 2 cell arrays	.xls
Image data	Various	`imread`	`imwrite`	True color, grayscale, or indexed image	various
Audio file	AU or WAV	`auread` or `wavread`	`Auwrite` or `wavwrite`	Sound data and sample rate	.au or .wav
Movie	AVI	`aviread`	`no`	movie	.avi

rows. Once the data are imported, all of our normal array and matrix processing tools can be applied. The exception to this rule is the plain text reader that must be provided with a format specifier to define the data, and the names of the variables in which the data are to be stored.

So when you are approached with a file, the file extension (the part of the file name after the dot) gives you a significant clue to the nature of the data. For example, if it is the output from a spreadsheet, you should open the data in that spreadsheet program to explore its contents and organization. [Typically, spreadsheet data will not open well in a plain text editor.] If you do not recognize the file extension as coming from a spreadsheet, try opening the file in a plain text editor such as that used for your scripts and functions and see if the data are legible. You should be able to discern the field delimiters and the content of each line if the file contains plain text.

8.3.2 Spreadsheets

Excel is a Microsoft product that implements spreadsheets. Spreadsheets are rectangular arrays containing labeled rows and columns of cells. The data in the cells may be numbers, strings, or formulae that combine the data values in other cells. Because of this computational capability, spreadsheets can be used to solve many problems, and most offer flexible plotting packages for presenting the results in colorful charts. There are occasions, however, when we need to apply the power of the MATLAB language to the data in a spreadsheet.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 171 02/02/12 4:38 PM user-f396 F-396 EQA

MATLAB provides a reader for Excel spreadsheets that gives you a significant amount of flexibility in retrieving the data from the spreadsheet. Consider the typical set of data in a spreadsheet named `grades.xls` shown in Table 8.2. The goal of your spreadsheet reader is to separate the text and numerical portions of the spreadsheet. The parameter consumed by your spreadsheet reader is the name of the file; you can ask for up to three return variables: the first will hold all the numerical values in an array of doubles; the second will hold all the text data in cell arrays; and the third, if you request it, will hold both string and numerical data in cell arrays (try Exercise 8.1).

Table 8.2 Sample spreadsheet date

name	age	grade
fred	19	78
Joe	22	83
Sally	98	99
Charlie	21	56
Mary	23	89
Ann	19	51

Exercise 8.1 Reading Excel data

```
>> [nums txt raw] = csvread('grades.csv')
% or xlsread('grades.xls') with MATLAB
nums =
    19     78
    22     83
    98     99
    21     56
    23     89
    19     51
txt =
    'name'      'age'     'grade'
    'fred'        ''          ''
    'joe'         ''          ''
    'sally'       ''          ''
    'charlie'     ''          ''
    'mary'        ''          ''
    'ann'         ''          ''
raw =
    'name'      'age'     'grade'
    'fred'      [ 19]     [   78]
    'joe'       [ 22]     [   83]
    'sally'     [ 98]     [   99]
    'charlie'   [ 21]     [   56]
    'mary'      [ 23]     [   89]
    'ann'       [ 19]     [   51]
    'ann'       [ 19]     [   51]
```

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 172 02/02/12 4:38 PM user-f396 F-396 EQA

The reader first determines the smallest rectangle on the spreadsheet containing all of the numerical data; we will refer to this as the "number rectangle." Then it produces the following results:

1. The first returned result is an array with the same number of rows and columns as the number rectangle and containing the values of all the numeric data in that rectangle. If there are non-numeric values within that rectangle, they are replaced by `NaN`, the built-in name for anything that is not a number.
2. The second returned result is a cell array with the same size as the original spreadsheet, containing only the string data; to ensure the consistency of this cell array, all numbers present are replaced by the empty string.
3. The third returned result is a cell array also with the same size as the original spreadsheet, containing both the strings and the numbers. Cells that are blank are presumed to be numerical, and are assigned as a cell containing an empty vector.

Frequently, after processing data, you need to write the results back to a spreadsheet. Excel spreadsheets can be written using:

```
xlswrite(<filename>, <array>, <sheet>, <range>)
```

where `<filename>` is the name of the file, `<array>` is the data source (a cell array), `<sheet>` is the sheet name, and `<range>` is the range of cells in Excel cell identity notation. The sheet name and range are optional.

8.3.3 Delimited Text Files

If information is not available specifically in spreadsheet form, it can frequently be presented in text file form. If the data in a text file are numerical values only and are organized in a reasonable form, you can read the file directly into an array. It is necessary that the data values are separated (delimited) by commas, spaces, or tab characters. Rows in the data are separated as expected by the new-line character. These values might be saved in a file named `nums.txt`. This type of numerical data (not strings) in general delimited form can be read using `dlmread(<file>,<delimiter>)`, where the delimiter parameter is a single character that can be used to specify an unusual delimiting character. However, the function can usually determine common delimiter situations without specifying the parameter.

The `dlmread(...)` function produces a numerical array containing the data values. Try reading delimited files in Exercise 8.2. Table 8.3 shows the content of the file `nums.txt`.

Notice that the array elements where data are not supplied are filled with zero.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 173 02/02/12 4:38 PM user-f396 F-396 EQA

Exercise 8.2 Reading delimited files

```
>> A = dlmread('nums.txt')
A =
    19    78    42
    22    83   100
    98    99    34
    21    56    12
    23    89     0
    19    51     0
```

Delimited data files can be written using: `dlmwrite( <filename>, <array>, <dlm>)` where `<filename>` is the name of the file, `<array>` is the data source (a numerical array), and `<dlm>` is the delimiting character. If the delimiting character is not specified, it is presumed to be a comma.

The `csvread(...)` function is a special case of `dlm read(...)` where the delimiter is presumed to be a comma, and it produces a numerical array containing the data values. As noted above, the MATLAB version of `csvread(...)` has been enhanced so that if the data contain only numerical values, it will return an array. However, if the data contain some strings, it produces the three results specified above for xls `read(...)`. The normal content of CSV files allows embedded strings to contain the comma character. This is accomplished by surrounding any such string with double quotes—for example, `"Jones, Tom"` in a CSV file would prevent the embedded comma from separating this string into the two strings: `'Jones'` and `'Tom'`.

Common Pitfalls 8.1

It is best not to provide the delimiter unless you have to. Without it, MATLAB will assume that repeated delimiters—like tabs and spaces—are single delimiters. If you do specify a delimiter, it will assume that repeated delimiter characters are separating different, absent field values.

Table 8.3 Sample delimited text file

19	78	42
22	83	100
98	99	34
21	56	12
23	89	
19	51	

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 174 02/02/12 4:38 PM user-f396 F-396 EQA

8.4 Lower-Level File I/O

Some text files contain data in mixed format that are not readable by the high-level file reading functions. The MATLAB language provides a set of lower-level I/O functions that permit general-purpose text file reading and writing. The following is a partial discussion of these functions that is sufficient for most text file processing needs. In general, the file must be opened to return a value to be used by subsequent functions to identify its data stream. We usually refer to this identifier as the "file handle." After the file contents have been manipulated, the file must be closed to complete the activity. Because these are lower-level functions used in combination to solve problems, we will need to discuss the behavior of several of them before we can show examples of their use.

8.4.1 Opening and Closing Files

To open a file for reading or writing, use `fh = fopen( <filename>, <purpose>)` where `fh` is a file handle used in subsequent function calls to identify this particular input stream, `<filename>` is the name of the file, and `<purpose>` is a string specifying the purpose for opening the file. The most common purposes are `'r'` to read the file, `'w'` to write it, or `'a'` to append to an existing file. See the help files for more complex situations. If the purpose is `'r'`, the file must already exist; if `'w'` and the file already exists, it will be overwritten; if `'a'` and the file already exists, the new data will be appended to the end. The consequence of failure to open the file is system dependent. In the standard version on a PC, this is indicated by returning a file handle of –1.

To close the file, call `fclose( fh ).`

8.4.2 Reading Text Files

To read a file, three levels of support are provided: reading whole lines with or without the new-line character, parsing into tokens with delimiters, or parsing into cell arrays using a format string.

- To read a whole line including the new-line character, use `str = fgets( fh )` that will return each line as a string until the end of the file, when the value –1 is returned instead of a string. To leave out each new-line character, use `fgetl(...)` instead (the last character is a lowercase `L`).
- To parse each line into tokens (elementary text strings) separated by white space delimiters, use a combination of `fgetl(...)` and the tokenizer function `[<tk>, <rest>] = strtok( <ln> );` where `<tk>` is a string token, `<rest>` is the remainder of the line, and `<ln>` is a string to be parsed into tokens.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 175 02/02/12 4:38 PM user-f396 | F-396 | EQA

- If you are using MATLAB, you could try to parse a line according to a specific format string into a cell array by using `ca = textscan( fh, <format> );` where `ca` is the resulting cell array, `fh` is the file handle, and `<format>` is a format control string such as we used for `sscanf(...)` in Chapter 6.

8.4.3 Examples of Reading Text Files

To illustrate the use of these functions for reading a text file, the script shown in Listing 8.1 shows a script that will list any text file in the Command window.

In Listing 8.1:

Line 1: Asks the user for the name of a file.

Line 2: Opens the file for reading and returns the file handle.

Line 3: Initializes the `while` loop control variable.

Line 4: When the file read reaches the end of the file, the reading function returns –1 instead of a string.

Line 5: Reads a string, including the end of line character.

Line 6: Classic loop-and-a-half logic that determines whether there is a line to process.

Line 7: Displays that line if present.

Line 10: Closes the file when finished.

As an example of the use of a tokenizer, consider the code shown in Listing 8.2, which performs the same function as Listing 8.1 but uses tokens.

Line 5: Uses `fgetl(...)` instead of `fgets(...)` because the tokenizer does not need the new-line character.

Line 7: Initializes the resulting cell array.

Line 8: The tokenizer will be finished when it leaves an empty line as the result.

Line 9: Creates a token from the remains of the line and puts the remains back into the variable `ln`.

Listing 8.1 Script to list a text file

```
 1. fn = input( 'file name: ', 's' );
 2. fh = fopen( fn, 'r' );
 3. ln = '';
 4. while ischar( ln )
 5.     ln = fgets( fh );
 6.     if ischar( ln )
 7.         fprintf( ln );
 8.     end
 9. end
10. fclose( fh );
```

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 176 02/02/12 4:38 PM user-f396 | F-396 | EQA

Listing 8.2 Listing a file using tokens

```
 1. fn = input( 'file name: ' , 's' );
 2. fh = fopen( fn, 'r' );
 3. ln = '';
 4. while ischar( ln )
 5.     ln = fgetl( fh );
 6.     if ischar( ln )
 7.         ca = [];
 8.         while ~isempty( ln )
 9.             [tk, ln] = strtok( ln );
10.             ca = [ca {tk}];
11.         end
12.         disp( ca );
13.     end
14. end
15. fclose( fh );
```

Line 10: Adds the current token to the result.

Line 12: Shows the tokens for one line.

Line 15: Closes the file.

Run the scripts in Listings 8.1 and 8.2. This will show the difference in output results between the conventional listing script and the tokenizing lister. With the tokenizer, we see each individual token (really, each word in a normal text file) separately listed.

8.4.4 Writing Text Files

Once a file has been opened for writing, the `fprintf(...)` function can be used to write to it by including its file handle as the first parameter. For example, Listing 8.3 is a minor alteration to Listing 8.1, copying a text file instead of listing it in the Command window.

Listing 8.3 Script to copy a text file

```
 1. ifn = input( 'input file name: ', 's' );
 2. ofn = input('output file name: ', 's' );
 3. ih = fopen( ifn, 'r' );
 4. oh = fopen( ofn, 'w' );
 5. ln = '';
 6. while ischar( ln )
 7.     ln = fgets( ih );
 8.     if ischar( ln )
 9.         fprintf( oh, ln );
10.     end
11. end
12. fclose( ih );
13. fclose( oh );
```

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 177 02/02/12 4:38 PM user-f396 | F-396 | EQA

In Listing 8.3:

Line 2: Fetches the output file name.
Line 4: Opens the output file for writing.
Line 9: Adding `oh` as the first parameter to `fprintf(...)` redirects the output to the specified file.
Line 13: Closes the output file.

8.5 Engineering Example—Spreadsheet Data

- Frequently, engineering data are provided in spreadsheets. Here we will adapt the structure assembly problem from Chapter 7. The script for that solution created the data using a constructor function. Consider the situation in which the data are provided in a spreadsheet such as that shown in Figure 8.3. We have to start by examining the layout of the data and the process necessary to extract what we need. Bearing in mind the three results returned from `xlsread(...)`, first we determine which of the three is most appropriate:
- The {`xlsread(...)`} function is going to include all the numerical cells from the spreadsheet in the numerical array. This is awkward because there are numbers in the first column; and since the primary interest in this problem is not the numerical data, we will not use the numerical array directly.
- However, this is not exclusively a text processing problem. Since we need the numerical coordinates, the second, text-only result is not what we need.
- Therefore, in this particular application, we will process the raw data provided by `csvread(...)`, giving both the string and numerical data.

The other concern is that there are a different number of connections on each row of the sheet. When a connection is present, it is a string. When it is

	A	B	C	D	E	F	G	H	I	J
1	Item	Name	X	Y	Connected to					
2	1	A-1	0.866	0.5	A	A-2	A-3	D-1		
3	2	A-2	0	1	A	A-3	B-1	B-2		
4	3	A-3	0.866	1.5	A-1	A-2	B-1	D-1		
5	4	B-1	0.866	2.5	A-2	A-3	B-2	B-3	D-1	D-2
6	5	B-2	0	3	A-2	A-3	B-1	B-3	C-1	C-2
7	6	B-3	0.866	3.5	B-1	B-2	C-1	C-2	D-1	D-2
8	7	C-1	0.866	4.5	B-2	B-3	C-2	C-3	D-2	
9	8	C-2	0	5	B-2	B-3	C-1	C-3	C	
10	9	C-3	0.866	5.5	C-1	C-2	D-2	C		
11	10	D-1	1.732	2	A-1	A-3	B-1	B-3	D-2	
12	11	D-2	1.732	4	B-1	B-3	C-1	C-3	D-1	

Figure 8.3 *Data in a spreadsheet*

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 178 02/02/12 4:38 PM user-f396 F-396 EQA

not there, we refer to the behavior of the raw data to discover that the contents of empty cells appear as [] of type `double`.

We need a function that will read this file and produce the same model of the structure used in Chapter 7. Such a function is shown in Listing 8.4.

In Listing 8.4:

Line 1: The function consumes the file name and produces a structure array with the fields described in the following comments.

Line 2: Reads the spreadsheet and keeps only the raw data.

Line 3: Gets the rows and columns in the raw data; we need to ignore the top row and left column.

Line 4: Initializes the output index for the structure array.

Line 5: Ignoring the first row, traverses all the remaining rows.

Line 6: The component name is in the second column.

Line 7: The coordinates of the component are in the third and fourth columns.

Listing 8.4 Reading structure data

```
1. function data = readStruct(filename)
   % read a spreadsheet and produce a
   % structure array:
   % name - the second column value
   % pos - columns 3 and 4 in a vector
   % connect - cell array with the remaining
   % data on the row

2.     [no no raw] = xlsread(filename);
3.     [rows cols] = size(raw);
       % ignore the first row and column
4.     out = 1;
5.     for row = 2:rows
6.         str.name = raw{row,2};
7.         str.pos = [raw{row,3} raw{row,4}];
8.         cni = 1;
9.         conn = {};
10.        for col = 5:cols
11.            item = raw{row, col};
12.            if ~ischar(item)
13.                break;
14.            end
15.            conn{cni} = item;
16.            cni = cni + 1;
17.        end
18.        str.connect = conn;
19.        data(out) = str;
20.        out = out + 1;
21.    end
22. end
```

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 179 02/02/12 4:38 PM user-f396 | F-396 | EQA

Lines 8–9: Initialize the search for the connections for this component. It is important to empty the array `conn` before each pass to avoid "inheriting" data from a previous row.

Lines 10–11: Extract each item in turn from the row.

Lines 12–14: If the item is not of class `char`, this is the blank cell at the end of the row; the `break` command exits the `for` loop moving across the row.

Lines 15–16: Otherwise, it stores the connection and keeps going.

Lines 18–20: When the connections are complete, it stores them in the structure, stores the structure in the structure array, and continues to the next row.

Line 21: When the rows are completed, the data are ready to return to the calling script.

To test this function, replace the structure array construction in lines 1–22 of Listing 7.6 in Chapter 7 with the following line:

```
data = readStruct('Structure_data.xls');
```

The script should then produce the same results as before.

Chapter Summary

We have described three levels of capability for reading and writing files:

- The save and load operators allow you to save variables from the workspace and restore them to the workspace
- Specialized functions read and write spreadsheets and delimited text files
- Lower-level functions provide the ability to open and close files, and to read and write text files in any form that is required

Special Characters, Reserved Words, and Functions

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`[nums,txt,raw] = csvread(<file>)`	Reads comma-separated text files	8.3
`csvwrite(<file>,<data>)`	Writes comma-separated text files	8.3
`dlmread(<file>,<dlm>)`	Reads text files separated by the given delimiting character	8.3
`dlmwrite (<file>, <data>, <dlm>)`	Reads text files separated by the given delimiting character	8.3

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 180 02/02/12 4:38 PM user-f396 F-396 EQA

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`fclose(fh)`	Closes a text file	8.4.1
`fgetl(fh)`	Reads a line, omitting the new-line character	8.4.2
`fgets(fh)`	Reads a line, including the new-line character	8.4.2
`fh = fopen(<file>, <why>)`	Opens a text file for reading or writing	8.4.1
`fprintf(...)`	Writes to the console, or to plain text files	8.3, 8.4.4
`load <filename>`	Loads the workspace from a file	8.2
`save <filename>`	Saves workspace variables in a file	8.2
`[tk rest] = strtok (<str>, <dlm>)`	Extracts a token from a string and returns the remainder of the string	8.4.2
`ca = textscan(fh, format)`	Acquires and scans a line of text according to a specific format	8.3, 8.4.2
`[nums, txt, raw] = xlsread(<file>)`	Reads an Excel spreadsheet	8.3.2
`xlswrite(<file>,<data>, <sheet>, <,range>)`	Writes an Excel spreadsheet in a specific row/column range	8.3.2

Self Test

Use the following questions to check your understanding of the material in this chapter:

True or False

1. All data files should be treated as a sequential series of characters.
2. When you save a workspace, you are actually saving the scripts that generate the data in the workspace.
3. MATLAB reads strings from tab- or comma-delimited files by recognizing the double quotes that delimit strings.
4. If you use `fopen(...)` to open an existing file and write to it, the original data in the file will be overwritten.
5. The function `fgets(fh)` does not always return a string.

Fill in the Blanks

1. In general, data files contain text that represents the _____ of the data and control characters that specify the _____ of the data.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 181 02/02/12 4:38 PM user-f396 F-396 EQA

2. The MATLAB `xlsread(...)` function returns three results: the _____ in a(n) _____, the _____ in a(n) _____, and _____ in a _____.
3. When using `dlmread(...)` to populate a(n) _____, any unassigned values are _____.
4. When using `fopen(...)`, the consequence of failure to open the file is _____.

Programming Projects

1. Write a script that performs the following operations:
   a. Set the value of variables `a, b, c1, c2, c3,` and `x`. The values don't matter, except you should set `c2` to `42`.
   b. Save the values of all the variable except `x` to `mydata.mat` using the save operation.
   c. Set the value of `c2` to `-99`.
   d. Load `myData.mat` and check that `c2` is now `42`.
   e. Clear all variables.
   f. Load `myData.mat` again and note that the variable `x` is not present.
2. One requirement for all freshmen classes is an issue of a `'Standing'` during the middle of the term. The results are either Satisfactory `(S)` or Unsatisfactory `(U)`. Since you are the office employee in charge of issuing these grades, you decide to write a function called `midtermGrades` to help yourself. You discover that the grades are on a spreadsheet organized like this:
   - Each student is represented by one row on the spreadsheet.
   - Unfortunately, since these sheets are created by different instructors, they are not necessarily consistent in their layout.
   - The first row will contain the following six strings in any order:

     `'name', 'math', 'science', 'english', 'history', and 'cs'.`

   - Under the name column will be a string with the student's name.
   - Grades in the other columns can be `'A', 'B', 'C', 'D', 'F',` or `'W'`.
   - A student's grade is `'S'` if there are more A's, B's and C's than not.

   Your function should print out grades ready to be entered consisting of a table with headings `'Name'` and `'S/U'`
3. Write a function called `genStats` that will compute statistics for a set of class grades. The grades will be stored in a spreadsheet, and your function will compute statistics and then write the grades along with the statistics to another spreadsheet.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 182 02/02/12 4:38 PM user-f396 F-396 EQA

You may assume that the initial spreadsheet will have a format similar too:

Student Name	Exam1	Exam2	Exam3	...
student 1	100	76	45	...
student 2	34	83	89	...

The first row is the header row, and the first column is the list of student names. There may be any number of exam grades, and there may be any number of students. Although, you may assume that there will be at least one student and that there will be at least one exam.

Also, every student will have a grade for every exam.

Your function should only have one input (a string containing the file name of the grades file) and no outputs. You must write your function to perform the following steps:

a. Calculate the average grade of each student (across the rows) and store it in a new column called `'Average'` (to the right of the last exam grade).
b. Calculate the deviation of each student's overall average (calculated in step a) from the maximum student average and store it in a new column called `'Deviation'` (to the right of the `'Average'` column). Note that deviation is just the difference between the maximum student average and a student's overall average.
c. Calculate the average of each column's data (each exam), the averages calculated in step a, and the deviations calculated in step b, then store these averages below the last row of the original data and name that row `'Total Average'`.
d. Write the original data along with all of the new data to a file named `'Stats_<name_of_original_file>'` (so if the inputted file name was `'Student_Grades.csv'`, the new data would be written to the file named `'Stats_Student_Grades.csv'`).
e. Construct a spreadsheet with suitable test data and use it to test your function.

4. Write a function called `replaceStr`. Your function should take in the following order:

`filename:` A string that corresponds to the name of a file

`wordA:` A string that is a word (contains no spaces)

`wordB:` Another string that is also a word (contains no spaces)

Your function should do the following:
a. Read the file a line at a time.
b. On each line, replace every occurrence of `wordA` with `wordB`.

72 DPI M08_SMIT8708_03_SE_C08.indd Page 183 02/02/12 4:38 PM user-f396 F-396 EQA

c. Write the modified text file with the same name as the original file, but preprended with `'new_'`. For instance, if the input filename was `'data.txt'`, the output filename would be `'new_data.txt'`.
d. Prepare a test file by downloading a text file from the Internet. For example, the complete works of Shakespeare are accessible at http://www.william-shakespeare.info
e. Examine the file for repeated words, and test your function by writing a script that replaces frequently repeated words.

72 DPI | M08_SMIT8708_03_SE_C08.indd Page 184 02/02/12 4:38 PM user-f396