

MM 319 Lab Seven: Line-by-Line Parsing

Introduction

In this activity, you will take the first steps toward using an external text file to control what appears on the screen. This activity provides the foundation for your second programming assignment, so pay close attention and make sure that you understand what is happening. The very first step is to look at a set of text operations called **chunk expressions**.

Chunk Expressions

A block of text is composed of many pieces, and can be divided in many ways. Although we can't multiply or divide text, there are useful operations that are legal. These allow us to manipulate parts of text strings, whether literals or contained in a variable. We won't look at all of the types of chunk expressions yet—just a couple.

One operation returns the number of lines (as delineated by carriage returns) that a string contains. The operation is **the number of lines in**. Clever name, huh? What follows the operator is the string, which may be a variable or the identifier for a text cast member. We can assign the value returned by this operation to a variable:

```
linesInField = the number of lines in member("textInput")
```

Another operation allows us to copy a "chunk" of one or more lines for use. This is the **line** operator. The syntax is *stringObjectReference.line [startLine..endLine]*. The "stringObjectReference" can refer to a cast member **or** a variable.

The following hands-on exercise should make the use of these operations clearer.

Experiment One

Create a new Director file with a stage of 640 by 480. Place a single text box on the stage, and put several lines of text in it, pressing the **return** key at the end of each line. Make one line long enough to wrap, so that it *looks* like more than one line. Now open the message window, and enter these commands. Pay attention to the results.

```
put the number of lines in member(1).text
put member(1).line [2]
put member(1).line [2..3]
lineCount = the number of lines in member(1).text
put member(1).line[1..lineCount]
```

Notice that the number of lines, as far as Director is concerned, is based on the number of carriage returns, not the appearance of the text box. Notice that you can access a single line or a range using the line **operator**. Now try these:

```
member(1).line[1] = "Georgie Porgie, puddin' and pie"
member(1).line[3] = "Kissed the girls and made them cry"
```

So we can modify individual lines of a text box. As the following statements illustrate, we can manipulate the individual lines of text variables as well, although the syntax is different.

```
x = member(1).text -- load screen contents into x
```

```

put x
put "And now for something completely different" into x.line[2]
put x

```

Experiment Two

This activity will build on what you learned in the previous lab about file input. Suppose we want to build the contents of the screen line by line, getting the content from an external text file? We will add a line each time the mouse is clicked, until there is no more text.

Create a file with five to seven lines of text in MS-Word, and use **Save as...** to save this file in plain text with line breaks format. Name it whatever you wish—the rest of this example will use the name **test.txt**.

Use the Director file you created for the previous experiment. Here is the information we'll be using. We must store the text from the file, so we'll need a variable for that. We'll need to keep track of which line of the file we are on, and which line of the screen we are on. We also need to remember how many lines of text there are. This makes three more variables. Since all of this information is needed for all the handlers, we'll declare them as globals. Armed with this information, you are now ready to input the movie script:

```

global theText, lastLine, screenLine, currentLine

-- open the text file and read it into a variable
-- initialize line counters and clear the screen
--
on prepareMovie
  fileHandle =new xtra("fileio")

  fileHandle.openFile("test.txt",1)
  theText = fileHandle.readFile()
  filehandle.closeFile()
  fileHandle = VOID

  lastLine = the number of lines in theText
  screenLine = 0
  currentLine = 0

  member(1).text = ""
end

-- if there are any more lines of text,
-- increment line counters and display the next line
--
on mouseUp
  if currentLine < lastLine then
    currentLine = currentLine + 1
    screenLine = screenLine + 1

    member(1).line [ screenLine ] = theText.line[ currentLine ]

  end if
end

```

MM 319 Lab Seven

Once you have entered this script, examine it and figure out how it works. Compile the script, and correct any typos. Now bring the stage into view, and the message window, and enter the message **prepareMovie**. If you get an error, it is possible that you don't have your text file save in the same location you have saved the Director movie, or that the name doesn't match the name in the script.

When you can send the message with no errors, go to the stage and play the movie. Every time you click the mouse on the stage, another line of text should appear. Believe it or not, this is the beginning of some very big stuff!

More Questions to Ponder

Here are some things to think about. What if there were too many lines of text in the file to fit on the screen? We would need to clear the screen and start over again when we ran out of space. How would you do this? What additional variables are necessary? *Think about it!*