APPLESCRIPT FOR ABSOLUTE STARTERS

By Bert Altenburg

INTRODUCTION

AppleScript is a revolutionary Apple technology that makes communication between computer programs possible. For example, with AppleScript you can
- retrieve e-mails from Mail and store them in a database;
- tell a picture editing program to change the resolution of a series of pictures, resize them, and send the resulting pictures to another computer or post them on the Web;
- and much, much more.

An AppleScript, or script for short, is a series of written instructions in a scripting language named AppleScript. This language resembles the English language, making AppleScripts both easy to read, write and understand.

Despite its power, AppleScript is heavily used in a couple of fields only. The publishing industry depends on it for workflow automation (PhotoShop, QuarkExpress, InDesign). Filemaker Pro developers use it for creating Mac-based kiosks, which you can find in malls and museums (k-Builder). Apart from the programs mentioned, many more major and minor Macintosh programs like GraphicConverter, BBEdit, and Word are AppleScriptable. That means you can use AppleScript to boss these programs around. Scripting applications is not the focus of this book, however. There are other books on the market that show you how to do that. If these books provide an introduction to AppleScript, it is usually cursory and they quickly dash to the really juicy stuff, which generally requires a modest or good knowledge of the basics of AppleScript. The aim of this book is to provide you just that.

It is intended to update and expand this book on a regular basis. So, you may want to check for new versions (see Chapter 15). A second book on scripting various programs is considered. This book is freeware, and you are encouraged to bring it to the attention of other Macintosh users. In this respect, please pay attention to Chapter 0 in this book on how you can promote the Mac.

Once you dive into the world of AppleScript, you'll notice that the term 'AppleScript' is used quite loosely for three different concepts.

- The AppleScript language: The English-like language which is used to give written instructions to your Mac;
- An AppleScript: A series of instructions, a.k.a. a script, written in the AppleScript language; and
- A part of the Mac operating system (Mac OS X), which actually reads an AppleScript and executes the instructions containing it.

In this book, if there is need to refer to one of these three concepts specifically, the following terms are used respectively:
- The AppleScript language;
- An AppleScript or a script (noun);
- The AppleScript component of Mac OS X.

Learning how to script with AppleScript is ideal as an introduction to programming. It leaves out most of the nitty-gritty work a programmer in a computer language such as Java has to do before she can even perform the easiest of tasks. AppleScript is easy enough that a 10 year old can learn it, yet so powerful that professionals enjoy it too. That leaves plenty of room for growth for you. While not covered in this book, you can even use AppleScript to build computer programs that look and work just like the commercial programs you use on your Mac, with buttons, menus, scrollbars and all. This requires AppleScript Studio, provided for free by your favorite computer company.

What is the difference between scripting and programming? I'd like to think that if it is easy, it is scripting and if it is difficult, it is programming. However, javascripting is not easy in my book, so perhaps that definition is wonky.

How to use this book?
As you will see, some paragraphs are displayed in a green font. We suggest you read each chapter (at least) twice. The first time, skip the green text. The next time you read the chapters, include the green paragraphs. You will in effect rehearse what you have learned, but learn some interesting tidbits which would have been distracting the first time. By using the book in this way, you will level the inevitable learning curve to a gentle slope.

This book contains dozens of script examples. To make sure you link an explanation to the proper script, every script is labeled by a number placed between square brackets, like this: [4]. Most scripts consist of two or more lines. At times, a second number is used to refer to a particular line. For example, [4.3] refers to the third line of script [4].

You will not learn riding a horse by reading a book. Similarly, you will not learn AppleScript if you don't get your mitts on your Mac. This is an electronic book. You have no excuse for not switching to the Script Editor (see Chapter 2).

CHAPTER 0

BEFORE WE START

I wrote this book for you. As it is free, please allow me to say a couple of words on promoting the Mac in return. Every Macintosh user can help to promote their favorite computer platform with little effort. Here is how.

1) The more efficient with your Mac you are, the easier it is to get other people to consider a Mac. So, try stay up to date by visiting Mac-oriented websites and reading Mac magazines. Of course, learning AppleScript and putting it to use is great too. For companies, the use of AppleScript can save tons of money and time.

2) Show the world that not everybody is using a PC by making Macintosh more visible. Wearing a neat Mac T-shirt is one, but you can even make the Mac more visible from within your home. If you run CPU monitor (in the Utilities folder which you find in the Applications folder on your Mac), you will notice that your Mac uses its full processing power only occasionally. Scientists have initiated several distributed computing (DC) projects, such as Folding@home, that harness this unused processing power. You download a small, free program, called a DC client, and start processing work units. These DC clients run with the lowest level of priority. If you are using a program on your Mac and that program needs full processing power, the DC client immediately takes a back seat. So, you will not notice it is running. How does this help the Mac? Well, most DC projects keep rankings on their websites of work units processed. If you join a Mac team (you'll recognize their names in the rankings), you can help the Mac team of your choice to move up the rankings. So, users of other computer platforms will see how well Macs are doing. There are DC clients for many topics, such as math, curing diseases and more. To choose a DC project you like, check out
www.aspenleaf.com/distributed/distrib-projects.html
One problem with this suggestion: It may become addictive.

3) Make sure the Macintosh platform has the best software. No, you don't have to learn programming. Just make it a habit to give (polite) feedback to the developers of programs you use. Even if you tried a piece of software and didn't like it, tell the developer why. Report bugs by providing a description of the actions you performed when you experienced the bug. For a great free multimedia-based tutorial on how to do this, visit
www.macinstruct.com/tutorials/crash/index.html

4. Pay for the software you use. As long as the Macintosh software market is viable, developers will provide the software.

5. Please contact at least 3 Macintosh users who don't know about this book and tell them where to find it. Or advise them about the above 4 points.

OK, while you download a DC client in the background, let's get started!

CHAPTER 1

A SCRIPT IS A SERIES OF INSTRUCTIONS

AppleScript as part of the Macintosh Operating System can perform only a very limited number of tasks. For example, it can produce a beep. Let's take a look at the script [1] needed to make your Mac beep.

[1]
```
beep
```

This must be the world’s shortest script, consisting of a single command or instruction. A line containing an instruction is called a statement, even if that line is just one word long. If the above script is executed by your Mac, your Mac beeps once.

To have more beeps than just one, you may provide the beep command with a number, which number indicates the number of beeps you want to hear [2].

[2]
```
beep 2
```

As you can see by comparing scripts [1] and [2], this additional piece of information is optional. If you don’t provide a number, AppleScript assumes you want just one beep. So, 1 is the default value.

If you think beeps are PeeCee-ish, why don’t we let AppleScript communicate with you the Macintosh way [3], using the following statement:

[3]
```
say "This is a spoken sentence."
```

You may even select another voice, such as “Fred”, “Trinoids”, “Cellos”, or “Zarvox” [4], to replace the default voice “Victoria”.

[4]
```
say "This is a spoken sentence." using "Zarvox"
```

#Note: Generally, AppleScript is not case sensitive. That is, it doesn’t mind if you use capitals or not. However, the voices, such as “Victoria”, and “Zarvox” must be properly capitalized. Grrr.#

As you can see, AppleScript instructions resemble English, making the script quite readable and understandable, even if you have never had any scripting experience. But while the scripts [1-4] are probably fun; they are not very useful. The AppleScript language has a couple more commands, but probably not much to impress you. AppleScript derives its strength from the fact that it allows you to communicate with other programs. This works if these programs are AppleScriptable. Fortunately, many Macintosh programs are. As a result, you have at your disposal not only the, granted, limited command set of the AppleScript component of Mac OS X, but also the vast number of commands provided by your programs.

Some Mac programs are more popular than others. One is used by every Macintosh user: the Finder. Yes, the Finder is a program. When you turn on your Mac, it starts-up automatically and it is always running. It allows you to move files around, find files on your hard disk, create folders, copy and rename them and much more. For example, if you empty the trash, it is the Finder that does it for you. While you can perform the empty-the-trash operation with the mouse or keyboard, you can do it with an AppleScript [5] too.

[5]
```
tell application "Finder"
	empty the trash
end tell
```

Like a boss, you must tell
- who is to perform a task, and
- which task is to be performed.
It is no use telling, for example, PhotoShop that it is to empty the trash. PhotoShop does not know how to do that. So, the instruction to empty the trash must be conveyed to the Finder.

Like in the real world, the job a boss has ordered you to do may be less than wise, but your Mac is a most faithful employee, and does what it is told. If there were an important file in the trash, once you have executed the above AppleScript [5], you have lost it forever.

The first statement [5.1] is the ‘tell’ statement where we ask the AppleScript component of Mac OS X to convey one ore more statements to another program, here to the Finder. The AppleScript component of Mac OS X keeps doing that, until it encounters the obligatory ‘end tell’ statement [5.3]. In the above script [5] we order AppleScript to send the Finder the instruction to empty the trash and then to stop telling the Finder what to do. Taken together, the lines

```
tell application "xyz"

end tell
```

are called a ‘tell block’. The instruction to be executed by program ‘xyz’ is inside the tell block for program ‘xyz’. By the way, while the AppleScript language is not very finicky when it comes to notation in comparison with other scripting and especially programming languages, it is not without a couple of rules. One of the rules is that you must use double quotes around the application’s name, as in the first statement [5.1].

It is also possible to give the Finder more instructions. In the example [6] below, there are two statements [6.2, 6.3] destined for the Finder. Because they both are to be performed by the Finder, they must be inside the tell block for the Finder.

[6]
```
tell application "Finder"
	empty the trash
	open the startup disk
end tell
```

After emptying the trash, the Finder opens a window showing you the content of your hard disk.

As you can see, we can make the Finder do whatever we want to. We can even tell the Finder to resize the Finder window, put it at a desired position on the screen and way, way more. You will learn how to do that later on.

We can now create a script containing both instruction statements for the Finder, and for the AppleScript component of Mac OS X itself [7].

[7]
```
tell application "Finder"
	empty the trash
	open the startup disk
end tell
beep
```

First, the Finder is given a couple of instructions [7.2, 7.3]. Then the ‘beep’ instruction [7.5] is executed by AppleScript. In effect, it indicates that the script has been executed.

Interestingly, it doesn’t matter whether you put the beep instruction (and any other instruction understood by the AppleScript component of Mac OS X itself) inside or outside the tell block [8].

[8]
```
tell application "Finder"
	empty the trash
	beep
	open the startup disk
end tell
```

While the Finder doesn’t know the beep command, the AppleScript component of Mac OS X knows how to deal with it. This makes the script easier to read and understand. Otherwise, you would have to have a first tell block containing the first Finder-executable statement [8.2], then a statement consisting of the beep command, and finally a second tell block for the last Finder-executable statement [8.4].

Mind you, while commands understood by the AppleScript component of Mac OS X may be anywhere in a script, each and every instruction for a particular program, such as the Finder, <u>must</u> be within the tell block for that program. The following script [9] contains a fatal flaw (the last statement [9.5]).

[9]
```
tell application "Finder"
	empty the trash
	beep
end tell
open the startup disk
```

The AppleScript component of Mac OS X does not know how to open the startup disk, and is not willing to look for a program that can do it. The first part of this script (statements [9.2-3] within the tell block) will be executed nicely, but the last statement [9.5] can not be executed.

In a running script, once a problem is encountered, any further statements are not executed [10].

[10]
```
tell application "Finder"
	empty the trash
end tell
open the startup disk
say "I emptied the trash and opened the startup disk for you" using "Victoria"
```

After emptying the trash, the AppleScript component of Mac OS X will stop at the statement [10.4] that should have been addressed to the Finder. You will not hear the sentence of statement [10.5] being spoken, even though there is nothing wrong with the statement.

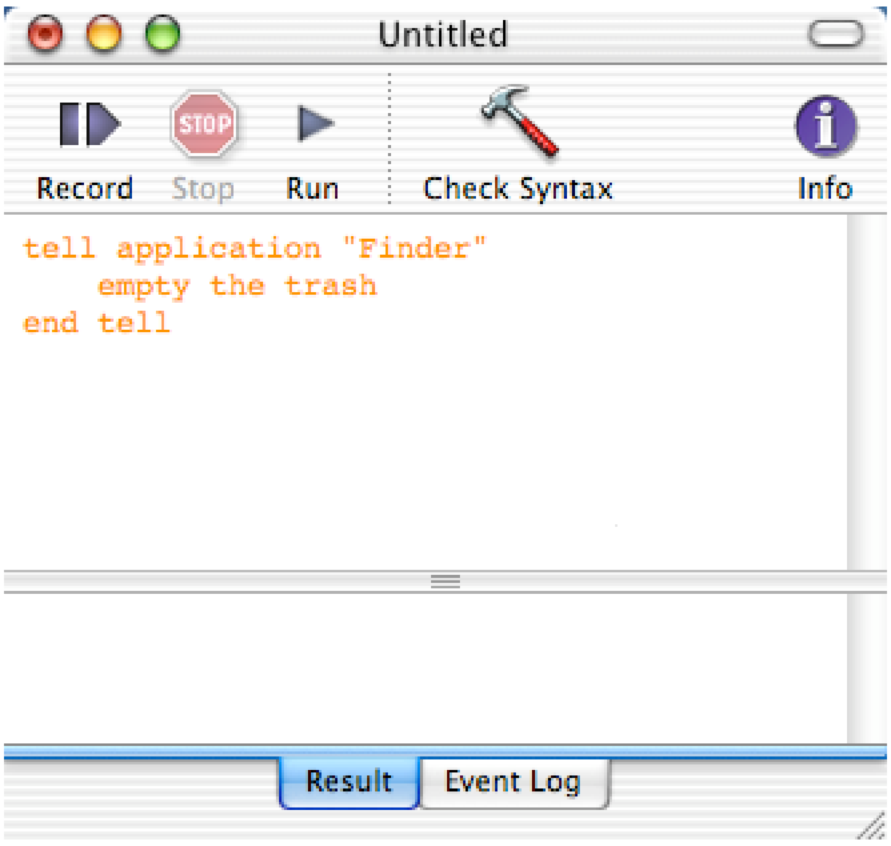
CHAPTER 2

EXECUTING AND SAVING A SCRIPT

You have seen a couple of scripts now., and there is no denying that they are very similar to English, making the scripts easy to read and understand. You could have performed commands in the script - like empty the trash - yourself, using mouse and keyboard. Let's see how your Mac can do that for you.

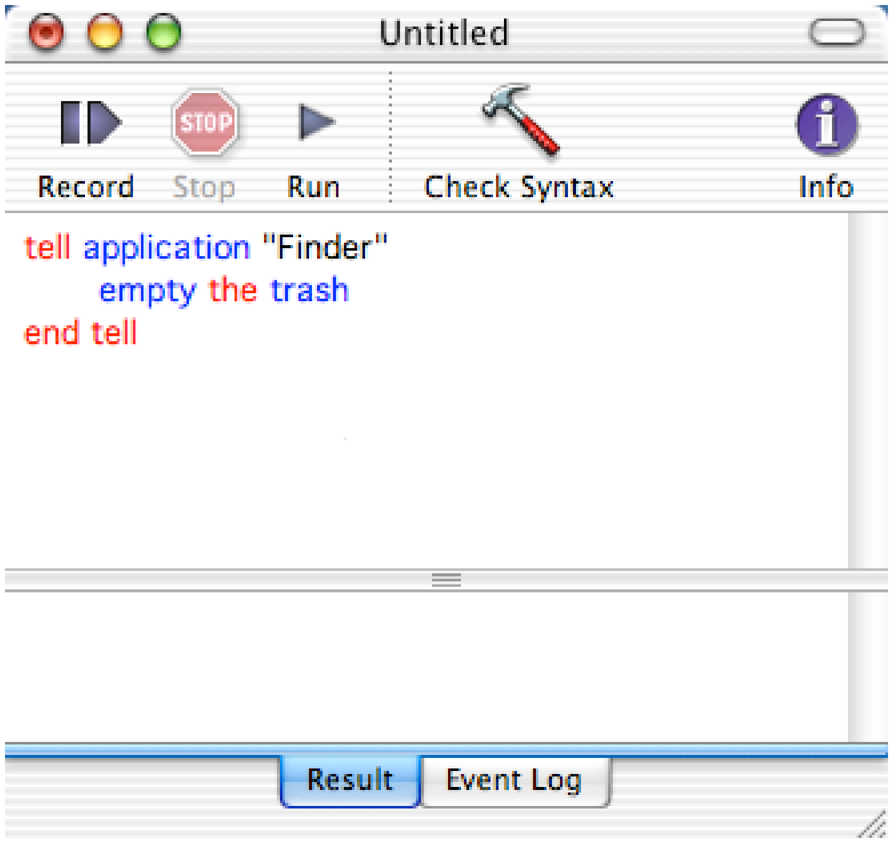
The Script Editor is a program where you can type in a script and execute it. You can find the Script Editor in the AppleScript folder that in turn is in the Applications folder. After starting it up, you will see two fields. The upper one is where you should enter the script [1].

[1]



```
tell application "Finder"
    empty the trash
end tell
```

Near the middle you will see a button labeled 'Check Syntax'. While an AppleScript may look like English, the AppleScript language is far from fully versed in English. "Yo Finder! Dump my garbage" or "Hey Finder, clean out the bin" is not what the Finder expects. By checking the syntax of your script, a.k.a. compilation, the AppleScript component of Mac OS X performs a course check whether it understands your script. If it thinks it does, it formats your script in a nice, colorful way. Uncompiled text is shown in orange, while after compilation the reserved keywords are displayed in red or blue.



```
tell application "Finder"
	empty the trash
end tell
```

If the script does not compile because you made a mistake, you will see a cryptic message indicating that there is something wrong. Try leaving out one of the double quotes in script [2] and see for yourself that the AppleScript component of Mac OS X no longer understands what you mean.

[2]
```
say "I'm learning AppleScript the easy way!" using "Zarvox"
```

If all is well, you may press the Run button, and your script is executed. Now fire up the Script Editor, peck in one of the scripts you've seen here, and try it!

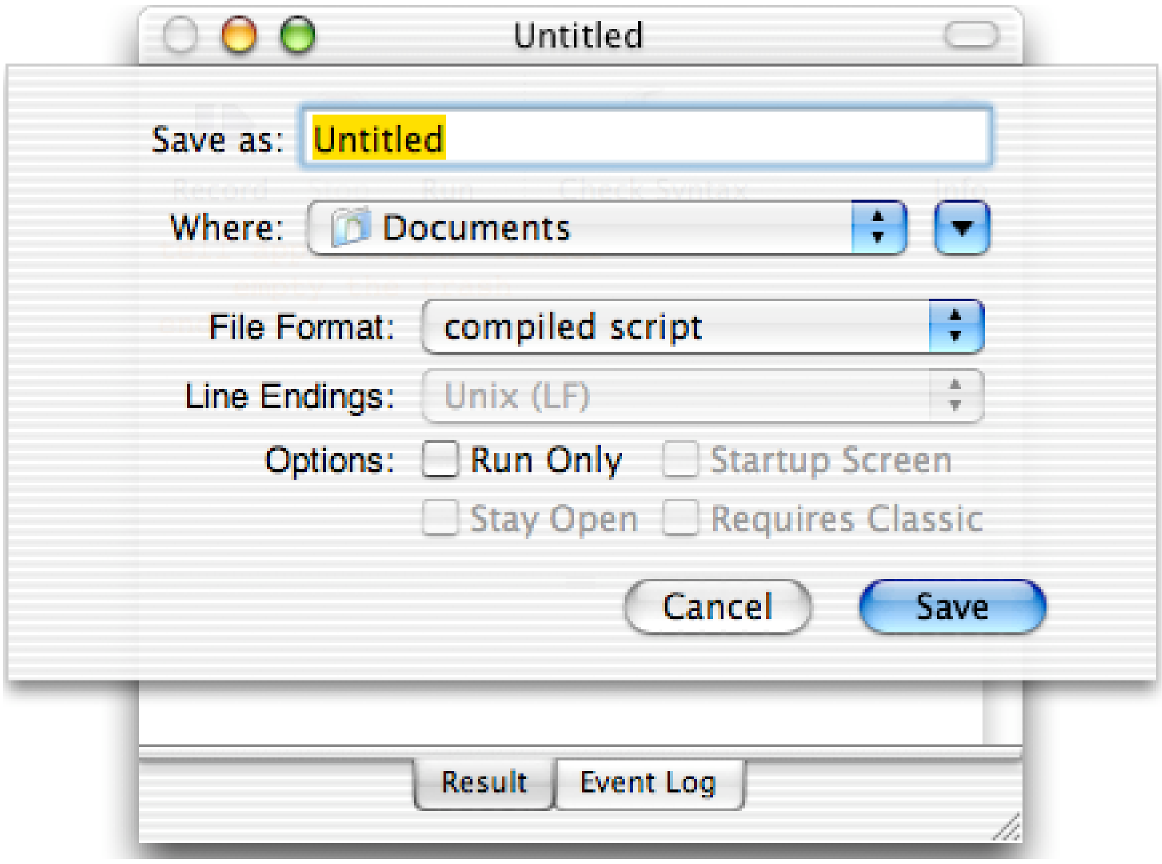
#You may hit the Enter key as a shortcut to check the syntax of your script. The Enter key is the key at the right next to your spacebar (for laptops) or numerical keypad (desktop Macs). The Return key (above the right shift key) works as you would expect, and creates a new line after the current line. You can not use the Return key to check the syntax.

It is not really necessary to press the Check Syntax button before you run the script. If you press the Run button, the script's syntax is checked and, if ok, the script is immediately executed.

Instead of clicking the Run button, you may press Command-R.#

Saving your script
There are several ways to save your script. If the script did not pass the syntax check succesfully, your only option is to save your script as mere text.

If syntax checking does not result in any problems, the dialog window below will appear, and you can save your text as a compiled script or as an application.



Untitled

Save as: Untitled

Where: Documents

File Format: compiled script

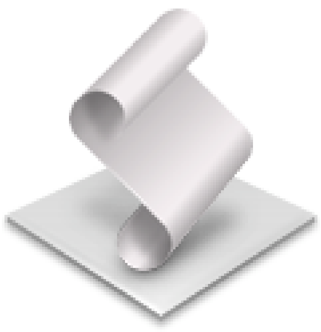
Line Endings: Unix (LF)

Options: Run Only, Startup Screen, Stay Open, Requires Classic

COMPILED SCRIPT: If you double-click the icon of your saved, compiled AppleScript,

the Script Editor is opened and you can run the script by pressing the Run button.

APPLICATION: If you double-click the icon of your saved, AppleScript application,



the script will be executed immediately. That is, the Script Editor is not opened. Saved as an application, you may use the script as a log-in item (in your System Preferences). After logging in, your Mac will perform the tasks detailed in your script. If you need to edit a script saved as an application, startup the Script Editor, and open the script by selecting Open from the File menu.

#WARNING: Setting the appropriate tick in the Save dialog allows you save your script: as run-only. Make sure you have a backup of your script, because a run-only script can not be opened and edited again. #

CHAPTER 3

EASIER SCRIPTING (I)

In Chapter 1 you came across the following script:

```
[1]
tell application "Finder"
        empty the trash
end tell
```

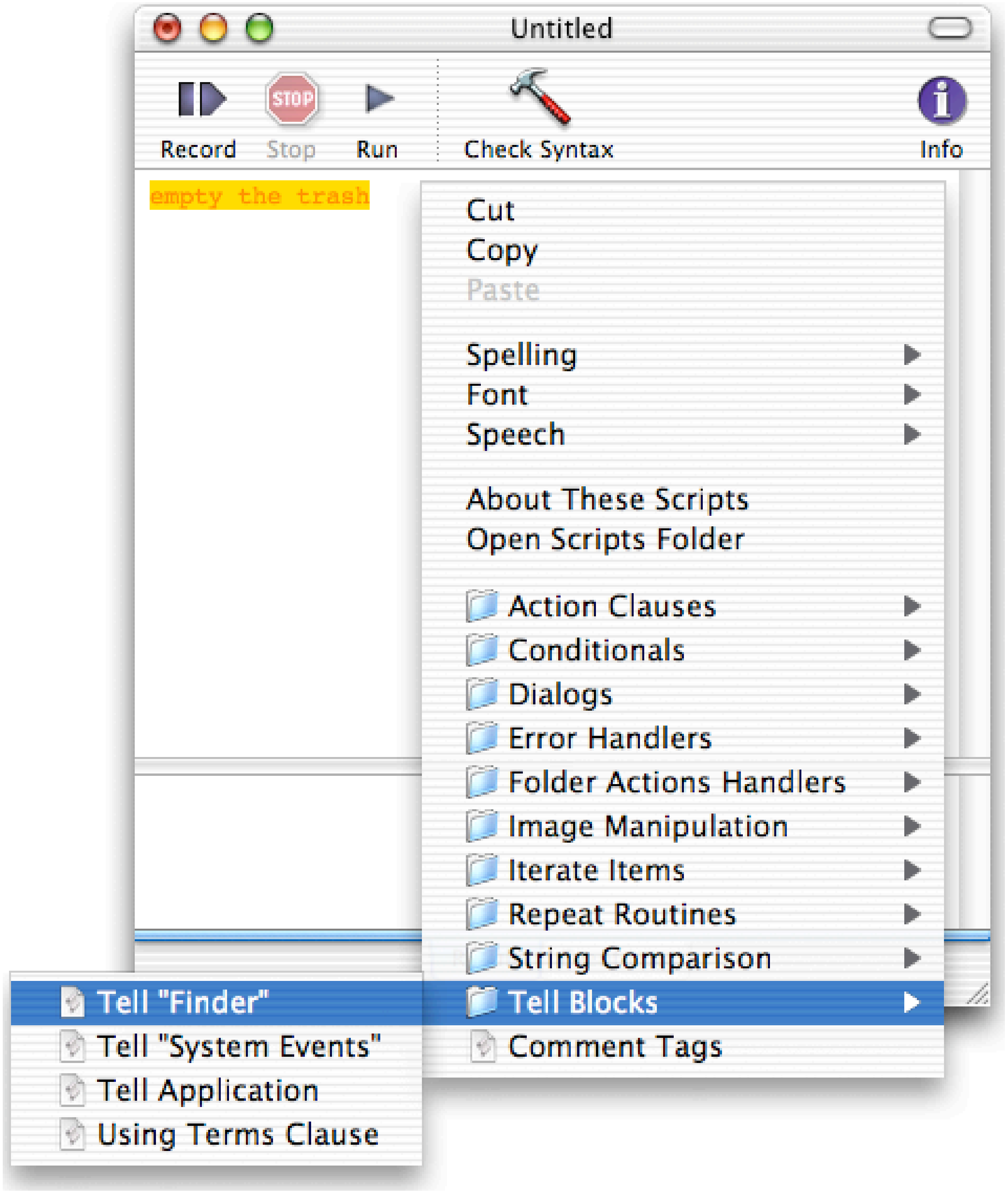
Let us see how AppleScript and the Script Editor try to make it easy for you to script.

In the first line of a tell block, instead of typing the word ‘application’ in full, you may write

```
tell app “xyz”
```

Upon compilation, the Script Editor expands ‘app’ to ‘application’. Even better, you do not need to type or know how to spell the name of the application ‘xyz’ either. Just type anything (provided it is not the name of another application), such as ‘pqr’. If you compile the script, AppleScript will provide you with a list of all scriptable applications on your Mac. You just pick the appropriate application, and AppleScript will replace ‘pqr’ with the correct name of the application, and in effect finish writing the tell statement for you.

Actually, the Script Editor allows you to create a tell block without any typing, by using a contextual menu. That is the kind of menu that appears if you hold down the control-key while clicking. This trick can be performed in two ways:
1) Control-click the upper field of the Script Editor. A menu appears (see picture on the next page), and near the lower end of this menu, you will see a menu item named ‘Tell Blocks’. Click it and a submenu appears from which you can select ‘Tell “Finder”’.
2) If your script already contains one or more statements for the Finder - such as ‘empty the trash’ - which are not yet enclosed by the required tell block, select the statement(s), and then do as under 1). You can see this in action in the picture below. Your statements are automatically enclosed by the tell block.



```
empty the trash
```

CHAPTER 4

DEALING WITH NUMBERS

In primary school you had to do calculations, filling in the dots:

2 + 6 = ...
... = 3 * 4

In secondary school, dots were out of fashion and variables called 'x' and 'y' were all the hype. Looking back, you may wonder why people felt so intimidated by this very small change in notation.

2 + 6 = x
y = 3 * 4

AppleScript uses variables too. Variables are nothing more than convenient names to refer to a specific piece of data, such as a number. Variable names are often referred to as 'identifiers', because they identify a piece of data. Here are two examples [1] of AppleScript statements where a variable is given a particular value, using the 'set' command.

[1]
```
set x to 25
set y to 4321.234
```

While the variable names themselves have no special meaning to AppleScript, to us human beings descriptive variable names can make a script much easier to read and hence easier to understand. That is a big bonus if you need to track down an error in your script (errors in scripts and programs are traditionally called 'bugs'). Hence, avoid using non-descriptive variable names like 'x'. For example, the variable name for the width of a picture could be called "pictureWidth" [2].

[2]
```
set pictureWidth to 8
```

Please note that a variable name consists of a <u>single</u> word (or single character, at a pinch). After checking the syntax, the variable name is displayed in green, so you can immediately see it is not a reserved keyword of AppleScript, which are shown in blue or red. Also, note that data (such as the number '8' in script [2]) is shown in black.

#While you have plenty freedom choosing variable names, there are several rules which a variable name has to conform with. While I could spell them all out, it would be boring. The prime rule you must obey is that your variable name may not be an AppleScript command or any other reserved keyword. For example, 'set', 'say','to', and 'beep' are words that have a special meaning to AppleScript. By compositing a variable name as contracted words, like 'pictureWidth', you are always safe. To keep the variable name readable, the use of capitals within the variable name is recommended.

If you insist on learning a couple of rules, finish this paragraph. Apart from letters, the use of digits is allowed, but a variable name is not allowed to start with a digit. Also allowed is the underscore _.#

Now we know how to give a variable a value, we can perform calculations. AppleScript is capable of performing basic math operations, so there is no need to tell a particular program that it should perform the calculation to determine the surface area of a picture. Here is the script [3] that does just that.

[3]
```
set pictureWidth to 8
set pictureHeight to 6
set pictureSurfaceArea to pictureWidth * pictureHeight
```

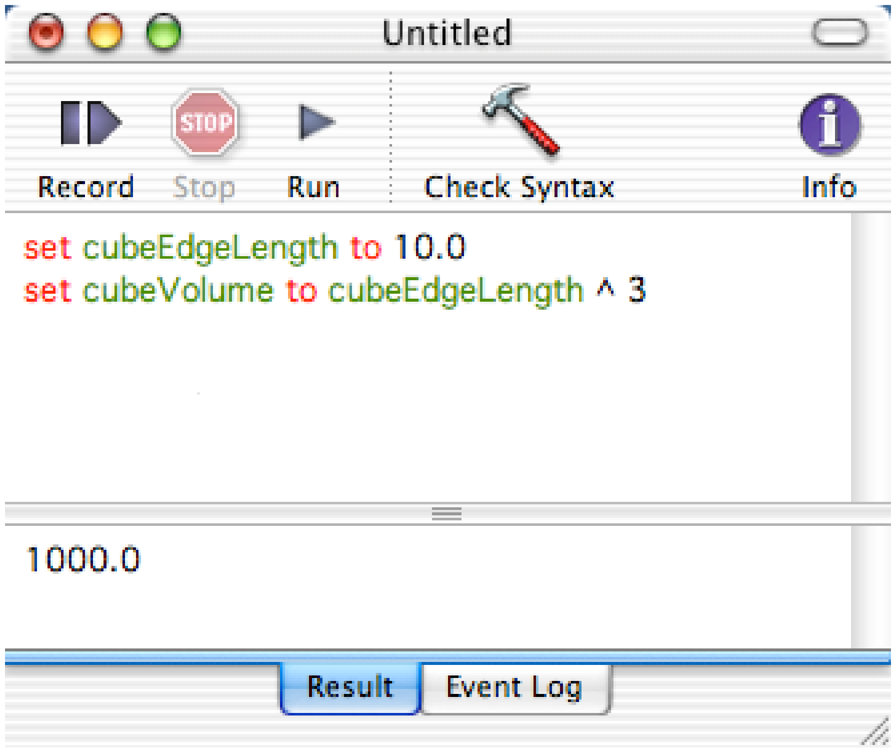
Use the following symbols, officially known as operators, for doing basic mathematical calculations.

+	for addition
-	for subtraction
/	for division
*	for multiplication

The exponent is written using the exponent symbol ^. Here is a script [4] that calculates the volume of a cube.

[4]
```
set cubeEdgeLength to 10.0
set cubeVolume to cubeEdgeLength ^ 3
```

If you run this script [4] in the Script Editor, the result is displayed in the lower field. If you don't see the result, move the horizontal bar near the two tabs up. The result field displays the result of the last statement executed. If your script consists of statement [4.1] only, the result field displays 10.0. For the full script [4], the result displayed is 1000.0. That is, the expression'cubeEdgeLength^3' is evaluated, and the result is displayed.



```
set cubeEdgeLength to 10.0
set cubeVolume to cubeEdgeLength ^ 3
```

```
1000.0
```

Numbers can be distinguished into two types: integers and fractional numbers. You can see an example of each in the statements [1.1] and [1.2], respectively. Integers are used for counting, which is something we will do when we have to repeat an series of instructions a specified number of times (see Chapter 13). You know fractional or real numbers ('reals' for short), for example, from baseball hitting averages. By the way, both integers and reals can be negative, as you know for example from your bank account.