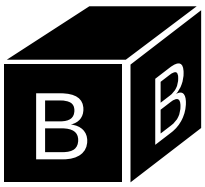


Chapter *1*

The BlackBox Framework

Computers, as useful as they are, have no innate intelligence. Any intelligent behavior that a computer exhibits is a reflection of the program that resides in its memory. That program was conceived, designed, and written by one or more human beings. How friendly a program is to the user, or how useful, or how frustrating is determined largely by the quality of the program design. When we marvel at the power of computers we are paying tribute to programs that are designed well. And when we blame the computer for making a mistake that blame must inevitably fall on a person—either the designer, the programmer, or someone who earlier entered incorrect data into the system. Computers are so new to the human scene that getting them to work efficiently and effectively is a great challenge. Reports of computer failure in the popular press attest to the fact that developing computer systems that are both error free and easy to use is difficult.

The software design problem

The BlackBox system provides the human software designer a tool to develop such systems. This chapter introduces the BlackBox framework and its text subsystem and shows how to encode your BlackBox documents for electronic transmission.

The purpose of the BlackBox system

The BlackBox framework

The BlackBox framework is a powerful tool based on modern software design principles. The framework has many characteristics, four of which are:

Four characteristics of the BlackBox framework

- The Component Pascal language
- Cross-platform capability
- Graphical user interface
- Object-oriented language and system

A framework consists of a complete programming environment including a text editor and all the other tools needed to write and execute programs. Part of the framework is a programming language. In the BlackBox framework, the programming language is Component Pascal. The Component Pascal language is nearly identical to the Oberon-2 language, which is used in systems other than BlackBox. This book is an introduction to both the BlackBox framework and the Component Pascal language.Those features of the language that you will learn will be applicable to other systems that incorporate Oberon-2. However, much of the power of the programming environment is derived from the framework. Although they will be similar, the

features of the framework that you learn will not in general be applicable to other programming environments.

The Component Pascal language

The programming language Component Pascal is the newest descendent in the Algol family of languages. Algol stands for *algo*rithmic *l*anguage. It was designed in 1960 and had a major impact on programming languages. Although several shortcomings surfaced later after extensive experience, it quickly gained a reputation as a simple and elegant language. Two members of the committee that designed the language were John Backus and Peter Naur. They devised a method for specifying the grammatical rules of the language. Their specification method is still used for many programming languages including the Component Pascal language.

The Algol language

Another member of the Algol design committee was Niklaus Wirth, a professor of computer science at ETH Zürich, the Swiss Federal Institute of Technology. He proposed some changes to the language that were eventually incorporated into a dialect called Algol-W. He made more substantial changes in that language with his design in 1970 of the popular Pascal programming language. Wirth named Pascal after Blaise Pascal, a French religious thinker, mathematician, and physician who lived in the seventeenth century. Pascal is credited with inventing the Pascaline, a calculating machine that used gears and dials to represent and manipulate numeric values. Wirth's two design goals for the Pascal language were that it be a good vehicle for teaching structured programming and that it be easy to implement on the computers of that day. He was especially successful in meeting the teaching goal for the language, because it quickly gained widespread use in universities around the world.

The Pascal language

Not content with the success of Pascal, Niklaus Wirth improved on his design in 1979 with the language Modula-2, so called because of its ability to subdivide a program into modules. With Pascal, a large program must be written as if it were a single document. The concept of dividing a program into modules is like the concept of dividing a book into chapters. In the same way that a chapter of a textbook collects similar topics under one subheading, a module collects similar program parts under one "compilation unit". To show the power of Modula-2 Wirth designed an entire computer, which he called Lilith, based solely on that language.

The Modula-2 language

The Oberon project was begun in 1985 at ETH. By this time computer hardware and its associated software had become powerful and complex. With the complexity came difficulty in software design and programming. A primary goal of the Oberon project was to develop a programming language that would be simple to learn and use, but powerful enough to solve complex problems. Wirth's philosophy of programming languages is that a complex language is not required to solve a complex problem. On the contrary, he believes that languages containing complex features whose purposes are to solve complex problems actually hinder the problem solving effort. The idea is that it is difficult enough for a programmer to solve a complex problem without having to also cope with the complexity of the language. Having a simple programming language as a tool reduces the total complexity of the problem to be solved.

The Oberon language incorporated a modern software design technique known

as object-oriented programming, while maintaining the successful features of modularity from Modula-2 and structure from Pascal. In 1992, Wirth and H. P. Mössenböck added a few features to the original Oberon language which resulted in the language Oberon-2, named after the second largest moon of the planet Uranus. Inspiration for the name came from the Voyager 2 space probe whose small computer controlled the spacecraft's data acquisition and transmission system.

The Oberon-2 language

In 1997, the Oberon microsystems company made a few additional changes to Oberon-2, some of which make the language more consistent with the popular Java language, and some of which made the language more useful in a framework environment. They named the new language Component Pascal. The word "component" in the name of the language is to emphasize the suitability of the language for writing objects that are components of containers. An example of a component is a spreadsheet object that is contained in a word processing container. Because this is an introductory book, it will not be able to delve into those advanced features of component programming. But it is good to know that with Component Pascal you will be learning a language that is capable of solving industrial strength problems. Unlike Pascal, the primary goal of Component Pascal is not a teaching goal. However, Component Pascal has its roots in Pascal, and because of that it is an excellent language to learn principles of software development. Because the difference between Oberon and Component Pascal is so small, Niklaus Wirth is considered to be the designer of Component Pascal.

The Component Pascal language

Niklaus Wirth is the designer of the Component Pascal language

If you have experience with Pascal you will notice many similarities with Component Pascal. There may even be a few features that you miss. If so, remember that missing features do not equate to less power. If you have no prior programming experience you will surely not miss a thing, and you may even have the advantage of not having to unlearn some old ways of thinking.

Cross-platform capability

Imagine a country in which the same company that owned the local electrical power plant also owned the local appliance factory. The company designs the electrical wall outlet to accommodate the appliances that it makes in its factory. If you buy a toaster from the company there is no problem plugging it into the electrical outlet, because the same company that makes the outlet makes the appliance, and it designs the plug on the toaster to fit the outlet on the wall.

Suppose that another company moves into the neighborhood and sets up a competing electrical power plant and a competing appliance factory. The new company thinks that it can provide a better wall outlet and corresponding plug in its appliances, so it makes them differently from the first company. Unfortunately, a plug from the first company will not fit a wall socket from the second company and vice versa. This situation forces people to choose who they will get their electric power from and commits them to buy their appliances from the same company.

The compatibility issue

Now suppose that a third company moves to town and is not interested in setting up a local power plant, but is only interested in making appliances to sell to customers of both the other companies. To do so this third company will have to design its appliances to be compatible with both types of outlets. Each appliance will come in two models—one model with a plug compatible with the first company's outlet and

another model with a plug compatible with the second company's outlet.

The scenario described above is certainly not optimal for the third company. If the first two competing companies would simply standardize on their outlet design the third company would only need to manufacture one model of its appliance, which it could sell to either type of customer. Manufacturing costs would be lower because of increased economy of scale and customers would benefit from the lower prices.

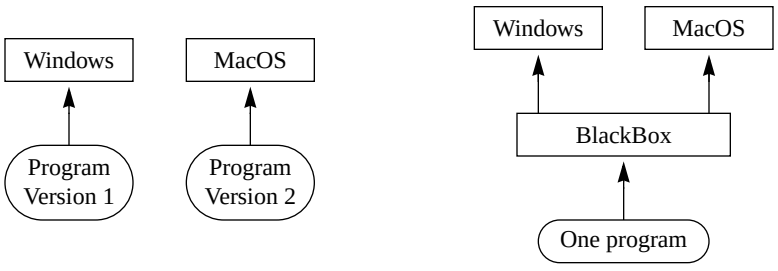
This scenario is not much different from the world of computing today. Every computer is controlled by an operating system. Like the wall outlet designed by the power plant, each operating system has its own standards to which applications must adhere. There are at least three different widespread operating systems with three different standards—various Microsoft Windows operating systems, MacOS, and UNIX. The standard specified by an operating system is called an *application programming interface*, or API for short. An operating system's API is a specification to which programmers must adhere if they want their programs to operate with that operating system. The API of each of these operating systems is different from the others. Consequently, if you write a program that you want to use with Windows98 it will not work on a computer that is running the MacOS or UNIX.

The application programming interface (API)

A remarkable feature of the BlackBox framework is its cross-platform capability. Of all the software development environments on the market at the time of this writing, BlackBox and the various Java systems are the most successful in providing a programmer with the ability to write a complex general-purpose program that will work with little or no modification on any of the above three operating systems. It is as if an additional company moved into town and supplied a converter to be positioned between the wall socket and the appliance plug.

Cross-platform capability

The appliance company is now free to make only one kind of appliance that will work with both customers. It only needs to include the converter with its appliance. Similarly, the programmer is now free to write only one program that will work with any operating system. The BlackBox framework is the converter between the computers as shown in Figure 1.1.



(a) Multiple versions without the BlackBox framework.

(b) One program with the BlackBox framework.

Figure 1.1
The cross-platform capability of BlackBox.

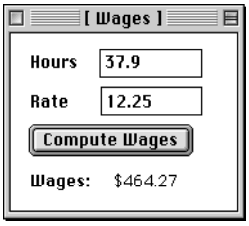
Graphical user interface

The graphical user interface (GUI)

The cross-platform capability of BlackBox would not be that remarkable if it were confined to the programming language Component Pascal. Programs written in all the older languages mentioned previously have the capability to run on different computers with their different operating systems. However, such portable programs based on a language instead of a framework do not have the capability of providing the user with the *graphical user interface* (GUI) of modern operating systems. Instead, such programs execute in what is called a command-line interface. If you have experience with the DOS operating system you know how a command line interface works. The user is presented with a prompt character on the left side of the screen—usually the % character—and must type a command to instruct the operating system to perform a task.

The GUI was invented by the Xerox company and later popularized by Apple with its Macintosh computer. A GUI is characterized by input that is controlled by a mouse as well as the keyboard. Rather than having to memorize and enter commands from the keyboard the user simply makes selections from a menu bar at the top of the screen or from a dialog box that pops up in the middle of the screen. Each operating system has its own GUI with its own look and feel.

The real advance made by the BlackBox framework is not just its portability across operating systems, but the fact that its portability adopts the GUI of the host operating system and is not limited to a command-line interface. This book assumes you have access to and are familiar with either the Microsoft Windows operating system or the Macintosh. The BlackBox system can be installed under either system and every program described in this book will run under both MSWindows and the MacOS. BlackBox is impressive because the same program will produce a dialog box that conforms to the MSWindows GUI if it is executed on a MSWindows machine and to the MacOS GUI if it is executed on a Macintosh. Figure 1.2 shows an example of dialog boxes that were developed on a Macintosh and a MSWindows machine. Each dialog box maintains the style of its host GUI. Furthermore, the program that performs the processing for both dialog boxes is identical. The programmer who develops a program for one machine need not change the program for it to run on the other machine.



(a) A Macintosh dialog box developed in the BlackBox framework.

(b) A MSWindows dialog box developed in the BlackBox framework.

Figure 1.2
The same dialog box with two different operating systems.

Object-oriented language and system

Abstraction is the hiding of details.

Object-oriented languages are based on the concept of abstraction. In computer science, abstraction is the subdivision of a system into layers in which the details of one layer of abstraction are hidden from layers at a higher level. A computer scientist uses abstraction as a thinking tool to understand a system, to model a problem, and to master complexity.

An example of levels of abstraction that is closely analogous to computer systems is the automobile. Like a computer system, the automobile is a man-made machine. It consists of an engine, a transmission, an electrical system, a cooling system, and a chassis. Each part of an automobile is subdivided further. The electrical system has, among other things, a battery, headlights, and a voltage regulator.

People relate to automobiles at different levels of abstraction. At the highest level of abstraction are the drivers. Drivers perform their tasks by knowing how to operate the car: how to start it, how to use the accelerator, and how to apply the brakes, for example. At the next lower level of abstraction are the backyard mechanics. They understand more of the details under the hood than the casual drivers do. They know how to change the oil and the spark plugs. They do not need this detailed knowledge to drive the automobile. At the next lower level of abstraction are the master mechanics. They can completely remove the engine, take it apart, fix it, and put it back together again.

The history of the development of programming languages is a progression from lower levels of abstraction to higher levels. Early programming languages required the programmer to know details of the computer system that are hidden from the programmer using modern programming languages. Object-oriented languages are the latest step in this evolution. In the same way that automobile drivers no longer need to be master mechanics to use their automobiles, programmers can use the capabilities of software objects to produce graphical user interfaces without knowing many of the details that would be visible in older languages.

Programming languages have evolved through four levels of abstraction.

The four levels of abstraction in programming languages

- Type and Statement abstraction
- Structure and Procedure abstraction
- Class abstraction
- Behavior abstraction

The last two levels encompass object-oriented programming. Chapter 23 describes each of the levels in detail. This book mirrors the historical development of programming languages. It presents software design first using type and statement abstraction, then structure and procedure abstraction, then class abstraction, and finally behavior abstraction. Component Pascal is a particularly good language to learn for this journey, because it provides all four levels of abstraction in one language.

Project folders

All software development systems have some pattern of organization for storing program and documentation files. These patterns vary from one framework to the

next. This section describes the way you should organize your files in the BlackBox framework in order to work the problems in this book. It assumes you know the commands to create directories or folders on the hard drive in your computer.

The first task in setting up the BlackBox framework is to acquire it and install it. Fortunately, this powerful system is available free for personal and educational use from the Swiss company Oberon microsystems. The Preface gives instruction for downloading it from the World Wide Web.

After downloading the software, execute the installation program. Then, simply follow the installation instructions on the screen, which should result in a folder named BlackBox on your hard drive.

The BlackBox folder will have many subfolders within it, including folders named Form, Text, and Sql, which are examples of subsystems of the framework. Form is the subsystem that enables you to write dialog boxes, Text makes possible the creation and modification of text documents such as program listings and documentation, and Sql is a database subsystem that you may wish to investigate, but which is beyond the scope of this book. You can see that there are many other subsystems that have been installed in the BlackBox folder.

To hand in the problems from this book to your instructor, you need to establish your own project folder alongside Form, Text, and Sql. Assume that the typical class contains fewer than 100 students. Each student is assigned a unique two-digit number. Say you are the 99th student on the roster and are assigned the number 99 on the first day of class. You should then create a new project folder named Hw99 within the BlackBox folder. Hw stands for homework, and Hw99 is the folder that will contain your homework assignments. (The folder names described here may differ somewhat from the names your instructor prefers.) Figure 1.3 shows the placement of this project folder among some other folders.

Now that you have created your Hw99 project folder, you should create six subfolders named Code, Docu, Mod, Rsrc, Sym, and Enc. The first five names are special names that the BlackBox framework recognizes. Their contents will be as follows.

Folder names with special meaning to the BlackBox framework

- Mod—the programs you write with the text editor
- Code—the machine language version that the Component Pascal translator will create from your programs in Mod
- Sym—the symbol table that the Component Pascal translator will create from your programs in Mod
- Docu—the documentation you write with the text editor that accompanies the programs in the Mod folder
- Rsrc—the resources that you create with the Form subsystem that accompany the programs in the Mod folder

The last folder, Enc, is not standard for the BlackBox framework but is recommended for completing the programs in this book. Enc stands for encode. As explained later in this chapter, to hand in your homework electronically you will need to store your work in a standard encoded form. Enc will be a convenient folder in which to place your encoded assignments before transferring them electronically to your instructor.

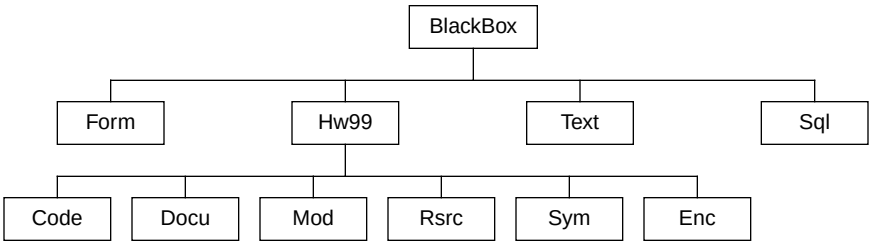


Figure 1.3
The homework project folder.

The text subsystem

After creating your project folder with its subfolders, you should execute the BlackBox program and explore its on-line help system. You may notice that when you execute BlackBox, a window labeled Log is displayed. You will normally want to keep the Log window visible as you work. In the help system you will find a table of contents of the help subjects that are available for your perusal. Locate the topic named User Manuals, which contains the following links, similar to links on a World Wide Web browser: Framework, Text Subsystem, Form Subsystem, Dev Subsystem, and Custom Commands. Clicking on one of these links will bring up a documentation window appropriate to the topic on which you clicked. You can click on the Text Subsystem link to learn how to use the text editor to create and modify documents. Its capabilities are similar to those of word processors.

The right arrow menu convention

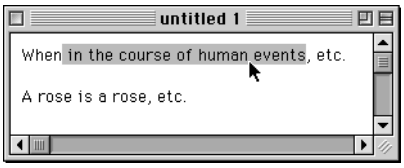
This book will use the same notation that the on-line help system uses to indicate a menu selection. A right arrow will indicate the selection of a topic from a list of menu items from the menu bar at the top of the window. For example, to create a new text document, you select the New command from the menu bar item labeled File. The arrow notation indicates this selection as File→New. Another documentation convention is the definition of the modifier key. If you are using a MSWindows machine the modifier key is the key labeled *Ctrl*. If you are using a MacOS machine the modifier key is the key labeled *option*.

The modifier key

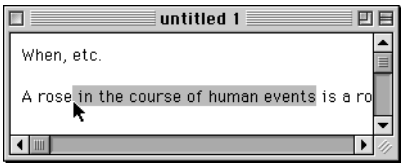
Two features of the text subsystem are especially nice for writing programs—the Undo command and the Drag and Drop feature. After you have performed several operations such as inserting and deleting text in a text document experiment with Edit→Undo and Edit→Redo. You will discover that the Undo command can reverse not only the previous operation, but all operations up to the one after the most recent File→Save command.

Figure 1.4 shows the Drag and Drop feature. Figure 1.4(a) shows the selection of a stretch of text, which you are probably familiar with from your word processing experience. With Drag and Drop, you can put the arrow cursor in the selection, press the mouse button, and while the button is pressed drag the selected text to an insertion point at some other part of the document. Figure 1.4(b) shows the result of dragging the text to the second line and releasing the mouse button. Note that the original text has been moved from the first line and no longer appears there. If you want to duplicate the selected text, press the modifier key as you are releasing the mouse button. Figure 1.4(c) shows the result of this action. Note that the first line contains the original selection of text, which has been duplicated on the second line. This fea-

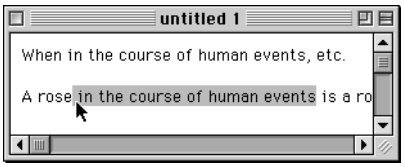
ture is useful in programming when you must type several lines of text that are identical except for a few small differences. After typing the first line you can simply duplicate the whole line and enter the difference on the duplicated lines rather than type them all from scratch.



(a) Selecting a stretch of text.



(b) Drag and drop the selection.



(c) Drag and drop with copy.

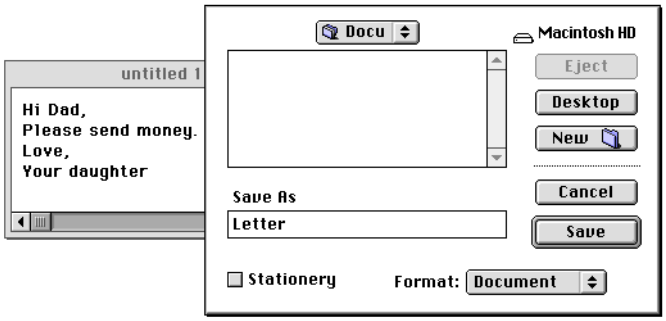
Figure 1.4
The Drag and Drop feature.

Encoding BlackBox documents

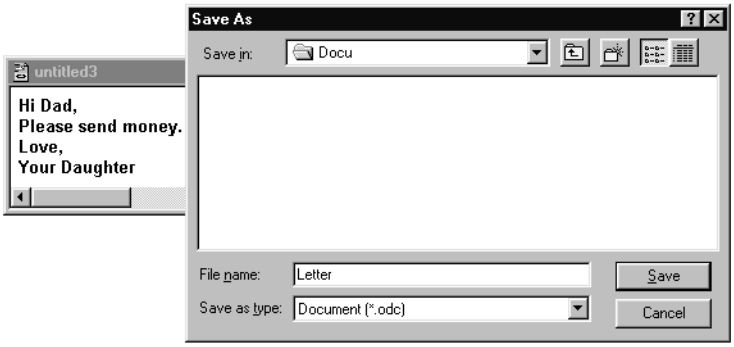
You will eventually use the text subsystem to create programs written in the Component Pascal language. After you write your programs and test them, you will need to hand them in to your instructor electronically. Before you hand them in you must encode them in a format that is safe for transmission via email over a network. The BlackBox framework has a special encoding feature that is designed for such transmissions. To describe how to use the encoding system, this section will explain how to encode a document that you create with the text subsystem.

Suppose you have a document such as a personal letter that you have entered using the features of the text editor. Select File→Save As to save your document. In the dialog box from the Save As command, manipulate the controls so the document will be saved in the folder named Docu within the folder named Hw99 (assuming your assigned number is 99) that you have previously installed in your BlackBox folder. If you are using MacOS, check the dialog box to make sure the Format control specifies Document. If you are using MSWindows, check the dialog box to make sure the Save As type control specifies Document. Unlike the MacOS system, MSWindows will append the suffix .odc to the end of a document file. Make up a name for your document and save it. Let us suppose for the time being that you name your document Letter. Figure 1.5 shows the dialog box just before the save.

Now that you have saved the document close the document window. The Log window should still be showing somewhere on the screen. If you have previously closed the Log window you can bring it back up again by selecting Info→Open Log.



(a) MacOS.



(b) MSWindows.

Figure 1.5
Saving a document.

Now click on the Log window to make sure it is the active window. An active window such as this is called the focus window. Enter the following line of text in the Log:

The focus window

```
Hw99/Docu/Letter
```

if you are using a Macintosh or

```
Hw99/Docu/Letter.odc
```

if you are using a MSWindows machine. If you are an experienced MSWindows user you may be tempted to substitute back slash characters for the forward slashes shown above. Resist that temptation. The forward slash will work fine even with MSWindows and the back slash character will cause grief for users of other systems when you exchange documents between your computers. Now select that line of text in your Log as shown in Figure 1.6.

Do not use the back slash character.

With the text highlighted, select Tools→Encode File List. There are other encode options under the Tools menu, but in a class based on this book you should never select them. To grade your homework efficiently, your instructor always needs you

Always select Encode File List

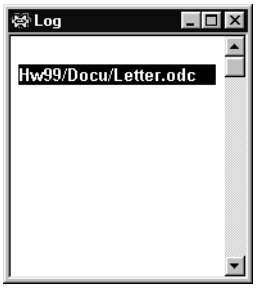


Figure 1.6
Entering a file list to be encoded.

to select the Encode File List option. If you select any of the other encode options you will probably be giving your instructor extra unnecessary work. If you have made all the correct entries so far, the result of selecting Tools→Encode File List will be a new window that contains the encoded file similar to that shown in Figure 1.7.

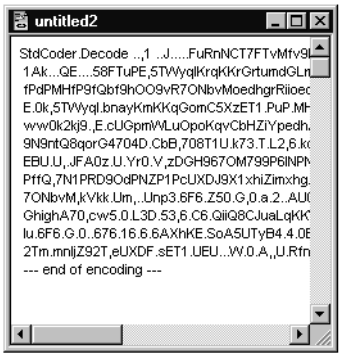


Figure 1.7
The result of selecting Tools→Encode File List.

The first line of the encoded file should begin with

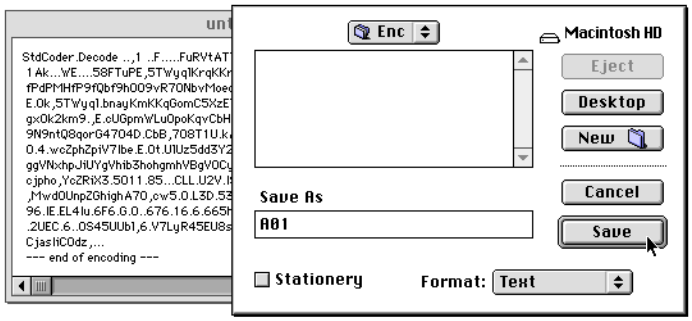
```
StdCoder.Decode
```

and the last line should be

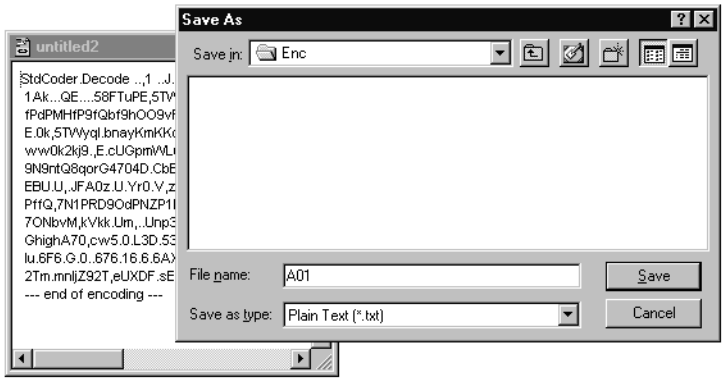
```
--- end of encoding ---
```

If your encode window does not begin and end with these lines, you need to review the above procedure and try it again.

Now it is time to save your encoded file. With the encode window similar to Figure 1.7 in focus, select File→Save As. In the dialog box from the Save As command, manipulate the controls so the document will be saved in the folder named Enc within the folder named Hw99. If you are using MacOS, verify that the Format control specifies Text. If you are using MSWindows, verify that the Save As type control specifies Plain Text. Select a name for your encoded file according to the guidelines specified by your instructor. For example, she may assign a two-digit number to each assignment. If this document is to be handed in as the first assignment she may specify that you name the document A01 as Figure 1.8 shows.



(a) MacOS.



(b) MSWindows.

Figure 1.8
Saving the encoded document.

It is possible to encode many documents into one file. To encode several documents simply list the name of each one you want to include in the Log window. Separate the names with spaces or tabs, or put each name on a separate line. Then, instead of highlighting a single file name as in Figure 1.6, highlight the entire list of names. When you select Tools→Encode File List, all the listed files will be encoded into one window, which you can then save to a file. This technique will be useful for your homework submissions because you will typically need to hand in more than one file for each assignment.

Now that the document is saved in your Enc folder you will need to hand it in electronically. The procedure for doing this will vary depending on the particular networks and computer systems that are used at your institution.

Decoding a document

If you ever receive an encoded document you must first open it. Select File→Open. If you are using MacOS, put a check in the check box labeled “more files”. If you are using MSWindows, change the control labeled Files of type: to specify All Files. These settings will make the encoded files visible in the dialog box. When you open the encoded file a window should appear similar to Figure 1.7. With that window in focus, select Tools→Decode. The files will be decoded and

placed in folders with the same names as the ones they came from. If you do not have folders in your BlackBox folder with the same names from which the encoded files come, the decode command will create folders with those names on your disk before storing the files. For example, when you encoded

Hw99/Docu/Letter

you specified that the file named Letter in the folder named Docu in the folder named Hw99 was to be encoded. When the recipient decodes your file, his disk may not have a folder named Hw99 in his BlackBox folder. When he selects Tools→Decode the framework will create a new folder named Hw99 in his BlackBox folder, and will create a new folder named Docu in the Hw99 folder. It will then decode the file and place it in the Docu folder, giving it the name Letter.

Exercises

At the end of each chapter in this book is a set of exercises and problems. Work the exercises on paper by hand. The problems are programs to be entered into the computer and submitted electronically.

1. Identify the following acronyms.

(a) Algol **(b)** ETH **(c)** API **(d)** GUI

2. **(a)** Who devised a method for specifying the grammatical rules of the Algol language?
(b) How did the Pascal language get its name?
(c) How did the Oberon-2 language get its name?
(d) Who designed the Component Pascal language?
(e) What is the significance of the name Component in Component Pascal?

Problems

3. Create the project folder for the problems you will write for this book. Create the six subfolders within the project folder as described in this chapter. Read briefly the documentation in the on-line help titled Text Subsystem in the User's Guide. Write a document with the BlackBox text editor. Include your name, address, and a description of your favorite hobbies. Include more than one font style and more than one color of text. Experiment with the drag-and-drop and multiple undo features of the text subsystem. Store your document in your Docu folder. Use the Encode File List command to encode your document. Store the encoded document in your Enc folder. Submit your assignment electronically as specified by your instructor.

4. Your instructor has placed an encoded document for you to retrieve electronically. Retrieve it, decode it, read it, and follow the instructions contained in the document.