

chapter

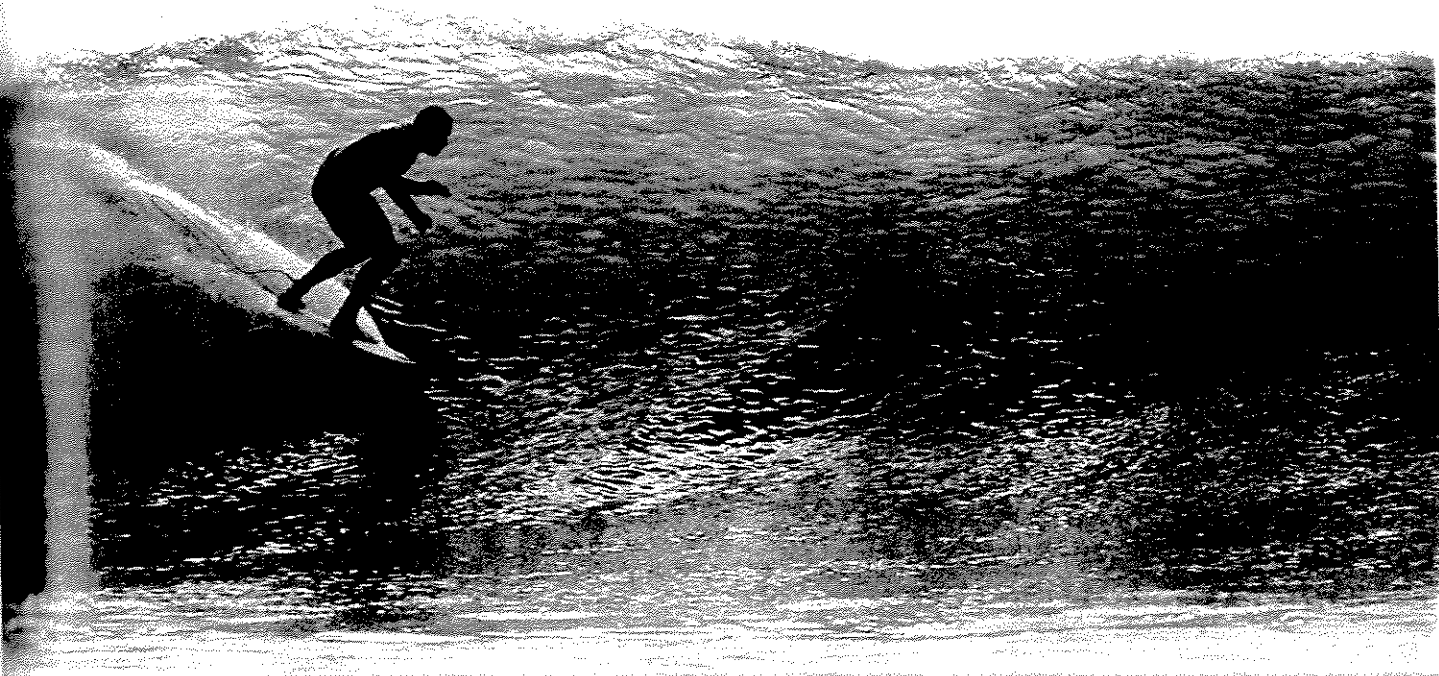
2

WHAT THE DIGERATI KNOW

Exploring the Human-Computer Interface

learning objectives > Explain key ideas familiar to experienced users (the digerati):

- The advantages of having consistent features in information technology
 - The benefits of using feedback of clicking around and blazing away in exploring new applications
 - The basic principle of IT: Form follows function
- > Explain how a basic search is done
- > Use common methods to search and edit text:
- Find (words, characters, spaces)
 - Shift-select
 - The placeholder technique
 - Search-and-replace (substitution)



WHAT THE DIGERATI KNOW

Exploring the Human-Computer Interface

Exploration is really the essence of the human spirit.

—FRANK BORMAN, US ASTRONAUT

Text processing has made it possible to right-justify any idea, even one which cannot be justified on any other grounds.

—J FINNEGAN, USC

PERHAPS the most uncomfortable part of being an inexperienced computer user is the suspicion that everyone but you knows how to use technology. They seem to know automatically what to do in any situation.

Of course, experienced users don't really have a technology gene. Through experience, however, they have learned a certain kind of knowledge that lets them figure out what to do in most situations. Most people don't "know" this information explicitly—it's not usually taught in class. They learn it through experience. But you can avoid long hours of stumbling around gaining experience. In this chapter, we reveal some secrets of the digerati so that you too, can "join the club." (Digerati, analogous to the word *literati*, means people who understand digital technology.)

Our major goal in this chapter is to show you how to think abstractly about technology. We do this by asking how people learn technical skills and by considering what technology developers expect from us as users. This chapter will also help you understand that:

- > Computer systems use consistent interfaces, standard metaphors, and common operations.
- > Computer systems always give feedback while they are working.
- > Making mistakes does not break the computer.
- > The best way to learn to use new computer software is to experiment with it.
- > Asking questions of other computer users is not evidence of being a dummy, but proof of an inquiring mind.

These ideas can help you learn new software quickly. A key abstract idea about software is that it obeys fundamental laws. This core idea can help you in your everyday software use, as we illustrate when we explain the principle "form follows function." We show how this principle applies to basic text searching by teaching the subject without using any specific software. Such knowledge applies to every system and makes us versatile users. You too, can become one of the digerati.

LEARNING ABOUT TECHNOLOGY

Human beings are born knowing how to chew, cough, stand, blink, smile, and so forth. They are not born knowing how to ride a bicycle, drive a car, use a food processor, or start a lawnmower. For any tool more complicated than a stick, we need some explanation about how it works and possibly some training in how it's used. Parents teach their children how to ride bicycles and most products come with an owner's manual.

Some tools, such as portable CD players, are so intuitive that most people living in our technological society find their use "obvious." We don't need to refer to the owner's manual. We can guess what the controls do because we know what operations are needed to play music. (Without this knowledge, the icons on the buttons would probably be meaningless.) And we can usually recover from our mistakes. For example, if you were to insert a CD upside-down, it wouldn't work, so you'd turn it over and try again.

But the fact that we live in a technological society and can figure out how a CD player works doesn't mean that we have any innate technological abilities. Instead, it emphasizes two facts about technology that are key to our success:

- > Our experience using (related) devices, including software, guides us in what to expect.
- > Designers who create these devices, including software, know we have that experience and design products to match what we already know.

The Desktop

When a personal computer starts up, the image it displays on the monitor is called the **desktop**. It usually has a colored background, but photos, patterns, logos, etc. are also common background images. Information is displayed along the top, bottom, or side of the desktop. Small icons of three basic kinds are displayed on the "desk":



- > Applications (programs), such as Internet Explorer, which are identified by their logos
- > Folders (directories), which have icons like small file folders
- > Files (documents), which are identified by an icon corresponding to the application, such as Adobe Acrobat, that created them

There are other icons too, such as the wastebasket for trash.

The image is called the desktop because it implies a metaphor. In software, a **metaphor** is an object or idea used as an analogy for computation. Working with a personal computer is analogous to working at a desk: You keep your work in files, you organize your files into folders, and you use applications (programs) such as a calculator or photo-editing tool to perform your tasks. The designers at

Xerox PARC and Apple Computer who created the idea of personal computing, wanted an analogy to represent the interaction between humans and computers. They chose the analogy of working at a desk; all personal computing software since is designed around that same metaphor.

fitBYTE

Desktop. Many creative people have contributed to the invention of personal computing, beginning in the late 1960s with Douglas Englebart and his team at SRI who created devices such as the mouse. In the 1970s researchers at Xerox’s Palo Alto Research Center (PARC) applied the ideas to office automation by creating the Alto, the first personal computer with the features we’ve come to expect: bit-mapped display, mouse, windows, desk-top metaphor, etc. Although it was never marketed, the Alto motivated Apple to create the Macintosh, launched in 1984. Eventually, Microsoft upgraded its DOS (disk operating system) to have these features, making them effectively universal.

Playing Recorded Music

To illustrate how metaphors are used, consider listening to music. When we listen to a CD on a computer, we control the software that plays the CD using a graphic user interface (GUI), as shown in Figure 2.1. This GUI (GOO-ey) is for the iTunes software for the MacOSX operating system, standard on Apple computers. Similar software plays CDs on the Windows operating system (see Figure 2.2). Even if the iTunes GUI is not familiar, anyone who has ever used a CD player can look at it and figure out how it works.

We can successfully guess how the software works because the GUI shown in Figure 2.1 graphically presents a familiar “music player” metaphor. In addition to



Figure 2.1. Graphical user interface for the iTunes audio CD player on an Apple Macintosh.

the red (close), yellow (minimize), and green (maximize) buttons common to all MacOSX GUIs, it shows three white (plastic-looking) buttons with icons at the upper left that are meaningful to us in the context of playing recorded music: last track, pause, and next track. Below those buttons is a volume control. To the right an LCD window shows the name of the CD—The Beatles—with elapsed time and a diamond moving along a slot, which we can guess is a visual description of how much of the track has been played. In the main window is the playlist titled “Song Name” giving the usual information. Notice the icon by the track that is playing. At the bottom are other buttons with icons, some of which are familiar, e.g., shuffle, repeat, and eject. We also see the number of songs, total time, and memory size (MB is megabytes). On the left is a list of “Sources.” The highlighted item is “Revolver”—the Beatles album we’re playing. The other items, we can guess, are places where other recorded music is stored.

Because we know from experience that one “pushes” a button on a computer by clicking it with the mouse, anyone who has played recorded music can intuitively learn to control the iTunes GUI. We don’t need instruction because the software designers present a metaphor that we immediately understand. We apply what we know about the metaphor; therefore, we can use the software without reading the user’s manual.

That’s the idea behind all personal computing software: When you have a task, e.g., playing music, expect the software’s GUI to present a familiar metaphor, e.g., a physical CD player. Apply your knowledge about the metaphor as a guide for using the software.

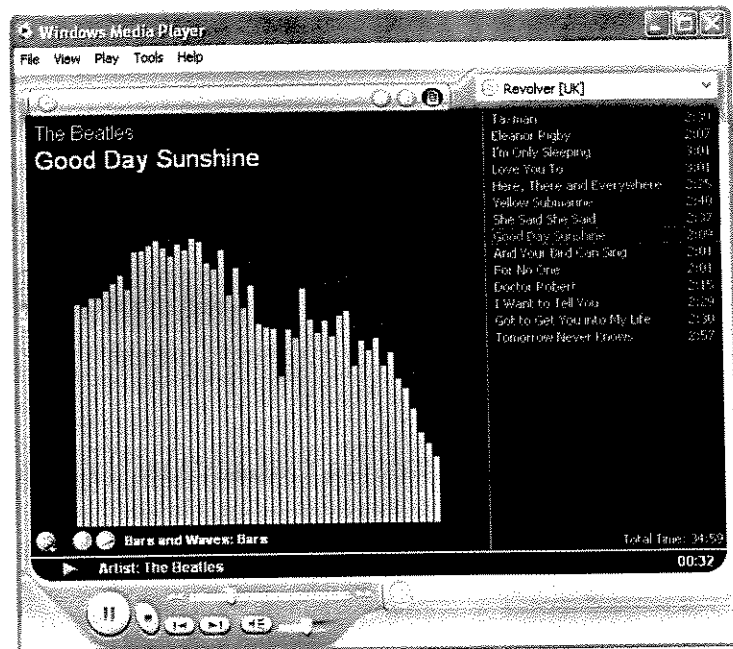


Figure 2.2. Graphical user interface for playing audio CDs with Windows Media Player.

Understanding the Designer's Intent

The designers of the iTunes player invested plenty of effort to make the GUI familiar. For example, the volume control looks like the volume control on a physical CD player. Its operation is identical to the blue slider bar at the bottom, but the designers didn't use that kind of standard slider. They took the trouble to customize the iTunes volume control to resemble one found on physical CD players, including the “soft” and “loud” icons. They designed the buttons in the same way. The buttons didn't have to be special or look plastic. Instead of the icon, the word “pause” could have been printed on the large button. The oval LCD display, common on physical CD players, was created purposely to match the look and feel of a physical CD player. There are easier and fancier ways to display the information—so why do software designers go to so much trouble?

Everyone who invents a new tool, including software designers, has to teach users how to operate their inventions. Developers do write manuals explaining the software's slick features, but it's much faster and easier if users can figure out the software without studying the manual. So software designers, like CD player designers, try to pick easy-to-understand user interfaces. Instead of creating a GUI that requires explanation, the designers guessed that the metaphor of the familiar physical CD player would be intuitive. And they guessed right. By analogy, the basic features of the software are obvious. The audio CD software for the Windows operating system uses a similar interface, as shown in Figure 2.2.

To summarize, software designers want to make the GUI intuitive so that users can figure it out. We should expect that we can “brain out” how it works. We use this idea every time we use new software.

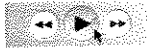
CONSISTENT INTERFACES

Because computers can do many things, GUIs use many different metaphors. GUIs are built using simple metaphors, such as buttons. Software designers use these metaphors similarly; they make them look and work alike (for their operating system). The result is a **consistent interface**. The consistent interface is one of the secrets of the digerati: Whenever they see an icon or metaphor they have seen before, they know immediately how it works, which explains why they always seem to know what to do even if they have never used the software before. You will do the same thing!

Although the iTunes GUI contains many custom features to make it more intuitive, it also uses many standard metaphors found in all GUIs. We'll take a moment to look more closely at some of them.

Command Buttons

As we have seen command buttons take a variety of forms such as a 3D rectangle, an oval or a circle. They are highlighted as explained in Chapter 1 with an icon or text centered on the button. This label says what the command does. To **invoke**



the command—that is, to tell the software to perform the operation shown on the label—we “press” the button by clicking it with the mouse. We receive feedback telling us that the button has been clicked, usually by means of a color change, shadow, highlight, text/icon change, or audible click. Some people think audible clicks are obsessive attempts at realism by developers, but some form of feedback is essential for effective computer use, as explained below.

fit TIP

A Click Is Enough. When clicking on a button, it is not a good idea to press down on the mouse button slowly or for a long time, because the computer may interpret a too-long click as a different action.

Slider Control

The volume control shown in Figure 2.3(a) is a slider control. A **slider control** sets a value from a “continuous” range, such as volume. To move the slider, place the mouse pointer on the slider, hold down the (left) mouse button, and move in the direction of change. The most common examples of sliders are the scroll bars in a window display, usually shown at the right and bottom of the window, as shown in Figure 2.3(b). When the window is not large enough to display all of the information in the horizontal or vertical direction, a scroll bar is shown for each direction in which information has been clipped. For example, the complete information of the playlist is not shown, so the horizontal scroll bar has been placed at the bottom of the window. The range is the size of the information in the direction that’s hidden. Often the size of the slider is scaled to show what proportion of information is displayed. Thus, if the slider takes up half of the length of the “slot,” about half of the information is displayed. There are usually directional triangles ◀ ▶ at one or both ends of the scroll bar; clicking on them moves the slider one “unit” in the chosen direction.

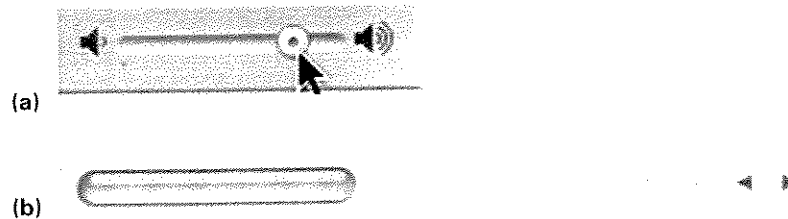



Figure 2.3. Slider controls. (a) A volume control. (b) A scroll bar.

Triangle Pointers

To reduce clutter, GUIs hide information until the user needs or wants to see it. A triangle pointer indicates the presence of hidden information or an alternative form of the information. Clicking on the triangle reveals that information. So, at the end of slider bar, the triangles ◀ ▶ allow you to shift the contents of the

	Song Name
1	✓ Taxman
2	✓ Eleanor Rigl
3	✓ I'm Only Ste

window. You can see other triangles in Figure 2.1. There is a triangle shown in the LCD display ; clicking on it reveals the familiar sound levels display, as shown in Figure 2.4. A triangle in an iTunes column header indicates the order in which the songs are displayed. In Figure 2.1, for example, the songs are displayed in ascending numeric order as they occur on the CD. Clicking on the triangle reorders the listed songs in reverse order.

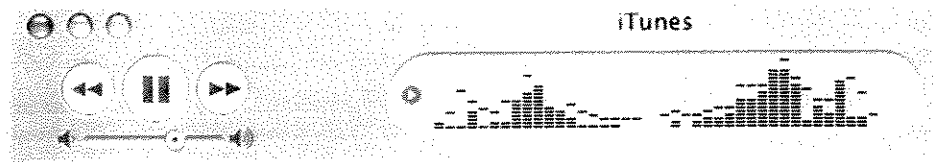


Figure 2.4. The iTunes GUI displaying the hidden sound level information.

We have discussed a few examples to illustrate the metaphor concept. There are many others, and beginning users should get to know them quickly. The point is to emphasize that computer applications have many operations in common, and software designers purposely use them consistently so that they can take advantage of the user's knowledge and experience. Experienced users look for familiar metaphors; and when they recognize a new one they add it to their repertoire.

fitBYTE

Mac or PC? Is the PC better than the Macintosh, or vice versa? The question usually sets off a pointless argument. Listening to the battle, many wrongly guess that the other system must be very different and difficult to use. In fact, the two systems are much more alike than they are different, sharing the concepts we discuss in this chapter and much, much more. Any competent user of one system can quickly and easily learn to use the other. And every Fluent user should.

ANATOMY OF AN INTERFACE

In addition to the window where users conduct most of their interactions by typing or clicking, there are menus. A **menu** is a list of operations that the software can perform. Menus are grouped by the similarity of their operations and listed across the top of the screen in the **menu bar**, as shown in Figure 2.5. All operations performed by the software are listed in a menu.

Menu Operation

Menus listed in the menu bar across the top of a window are called **pull-down** or **drop-down** menus. All of the operations available with the software are listed “under” the menus, even if they are also available by clicking on an icon elsewhere in the window. In some situations, menus are also displayed at the spot where the mouse is pointing when the mouse button is clicked; these are called **pop-up** menus. Both menu types work the same way.

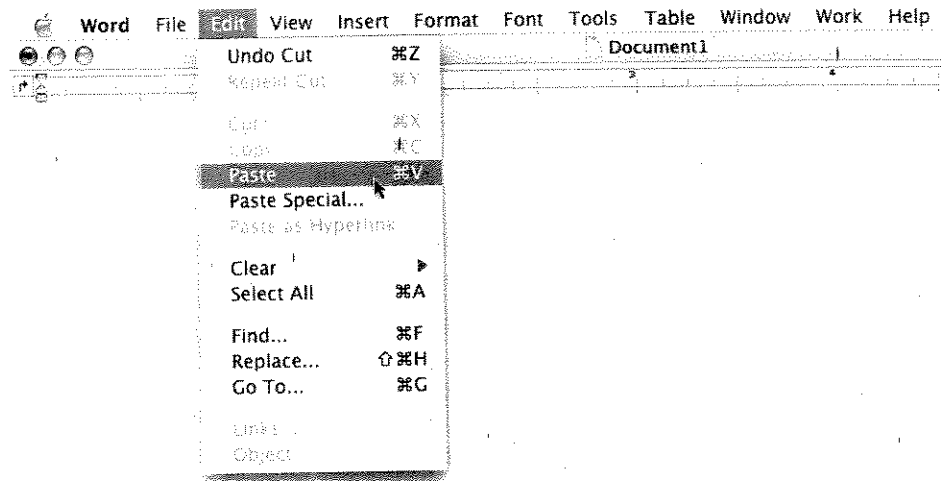


Figure 2.5. The Microsoft Word menu bar for Mac OS X with the Edit menu selected.

To pull down or pop up a menu requires a mouse click, which reveals a list of operations. Sliding the mouse pointer down the list highlights the items as it passes over them. That is, they reverse color like the **Paste** command shown in Figure 2.5. Clicking or releasing the mouse button on the highlighted (selected) menu item causes the computer to perform the listed operation.

Reading a Menu

Menus give more information than just a list of items. They tell you whether an operation is available or not, they indicate when more user input is needed, and sometimes they give you shortcuts. Refer to Figure 2.5 as you read these descriptions.

Which Operations Are Available? Unlike restaurant menus that are printed once and reused, GUI menus are created each time they are opened. So they specify exactly which operations are available. An operation may not apply in every context. For example, **Copy** is not available if nothing has been selected to be copied. Operations that can be applied immediately are shown in solid color and operations that are not available at the moment are shown in a lighter color or “grayed out,” as shown for the **Copy** operation in Figure 2.5. Unavailable items are not highlighted as the cursor passes over them, and of course, they cannot be selected.

Is More Input Needed? Some operations need further specification or more input from the user. Menu items that need further specification have a triangle pointer \blacktriangleright at the right end of the entry (see **Clear** in Figure 2.5). Selecting such an item pops up a menu with the additional choices. Making the selection causes the operation to be performed unless it still needs more specification. Menu items show that they need more input with an ellipsis \dots after their name.

Selecting the item opens a dialog box for specifying the extra input. For example, in Figure 2.5, the operation **Find** has an ellipsis because it needs the user to specify what to look for.

When the software has enough information, it performs the operation immediately and closes the menu(s) and window(s). If not, it continues to ask for more information by opening another window. Answering these questions may lead to more information requests. Eventually the command will be fully specified and can be performed. You can stop the dialog at any time by simply moving your mouse pointer away from the menu or by clicking **Cancel**. Clicking **Cancel** is the same as never having looked at the menu in the first place, no matter how much information you have entered.

Is There a Shortcut? Sometimes it's more convenient to type a keyboard letter than to pull down a window with the mouse and slide the cursor down the list. So some menu items have shortcuts. A **shortcut** is a combination of keyboard characters, shown next to the menu item, that have the same effect. The shortcut is specified by a combination of a special key and a letter. In Figure 2.5, the shortcut for **Cut** is **⌘-X**, shown to the right of the operation in the menu entry; the shortcut for **Copy** is **⌘-C**.

The special character for the Mac is Command (**⌘**), sometimes referred to as a clover. The special character for the Windows operating system is Control (**Ctrl**) (see Figure 2.6). To use the shortcut it is not necessary to pull down the menu. It is enough to hold down the special key—Command (**⌘**) or Control (**Ctrl**) depending on which operating system you are using—and type the letter. Even though the letter is shown as a capital, it is not correct to hold down the **Shift** key while performing this action.

The important thing to notice about the shortcuts is that the same letter command is used for both operating systems. Compare Figures 2.5 and 2.6. That is, once you have learned the shortcuts for an application running on one operating system, you can easily switch to another operating system because the vendors keep the shortcuts—and nearly everything else—consistent. In fact, basic operations like **Copy**, **Paste**, **Print**, **Find**, etc. that are used in most applications use these same letters. This is another way in which the interface is kept consistent.

Shortcuts are not very important for a casual user, but they are extremely handy for people who use a single application intensively.

fitBYTE

A Win for Users. The Microsoft Windows operating system includes most of the GUI metaphors developed for the Apple Macintosh, so in 1988 Apple sued Microsoft for patent infringement. Apple claimed Microsoft illegally used the “look and feel” of its Mac. The legal issues were complex, but the judge ruled that Microsoft could freely use the metaphors Apple had developed. This might not seem fair to Apple, but it was a great win for users, because it meant that GUIs could work pretty much the same on the Mac and the PC.

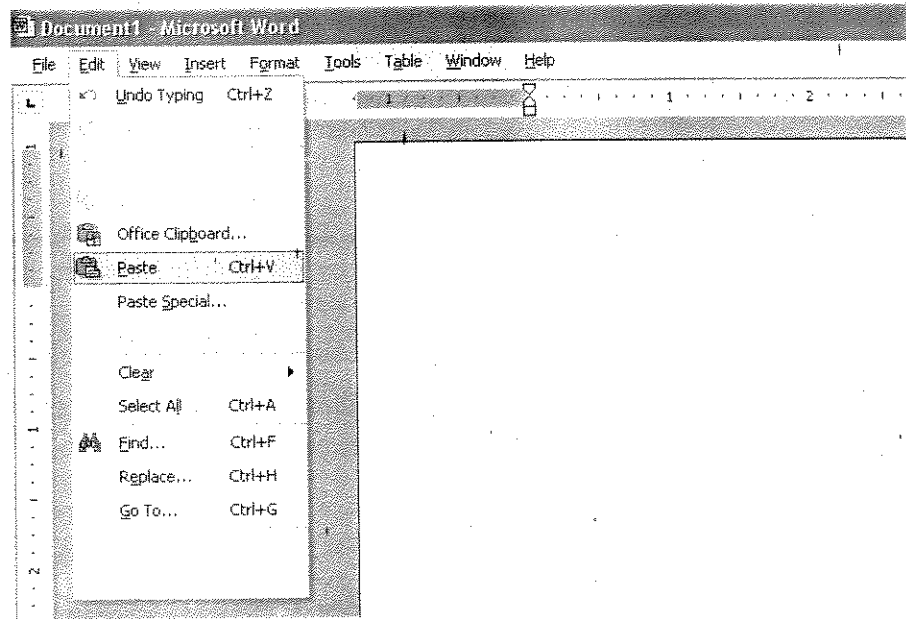


Figure 2.6. The Microsoft Word menu bar for Windows with the Edit menu selected.

STANDARD GUI FUNCTIONALITY

There are some operations that almost all personal computer applications should be expected to perform simply because they process information. That is, whether the information is text, spreadsheets, circuit diagrams, or digitized photographs, the fact that it is information stored in a computer means that certain operations will be available in the software. We call these operations the **standard functionality**. For example, it should be possible to **Save** the information to a file, **Open** a file containing the saved information, **Print** the file, and so on. You should expect to find these functions in almost every software application.

File Operations

To help users, the standard operations are grouped—usually with other operations specific to the application—into two menus labeled **File** and **Edit**. Generally, the operations under the **File** menu apply to whole instances of the information being processed by an application. An **instance** is one of whatever kind of information the application processes. For example, for word processors, an instance is a document; for MP3 players, an instance is a song; for photo editors, an instance is a picture. The **File** menu items treat a whole document. The operations you can expect to see under the **File** menu and their meanings are as follows:

- > **New** Create a “blank” instance of the information.
- > **Open** Locate a file on the disk containing an instance of the information and read it in.

- > **Close** Stop processing the current instance of the information, close the window, but keep the program available to process other instances.
- > **Save** Write the current instance to the hard disk or a floppy disk, using the previous name and location.
- > **Save As** Write the current instance to the hard disk or a floppy disk with a new name or location.
- > **Page Setup** Specify how the printed document should appear on paper; changes to the setup are rare.
- > **Print** Print a copy of the current instance of the information.
- > **Print Preview** Show the information as it will appear when printed.
- > **Exit** or **Quit** End the entire application.

There are usually other operations unique to the application.

New Instance

Notice that **New** under the **File** menu creates a “blank” **instance**. What is “blank information”?

To understand this fundamental idea, notice that all information is grouped into types, based on its properties. Photographs (digital images) are a type of information; among the properties of every image is its height and width in pixels. Monthly calendars are a type of information with properties such as the number of days, day of the week on which the first day falls, and year. Text documents are another type of information and the length of a document in characters is one property. Any specific piece of information—an image, month, or document—is an **instance** of its type. Your term paper is an instance of the document type of information; June, 2005 is an instance of calendar type information.

To store or process information of a given type, the computer sets up a structure to record all of the properties and store its content. A “new” or “blank” instance is simply the structure without any properties or content filled in. For example, imagine an empty form for contact information in an electronic address book, as shown in Figure 2.7. That’s a **New** contact, ready to receive its content.

Edit Operations

The **Edit** operations let you make changes within an instance. They often involve selection and cursor placement. The operations are performed in a standard sequence: select, cut/copy, indicate, paste, and revise. Selection identifies the information to be moved or copied. Selection is usually done by moving the cursor to a particular position in the instance and, while holding down either the (left) mouse button or keyboard keys, moving the cursor to a new position. All information between the two positions is selected. Highlighting, usually color reversal, identifies the selection. If the information is to be recorded and deleted

Create Contact

Cancel

Name & E-mail

First: _____

Last: _____

Title: _____ Suffix: _____

Company: _____

Department: _____

Job title: _____

Work E-mail: _____

Home e-mail: _____

Phone Numbers

Work phone: _____

Home phone: _____

Mobile: _____

Address

Work: _____

City: _____

State/Province: _____

ZIP/Postal code: _____

Country/Region: _____

More

Figure 2.7. A **New** contact (i.e., a “blank” instance) in an electronic address book.

from its current position, the **Cut** command is used. The **Copy** command records but does not delete the information. Then, the new location for the information is indicated in preparation for pasting it into position, although in many applications the indicate step is skipped and the text is pasted into a standard place. The **Paste** command copies the information recorded in memory into the indicated position. Because a copy is made in memory, the information can be pasted again and again. Often, revisions or repositioning are required to complete the editing operation.

The operations under the **Edit** menu and their meanings are as follows:

- > **Undo** Cancel the most recent editing change, returning the instance to its previous form.
- > **Repeat** Apply the most recent editing change again.
- > **Cut** Remove the selected information and save it in temporary storage, ready for pasting.
- > **Copy** Store a copy of the selected information in temporary storage, ready for pasting.
- > **Paste** Insert into the instance the information saved in the temporary storage by **Cut** or **Copy**; the information is placed either at the cursor position or at a standard position, depending on the application.

- > **Clear** Delete the selected information.
- > **Select All** Make the selection be the entire instance.

Undo is not always available because not all operations are reversible. **Redo** may not be available because some operations cannot be repeated.

Because these operations are standard—available for most applications and consistent across operating systems—it is a good idea to learn their shortcuts, as shown in Table 2.1. (To prevent accidents, **Clear** often does not have a shortcut.) In addition, “double-click”—two (rapid) clicks with the (left) mouse button—often means **Open**.

Table 2.1. Standard Shortcuts. These common shortcut letters for standard software operations combine with “Control” (Ctrl) for Windows or “Command” (⌘) for MacOSX.

File Functions		Edit Functions	
New	N	Cut	X
Open	O	Copy	C
Save	S	Paste	V
Print	P	Select All	A
Quit	Q	Undo	Z
Redo	Y	Find	F

TIP





Command and Control. Sometimes it is necessary to refer to an operation like **Copy** by its shortcut without being specific about which operating system is used. In such cases we write ^C to indicate that the operation takes either Command (⌘) or Control (Ctrl), depending on the OS.

Expecting Feedback

A computer is our assistant, ready to do whatever we tell it to do. It is natural that when any assistant performs an operation, he, she, or it must report back to the person who made the request, describing the progress. This is especially true when the assistant is a computer, because the user needs to know that the task was done and when to give the next command. So a user interface will always give the user feedback about “what’s happenin’.”

In a GUI, **feedback** is any indication that the computer is still working, or has completed the request. Feedback takes many forms, depending on what operation a user has commanded. If the operation can be performed instantaneously—that is, so fast that a user does not have to wait for it to complete—the GUI simply

indicates that the operation is complete. When the operation is an editing change, for example, the proof that it is done is that the revision is visible. When the effect of the command is not discernable—say, when one clicks a button—then there is some other indication provided; for example, highlighting, shading, graying, underlining, changing color, or an audible click.


The most common form of feedback is the indication that the computer is continuing to perform a time-consuming operation. As the operation is carried out, the cursor is replaced with an icon such as an hourglass  (on Windows systems) or a rainbow spinner  (on Macintosh systems). Applications can also give the user custom feedback. A common indicator is the busy spinner , a revolving circle divided into quarters, two white and two black. The file transfer application Fetch turns the cursor into a running dog . When the completion time can be predicted, applications show a meter that "fills" as the operation progresses. Often these displays give a time estimate for 100 percent completion. Finally, when an operation is processing a series of inputs, the "completion count" gives the tally of the completed instances, or equivalently, the number remaining.

*fit*TIP

Be Selective. New users can get confused when an operation they want to use is not available (that is, it is "grayed out"). This happens because the operation needs the user to select something and nothing is selected. For example, the computer cannot perform **Copy** until you have selected what you want to copy.

 "CLICKING AROUND"

When the digerati encounter new software, they expect a consistent interface. They expect to see the basic metaphors, find standard operations, and receive feedback while the application is working. Digerati automatically look for these interface features and begin to explore. The purpose of their exploration is to learn what the software can do.

We call the act of exploring a user interface **clicking around**. It involves noting the basic features presented by the GUI and checking each menu to see what operations are available. For example, when experienced users see a slider bar, they slide it to see what happens. When we slide the volume slider in iTunes we notice a change in volume, and in accordance with the feedback principle, we see in Figure 2.3 that the circle in the middle of the slider is colored blue  while we drag it.

When the digerati see a button, they hover their cursor over the button—not clicking it, yet—until the 'balloon help' explanation tells them what the button does. For example, in the iTunes GUI, as shown in Figure 2.1, there are several buttons at the bottom that we don't immediately understand. Hovering over them reveals the explanation, as seen in Figure 2.8. Then, like the digerati, if our

curiosity isn't satisfied, we click the button to see what happens. Perhaps we don't know what "Visual Effects" means, but noting that it is an "off and on" type of control, we click it (see Figure 2.9).

fit TIP

A Fast Start. When you're using software for the first time, practice "clicking around":

- Take a minute to study the GUI graphics.
- Open each window to see what operations are available.
- Determine the purpose of icons and controls.
- Hover the cursor over unknown buttons or GUI features for a short explanation of their purpose.

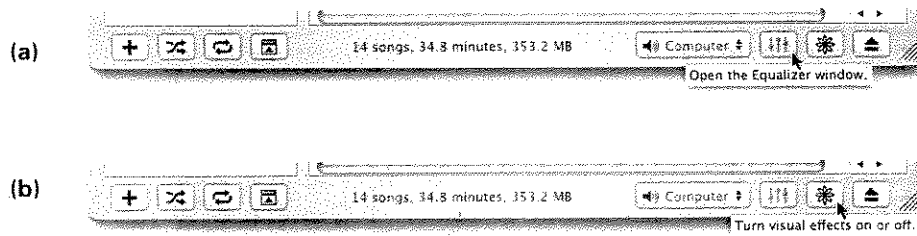


Figure 2.8. Hovering the cursor over unknown buttons shows the help description: (a) Equalizer window, (b) visual effects.

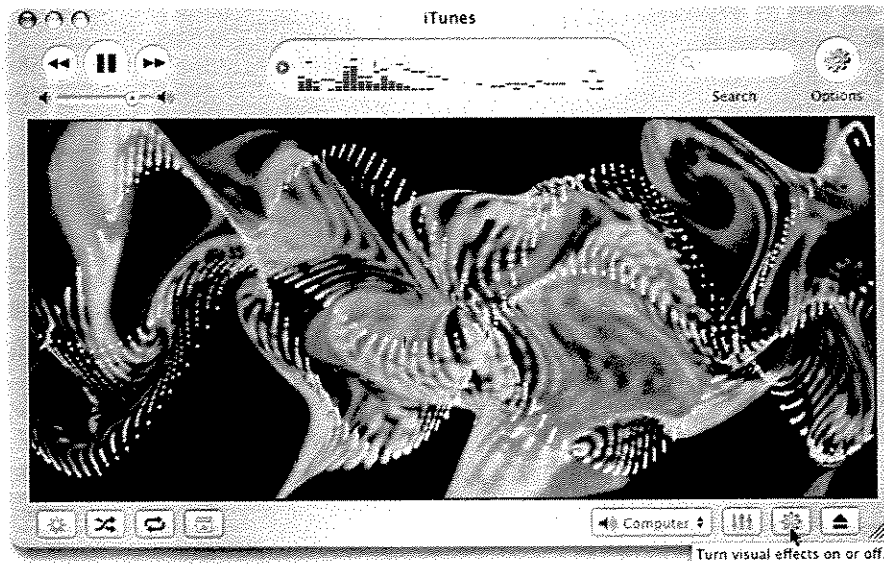


Figure 2.9. Clicking on the Visual effects button in iTunes.

"Clicking around" can help us figure out what operations are available without having to be taught or to read the manual. Software manuals are notoriously dull reading and hard to use. But "clicking around" does not make them obsolete. Manuals—mostly online **Help** resources—are still necessary and useful. "Clicking around" works because (a) we come to the new software with technological experience and (b) software designers try to build on what we know by using metaphors and consistent interfaces. When new software works like the last software did, we already "know" how to use it. The manual is usually needed only to understand advanced features or subtleties of operation. Ironically then, manuals are most useful for experienced users, not beginners.

Returning to the iTunes GUI shown in Figure 2.1, we previously explored the triangle by the "Song Name," and when we clicked on it, the playlist was shown in descending numerical order. But why is the box containing the triangle colored blue? On closer inspection, its shading somewhat resembles a button. Is it a button? Hovering over it doesn't give a short description, so it isn't an 'official' button. But since we're exploring by clicking around, we should try to figure out what the color means. What happens if we click in the box containing the words "Song Name"? When we click in the box, it changes color; it has an up-pointing triangle; the playlist is alphabetized by title; and the square next to it is no longer blue nor has a triangle pointer (see Figure 2.10). We then click on the up-pointing triangle in the "Song Name" box to see what happens and the list is rearranged in reverse alphabetical order. From our clicking around we figure out that these are the controls provided for organizing our playlist. We can order the songs by ascending or descending number, by increasing or decreasing time, alphabetically by title (a-to-z or z-to-a), alphabetically by artist, etc. To choose how we want to organize our



Figure 2.10. iTunes GUI with the playlist organized by song title.

playlist, we click on the box above the column, and possibly on the triangle itself. This is intuitive; no one has to teach us how to use these controls.

“Clicking around” is exploration and may not reveal all of the software features. We may need to experiment, test repeatedly, and try again. But this clicking around technique usually gives useful information quickly. If it doesn’t, the software design has undoubtedly failed to some extent.

fit TIP

Following Protocol. Our normal interactive use of computers alternates between our commanding the computer to do something and the computer’s doing it. If the computer can’t finish immediately, it gives feedback showing the operation is in progress. If the computer is finished, we can see the effects of the command. Be attuned to this alternating protocol. If nothing seems to be happening, the computer is waiting for you to give a command.

 **“BLAZING AWAY”**

After getting to know a software application by “clicking around,” the next step is to try it. We will call this **blazing away**. The term suggests a user’s trying an application assertively—exploring features even without a clear idea of what they will do. “Blazing away” is sometimes intimidating for beginning users because they’re afraid they’ll break something if they make a mistake. A basic rule of information processing is: *Nothing will break!* If you make a mistake, the software is not going to screech and grind to a halt, and plop on the floor with a clunk. When you make a mistake, the software may “crash” or “hang,” but nothing actually breaks. Most of the time nothing happens. The software catches the mistake before doing something wrong and displays an error message. By paying attention to these messages, you can quickly learn what’s legal and what isn’t. Therefore, “blazing away” is an effective way to learn about the application even if you make mistakes.

Of course, saying that nothing will break is not the same as saying that it’s impossible to get into a terrible mess by “blazing away.” Creating a mess is often very easy. Beginners and experts do it all the time. The difference between the two is that the experts know another basic rule of information technology: *When stuck, start over.* That may mean exiting the program. It may mean rebooting the computer. It may simply mean “undoing” a series of edits and repeating them. The simple point is that the mess has no value. It does not have to be straightened out or fixed, because it didn’t cost anything but your time to create in the first place. Because this time is chalked up to “experience” or “user training,” there’s no harm in throwing the mess out.¹ Therefore, an experienced user who is “blazing away” on a new software system will probably exit the software and restart the application over and over, without saving anything.

Usually, we are working with new software because we have something specific we want to do, so it pays to focus on getting that task done. This means that we

should “blaze away” on those operations that will contribute to completing the task; we don’t have to become experts. It’s common for Fluent users to know only the most basic functions of the software systems they use infrequently. And, because they are not regular users of these programs, they usually forget how the applications work and have to “click around” and “blaze away” each time.

filBYTE

Getting Out and Getting Back In. Starting over is so common for computer users—it’s called, *getting out and getting back in*—that it’s become the subject of some geek humor. A mechanical engineer, an electrical engineer, and a computer engineer are camped at Mt. Rainier. In the morning, they pack up to leave and get into their car, but it doesn’t start. The ME says, “The starter motor is broken, but I can fix it,” and he gets out of the car. The EE says, “No way. It’s the battery, but I know what to do,” and she gets out of the car. The CE says while getting out of the car, “Now, let’s get back in.”

Obviously, if you are “blazing away” and starting over when you get into trouble, you shouldn’t spend too much time creating complicated inputs. For example, if the software asks for text input and gives you space for several paragraphs, just enter `Test text` and continue to explore. Once you understand how to do the task, you can focus on using the software productively.

WATCHING OTHERS

“Clicking around” and “blazing away” are the first steps when learning new software because you are likely to succeed using your own observation and reasoning skills. And, if you need to know something very specific about the software, you can always read the manual or online help. However, these two extremes may not cover all of the possibilities. Complicated software systems usually have some features that are not obvious, too advanced, or too specialized to learn on our own. They include GUI features that most of us do not think to look for and they provide capabilities that we may not even know we need.

The Shift-Select Operation

An example of a not-so-obvious feature is the **Shift** key in selection operations. Suppose we want to select only the red and green circles of the stoplight in Figure 2.11(a). Clicking on the red circle selects it (Figure 2.11(b)), as shown by the small boxes around the circle. Clicking on the green circle selects it and deselects the red circle (Figure 2.11(c)). Dragging the cursor vertically from the red circle to the green circle selects all the circles (Figure 2.11(d)). So how do we select just red and green without the yellow? The problem is that when we select something (e.g., the green circle), anything that is already selected (e.g., the red circle) becomes deselected automatically. We need some way to bypass that automatic protocol.

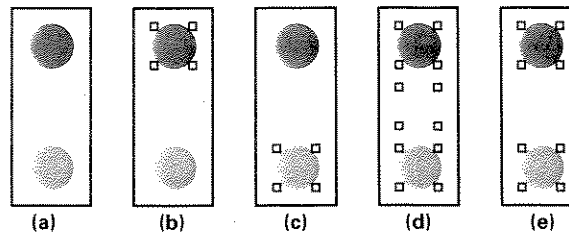


Figure 2.11. Examples of selection.

The solution is to select the first item (e.g., click on the red circle) and then hold down the **Shift** key while selecting the second item (e.g., clicking on the green circle). Using the **Shift** key during a selection means to “continue selecting everything that is already selected.” Because the red circle is already selected when the green circle is shift-selected, both become selected, completing the task.

Learning from Others

The **shift-select** operation, meaning “continue to select the item(s) already selected,” is a common feature in commercial software. Without knowing about shift-select, however, we probably wouldn’t discover it by “clicking around” or “blazing away.” We would not think to try it. We might not even know that we need the feature in the first place. So how do we learn about this kind of feature?

We can take a course on the specific software or read the user’s manual, but an alternative is to observe others as they use a program we are familiar with. As we watch, we should be able to follow what they are doing, though it might seem very fast. If we see an operation that we do not understand, we ask about it. Most people are eager to share their expertise. Many an obscure feature, trick, or shortcut is learned while looking over the shoulder of an experienced user, so it pays to pay attention.

fit TIP

Toggling shift-select. Generally when you use shift-select, one or more additional items is selected, because you usually click on an unselected item. But what happens when you use shift-select on an item that is already selected? It deselects that item only, leaving all other items selected. This property of changing to the opposite state—selecting if not selected, deselecting if selected—is called **toggling**. It’s a handy feature in many situations.

PRINCIPLE: FORM FOLLOWS FUNCTION

A theme of this chapter is that computer systems are very similar because software designers want us to figure out how to use them based on our previous experience with other software. So, they use consistent interfaces and metaphors. When

features of a new system work like the same features in a familiar system, we already know how to use parts of the new system. But there is a much deeper principle at work to explain the similarity among software systems. We state this principle as Form Follows Function.

The **Form Follows Function** principle states that the fundamental operations of a software system and the way they work are determined by the task being solved. This doesn't mean that two software systems for the same task look alike; it means that they will have the same basic operations and those operations will work similarly. Of course, their GUIs can be very different with fancier icons and glitzier buttons, but those differences are superficial. At the core, the application defines the operations and how they work. Performing the task requires that the information is processed in a specific way.

Similar Applications Have Similar Features

To illustrate the principle, text processing applications such as Word, Word Perfect, BBEEdit, Simple Text, Apple Works, NotePad, and a dozen others use a cursor to mark your place in text. They all have operations for typing text, deleting text, selecting text, copying text, searching text, replacing text, etc. The software vendors did not invent these operations; they are fundamental in text processing. Furthermore, the operations work in much the same way in every system. For example, the **Backspace** (or **Delete**) key removes the character to the left of the cursor's present position and it is impossible to select disconnected blocks of text. These are the natural and sensible meanings of those operations in the context of text, which is why they share many features.

If vendors cannot make better software by changing the fundamental operations, how can they compete? Easy, they add other non-fundamental features that make their systems more convenient, friendlier, faster, less error prone, etc. For example, in some, but not all text processing systems, it's possible to select text and drag it to a new position. This is not a fundamental text processing operation, since it can be achieved with a select, cut, reposition-the-cursor, and paste sequence of operations. But text dragging is convenient, and so it has been added to many systems. Software vendors also try to attract new customers by making their software more appealing by adding cool icons or animated "helpers" like paperclips.

Take Advantage of Similarities

Form Follows Function in browser software, spreadsheet software, drawing software, photo-editing software, and so on. Because it's a general principle, when we learn an application from one software maker, we learn the core operations for that task as well as the handy features and annoying quirks of that vendor's product. Then when we use software from a different vendor, we should look for and expect to recognize the basic operations. The features and quirks may or may not be present. The basic operations will have a different look and feel, but they will still be there and work roughly the same way.

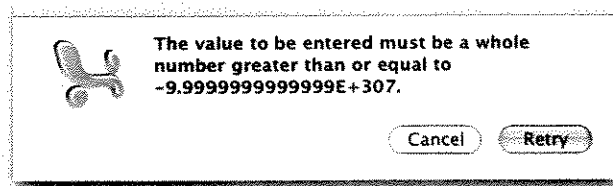
The Form Follows Function principle is important to our everyday use of computers for three reasons:

- > When a new version of familiar software is released, we should expect to learn it very quickly because it shares the core functions and many of the features and quirks of the earlier version.
- > When we must perform a familiar task using unfamiliar software, we should expect to use its basic features immediately because we're already familiar and experienced with them.
- > When we are frustrated with one vendor's software, we should try another's. Because of our experience with the first system, we will learn the new system quickly. (And "voting" by purchasing better software helps to improve overall software quality.)

In summary, because the function determines how a system must work, different software implementations for a task must share basic characteristics. You don't need to feel tied to a particular software system that you learned years ago. You should experiment with new systems because you already know the basic functional behavior of the program.

fit TIP

Mixed Messages. Software is notorious for confusing error messages. It's a difficult problem to fix: Finding errors is easy, but diagnosing the cause is difficult. And programmers explain errors in programming jargon. The result is an incomprehensible error message. But, don't ignore it entirely; a hint of the cause may be buried in the message. For example, this error message



resulted from typing a letter rather than a number. Why doesn't it say, "Enter a number!"? In fact, it does, but in a very complicated way. The conclusion: Spend a moment trying to understand the error message; a useful hint may be hidden there.

SEARCHING TEXT USING FIND

The principle that form follows function has another advantage: It lets us learn how certain computer operations work without referring to any specific software system. Of course, we must focus only on the basic processing behavior rather than

on the “bells and whistles” of the GUI, but learning in this way lets us apply our knowledge to any vendor’s software. We illustrate this idea with text searching.

Many applications let us search text. Text searching, often called **Find**, is used by word processors, browsers (to look through the text of the current page), email readers, operating systems, and so on. **Find** is typically available under the **Edit** menu, because locating text is the first step in editing it. In cases where editing doesn’t make sense—for example, when looking through a file structure in an operating system—**Find** may be listed under the File menu or as a “top level” application. The shortcut for Find—**Ctrl**-**F** for Windows and **Command**-**F** for MacOSX—is standard with most applications.

Things to be searched are called **tokens**. Most often, tokens are simply the letters, numbers, and special symbols like **@** and **&** from the keyboard, which are also called **characters**. However, sometimes we search for composite items, such as dates that we want to treat as a whole. In such cases, the date is the token, not its letters and digits. For the purposes of searching, tokens form a sequence, called the **search text**, and the tokens to be found are called the **search string**. One property of the search string is that it can contain any tokens that may be in the text. In other words, if the text contains unprintable characters like tabs, the search string is allowed to contain those characters.

How to Search

To illustrate searching, suppose the search string is `content` and the text is a sentence from Martin Luther King’s “I Have a Dream” speech:

```
I have a dream that my four little children will one day live
in a nation where they will not be judged by the color of their
skin, but by the content of their character.
```

Searching begins at the beginning or at the current cursor position. Although computers use many clever ways to search text, the easiest one to understand is to think of “sliding” the search string along the text. At each position, compare to see if there is a token match. This simply means looking at corresponding token pairs to see if they are the same:

```
I have a dream ...
```

```

^ ^ ^ ^ ^ ^ ^ ^
^ ^ ^ ^ ^ ^ ^ ^
content
```

(Notice that spaces are characters too.) If there is a match, then the process stops and you see the found instance. But if there is no match, you slide the search string one position along and repeat:

```
... by the content of ...
```

```

^ ^ ^ ^ ^ ^ ^ ^
^ ^ ^ ^ ^ ^ ^ ^
... oooooocontent
```

If the search string is not found when the end of the text is reached, the search stops and is unsuccessful. (Search facilities typically give you the option to continue searching from the beginning of the text if the search did not start there.) The search ends where it began when the search string is not found.

Search Complications

Character searching is easy, but to be completely successful, you should be operationally attuned. There are four things to keep in mind when you are searching: case sensitivity, hidden text, substrings, and multiword strings.

Case Sensitivity. One complication is that the characters stored in a computer are case sensitive, meaning that the uppercase letters, such as *R*, and lowercase letters, such as *r*, are different. So a match occurs only when the letters *and* the case are identical. A case-sensitive search for `unalienable rights` fails on Jefferson's most famous sentence from the *Declaration of Independence*:

We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.

To find `unalienable rights` in a text that uses the original capitalization, we would have to ignore the case. Search tools are case sensitive if case is important to the application. For example, word processors are usually case sensitive, but operating systems are not. If the search has case sensitive capabilities, the user has the option to ignore them.

Hidden Text. Characters are stored in the computer as one continuous sequence. There are two types of characters: keyboard characters that we type and formatting information added by the software application using tags. Because every system uses a different method for the formatting information and because it is usually not important to the search anyhow, we will show the formatting information using our own invented tags:

Tags are abbreviations in paired angle brackets, such as `<Ital>` that describe additional information about the characters (in this case that they should be in italics). Tags generally come in pairs so that they can enclose text like parentheses. The second of the pair is the same as the first, except with a slash (/) or backslash (\) in it. (Tags are used often in our Fluency study, so backslash (\) is used here for the invented tags of our generic application to distinguish them from later uses in HTML and the OED, digitization, which use slash.) For example, to show that the word "Enter" should appear in italics, a software application might represent it as `<Ital>Enter<\Ital>`. These formatting tags are invisible to the reader.

For example, the balcony scene from *Romeo and Juliet* appears in a book of Shakespeare's plays as:

SCENE II. *Capulet's orchard.*

Enter Romeo.

Romeo. He jests at scars that never felt a wound.

[Juliet appears above at a window.

But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.

But, this scene might be stored in the computer as follows:

```
SCENE·II·♦<Ital>Capulet's·orchard.<\Ital>·¶¶<Center><Ital>Ente
r<\Ital>Romeo.<\Center>¶¶<Ital>Romeo.<\Ital>♦He·jest·at·scars
·that·never·felt·a·wound.¶<Right>[<Ital>Juliet·appears·above·a
t·a·window.<\Ital><\Right>¶But,·soft·!·what·light·through·yon
der·window·breaks?·¶It·is·the·east,·and·Juliet·is·the·sun.¶
```

The word processor's tags surround the italic text (<Ital>, <\Ital>), and the text to be centered (<Center>, <\Center>) or right-justified (<Right>, <\Right>). The user typed the other characters and they are the ones we are interested in now. These characters include the text we see as well as formatting characters we can't see: **spaces** (·), **tabs** (♦), and **new lines** (¶). Because these characters control formatting and have no printable form, there is no standard for how they are displayed; for example, the new line character is the **paragraph symbol** (¶) in some systems. Users can ask that all the characters they type be displayed:

```
SCENE·II·♦ Capulet's·orchard.¶
¶
¶ Enter·Romeo.¶
¶
Romeo·♦ He·jest·at·scars·that·never·felt·a·wound.¶
¶ [Juliet·appears·above·at·a·window.
¶
But,·soft·!·what·light·through·yonder·window·breaks?¶
It·is·the·east,·and·Juliet·is·the·sun.¶
```

Because the effects of the formatting are shown, it is easy to see where the non-printed formatting characters are. During a search, the software's formatting tags are generally ignored, but all of the characters typed by the user are considered.

Some systems allow tags to be searched by giving you a way to search for formatted text such as italic.

It gets more complicated when we think of search strings as having a meaning more complex than tokens. For example, we often look for words, although the tokens are characters. The problem is that the software may be searching for token sequences, not the more complicated objects that we may have in mind. So searches for the search string *you* in President John Kennedy's inaugural address turn up five hits:

```
And so, my fellow Americans: ask not what
your country can do for you—ask what you
can do for your country.
My fellow citizens of the world: ask not
what America will do for you, but what
together we can do for the freedom of man.
```

Of the five hits, only three are the actual word we're looking for; the other two hits *contain* the search string. To avoid finding *your*, we can search for *•you•* because words in text are usually surrounded by spaces. However, that search discovers *no* hits in this quote because *you* doesn't appear with spaces on both sides. The five hits for *you* are followed by *r*, a dash, a new line, an *r*, and a comma, respectively. The *you* at the end of the second line probably should have had a space between it and the new line, but the typist left it out.

Because looking only for the word *you* and avoiding *your* means checking for all of the possible starting and ending punctuation characters and blanks, it's probably better to relinquish finding the exact word matches and simply ignore the cases where the search string is part of another word. If the search is part of a system such as a word processor, where words are a basic element, the ability to search for words is available. Such cases amount to changing the tokens from characters to words.

Multiword Search Strings: A similar problem occurs with multiword search strings. The words of a multiword string are separated by spaces, but if the number of spaces in the search string is different from the number in the text being searched, no match is found. For example, the search string

```
That's•one•small•step•for•man
```

Neil Armstrong's words on first stepping on the moon, will not be found in the quote

```
That's•one•small•step•for••man,•one•giant•leap•for•mankind.
```

because there are two spaces between *for* and *man* in the text.

fit TIP

One Small Step. It is a good idea to look for single words in your search instead of longer phrases. For example, looking for `leap` or `mankind` might work because they are probably not used again in the moon walk transcript.

In summary, searching is the process of locating a sequence of tokens, the search string, in a longer sequence of tokens, the text. Character searches are usually limited to the characters typed, even though other characters may be present. Typed characters can include nonprinting formatting characters such as new line characters. Searches look for token sequences and the tokens (for example, characters) are often more basic than what we can build from them (for example, words). To be successful, we must create search strings so that we find all the matches we're interested in.

EDITING TEXT USING SUBSTITUTION

Search-and-replace, also known as **substitution**, is a combination of searching and editing documents to make corrections. The string that replaces the search string is called the **replacement string**. Although substitution can apply to a single occurrence of the search string in the text if necessary, there is little advantage to using a search-and-replace facility over simply searching and editing the single occurrence directly. The real power of substitution comes from applying it to all occurrences of the search string. For example, if you typed "west coast" in your term paper but then realized that regions are usually capitalized, it is simple to search for all occurrences of `west coast` and replace them with `West Coast`.

Because substitution is a powerful tool that we want to study closely, we will express it in this book using a left-pointing arrow (\leftarrow) between the search string and the replacement string. The capitalization example is shown as

`west coast` \leftarrow `West Coast`

Such an expression can be read "west coast is replaced by West Coast" or "West Coast substitutes for west coast." Another example is

`Norma Jeane Mortensen` \leftarrow `Marilyn Monroe`

describing her 1946 name change when she signed her first movie contract.

We emphasize that the arrow is only a *notation* that helps us discuss substitutions in this book; it doesn't appear within the application. When using an application, a GUI is used to specify the replacement (see Figure 2.12). For example the two text fields of the GUI correspond to the information on each side of the arrow.

Find is the left side of the arrow and **Replace** is the right side. We don't type the arrow in applications. It is only for our use here.

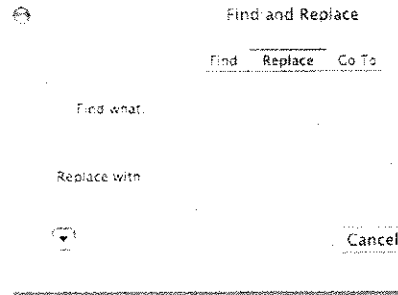


Figure 2.12. A Find and Replace GUI.

Unwanted Spaces

In the last section, we noted that multiple spaces separating words in a text complicates searching for multiword strings. Substitution can fix the “multiple spaces in a document” problem: Simply collapse double spaces to single spaces. That is, if the search string is `..` and the replacement string is `.`, a search-and-replace of the whole document results in all double spaces becoming single spaces. Expressed using the arrow notation, the “two spaces are replaced by one” substitution is

`.. ← .`

Changing from smart quotes to simple quotes is easy enough by writing

`" ← "`

`" ← "`

Can we change simple quotes to smart quotes? We can’t write

`" ← "`

because that changes all of the simple quotes to opening smart quotes, including the closing quotes. But we can use the context around the quotes: Opening quotes are preceded by a space, and after changing them, we replace the remaining quotes with close quotes. So we write

`. " ← ."`

`" ← "`

solving the problem.

Formatting Text

One situation where substitution is particularly handy is when text is imported into a document from another source and the formatting becomes messed up. For example, suppose you find the Articles from the UN’s Universal Declaration of Human Rights on the Web:

Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political, or other opinion, national or social origin, property, birth or other status.

Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.

Article 3 Everyone has the right to life, liberty and security of person.

But when you copy the first three articles and paste them into your document, they come out looking like this:

Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.

Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.

Article 3 Everyone has the right to life, liberty and security of person.

The formatting is a mess. Displaying the text with the formatting characters reveals:

.....Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

.....Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.

.....Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.

.....Article 3 Everyone has the right to life, liberty and security of person.

1-
2S
3,
f

g
otes

For

We see that extra leading spaces and new lines have been inserted when we imported the text into the document.

Clearly, removing the groups of eight spaces at the beginning of lines is simple: we replace them with nothing. When writing the substitution expression, we express “nothing” with the Greek letter epsilon, which is called the empty string; that is, the string with no letters. (Notice that epsilon is used only for writing out substitution expressions for ourselves. In the Find-and-Replace facility of an application, we simply leave the replacement string empty.)

..... ← ε

Removing these leading spaces is easy because they appear only at the beginning of the lines. Correcting the new line characters is more difficult.

We want to get rid of the new line characters that have been inserted within a paragraph but we want to keep the double new lines that separate the paragraphs. But getting rid of the single new line

↵ ← ε

will get rid of *all* the new lines! How can we keep the double new lines but remove the singles?

The Placeholder Technique

An easy strategy, called the **placeholder technique**, solves our problem. It begins by substituting a placeholder character for the strings we want to keep; that is, the new line pairs in the example. We pick # as the placeholder because it doesn't appear anywhere else in the document, but any unused character or character string will work. The substitution expression is

↵↵ ← #

Our text without the leading blanks and double new lines now looks like this:

Article 1. All human beings are born free and equal in dignity and ↵
rights. They are endowed with reason and ↵
conscience and should act towards one another in a spirit ↵
of brotherhood. # Article 2. Everyone is entitled to all the rights and freedoms set forth ↵
in this Declaration, without distinction of any ↵
kind, such as race, color, sex, language, religion, political ↵
or other opinion, national or social origin, ↵
property, birth or other status. # Furthermore, no distinction shall be made on the basis of ↵
political, jurisdictional or international status of ↵
the country or territory to which a person belongs, whether ↵
it be independent, trust, non-self-governing, ↵
or under any other limitation of sovereignty. # Article 3. Everyone has the right to life, liberty and security of person.

The new lines that remain are the ones to be removed, so we need to replace them with nothing

↵ ← ε

The resulting text has no new line characters left:

Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.#
 Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.#Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.#Article 3 Everyone has the right to life, liberty and security of person.

Finally, replace the placeholder with the desired character string

← **Article 1**

which gives us

Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.
 Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.
 Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.
 Article 3 Everyone has the right to life, liberty and security of person.

Except for the bold font for the Article titles and numbers, the resulting file looks like the original document showing the double new lines as appropriate. The final replacements:

Article 1 ← **Article 1**

Article 2 ← **Article 2**

Article 3 ← **Article 3**

complete the task. You may also change the titles to bold directly by editing.

To summarize, the placeholder technique is used to remove short search strings that are part of longer strings that we want to keep. If we were to remove the short strings directly, we would trash the longer strings. The idea is to convert the longer strings into the placeholder temporarily. Of course, a single placeholder character can replace the long strings because all we're keeping track of is the position of the longer string. With the longer strings replaced by the placeholder, it is safe to remove the short strings. Once they are gone, the longer string can replace the placeholder. The substitution expressions

LongStringsContainingInstance(s)OfAShortString ← Placeholder

ShortString ← ϵ

Placeholder ← *LongStringsContainingInstance(s)OfAShortString*

summarize the idea.

TECHNOLOGY: TAKE IT PERSONALLY

We have revealed some secrets known to expert computer users. Now, it's not miraculous that the digerati appear to know how to use software they have never seen before. They expect consistent interfaces and suggestive icons that allow them to apply their previously learned knowledge intuitively. Now we can exploit that strategy, too.

The important thing about our discussion of learning to use technology is how we figured it all out. We thought about technology as it relates to us. We thought, "We need to learn how to use this technology," so we asked, "How did the inventors of this technology expect us to learn it?" We thought about tool inventors who want people to use their inventions. Inventors can write manuals and we can read them, but they will have more users and users will have more success if the inventions are intuitive, allowing people to "brain out" how to use them. We concluded that we should expect intuitive interfaces. This approach of thinking about technology as it relates to us personally and analyzing the situation in that context is essential for any technology use.

✓checkLIST>>

There is a series of similar questions that we should ask ourselves when we need to use new technology, especially software:

- ☞ What do I have to learn about this software to do my task?
- ☞ What does the designer of this software expect me to know?
- ☞ What does the designer expect me to do?
- ☞ What metaphor is the software showing me?
- ☞ What additional information does the software need to do its task?
- ☞ Have I seen these operations in other software?

✓checkLIST>>

When we think about information technology in terms of our personal or workplace needs, we may also ask questions such as:

- ☞ Is there information technology that I am not now using that could help me with my task?
- ☞ Am I more or less productive using this technological solution for my task?

- ✓ *Can I customize the technology I'm using to make myself more productive?*
- ✓ *Have I assessed my uses of information technology recently?*

These and similar questions can help you use technology more effectively. Information technology, as a means rather than an end, should be continually assessed to ensure that it is fulfilling your needs as the technology changes and evolves.

SUMMARY

This chapter began by exploring how we learn to use technology. We concluded, that:

- > People are either taught technology or they figure it out on their own.
- > We can figure out software because designers use consistent interfaces, suggestive metaphors, and standard functionality.
- > We apply our previous experience to learn new applications.
- > Features of the iTunes GUI are familiar, even if we have never seen it before.
- > In computer software, if we make a mistake, nothing breaks.
- > We should explore a new application by “clicking around” and “blazing away.”
- > We should watch other users and ask questions.
- > Form follows function.
- > Searching and substituting, which are available with many applications, work consistently, demonstrating the idea behind form follows function.
- > We should think personally about technology and apply general principles and ideas to become more expert users.

EXERCISES

Multiple Choice

1. Experienced computer users are known as
 - A. digerati
 - B. literati
 - C. mazzerati
 - D. culturati
2. What is a GUI?
 - A. graphical update identification
 - B. general user identification
 - C. graphical user interface
 - D. general update interface
3. Software designers use analogies to help a user understand software because doing so
 - A. makes it easier for the user to learn and use the software
 - B. makes the software more popular
 - C. is required by law
 - D. more than one of the above
4. An example of a metaphor is
 - A. The player played with the heart of a lion.
 - B. The silence was deafening.
 - C. The computer played chess as well as the best humans.
 - D. all of the above
5. Which of the following is not a common computer metaphor?
 - A. buttons
 - B. door handles
 - C. menus
 - D. sliders
6. A slider control is used for selecting
 - A. one of several options
 - B. one or more of several options
 - C. within a continuous range of options
 - D. one or more items from a list
7. In Windows, closing a subwindow
 - A. is not allowed
 - B. leaves the application running
 - C. exits the application
 - D. automatically saves your file

8. A dialog will open when a menu has a(n) _____ in it.
- A. shortcut
 - B. ellipsis
 - C. check mark
 - D. separator
9. Menu options that are unavailable
- A. have a check mark by them
 - B. are gray
 - C. have a line through them
 - D. are hidden
10. Which of the following is not an instance?
- A. an image
 - B. a song file
 - C. a word processing document
 - D. a menu
11. The Greek letter epsilon ϵ can be used to represent "nothing." In a find and replace, you would have to use
- A. _
 - B. null
 - C. nothing (empty)
 - D. \emptyset

1. _____ is the word used to describe people who understand digital technology.
2. GUI stands for _____.
3. Software designers help users understand their software through the use of _____.
4. A(n) _____ is a figure of speech where one object is compared to another.
5. _____ are used to indicate that there is hidden information available.
6. **Open**, **New**, **Close**, and **Save** can usually be found in the _____ menu.
7. To avoid cluttering the screen with commands, software designers put most of their commands in _____.
8. **Undo**, **Cut**, **Copy**, and **Paste** are usually found in the _____ menu.
9. The online manual is usually found in the _____ menu.
10. Menus that can show up anywhere on the screen are called _____.
11. Another name for a pull-down menu is a(n) _____.
12. When the computer needs more information from the user before it completes an action, it gets the information via a(n) _____.

13. Menus that are unavailable are generally colored _____.
14. Menu options that open a dialog display are identified because they contain a(n) _____.
15. The clover-shaped shortcut key on a Macintosh is called the _____ key.
16. Menus are grouped by similarity of operation and listed across the top of the screen in the _____.

Exercises

1. Explain the desktop metaphor.
2. Discuss the advantages of a consistent interface. Look at it from the consumer's view and from the developer's view.
3. List the technology tools you can typically use without reading the owner's manual.
4. What are the two keys to success with information technology?
5. Match the buttons on the two CD interfaces in Figures 2.1 and 2.2. Label the commands. Speculate on how the features found on only one are implemented on the other.
6. What happens when we apply the `** ← *` replacement to `*****`? Try this out with your text editor. How many times did it find and replace? How many were left? Explain how this process worked.
7. Describe the similarities and differences between the Windows CD Player and the Windows Media Player. If you have a Mac, use the CD Player and the QuickTime Player.
8. Using Lincoln's Gettysburg Address, what appears more often, "that" or "here"?
9. How many times does the word "the" appear in Lincoln's Gettysburg Address? How many times do the letters "t-h-e" appear together?

ANSWERS TO SELECTED QUESTIONS

Chapter 1

Multiple Choice

1. A. Monitors use bit-mapped technology generated by the computer whereas TVs use recorded images.
3. C. A laptop has an LCD display and uses RGB color.
5. D. Follow the acronym, PILPOF, plug in last, pull out first.
7. A.
9. D.

Short Answer

1. microprocessor
3. screen saver
5. 786,432 or 1024×768
7. tip of the arrow
9. Execute or run
11. abstraction
13. word processor
15. generalization

Chapter 2

Multiple Choice

1. A. Those with computer skills are sometimes called digerati.
3. D. Both A and B are correct. Ease of use is one of the driving forces behind software development and one of the reasons for the popularity of some software.
5. B. You'll find all except door handles in a typical GUI.
7. B. In Windows, you can close a subwindow, such as a spreadsheet or word processing file, and the application will keep running.
9. B. Operations that can be applied immediately are shown in solid color and operations that are not available at the moment are shown in a lighter color or "grayed out."
11. C. Use nothing.

Short Answer

1. digirati
3. analogies
5. Triangle pointers
7. menus
9. Help
11. drop-down menu
13. gray
15. command

Chapter 3

Multiple Choice

1. D. The Internet is a great medium for asynchronous communication.
3. B. The ability to create and publish Web pages has made it easier for people to express their opinions and creativity.
5. C. For every n computers attached to the Internet, an additional computer adds n connections.
7. B. The right side shows the domain. That will get the message to the correct location. From there, the mail server will have to get it to the correct address. This is very similar to the postal service.
9. C. The government, educational institutions, and big business put together the first system in the late 1960s.
11. D. The client/server relationship is very efficient; the server can handle hundreds or thousands of requests at a time.
13. C. Most countries allow people access to the Internet with few restrictions.
15. D. A wireless hub can be used to connect wireless devices to a network and the Internet.

Short Answer

1. electronic commerce
3. multicast
5. peers
7. channel
9. gateways
11. Web servers
13. FTP
15. Hypertext Markup Language
17. blog

Chapter 4

Multiple Choice

1. C. The commands used in HTML are called tags. They are enclosed in $\langle \rangle$.
3. A. The first one has the tags properly paired.