Apple® HyperTalk™

Beginner's Guide

An Introduction to Scripting

Previously titled
Scripting With HyperTalk

Confirmation Draft

Jody Larson
Customer Communications
February 21, 1989

Apple Confidential

APPLE COMPUTER, INC.

Copyright © 1989 by Apple Computer, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

© Apple Computer, Inc., 1989
20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Apple, the Apple logo, HyperCard, and Macintosh, are registered trademarks of Apple Computer, Inc.

HyperTalk is a trademark of Apple Computer, Inc.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Linotronic is a registered trademark of Linotype Co.

POSTSCRIPT is a registered trademark, and Illustrator is a trademark, of Adobe Systems Incorporated.

Varityper is a registered trademark, and VT600 is a trademark, of AM International, Inc.

Simultaneously published in the United States and Canada.

Contents

Preface

About This Guide

Welcome to the *HyperTalk Beginner's Guide*. This guide provides you with a starting point for exploring HyperTalk™, the language used by the HyperCard® software. With HyperTalk, you can write your own instructions, called **scripts,** for HyperCard to carry out. Writing scripts is called **scripting.**

HyperCard allows you to create, customize, and personalize your own stacks without your having to do any scripting. But HyperCard is a extension of Apple Computer's goal —to bring the power of technology to the individual. Scripting takes you one step further toward having power over what your computer does for you.

If writing scripts sounds a lot like programming to you, you're right—they are very similar; however, you do not need *any* previous experience with programming to be able to write scripts. If you can read this paragraph, then you can write a script.

This guide introduces you to some basic scripting using a practice stack you create yourself. In this stack, you'll learn how to write scripts for traveling between cards, creating special effects, simulating animation, performing calculations, and more. Later on after you've completed this book, you can use your practice stack on your own as a place to try out new scripts.

What you need to know to use this guide

To get the most out of this guide, you should already know the basics of using your Macintosh® computer; for instance, how to use the mouse and the screen windows. You should also be familiar with how to get around in HyperCard. Specifically, you should know about using buttons to get around in stacks and how to use the menus and tools. You should have some working familiarity with HyperCard objects: buttons, fields, cards, backgrounds, and stacks. You should have looked through the Help system, browsed through other stacks, and personalized some stack—for example, used the Address stack to store some personal information. If you have gone through the first three chapters of the *HyperCard User's Guide*, you probably know all you need to know.

If you have experience with programming in another language, you might want to go directly to the *HyperCard Script Language Guide*, which is geared for people with prior experience.

The intent of this guide is to help you get started and let you get a feel for scripting on your own. You won't find long, technical explanations of HyperTalk concepts here; but you will be able to see clearly how specific scripts work.

How to use this guide

Each chapter builds on what you've done in previous chapters, so it's important that you start at Chapter 1 and work through the book sequentially. You should be able to go through an entire chapter in a single session at your computer, but you can take a break any time you like—or keep right on going, if it suits you.

- In Chapter 1, "Getting Started," you'll create a practice stack with which you'll work with scripting throughout this book. You'll make buttons to use with the stack and complete their scripts.
- In Chapter 2, "Special Effects," you'll learn about visual and sound effects in HyperTalk and add them to your stack.
- In Chapter 3, "More About Messages," you'll explore how buttons and other objects receive and send messages, and you'll further increase your HyperTalk vocabulary.
- In Chapter 4, "Fields, 'It,' and Other Containers," you'll get an introduction to how HyperCard stores information and performs calculations.

- ▫ In Chapter 5, "Animation," you'll learn two ways to create "moving pictures" with commands.
- ▫ In Chapter 6, "Stacks You Can Build," you'll look at two examples of useful stacks that you could create and script yourself, starting with materials available in the Idea Stacks that came with HyperCard.
- ▫ The Appendix, "HyperTalk Summary," contains a list of all HyperTalk commands, functions, and other elements.

You'll also find a glossary of terms, an index, and a Quick Reference Card containing the command and functions, which you can remove from this book and keep handy.

Conventions used in this guide

When a new term is defined, you'll see the term in **boldface.** All such terms and other, related terms are included in the glossary.

Sometimes definitions or cross-references appear in the margin.

Important Material set off like this is especially worth reading. Information in these boxes advises you of noteworthy circumstances or helps you avoid misfortune.

❖ *By the way:* Paragraphs like this one contain additional information or interesting sidelights.

A special font (Courier) is used to show HyperTalk words and words and statements you should type. It looks like this:

```
set userLevel to 5
```

Sometimes commands are shown in a generalized form; for example,

set *property* [of *object*] to *value*

Words in italic are simply placeholders. The square brackets ([ ]) are used to indicate optional parts; the brackets shouldn't be included in an actual command.

For more information

Because this guide is intended as an introduction for beginners, it is not comprehensive. HyperTalk comprises many commands, functions, keywords, and other elements that are not explained in this book.

The *HyperCard User's Guide* contains reference information for all menus and tools available.

The HyperCard Help system provides on-line help while HyperCard is running. The Help system contains a HyperTalk reference section.

The *HyperCard Script Language Guide,* published by Addison-Wesley Publishing Co. as part of the Apple Technical Library, is a complete reference to HyperTalk. It's intended for those with some programming or scripting experience.

The *HyperCard Stack Design Guidelines,* also published by Addison-Wesley, provides information on how to design and build professional-quality stacks. Its focus is the presentational aspect of stacks (for example, navigation methods and card layouts) rather than the mechanics of scripts.

Other excellent books on HyperCard and on HyperTalk scripting can be found in almost any bookstore.

Chapter 1

Getting Started

Have you ever wanted to create your own software—make an application program that does things the way *you* want, rather than someone else's way? That's what HyperCard® software allows you to do.

This book takes you a step further into the power of HyperCard by introducing you to **scripting**—the writing of sets of instructions, called **scripts,** to customize HyperCard's actions. Everything that happens in HyperCard is directed by a script.

HyperCard scripts are written in HyperTalk,™ a language very much like the language people use in daily life. Believe it or not, you probably already know how to "say" things in HyperTalk—things that HyperCard would probably be able to understand and perform.

You do not need any prior experience with computer languages to use this book. You should, however, be familiar with how to use HyperCard and how to get around in HyperCard stacks.

In this book, you'll practice scripting in a stack you'll build from scratch. In this chapter, you'll create the practice stack and write some simple scripts to control actions of buttons.

Start up HyperCard

This book is meant to be used with HyperCard "up and running" on your Macintosh® system. You'll need to perform the steps as directed in the sections that follow to get the most out of the material.

Start up HyperCard following the instructions in the *HyperCard User's Guide*. If you already have HyperCard running, go to the Home card. You're ready to go on when you see the Home card on your screen (Figure 1-1).

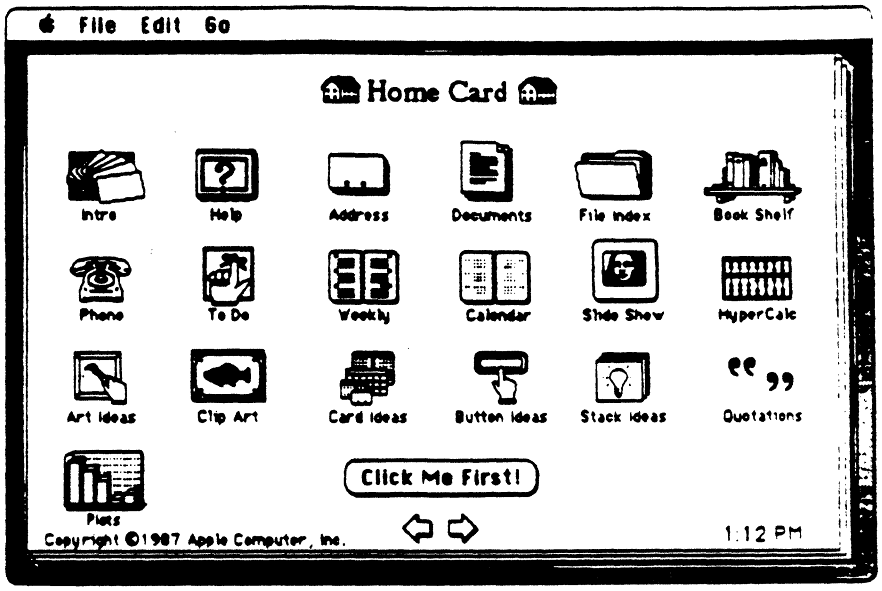


Figure 1-1
The Home card

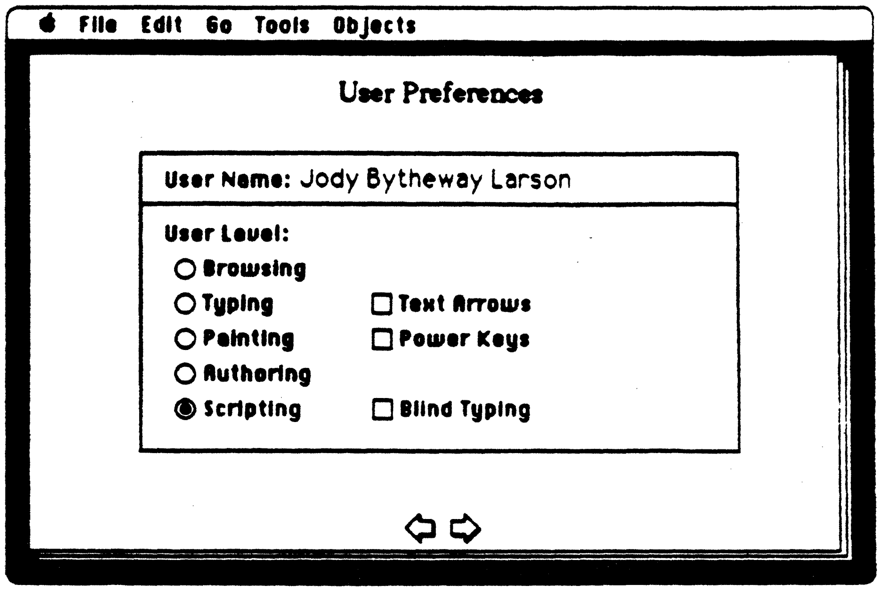
Set your user level

To work with scripts, your user level must be set at Scripting. Change the user level on the User Preferences card of the Home stack following these steps:

1. **Click the left arrow at the bottom of the Home card to go to the User Preferences card.**

2. **Click the Scripting button.**

   For now, the check box options Text Arrows and Blind Typing should be unchecked. You won't need the Power Keys option either, but if you prefer to use Power Keys with the Paint tools you may. Figure 1-2 shows the User Preferences card with Scripting selected. (Earlier versions of HyperCard may not have the Text Arrows option.)



User Preferences

User Name: Jody Bytheway Larson

User Level:

- ○ Browsing
- ○ Typing — ☐ Text Arrows
- ○ Painting — ☐ Power Keys
- ○ Authoring
- [✓ Scripting] — ☐ Blind Typing

Figure 1-2
The Scripting user level on the User Preferences card.

When the user level is set at Authoring or Scripting, a new menu title, Objects, appears in the menu bar. Commands in this menu allow you to get information and change properties of HyperCard **objects**—buttons, fields, cards, backgrounds, and stacks. (You'll learn more about objects later on.) The user level must be set at Scripting before you can look at, write, or change these objects' scripts.

Create a practice stack

Now that you've set the user level to Scripting, the next task is to create a stack where you can experiment with scripts. You can make a new stack at any time from anywhere in HyperCard; you don't have to go back to the Home card. Just follow these steps:

1. **Choose New Stack from the File menu.**

   A dialog box appears in which you can name the stack and specify its background.

2. Click the check box to remove the check mark from "Copy current background."

You don't want to copy the background for this practice stack, so "uncheck" the box. The new background will be completely blank.

3. Type a name for the stack—for example, `Practice Stack`

In this book, your practice stack is referred to simply as that—but you can name your stack anything you like; "Practice Stack," "Test Stack," "Pilgrim's Progress," or whatever. If you make an error while typing the name, use the Backspace (Delete) key to erase it and retype. The dialog box should look similar to the one in Figure 1-3.

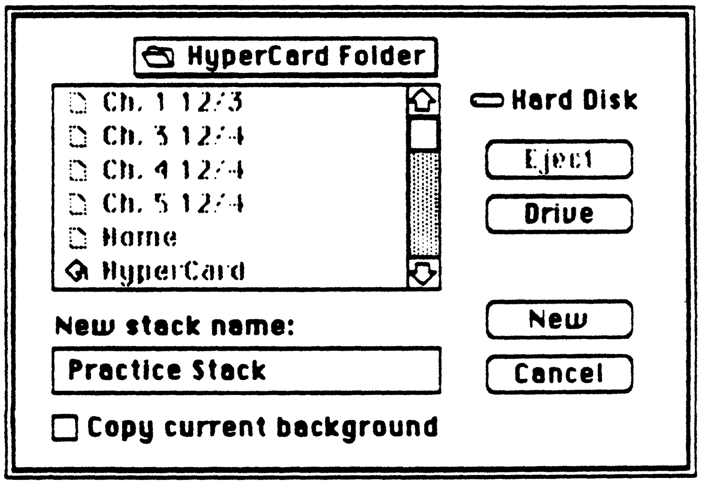


Figure 1-3
The New Stack dialog box

4. When you're ready, click New (or press Return).

You should see a completely blank card on your screen with only the menu bar showing along the top. This card is the first—and right now, the only—card of your practice scripting stack.

Set up the background

You can think of the **background** in HyperCard as a kind of "holding area" for general elements. If a button, a field, or a picture is in the background, then it appears on every card that shares that background. Putting a button in the background, for example, allows you to have that button constantly available throughout a number of cards without having to re-create it on every card. So far, the practice stack has only one background, so all cards you create will share that background.

In this section you'll first create a title that will appear on all cards of the stack. Then you'll put a Home button and some buttons for traveling into the background, and you'll write scripts for the buttons.

Before you go on,

- **Press Command-B to work in the background.**

  (You could also choose Background from the Edit menu.)

  The menu bar appears with striped lines top and bottom, indicating that you're working in the background (Figure 1-4).

Callout and lines will be evened up in production

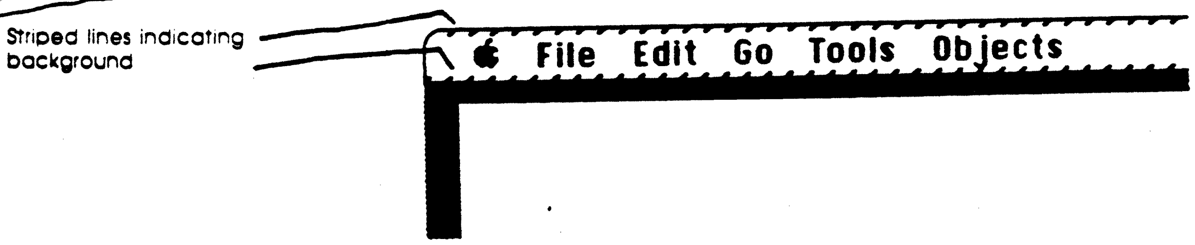

Striped lines indicating background
File Edit Go Tools Objects

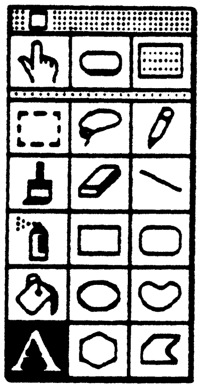
Figure 1-4
Working in the background

The steps you follow throughout this book make use of a number of shortcuts for menu commands and for getting around in HyperCard. Some of these shortcuts may be new to you at first, and you'll have plenty of opportunity to practice them.

Putting a title on the stack

It's a good idea to include a visible title or other identifier on each card of a stack, so you can always tell which stack you're in. Put a title on your practice stack using the steps that follow.

✦ *Are you in the background?* You should see stripes in the menu bar to indicate you're working in the background. If you don't see stripes, press Command-B.


The Paint Text tool

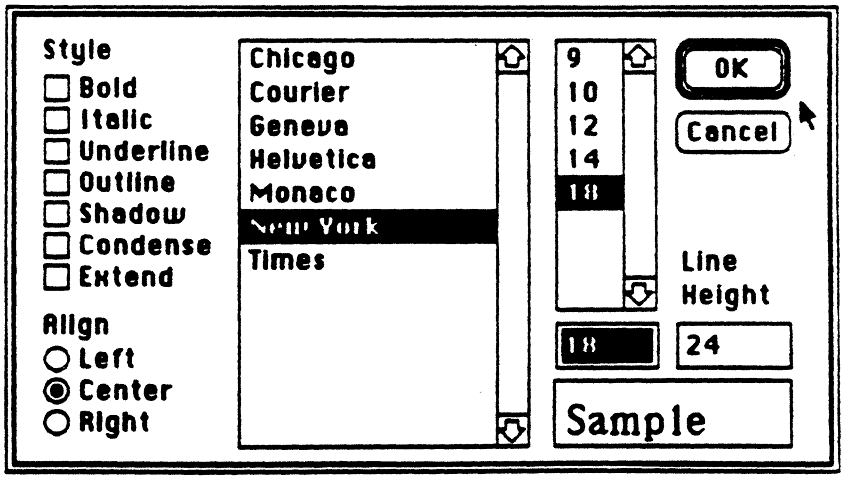
1. **Choose the Paint Text tool from the Tools menu.**

   If you prefer to work with a palette, you can turn the Tools menu into a palette by dragging past its bottom edge to "tear" it off the menu bar.

2. **Press Command-T to select the text style.**

   (You could also choose Text Style from the Edit menu or double-click the Paint Text tool on the Tools palette.)

   The Text Style dialog box appears as shown in Figure 1-5.


Style
☐ Bold
☐ Italic
☐ Underline
☐ Outline
☐ Shadow
☐ Condense
☐ Extend

Align
○ Left
● Center
○ Right

Chicago
Courier
Geneva
Helvetica
Monaco
[✓ New York]
Times

9
10
12
14
[✓ 18]

18

Line Height 24

Sample

OK
Cancel

Figure 1-5
Text Style dialog box

3. **Choose a font you like in a large, readable size—for instance, New York 18.**

4. **Click the "Center" button in the lower-left corner of the box so that your text will be centered as you type it.**

5. **Click OK.**

   You're ready to put the title on the stack.

6. **Click in the center of the card near the top to set the insertion point and then type the name of your stack.**

   When you're finished, the screen should look something like Figure 1-6. All you've added so far is the title.

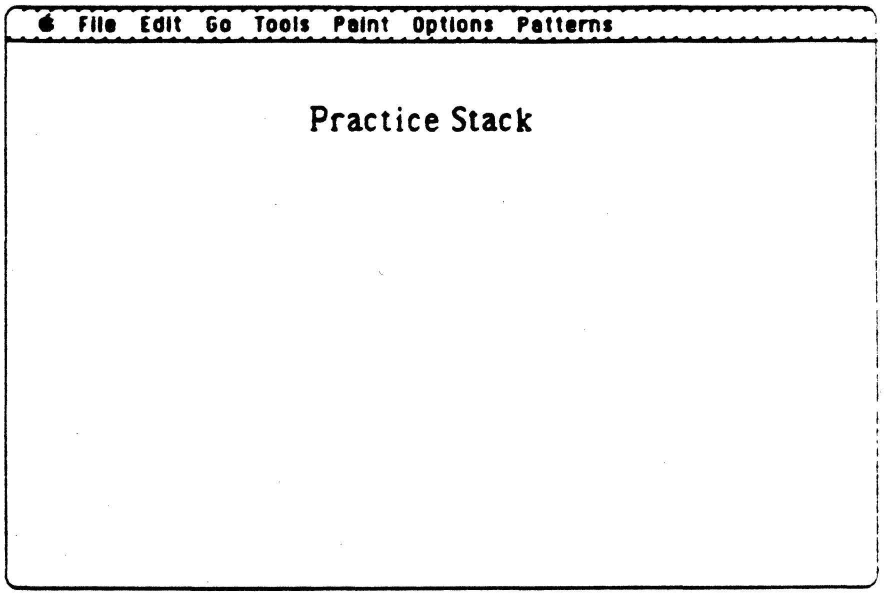


Figure 1-6
The practice stack with a title

Home, sweet Home

Whenever you see a small picture of a house in HyperCard, you can be pretty sure that clicking it will take you to the Home card. In the following sections, you'll add a Home to your stack and complete its script.

❖ *By the way:* In the *HyperCard User's Guide* you learned how to copy and paste buttons with prewritten scripts, such as Home buttons. In this book, you'll complete scripts yourself for practice.

Making a button

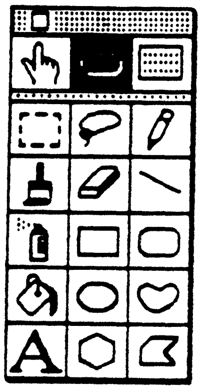
You can always get a new button by choosing New Button from the Objects menu. In this book, you'll use a keboard shortcut to make buttons. Follow these steps:

1. **Make sure you are working in the background.**

   You should see stripes in the menu bar. If you don't see stripes, press Command-B.

2. **Choose the Button tool from the Tools menu.**

   The Browse tool changes to the Button tool, which is an arrow pointer.



The Button tool

3. **With the pointer anywhere on the card, hold down the Command key.**

   Notice that the arrow pointer changes to a crosshair.

4. **While holding down the Command key, drag to create a small square button.**

   Release the mouse button when the button is about half an inch square. The new button is automatically selected so you can move it or change its size—you can tell it's selected by the moving dotted lines around its edges. (This effect is sometimes referred to as "marching ants.")

5. **Move the button to the lower-left corner of the card.**

   Drag the button by its center. Because it's in the background, the button will appear in this position on every card.

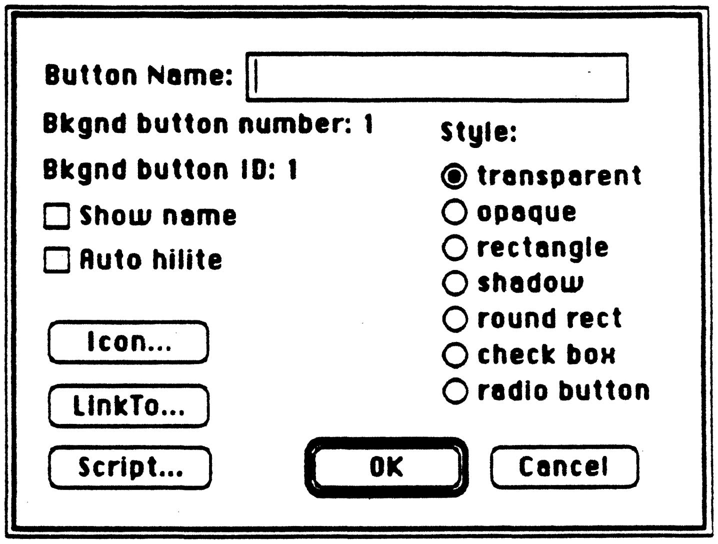
Customizing the button

HyperCard buttons have a variety of styles and features from which to choose. You customize a button's appearance and actions through the Button Info dialog box.

1. **Double-click the button to see the Button Info dialog box.**

   (You could also choose Button Info from the Objects menu.)

   Figure 1-7 shows this box.



Button Name:

Bkgnd button number: 1

Bkgnd button ID: 1

☐ Show name

☐ Auto hilite

Style:

[✓ transparent]

opaque

rectangle

shadow

round rect

check box

radio button

Figure 1-7
The Button Info dialog box

Notice that the insertion point is blinking in the Button Name box, ready for you to type a name.

2. **Type `Home` (but don't press Return)**

   If you press Return prematurely, don't worry; just double-click the button again to get back to the Info dialog box.

3. **Click "Auto hilite" to select it.**

   The "Auto hilite" option causes the button to become highlighted when it's clicked, which gives you a visual signal that you've clicked it.

   Leave the "Show name" option unchecked; you'll put an icon on this button instead.

4. **Click the Icon button.**

   Another dialog box appears in which you can select an icon for the button.

Some house icons

5. **Choose one of the house icons.**

   Scroll through the window until you find the house icons and click the one you want.

6. **Click OK.**

   All the dialog boxes disappear. Your new button now has the house icon on it.

Next, you'll write a script for this button.

And now, a little scripting

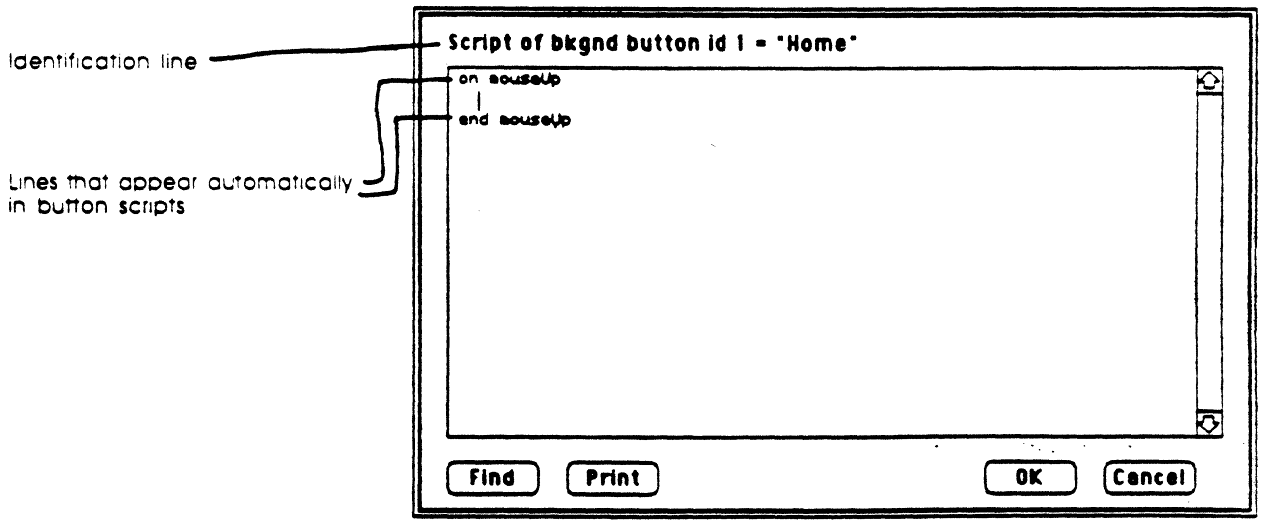
Scripts are created and changed in a special box called the **script editor.** To see the script for the new Home button:

1. **Double-click the Home button.**

   You see the Button Info dialog box again.

2. **Click the Script button.**

   You see a large dialog box with two lines of text already in the window. This box is the script editor for the Home button. (See Figure 1-8.)



Script of bkgnd button id 1 = "Home"

```
on mouseUp

end mouseUp
```

Caption here

Caption will be fixed in production

Figure 1-8
The script editor

Notice that the top line identifies which the script this is: "Script of bkgnd button id 1 = Home"—your new button. Notice also that two lines of text appear already—on `mouseUp` and `end mouseUp`—with the insertion point blinking in between. All scripts for new buttons have the first line and last line filled in for you.

The next step is to type the statement that defines the action of the button.

3. Type `go Home`

The new line should appear between the existing lines. If you make a mistake, use the Backspace (Delete) key to erase and type over.

The three lines constitute the completed script for the Home button:

```
on mouseUp
  go Home
end mouseUp
```

As you might guess, these instructions describe what should happen when someone clicks the Home button. You have one more step before you're finished, but first, here's a brief description of how the script works:

Whenever you move the mouse, the Macintosh computer and HyperCard software track the movement electronically. You see the movement as a change in the position of the pointer on the screen. When you press and release the mouse button, electrical signals are sent, something like when you turn a switch on and off. The same thing is true when you press different keys on the keyboard. The HyperCard software interprets these signals from the system and translates them into HyperTalk **system messages.**

MouseUp is a system message that means the mouse button has been released; an on-screen HyperCard button receives this message when someone clicks it (that is, positions the Browse tool on it and then presses and releases the mouse button). Actually, the button receives both mouseDown (the mouse button is pressed) and mouseUp (the button is released), but the mouse button must be released before a click is complete, so mouseUp is more frequently used in scripts.

Whether something happens when the button receives the mouseUp message depends on whether the button's script contains any instructions for that message.

The first line, on mouseUp, signals HyperCard that further instructions exist. Any subsequent lines contain HyperTalk statements that make up the instructions. The last line, end mouseUp, indicates the end of the instructions.

The word go is a HyperTalk command; it means what you might expect. Go must be followed by a destination—a description of a card or a stack. In this case, you used the name of the card. You could also have typed a more elaborate description, such as

```
go to card 1 of stack "Home"
```

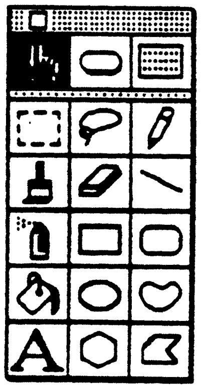
Translated into English, the instructions say

"When this button is clicked, go to the first card of the Home stack. That's all."

To leave the script editor,

4. Click OK.

The script editor disappears, and you're back to the practice stack. (If you click Cancel the same thing will happen, but your instructions won't be in the script.)


The Browse tool

Trying it out

Now see if the Home button works as it's supposed to.

1. Choose the Browse tool from the Tools menu or palette.

2. Click the Home button.

The next thing you see on the screen should be the Home card. Welcome Home!

If that's not what happened, switch to the Button tool and double-click the Home button to check the script. Make sure everything is typed correctly. Then click OK and repeat the steps.

To get back to the practice stack:

1. **Press Command-M to see the Message box.**

   (You could also choose Message from the Go menu.)

   The insertion point should be blinking inside the Message box, ready for you to type.

   If for any reason you previously typed something into the box, the earlier entry would still be there. Just start typing and the old text will be replaced.

2. **Type**
   ```
   go to stack "Practice Stack"
   ```
   (use the actual name of your stack in quotation marks).

3. **Press Return.**

   You should now see your practice stack on the screen.

As you see, you can use the `go` command both in scripts and in the Message box. Most HyperTalk commands work in both places; you can communicate directly with HyperCard through the Message box.

Buttons for traveling

Next you'll create two "travel buttons" to allow you to go back and forth between cards in the stack. (Right now there's still only one card, but you'll add more shortly.)

Making two new buttons

Use the same steps as you did for the Home button:

1. **Make sure you are working in the background.**

   You should see stripes in the menu bar. If you don't see stripes, press Command-B.

2. **Switch to the Button tool and use Command-drag to create two new transparent buttons.**

   Make them about the same size as the Home button.

3. **Position these two buttons side-by-side at the bottom of the card, roughly in the center.**

   Drag each button by its center to move it as needed.

Customizing the button on the right

Make the button on the right into a "move forward" button:

1. **With the Button tool still selected, double-click the button on the right.**

   The Button Info dialog box appears.

2. **Name the button** `Next`

3. **Click the check box to select "Auto hilite."**

4. **Click the Icon button to see the available icons.**

5. **Choose an icon that points to the right.**

   You can choose any size arrow or pointing finger. Click the one you want.

6. **Click OK.**

   The boxes disappear. You should see the arrow or finger on the button.

Customizing the button on the left

Repeat the steps for the remaining button:

1. **With the Button tool still selected, double-click the button on the left.**

   The Info dialog box appears.

2. **Name the button** `Previous`

3. **Click the check box to select "Auto hilite."**

4. **Click the icon button to see the available icons.**

5. **Choose an icon that points to the left.**

   It's best to use the same icon as you chose for the first button, but pointing the opposite way.

6. **Click OK.**

The two buttons should now have matching icons pointing away from each other.

Completing the scripts

You want the button on the right to take you to the next card in the stack and the button on the left to take you to the previous card. Put your instructions into the buttons' scripts:

1. **Hold down the Shift key and double-click the right-arrow button to see the script editor.**

   (You could also double-click the button and then click Script in the Info box. The Shift–double-click shortcut doesn't work in HyperCard versions earlier than 1.2.)

2. **Type**
   ```
   go to next card
   ```
   between the existing lines.

3. **Click OK.**

   The script editor disappears. Repeat the steps for the remaining button.

4. **Hold down the Shift key and double-click the left-arrow button to see the script editor.**

5. **Type**
   ```
   go to previous card
   ```
   between the existing lines.

6. **Click OK.**

You have now completed both buttons' scripts. The script for the button on the right contains

```
on mouseUp
  go to next card
end mouseUp
```

For the button on the left, it's

```
on mouseUp
  go to previous card
end mouseUp
```

These buttons can now be used to travel back and forth in the practice stack, card by card. Moving to adjacent cards isn't the only possiblity, of course; you can create other buttons to take you to any card of any stack you want by specifying in a script where you want to go.

❖ *By the way:* The LinkTo button in the Button Info box provides a shortcut for linking a button to any destination without going to the button's script. HyperCard completes the script for you. See the *HyperCard User's Guide* for details.

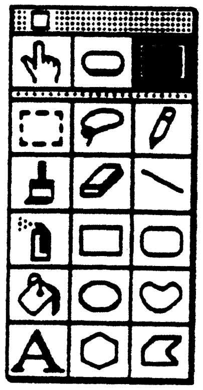
So far in this stack, there's nowhere else to go. It's time to add some cards.

Fill out the stack

Before you add new cards, it's a good idea to label this card in some way so that you know when you're at the first card. Later in this chapter you'll learn a way to label other cards you add as well.

Adding a label field

First, create a text field in the background to hold the label. Follow these steps:


The Field tool

1. **Make sure you're still working in the background.**

   If you don't see stripes in the menu bar, press Command-B.

2. **Choose the Field tool.**

3. **Hold down the Command key and drag to create a new field.**

   Make the field a rectangle roughly a quarter inch high and an inch and a half wide.

4. **Move the field to the card's upper right.**

   Drag it by its center, just as you did with the buttons.

5. **Double-click the field to see its Info box.**

6. **Click "Rectangle" to set the field's style.**

7. **Click the Font button.**

   The Text Style dialog box appears.

8. **Select a font and size:**

   Geneva 12, which may be already selected, is a good choice.

9. **Click OK (or press Return).**

   The Text Style dialog box closes, and you're back to the card.

Typing a label

The field you created will appear on every card because you put it in the background. Text in the field, however, can be different on every card. Type a label for this card into the field:

1. **Choose the Browse tool.**

   Notice that when you choose the Browse tool, the stripes disappear from the menu bar. You are no longer in the background of the stack.

2. **Click inside the field to set the insertion point, and then type the words "This is Card 1." (You don't need to include quotation marks.)**

The text you just typed will appear only on this card; the field, however, will appear on all cards, and you can type different text into it.

Adding five new cards

At last, you're ready to add some cards to this slim stack.

1. **Press Command-N five times.**

   (You could also choose New Card from the Edit menu five times.)

Although you haven't seen much happen on the screen, you've just increased the size of your stack from one card to six cards. Notice that the field in the upper-right corner is blank, indicating that you are no longer on the first card.

2. **Click the right-arrow button, and you should find yourself on Card 1 again.**

A script to label all cards

You could label all cards in your stack by going to each one and typing its number into the field, just as you did for Card 1. Instead, you can write a script telling HyperCard to do it for you. Here's how:

1. **Choose Stack Info from the Objects menu.**

   The Info dialog box for the stack appears.

2. **Click the Script button.**

   The script editor for the stack appears. Notice that, unlike the script editor for the buttons you created, the script editor for the stack does not contain the `on mouseUp` and `end mouseUp` statements.

❖ *Keyboard shortcut:* Press Command-Option-S to go directly to the script editor for the current stack, without going to the Info box. (This shortcut doesn't work with HyperCard versions earlier than 1.2.)

3. **Type the script that follows exactly as written. Press Return at the end of each line.**

```
on openCard
put "This is Card" && number of this card into field 1
end openCard
```

   Be sure that you type two ampersands (&&) and that you include the quotation marks.

When you press Return after the final statement, you'll notice that the last line moves over to the left, but the middle line remains indented. This automatic indenting helps you check your scripts. `on` and `end` should always line up at the leftmost edge of the script editor box after you press Return the final time; if they don't, you might have left out something important and should check the script again. Pressing the Tab key also checks the formatting.

If everything looks correct,

4. Click OK.

The script editor disappears.

This script labels each card as you go to it by typing "This is Card" and the card's number into the field. Try it out:

5. Click the right-arrow button to go to each card.

You should see the phrase appear in the field automatically as you go, with the correct card number.

The `openCard` message is sent to the current card whenever you go to it. The `put` command does what you would expect—it puts something where you want it to go.

The double ampersand (`&&`) connects two pieces, or **strings**, of text together with a space in between. One piece of text is "This is Card" and the other piece is the card's number, which you specified as `number of this card`. If you wanted to join two strings of text together without a space, you would use a single ampersand.

In English, the script says:

"When a card opens, put the phrase "This is Card" and the card's number with a space in between into field 1. That's all."

All cards in the stack will be labeled by this script because it's a stack script. A stack script can have an affect on all backgrounds, cards, fields, and buttons belonging to that stack. You could have put the script at the card level, but you would have had to copy it to *every* card's script or it wouldn't work for every card.

The advantage of using a script to label cards is that you won't have to worry about labeling the cards yourself, even if you add or delete cards. HyperCard will take care of it for you. What's more, you can lock the field to prevent anyone from typing into the field, but HyperCard will still be able to change the text.

Add a button to the Home card

Wouldn't it be convenient to have a button on the Home card that would take you directly to your scripting practice stack? Create one now:

1. **Choose the Button tool and create a new button.**

   Use Command-drag to create the button; make it fairly wide. Move it to any open space you have on the Home card.

   ❖ *By the way:* The buttons already on the Home card are there only for your convenience, and you can change their position easily; just click them with the Button tool and drag them to a new location. If you need more room, you can cut buttons that you don't use often and paste them elsewhere; you could create a new card in the Home stack to hold them.

2. **Double-click the button to see its Info box.**

   You could also choose Button Info from the Objects menu.

3. **Name the button `My Stack`**

4. **Click "Show name" and "Auto hilite" to select them.**

5. **Click "round rect" to make the button style a rounded rectangle.**

6. **Click the Script button to see the script editor.**

7. **Type the command that will take you to your practice stack.**

   Can you do it? Give it a try. Here's a hint: you typed this command into the Message box earlier after testing your Home button.

8. **Click OK.**

   You should see the Home card with the new button. If the button is too small for the words, drag one of its corners to make it larger.

9. **Choose the Browse tool and click the My Stack button.**

If you went to Card 1 of your practice stack, congratulations!

If something else happened such as a message appearing on the screen saying "Can't understand . . ." then you might have misspelled a word or left out a space. If you got a directory dialog box asking where the stack is, you might have typed the name incorrectly.

Any of these statements would work in the button's script:

```
go to stack "Name"
go to "Name"
go "Name"
go Name
```

The placeholder word *Name* stands for whatever you named your stack in its Stack Info box. Be sure that you type the name exactly as it is in the box; for example, if you included the word *Stack* in your stack's name, you'll have to include it with the `go` command.

Important Although it's possible in many cases to omit the quotation marks and still have a working statement, as a general rule it's best to include the marks. Quotation marks remove any ambiguity.

You should now be at Card 1 of your practice stack, ready to go on. Or, if you'd like to take a break, go ahead. In the next chapter you'll write some more elaborate scripts.

What you've done so far

In this chapter you've created a stack in which you can practice scripting in the rest of this book and on your own. You've completed scripts for three background buttons using the HyperCard script editor. Finally, you've created a background field and written a script to label all cards by number in that field.

Here's a list of the HyperTalk words you have learned:

Commands

`go`	This command is used to move around in and between stacks. The word `go` must be followed by the name of a card or a stack. `Go` works in scripts or in the Message box.

`put`	As you might guess, this command takes something and puts it somewhere. The word `put` must be followed by the name of the thing you want to put somewhere and the name of the place you want to put it.
Messages	
`mouseUp`	A system message; when you click something, such as a button, the system sends `mouseUp` when the mouse button is released. (If the pointer is moved off the button before the mouse button is released, `mouseUp` is not sent.)
`mouseDown`	A system message sent when the mouse button is pressed.
`openCard`	A system message sent to a card when it is opened.
Modifiers	
`next`	This word means the same thing as the English word.
`previous`	Another word that means the same thing as the English word; it can be abbreviated `prev`.
Miscellany	
`&`	(Ampersand) This symbol joins two pieces, or strings, of text together.
`&&`	(Double ampersand) This combination symbol joins two pieces of text with a space in between.
`end`	This word that signals the end of a set of instructions. All HyperTalk scripts conclude with an `end` statement.
`on`	This word that signals the beginning of a set of instructions. It must be followed by the name of a message, such as `mouseUp`.
`to`	The word `to` is used different ways in HyperTalk. It's optional with the `go` command; `go to stack "Scripting"` means the same as `go stack "Scripting"`.

Chapter 2

Special Effects

You might already know that buttons, fields, cards, backgrounds, and stacks in HyperCard are called *objects*. More specifically, **objects** are HyperCard elements that can

- receive and send messages
- act on messages according to instructions in their scripts

Not all elements in HyperCard are objects. Elements that are not include any graphics or text you create with the Paint tools, the text inside fields, any dialog boxes that appear, the menu bar at the top of the screen, and the menus and palettes. The Message box is also not an object, even though you can send messages with it.

When you copy (or cut) and paste any object, its script goes along with it; thus, you don't have to build a button from scratch, as you did with the buttons in Chapter 1, every time you want one.

❖ *By the way:* The Button Ideas stack contains buttons with prewritten scripts that you can copy into your own stacks.

In this chapter you'll create more buttons and add some special effects to button scripts using new commands.

If you took a break and quit HyperCard at the end of Chapter 1, you need to start up HyperCard again. Use the button you added to the Home card to get to your practice stack. You're ready to go on when you see Card 1 of your practice stack on the screen.

Some visual effects

HyperCard's visual effects make movement between cards and stacks noticeable and visually interesting.

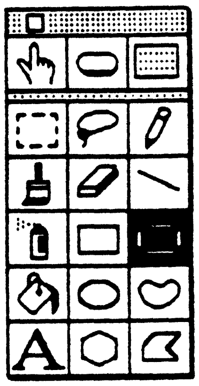
You add visual effects to scripts using the `visual` command. In the sections that follow, you'll add visual effects to your stack's buttons.

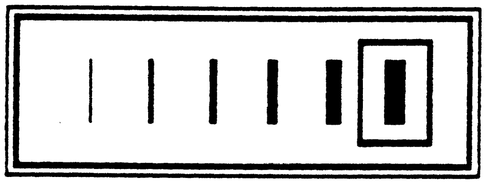
Right now, the practice stack is pretty bare visually, so visual effects won't be very effective. Let's add some graphics to the stack.

Important HyperCard visual effects aren't visible with color or with multiple grays selected. Use the Control Panel (available in the Apple menu) to change the Monitors setting to "Black & White/Grays" and the number of grays to 2.

Adding graphics

For demonstration purposes, use the Paint tools to put a border on all cards:


The Rounded Rectangle tool


The Line Size box

1. **Press Command-B to work in the background.**

   The menu bar becomes striped.

   Putting the border in the background means you'll have to draw it only once.

2. **Choose the Rounded Rectangle tool from the Tools menu.**

   The Browse tool changes to the crossbar pointer.

3. **Choose Line Size from the Options menu to see the Line Size dialog box.**

4. **Click the line size you want. Choose one of the wider sizes.**

   The Line Size box closes automatically when you select a size.

5. **If you want to make a patterned border, choose a pattern from the Patterns menu.**

   If you don't choose a pattern, the border will be a black line.

6. **Hold down the Option key, position the crossbar inside the top-left corner of the card, and drag to the bottom-right corner. Then release the mouse button.**

   Using Option-drag draws the rectangle with the selected pattern. If you didn't choose a pattern, you don't need to use the Option key.

   If you don't like the position of the rectangular border and want to try again, press Command-Z to undo the drawing before clicking anywhere else.

7. **Press Command-B or choose the Browse tool to stop working in the background**

You can add some of your own graphics at the card level if you like; however, leave cards 4, 5, and 6 blank. You'll need to draw on them later in this book. Figure 2-1 shows a sample of Card 1 with the background border completed and some optional, whimsical graphics added to the card.

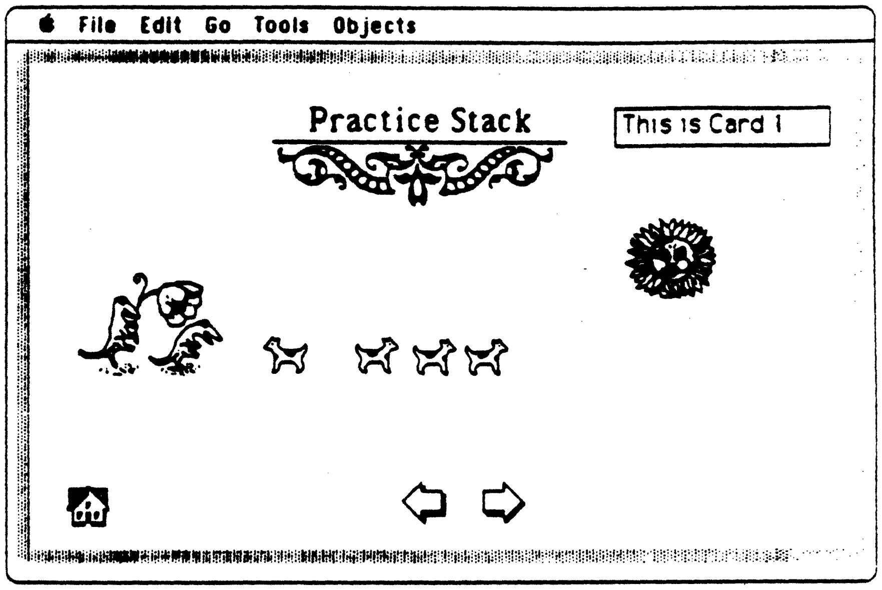


Figure 2-1
Sample Card 1 with graphics

The card border and other graphics you add to the practice stack are only for the purpose of being able to see certain visual effects; cards do not necessarily have to have borders. If you were creating a stack for some other purpose, you would want to consider the card layout and inclusion of graphics carefully. For information on designing stacks, see the *HyperCard Stack Design Guidelines*.

The Visual command

Before you go on, go back to Card 1, if you aren't there already, and choose the Browse tool.

The most common use of visual effects is during transition between cards. In this section you'll add effects to the arrow buttons and the buttons that take you back and forth from your stack to Home.

Adding effects to the arrow buttons

Follow these steps to add a visual effect to the right-arrow button:

1. **With the Browse tool still selected, hold down the Option and Command keys.**

   Pressing these two keys lets you see the outline of all buttons on the card—even invisible (transparent) ones.

2. **Still holding down Option and Command, click the right-arrow button.**

   The script editor appears showing the button's script.

   This shortcut allows you to go directly to the script without switching to the Button tool first—a handy feature when you're doing a lot of scripting. (In versions of HyperCard earlier than 1.2, this shortcut doesn't work. An alternative is to switch to the Button tool and Shift–double-click the button.)

❖ *By the way:* Even though you had to switch to the background when you created this button, you do not have to switch to the background to change its script.

3. **Click in front of the word `go` to place the insertion point.**

4. **Type**
   `visual effect scroll left`
   and press Return.

   The script should now look like this:

```
on mouseUp
  visual effect scroll left
  go next card
end mouseUp
```

5. **Click OK.**

   The script editor disappears. The Browse tool should still be selected if you used the Command-Option-click shortcut.

To see how the visual effect works, click the button with the Browse tool.

6. **Add the same effect, but going the opposite direction, to the left-arrow button.**

   Follow the same steps as you did for the right-arrow button, but type

```
visual effect scroll right
```

The `scroll` effect causes the entire screen image, including the background elements, to appear to move in the direction indicated. It's good for simulating pages turning.

❖ *By the way:* Notice that you use `scroll left` for the *right* arrow and `scroll right` for the *left* arrow to simulate page turning in the English language, which is read right-to-left. In other languages, pages might turn the opposite direction.

Adding an effect to the Home button

This time you'll use a different effect:

1. **Press Command-Option and click the Home button.**

   (You could also use Shift–double-click with the Button tool selected.)

   The script editor for the Home button appears.

2. **Place the insertion point in front of the `go` statment.**

3. **Type this line and press Return:**

   ```
   visual effect wipe left slowly
   ```

   Remember to press Return so that the statement is on its own line, but don't click OK to close the script editor yet.

4. **Leave the script editor on the screen for now.**

   If you already closed the script editor, just open it again with Command-Option-click.

The word `slowly` is a modifier that controls speed. You can choose from four options:

```
very fast
fast
slow[ly]
very slow[ly]
```
(The *-ly* is optional with `slow`.)

If you don't choose any of these, the effect runs at "normal" speed. The speed modifier should always follow the name of the effect.

Adding the same effect to the button on the Home card

Rather than type the command, you can just copy the command from the script editor of the Home button.

If you happened to close the script editor for the Home button, use Command-Option-click to open it again.

1. **Drag across the line with the visual effect as you would any text line to select it.**

   Make sure you select only the line with the visual effect.

2. **Press Command-C to copy the line.**

   The command statement is copied to the Clipboard.

3. **Click OK to save the script and close the script editor.**

4. **Click the Home button with the Browse tool.**

   The Home card appears. Notice the visual effect during transition—the wipe effect you just added. It's as though the first card is "wiped off" the next one.

5. **Command-Option-click the My Stack button to see the button's script editor.**

6. **Click in front of the `go` statement to place the insertion point.**

7. **Press Command-V to paste the visual effect.**

   You might also have to press Return to put the `go` command on a separate line after you paste.

8. **Click OK to save the script and close the script editor.**

9. **Click the My Stack button with the Browse tool.**

   You should go back to Card 1 of the practice stack, seeing again the wipe effect.

Being able to cut and paste scripts can save you a lot of typing. You must use the keyboard shortcuts for Edit menu commands when you're using the script editor, however; the Edit menu is not available. Table 2-1 lists the script editor keyboard commands.

Table 2-1
Script editor command summary

Key press	Action
Command-A	Select entire script
Command-C	Copy selection to Clipboard
Command-F	Find text (same as Find button)
Command-G	Find next occurrence of same text
Command-H	Find current selection
Command-P	Print selection or (if no selection) entire script (same as Print button)
Command-period	Close script without saving changes (same as Cancel button)
Command-V	Paste Clipboard contents at insertion point
Command-X	Cut selection to Clipboard
Enter	Close script and save changes (same as OK button)
Option-Return	Wrap line without return character ("soft" return—symbolized by ¬ in scripts. Don't use a "soft" return inside quotation marks.)
Return	Return character—indicates end of HyperTalk statement
Tab	Format script

More experiments with visual effects

You can make some test buttons on Card 1 of your practice stack to try out some of the visual effects. These test buttons will demonstrate the effects without your having to move to another card.

Here's a list of HyperCard visual effects:

`barn door close` (or `open`)
`checkerboard`
`dissolve`
`iris close` (or `open`)
`plain` (same as no effect)
`scroll down` (or `up`)
`scroll left` (or `right`)
`venetian blinds`
`wipe down` (or `up`)
`wipe left` (or `right`)
`zoom close` (or `open`)
`zoom in` (or `out`) (same as `zoom close`)

Some visual effects have a more noticeable effect than others, depending on the context. For example, the `scroll` effect creates a clearer transition than `wipe` does when only a few elements change from one card to another. `Wipe` is most effective when two cards have very different appearances. `Checkerboard` and `venetian blinds` can have an entertaining or humorous effect.

Barn Door

Make a button to see the `barn door` effect following these steps:

1. **Create a new button.**

   Choose the Button tool, hold down the Command key, and drag. Make the button wider than it is high.

2. **Double-click the button to get to its Info box**

   The Button Info box for the new button appears. Notice that this button is a card button, not a background button; it will appear only on Card 1.

3. **Name the button `Barn Door`**

4. **Select these options: "Show name," "Auto hilite," and "round rect."**

5. **Click the Script button to go to the script editor.**

   The insertion point is blinking at the beginning of the line between `on mouseUp` and `end mouseUp`.

6. **Type these statments, pressing Return after the first two lines (but not after the last line):**

```
visual effect barn door close to gray
visual effect barn door open to card
go to this card
```

❖ *By the way:* The word `effect` is optional after `visual`. You can leave it out and the command will still work.

7. **Click OK to close the script editor.**

   If the name is too big for the size of the button, drag the corner of the button to make it larger.

8. **Choose the Browse tool and try out the new button.**

   You should see gray "doors" close and then open. (This example is only one way to use the `barn door` effect; you don't necessarily have to pair the `open` and `close` versions.)

The `visual` command must be accompanied by a `go` command—the statement `go to this card` satisfies the requirement, even though it doesn't take you anywhere. (More specifically, it takes you to where you already are.)

The phrases `to gray` and `to card` determine the image HyperCard uses during transition. You can use any of the following words for the image:

`black`
`card` (the image of the destination card)
`gray` (or `grey`)
`inverse` (reverses the card image)
`white`

Dissolve

Create another button to test the `dissolve` effect:

1. **Create a new button and name it "Dissolve."**

   Follow the same steps and choose the same settings as you did for the Barn Door button.

2. **Click Script to see the script editor.**

3. **Type the following lines:**

```
visual dissolve slowly to black
visual dissolve slowly to white
visual dissolve slowly to card
go to this card
```

4. **Click OK, switch to the Browse tool, and try the button.**

   You should see the image fade to black, fade to white, and then fade to the card image.

The effects you've just created can be cut and pasted into scripts for other buttons to travel between cards.

When creating a stack for your own use or for others, you can combine a number of effects to give different visual impressions; for example, zooming in on a subject, turning pages, or changing the scene completely.

Create other test buttons on Card 1 as you like.

The syntax of the Visual command

You've seen several versions of the `visual` command. Each version follows a certain general structure, with or without the optional elements.

An expression of the general, underlying structure that a given command must follow is called its **syntax.** Knowing a command's syntax is as important as knowing its name and what it does; however, you don't have to try to memorize syntax just now; you can refer to this section whenever you need to.

Syntax is a description of the way in which words are put together to form meaningful phrases. All languages—for people and for computers—have rules of syntax.

Here's the syntax of the `visual` command:

`visual [effect]` *effectName* [*speed*] [to *image*]

Optional elements are shown enclosed by square brackets. (You do not include the brackets in an actual command.) Words in italic are placeholders: for example, in an actual command, you would replace *effectName* with any of the actual effect names: `barn door`, `checkerboard`, `zoom`, and so on. The same would apply for *speed* and *image*.

A statement's syntax shows you the correct order for elements in the statement; for example, if you were to write this command:

```
visual fast dissolve
```

HyperCard would not be able to understand the command because the *speed* element is in the wrong place. The correct order is

```
visual dissolve fast
```

HyperTalk syntax is much like English syntax, which makes HyperTalk an easy language to use. You can't always be sure, however, that a statement that makes sense in English will make sense in HyperTalk. Incorrect syntax will cause a "Can't understand" message; in such a case, check the statement's syntax if you find no spelling errors.

The Appendix and the Quick Reference Card both contain a list of HyperTalk commands showing their syntax.

Some sound effects

Two HyperTalk commands cause sound: the `beep` command, which causes the usual Macintosh system beep, and the `play` command, for adding other sounds and music.

Important To hear sounds, you must have the Speaker Volume in the Control Panel set to a value greater than zero.

To use the `beep` command with a button, you would write a script like this:

```
on mouseUp
  beep
end mouseUp
```

You can cause multiple beeps by adding a number after the command, as in `beep 3`. If you don't add a number, you get a single beep. In Chapter 3 you'll use this command when you create an alert box.

The `play` command lets you add music to scripts; you can specify a number of notes with different pitches and time values and thus have a melody play, or you can use digitized sounds (sounds recorded in a digital format that computers can understand). Make a new button to try out the `play` command:

1. Create a new button on Card 1 with the name "Sound."

Use the Command-drag shortcut with the Button tool as usual. Bring up the Button Info box by double-clicking the button, type the name, and choose the "Auto hilite," "Show name," and "round rect" options.

2. Click the Script button to see the script editor.

3. Complete the script by typing this line:

```
play "harpsichord" "c c g g a a g"
```

Be sure to include the quotation marks with the instrument name and the series of letters representing notes.

4. Click OK.

Now try the button with the Browse tool. You should hear the first line of a familiar childhood tune.

The syntax of the Play command

The `play` command allows you to control pitch and tempo as well as voice. Here's the command's basic structure:

play "*voice*" [tempo *tempoValue*] ["*notes*"]

Voice is either `harpsichord` or `boing`, which are included with HyperCard, or *voice* could be the name of a digitized sound from some outside source.

You can optionally set the tempo (speed of play) by including the word `tempo` followed by a number (*tempoValue*). The value 100 is a medium speed; higher numbers play faster. If you don't specify a tempo, `tempo 100` is assumed.

Notes make up the melody sequence. Include quotation marks around the voice and the notes. For example,

```
play "boing" tempo 200 "e4q d c d e e eh"
```

plays "Mary Had a Little Lamb."

Specifying the notes

Use this section for reference when writing out melodies; you don't have to try to memorize the information here.

The notes are represented by the letters A through G, corresponding to Western music notation (capitalization makes no difference). You can include further modifiers after the notes to indicate sharps or flats, pitch range, and duration (or how long the note lasts).

Use `#` for sharp or `b` for flat immediately after the note. A sharp makes a note a half tone higher; for example, `d#` is the pitch halfway between D and E. A flat makes a note a half tone lower.

Use a number following the note and any sharp or flat to specify the pitch range. For example, `g#4` would be the G-sharp note in the middle range, or what musicians call the *middle-C octave.* Higher numbers give higher ranges, and vice versa.

Use a letter code following the note, any sharp or flat, and any range number to specify how many counts, or beats, to hold the note before the next note sounds. The timing values are relative to each other. Here are the codes for note duration:

`w`	whole note (four counts)
`h`	half (two counts)
`q`	quarter (one count)
`e`	eighth (one-half count)
`s`	16th (one-fourth count)
`t`	32nd (one-eighth count)
`x`	64th (one-sixteenth count)

As an example, `Bb5q` would mean the note B-flat in the high-C range held as a quarter note.

A period (.) after the duration code means a value of half again as much; that is, `w.` would indicate six counts (four plus half of four). A numeral 3 after the duration code means a triplet.

The codes for pitch range and duration carry over to subsequent notes unless you change them; this feature saves you from having to type numbers and letters over and over. (See "Mary Had a Little Lamb" shown earlier.)

❖ *By the way:* Even if you have no formal music training and all these terms seem mystifying, you can still make melodies with the `play` command. The best way to gain an understanding of how to use the notes is to experiment on your own. Choose a short tune you already know and try to write it out. You can use the script for the Sound button you created earlier to test and change the tune until it sounds right to you.

Dealing with long lines

You can put a long sequence of notes into a script; however, the script editor doesn't wrap lines or let you scroll to see lines that extend beyond the window. You can press Return or Option-Return to wrap a long line temporarily while you type the notes; however, if you use this method you *must* eliminate the end-of-line breaks when you're finished or the script won't work properly. The reason is that HyperCard doesn't understand a line break of any sort occurring inside quotation marks.

You can, however, wrap a long line permanently by adding closing quotation marks and the double ampersand (&&) followed by an Option-Return (¬):

```
on mouseUp
  play "harpsichord" "c c g g" &&¬
  "a a g"
end mouseUp
```

Notice that you must begin the wrapped line with a quotation mark.

What you've done in this chapter

In this chapter you've used HyperTalk commands to produce special effects: visual effects and sound. You've also added to your vocabulary list.

Commands

`beep`	The command that produces the system beep. You can cause multiple beeps by including a number: `beep 3`.
`play`	The command that causes notes to play. You specify the sound and the sequence of notes.
`visual [effect]`	The command that causes the visual effects you specify. It must be followed by the `go` command.

Names of sounds

`"boing"`

`"harpsichord"`

Names of effects

`barn door`

`checkerboard`	
`dissolve`	
`iris`	
`plain`	(Same as no effect.)
`scroll`	
`venetian blinds`	
`wipe`	
`zoom`	

Miscellaneous

`fast`	A modifier used with visual effects.
`slow[ly]`	A modifier used with visual effect.
`tempo`	A word that you use with the `play` command to control the timing of the notes.
`very`	A modifier used with fast or slow; means "more."

Chapter 3

More About Messages

Earlier you learned about HyperCard *system messages*—information about system events such as clicks (mouseUp), keyboard actions, and events in HyperCard (openCard). System messages are sent constantly while HyperCard is running. There's even a message for when nothing is happening: idle. (See the Appendix for a list of HyperCard system messages.)

A script, as you've seen, can contain instructions to be carried out when a particular message is recieved—in other words, the script "handles" the message. Thus, a complete set of instructions dealing with a message is called a **message handler.** Message handlers always begin with the word on and end with the word end, and both words are followed by the name of whatever message the handler deals with; for example on mouseUp.

An object's script might contain a number of handlers, each one handling a different message. Strictly speaking, then, the word **script** refers to everything that appears in the script editor for a given object, and not just to a single handler.

❖ *By the way:* On and end belong to a group of HyperTalk words called *keywords*. **Keywords** have predefined meanings that can't be changed.

In this chapter you'll write new handlers and explore the way messages travel between objects.

Sending messages

When someone clicks a screen button, the action generates a mouseUp system message. The mouseUp message always goes first to the button that was clicked. If that button's script doesn't have a handler for mouseUp, the message is passed to the card, then to the background, then to the stack, then to the Home stack, and finally to HyperCard itself. This sequence is called the **message-passing hierarchy** or the **object hierarchy;** it's illustrated in Figure 3-1

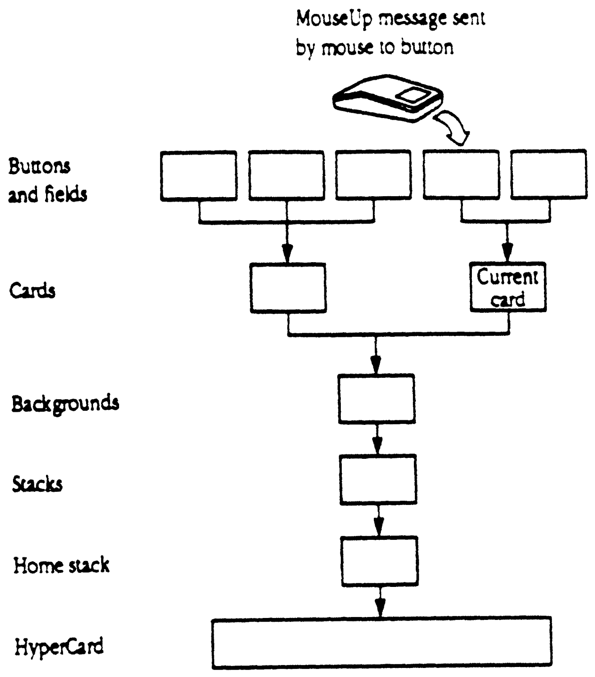


Figure 3-1
A message moving through the object hierarchy

You can place handlers at different levels; where you place a handler has an effect on its availability. For example, when you wrote the handler to label all cards of your practice stack, you placed it in the stack script; that placement meant that the handler was available for every card in the stack.

Messages can come from the system, from menus, from your actions with the mouse, keyboard, or Message box, or even from handlers themselves. You can write a handler that will send a message or pass on a message to another object. In this section you'll see how this feature works.

Create a "Receiver" Button

First, if you left HyperCard after the last chapter, start it up again and go to the practice stack.

Create a new button with the steps that follow; you'll use this button as a target for messages.

1. **Go to Card 2 of your practice stack.**

   You can think of Card 1 as your special effects card. Card 2 can be your message experiments card.

2. **Create a new card button and name it `Receiver`**

   You don't need to switch to the background because this is a card button.

   Follow the procedure you've used in previous chapters: use the Button tool and Command-drag to create a new button. Double-click the button to see its Info box. Type the name in the field at the top of the Info box.

3. **Select "Show name," "Auto hilite," and "round rect."**

4. **Click the Script button to see the script editor.**

5. **Type this line between `on mouseUp` and `end mouseUp`:**

   ```
   play "boing" tempo 80 "c4 e g"
   ```

   As you can see, this statement will cause three notes to play.

❖ *Alternative for hearing impaired people:* If you can't hear notes, type this line in place of or in addition to the `play` statement to see the effect of the handler:

   ```
   flash 3
   ```

   This command causes the entire screen image to flash rapidly three times when the button is clicked. (The white parts of the card switch to black and the black parts to white; then they change back again.)

6. **Click OK when you're finished.**

   The script editor closes and you're back to the card.

7. **Change to the Browse tool and try the button.**

   Notice that the button becomes highlighted when you click it (because of the "Auto hilite" setting) and the notes play immediately. you'll use the "Auto hilite" feature to distinguish between the sources of messages in this chapter.

In the next section, you'll send `mouseUp` without clicking.

Send a message with the Message box

You can send the Receiver button a message using the Message box:

1. **Press Command-M to see the Message box.**

2. **Type this sentence into the Message box:**

   ```
   Send mouseUp to button "Receiver"
   ```

3. **Press Return.**

   You should hear the notes play immediately; but notice that the button does not become highlighted. (To send the message again, just press Return.)

You selected "Auto hilite" when you created the button. But "Auto hilite" responds to `mouseDown` and `mouseUp` only when they are sent by the system as system messages—that is, when the button is actually clicked.

❖ *Well, almost:* You can "click" the button without a mouse using the `click` command. The button will respond to this command just as though it had been clicked manually. See the "Syntax Summaries" section later on.

The `mouseUp` message you sent from the Message box isn't a result of a click, so the button remains unhighlighted. The handler, however, still responds, and the notes play.

4. **Click the close box to hide the Message box again when you're finished.**

You can send messages from the Message box—and also from handlers—using the `send` keyword.It's the only keyword that works in the message box; it behaves a lot like a command does. Messages sent with `send` go directly to whatever object you specify, allowing you to bypass the usual hierarchy.

Create a "Sender" button

Follow these steps to make a button with which you'll practice sending messages from inside a handler:

1. **Create another card button anywhere on the card and name it `Sender`**

   Use Command-drag with the Button tool selected to make the button, then double-click the button to see its Info box and type the name.

2. Select "Show name," "Auto hilite," and "round rect."

3. Click the Script button to see the script editor.

4. Type these lines between `on mouseUp` and `end mouseUp`:

```
send mouseUp to button "Receiver"
wait 2 seconds
set hilite of button "Receiver" to true
wait 1 second
set hilite of button "Receiver" to false
```

5. Press Tab to format the script if necessary.

`On mouseUp` and `end mouseUp` should line up at the left edge of the window; all the other lines should be indented.

In English this script says

"When this button is clicked, send a `mouseUp` message to the Receiver button. Wait two seconds, and then highlight the Receiver button. Wait one second, and remove the highlighting. That's all."

6. Click OK when you're finished.

7. Change to the Brose tool and click the Sender button.

You should hear the notes play; after a two-second delay, you should see the Receiver button become highlighted. After one second the highlighting disappears.

You use the `set` command to change certain **properties** of objects or of HyperCard in general. In this example, the `hilite` property of the Receiver button is changed to `true` (button highlighted) and then back to `false`. Examples of properties you can change using `set` are the user level, the button style, the name of any object, a pattern from the Patterns palette, and many others. The Appendix contains a complete list of properties.

You can think of **properties** as characteristics of particular objects or of the HyperCard environment as a whole. You set values for properties with dialog boxes, palettes, check boxes, and radio buttons—or you can set them with scripts.

The `wait` command allows you to insert a delay. In this case, you used a two-second delay between when the button became highlighted and the notes started to play, and then a one-seocnd delay before removing the highlighting.

The buttons and handlers you've made in this section demonstrate sending a message in three different ways:

- As a system message: When you click the Receiver button, `mouseUp` is sent as a system message. The Receiver button becomes highlighted when you click it because of the "Auto hilite" option. The notes play (or the screen flashes, if you used that option) as indicated in the button's handler.
- As a Message-box message: When you use the `send` command in the Message box to send `mouseUp` to the Receiver button, the notes play as indicated in the handler, but because the button was not actually clicked, it doesn't become highlighted.
- From within a handler: When you use the `send` command in the handler of the Sender button to send `mouseUp` to the Receiver button, the button doesn't become highlighted right away because the button isn't actually clicked; but the notes play as indicated. However, you can add multiple commands to a handler to affect the Receiver button. In this case, you used the `set` command to change its highlighting.

Action at a distance

Where you place a handler in HyperCard affects its action. A handler at the "top" level, namely, in a button script or a field script, can respond only to a message received by that button or field. The same handler further "down" in the object hierarchy, such as at the card, background, or stack level, can respond to the message sent to any objects higher up, unless those objects intercept the message with their own handlers.

What the message-passing hierarchy means to you is that you can control whether your scripts act very locally, say, only for a particular button, or more globally, for an entire card, background, or stack.

In this section, you'll move the `mouseUp` handler of the Receiver button to different levels in the object hierarchy to experience the change in its response.

Remove the handler from the button script

Follow these steps to cut the handler from the Receiver button's script, placing it on the Clipboard automatically:

1. **Open the script editor for the Receiver button.**

   Use Command-Option-click with the Browse tool, or Shift–double-click with the Button tool.

2. **Press Command-A to select the handler.**

   Command-A selects the entire script, but in this case there's only one handler in the script.

3. **Press Command-X to cut the handler and place it on the Clipboard.**

   The script editor should now have nothing in it. If you still see the handler there, try steps 2 and 3 again.

4. **Click OK.**

   The script editor disappears and you're back to the card.

The script for the Receiver button is now **empty**. You can test it by clicking the Sender button with the Browse tool. You should see the Receiver button flash (because of "Auto hilite"), but hear no sound.

Every object has a script, even if there's nothing in it. Scripts with nothing in them are called **empty scripts.**

Move the handler to the card level

Paste the handler into the card's script.

1. **Choose Card Info from the Objects menu.**

2. **Click the Script button in the Info box.**

   The top line of the script editor tells you that it's the script for the card.

❖ *Keyboard shortcut:* You can press Command-Option-C to see the script editor of the current card without having to go through the Info box. (This shortcut doesn't work with HyperCard versions earlier than 1.2.)

3. **When you see the script editor, press Command-V to paste the handler.**

4. **Click OK.**

5. **Test the effects.**

   Switch to the Browse tool. First, click the Receiver button; you should see no difference in what happens: the button becomes highlighted and the notes play. The `mouseUp` message passes through the empty button script and goes on to the card script.

   Now, click the Sender button. Again, you should hear the notes, and then after two seconds see the button become highlighted and then change back.

   And now, click anywhere on the card (except on another button or in the field). The notes play because whenever you click the card, `mouseUp` goes directly to the card, which now has a handler for `mouseUp` in its script.

Move the handler to the background level

Take the handler out of the card script and move it to the Background script:

1. **Open the script editor for the card again.**

   Choose Card Info from the Objects menu and click Script, or simply press Command-Option-C.

2. **Press Command-A to select the handler.**

3. **Press Command-X to cut the script and place it on the Clipboard.**

   The card script should now be empty.

4. **Click OK.**

5. **Open the script editor for the background by choosing Bkgnd Info from the Objects menu and clicking the Script button.**

❖ *Keyboard shortcut:* Press Command-Option-B.

6. **Press Command-V to paste the handler.**

7. **Click OK.**

8. **Test the effects.**

Using the Browse tool, click the Receiver button, the Sender button, and the card, just as before. You should hear the notes play.

Now, move to any other card in the stack and click any area except a button or field—you should still hear the notes play. The handler is now available to any card sharing the background.

If you moved the handler to the script level, the same thing would happen because this practice stack has only one background; however, in cases where a stack has more than one background, only a handler at the script level or above would be available to all cards of all backgrounds.

❖ *Other handlers intercept messages:* The reason you don't hear the notes if you click one of the travel buttons or other buttons besides Sender and Receiver is that those buttons already contain `mouseUp` handlers. Once a message is handled, it's not passed on unless you specifically pass it using the `pass` keyword.

Change the handler

If you were to leave the `mouseUp` handler where it is, in the background, you'd hear notes any time you happened to click somewhere other than a button. You can do one of two things: take the handler out of the background and move it back to the button; or change the handler's name from `mouseUp` to something else—in other words, change the handler so that it no longer responds to `mouseUp`, but to some other message. These steps show you how to do the second alternative:

1. **Open the script editor for the background (Command-Option-B).**

   Use Command-Option-B or choose Bkgnd Info from the Objects menu and click Script.

2. **Select the word `mouseUp` in the first line**

   Drag across the word as you would when selecting any text.

3. **Replace it by typing the word `playTune`**

   `PlayTune` serves as the alternative name. You could use any other word (except a HyperTalk keyword); this name seems appropriate because it describes the action of the handler.

❖ *By the way:* If you are using the `flash 3` alternative instead of the notes, you could use a different name, such as `razzleDazzle` or something more fitting (don't use `flash` though). Be sure, however, that you use your alternative name in the steps that follow.

4. **Select the word `mouseUp` in the last line**

5. **Replace it also by again typing the word `playTune`**

   The name used after `on` must match the name after `end`.

You have now changed the handler from a `mouseUp` handler to a `playTune` handler. It will not longer respond to the `mouseUp` message, but instead to the message `playTune`. But where does a `playTune` message come from?

The answer is that you'll put a new handler in the Receiver button's script that will send a `playTune` message:

6. **Click OK to save the handler and close the script editor.**

   You're back to the card again.

7. **Open the script editor for the Receiver button.**

   Use Option-Command-click with the Browse tool, or Shift–double-click with the Button tool.

   You should see the `on mouseUp` and `end mouseUp` lines already in the script editor. HyperCard always adds the lines to "empty" button scripts.

8. Type the following word between the two lines:

```
playTune
```

The completed handler should look like this:

```
on mouseUp
  playTune
end mouseUp
```

8. Click OK to save the handler and close the script editor.

9. Test the effects.

Clicking the Receiver button or the Sender button should have the same effect as they did before you moved the handler. Sending `mouseUp` from the Message box to the Receiver button should also work the same. But clicking anywhere else on the card won't cause the notes to play, because the background handler isn't a `mouseUp` handler any more.

Now, when the Receiver button receives `mouseUp`, its handler in turn sends the message `playTune`. That message goes down the hierarchy until it's intercepted by the `playTune` handler in the background script.

Try this: go to some other card, open the Message box (press Command-M), type the word `playTune` and press Return. You'll hear the notes because the Message box sends the word as a message along the hierarchy.

(If you wanted to send `playTune` to some object not in the hierarchy, you would use the `send` keyword in the handler to specify the destination; otherwise, `playTune` alone is sufficient.)

What you've done in this section is essentially define a new command, which is named `playTune`. That's really all there is to defining your own commands: think of what you want a command to do, think of a name for it, and write a handler that uses the name after `on` and `end`, with the appropriate HyperTalk statements in between. Then, to make the command work, send the name to the object that has the handler in its script.

❖ *By the way:* It's probably best to avoid using the name of an existing HyperTalk command or function as the name of a command you create. See the *HyperCard Script Language Guide* for details on naming commands.

Confirming actions

Sometimes it's useful to be able to put a message on the screen and get a confirmation for an action someone has taken. For example, most Macintosh applications give you a chance to change your mind before erasing a disk by putting an alert box on the screen in which you confirm your choice. You can make your own alert boxes in HyperCard using a HyperTalk command. In this section you'll learn how to do it.

A disappearing act

Go to Card 2 of your practice stack if you are not there already; then follow the steps below.

1. **Create a new button on Card 2 and name it "Disappear."**

   Command-drag with the Button tool as before and double-click to see the Info box.

2. **Select the usual "Show name," "Auto hilite," and "round rect" options.**

3. **Click Script to see the script editor and type this line between the existing lines:**

   ```
   hide me
   ```

   `Me` always refers to the object that contains the handler—in this case, the button itself.

4. **Click OK.**

5. **Switch to the Browse tool and click the Disappear button.**

When you click this button with the Browse tool, it . . . disappears.

To get the button back again,

1. **Press Command-M to see the Message box.**

2. **Type this statement and press Return:**

   ```
   Show button "Disappear"
   ```

❖ *By the way:* Once you've typed a statement into the Message box, it stays there until you type something else—even if the box is invisible. All you need to do to send the message to HyperCard again is to press Return.

You can use the `hide` command to make a field, a button, a window (such as the Message box), the menu bar, the background picture, or the card picture invisible. (The **card picture** is any graphics on the card that aren't on the background). The `show` command does just the opposite.

Next, you'll create an alert box that will appear whenever you click the Disappear button.

A command to put up an alert box

Suppose you wanted anyone using your stack to think twice about making the button disappear. You can write a handler to make sure that happens:

1. **Open the script editor for the Disappear button.**

   Use Command-Option-click with the Browse tool, or use Shift–double-click with the Button tool.

2. **Place the insertion point after `mouseUp` in the first line and then press Return to start a new line.**

3. **Type the following lines:**

```
beep
answer "Do you really mean that?" with "Yes" or "No"
if it is "Yes" then
```

4. **Click in front of `end mouseUp` to reposition the insertion point.**

5. **Type this line and press Return:**

```
end if
```

Here's what the complete handler should look like:

```
on mouseUp
  beep
  answer "Do you really mean it?" with "Yes" or "No"
  if it is "Yes" then
    hide me
  end if
end mouseUp
```

If you have extra lines, you can delete them, although they won't make a difference in how the handler works. Press Tab to format the script, if necessary (the lines indent automatically).

6. **Click OK.**

7. **Click the Disappear button with the Browse tool.**

   When you click the Disappear button now, you should hear a beep and then see the alert box shown in Figure 3-2.

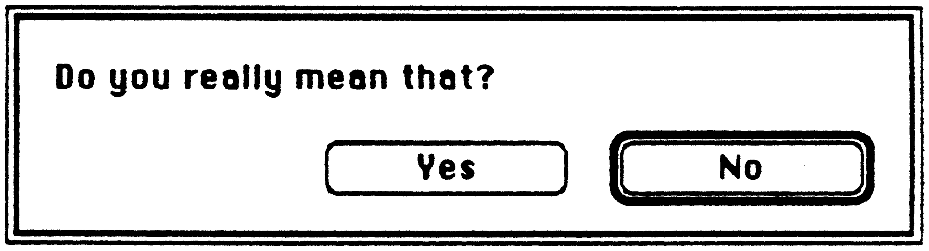

Do you really mean that?

Figure 3-2
The alert box you created with the Answer command

✣ *Something else happened?* If you get a "Can't understand . . . " message instead, go to the button's script and check the typing. Make sure no lines are left out and that the lines are in the correct order. Then try again.

8. **Click No so the button doesn't disappear.**

   If you click Yes by mistake, just press Return; your message to show the button should still be in the Message box.

In the alert box you have the choice of clicking a Yes button or a No button. These buttons are labeled with whatever you specify in quotation marks in the `answer` command statement. You can have up to three choices. Whichever choice you put last in order will be the button farthest to the right with the extra dark border; use this place for the the "best" or "safest" choice—the choice that can do no damage. This farthest-right button is also the one chosen by pressing Return or Enter.

The handler includes directions for what HyperCard should do if the Yes button is clicked:

```
if it is "Yes" then
  hide me
end if
```

The word `it` refers to whatever button—Yes or No—you click. (The word `it` has a specific identity in HyperTalk; you'll learn more about `it` in the next chapter.)

If you click the No button, nothing is specified, so nothing happens; the button won't disappear.

In English, the complete script says

"When this button is clicked, sound the system beep and let the user answer the question 'Do you really mean it?' by clicking either a button labeled 'Yes' or a button labeled 'No,' with 'No' being the emphasized choice. If the answer is 'Yes,' then make this button invisible. That's it, and that's all."

You can use a handler like this anytime you want yourself or someone else to have a second chance at something. For instance, it would be nice to have an opportunity to change your mind before deleting an important button or making some other potentially disruptive change.

An additional action

You can include an action for each button in the dialog box. Add one for the "No" choice:

1. **Go to the script editor for the Disappear button**

2. **Click to position the pointer in front of `end if`.**

3. **Type the following lines (press Return after each line):**

   ```
   else
   answer "Glad you reconsidered." with "No problem"
   ```

   The lines will automatically indent. When you press Return for the final time, `end mouseUp` should line up at the leftmost margin.

   When you have typed everything correctly,

4. **Click OK.**

5. **Try the Disappear button.**

   Now, when you click the Disappear button with the Browse tool you get the beep and the alert box just as before. Clicking Yes causes the button to disappear. Clicking No makes another alert box appear with a gratuitous comment and reply—just for fun.

Here's the completed handler in the Disappear button script:

```
on mouseUp
  beep
  answer "Do you really mean that?" with "Yes" or "No"
  if it is "Yes" then
    hide me
  else
    answer "Glad you reconsidered." with "No problem"
  end if
end mouseUp
```

If structures

`If`, `then`, and `else` are HyperTalk keywords that work together in specific arrangements called `if` structures. `If` structures are used to test things and to take different actions, depending on the results. You included an `if` structure in the handler for the Disappear button to specify the action HyperCard should take when someone clicks a button in the alert box resulting from the first `answer` command.

`If` structures come in a few varieties; three of them are shown here. In the examples that follow, the placeholder word *condition* stands for something that can be tested as either true or false. *Statement* is a HyperTalk command line, and *statementList* is a series of command lines.

```
if condition then statement [else statement]
```

This structure is a single line in a handler. You can use a single line as long as ***statement*** is a single HyperTalk command. (The statement you would use following `else` would be different from the one following `then`.)

```
if condition then
  statementList
[else
  statementList]
end if
```

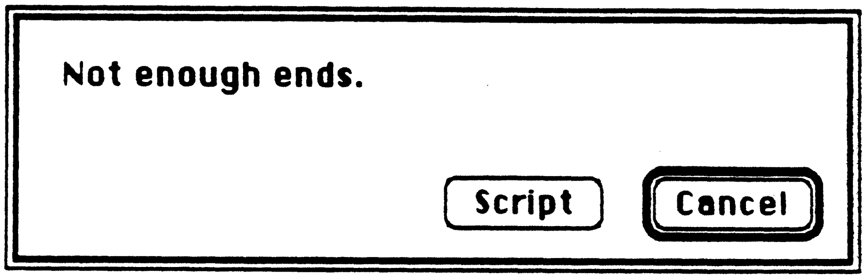
This version contains two lists of commands; one list following `then` and another, of alternatives, following `else`. Each statement must begin on a separate line. In this structure, you must include `end if` to signal the end of the statements.

```
if condition then
  statementList
[else
  if condition then
    statementList
  [else
    if condition then
      statementList
    end if]
  end if]
end if
```

This elaborate-looking structure contains several **nested** `if` structures, each of which requires an `end if`. Nested structures are useful when you have a number of different possible conditions and want to specify a number of different possible actions.

Nested means one inside another, inside another, inside another . . . for as many times as you want. The limit in HyperCard for nesting is 32 levels.

Anytime you use a several-line structure, you need to include `end if` to complete the structure, making it clear to HyperCard that you're finished. If you put too few (or too many) `end if` statements, HyperCard will put up a box like that in Figure 3-3 when you try to run the script.



Not enough ends.

Figure 3-3
An alert box telling you to add one or more End statements

This feature of HyperCard is especially useful when you write nested structures like the last one of the four syntax examples. The deeper your nesting, the harder it can be to keep track of how many `end` statements you need. HyperCard helps you out.

Syntax summaries

This section describes the syntax (most generalized form) of the `send` keyword and each command you used in this chapter.

You don't have to try to memorize these statements; refer to them as needed when writing your own handlers.

Answer

The basic structure for `answer` is this:

```
answer "question" [with "reply" [or "reply2" [or "reply3"]]]
```

Question can be any statement you like—usually a question invites the user to answer. *Reply, reply2,* and *reply3* are the labels for buttons representing the choices. The quotation marks are required.

You can have as many as three different replies; if you don't put a reply, HyperCard displays a single OK button in the box. The size limit for a reply is 13 characters, depending on the width of the characters.

The label of whatever button gets clicked is put into a special place named `it`. You can write other commands to use `it` or to evaluate `it`. In the handler you wrote for the Disappear button, the action of the `if` structure evaluated what `it` was. You'll learn more about `it` in Chapter 4, "Fields, It, and Other Containers."

Click

The `click` command has this general form:

click at *location* [with *key*], *key2*], *key3*]]]

The `click` command has the same effect as clicking manually with the mouse. *Location* is a description of a screen location; for example, `click at the location of card button 1`. *Location* could also be horizontal and vertical screen coordinates. (In Chapter 5 you'll learn more about screen coordinates.)

Key, key2, and *key3* are optional keys you can include with the click. You can use only these key names: `commandKey`, `optionKey`, and `shiftKey`. For example, a Shift-click would be written as click at *location* with shiftKey.

Hide

Here are the four structures of the `hide` command:

hide menuBar

hide *windowName*

hide *object*

hide *picture*

`MenuBar` is, obviously, the HyperTalk name for the menu bar. *WindowName* is the card window, one of the palettes (Tools or Patterns), or the Message box:

```
card window
tool window
pattern window
[the] message [box]
```

Object is the name or description of a button or field; for example, `background button 1`. *Picture* is either `card picture`, for all elements on the card level created with a Paint tool, or `background picture`, for graphic elements on the background level. You can also use the form `hide picture of` *description*, where *description* is the name or identifier of a card or background.

Send

The syntax of the `send` statement you used is

`send "`*messageName*`" [to` *object*`]`

The quotation marks around the name of the message aren't needed if the message is a single word, like `mouseUp`. *Object* is an identifier for an object, such as its number, ID, or name. The name must be in quotation marks.

`Send` directs a message to any object in the current stack or to another stack, but not to a specific object in another stack. The `send` keyword sends a message directly to the specified object, bypassing any other objects in the usual message-passing hierarchy.

Set

The general structure of the `set` command is

`set [the]` *property* `[of` *object*`] to` *value*

Property stands for a changeable characteristic of the HyperCard environment or of an object. For example, the user level is a property of HyperCard;the statement `set userLevel to 5` within a handler or typed into the message box would set the user level to Scripting (value 5). *Object* is an identifier for an object, such as its number, ID, or name.

What *value* is depends on the property; some properties, such as `hilite`, have the values `true` or `false`, while others take numerical values.

A complete description of properties is beyond the scope of this book. The Appendix contains a list of the properties.

Show

The `show` command also has four versions:

`show menuBar`

`show` *`windowName`* `[at` *`h,v`*`]`

`show` *`object`* `[at` *`h,v`*`]`

`show` *`picture`*

See the `hide` command, just previous, for a description of the placeholders. In the optional phrase `at` *h, v,* the *h* is a number specifying horizontal location on the screen, and the *v* specifies vertical location. The two numbers are separated by a comma. This optional phrase lets you place the window or object wherever you want. If you don't include it, the window or object appears wherever it was before it was hidden.

Later on, in Chapter 5, you'll learn more about the horizontal and vertical screen coordinates.

Wait

The `wait` command can have any of three forms, depending on what you want it to do:

`wait [for]` *`number`* `[seconds]`

`wait until` *`condition`*

`wait while` *`condition`*

Number is a whole number. If you want seconds, you must add `seconds` or the abbreviation `sec` or `secs`; otherwise, HyperCard uses **ticks,** which have a value of 1/60 second. No other measurements (such as minutes) can be used.

In the second and third forms, *condition* has to be some state that can have either the value `true` or `false`. In the second form the command waits until the condition has the value `true`. In the third form, the command waits while the condition has the value `true`.

What you've done in this chapter

You've done a lot. You learned about HyperCard's message-passing hierarchy and saw how placement of handlers can affect the range of their actions. You also used the `if` structure—a useful structure for taking action in a specific case or condition.

Here are the terms you've added to your vocabulary:

Commands

`answer`	This command puts an alert box on the screen containing a question and up to three response buttons.
`click`	The command that has the same effect as clicking with the mouse button.
`flash`	A command that causes the card image to flash. It's an **external command** (sometimes called an *XCMD* for short) included with HyperCard. External commands are written in a language other than HyperTalk.
`hide`	A command that hides buttons, fields, windows, and pictures.
`set`	A command that changes the value of properties.
`show`	A command that causes hidden buttons, fields, windows, and pictures to appear.
`wait`	A command that causes HyperCard to wait for something to happen or for a certain length of time.

Keywords

`else`	A word used when you want to specify a second alternative in an `if` structure.
`end`	You first encountered this keyword in Chapter 1; it signals the end of a handler.
`end if`	The last statement of an `if` structure.
`if`	The keyword that begins special structures called `if` structures..
`on`	You first encountered this keyword in Chapter 1. All handlers begin with `on`.

`send`	Sends messages to objects directly. It works in the Message box as well as in handlers.
`then`	A keyword used in `if` structures before the list of statements to be carried out.
Properties	
`hilite`	A button property; if its value is `true` the button is highlighted.
Miscellaneous	
`it`	The place where the `answer` command puts the label of the button chosen.
`secs`	An abbreviation for seconds.
`with`	A preposition; used in the `answer` command and some other commands.

Chapter 4

Fields, "It," and Other Containers

In everyday life, a container is something you can put things into. In HyperTalk, a **container** is a place in the computer's memory where you can put something of value, such as a text or numbers. You can then get whatever you have put into a container and use it elsewhere as needed.

In this chapter you'll learn about different kinds of containers, and you'll see how handlers can work with values in containers to do such things as calculations.

As in previous chapters, if you took a break, start up HyperCard and go to the practice stack before you go on.

Fields as containers

Fields are objects—they can receive and send messages and can have scripts. Fields are also containers. They usually contain text; specifically, regular (field) text rather than Paint text.

You already used a field as a container in Chapter 1, when you wrote the openCard handler to label the cards. Here's the handler (you can also see it in the script editor by opening the stack script):

```
on openCard
  put "This is Card" && number of this card into field 1
end openCard
```

This handler uses background field 1 to hold a string of characters made up of the text string "This is Card" and the card's number. Every time a card opens, this handler puts the same thing, but with a new card number, into field 1.

If you place a field in the background, it appears on every card sharing that background; but the text that field contains can be different on every card. An interesting feature of HyperCard is that even though a background field is the container for the text, the text itself remains with the card. This feature allows you to have card-specific text that appears in the same place and in the same style on each card, even though its content changes.

Important Deleting a background field deletes all the text for that field on all cards, even though the text "belongs" to the cards. Once the text is gone, you can't get it back.

HyperCard presents an alert box when you use the Cut or Clear commands on a background field so that you can reconsider.

Although fields most often contain text, they can also hold numerical values.

❖ *By the way:* Numerals can be interpreted either as numeric values or as text strings, depending on what a handler does with them.

A simple calculation

In this section you'll create some fields to hold numbers and then write a handler to use those numbers to calculate simple interest on a one-year loan. The handler then will put the results—amount of interest, total amount of loan, and monthly payment—into other fields. No expertise with mathematics is required on your part!

Set up the fields

You'll need five fields as containers for the numbers. You'll make card fields instead of background fields because you don't need the fields to be on every card of your practice stack.

Creating the first field

Follow these steps:

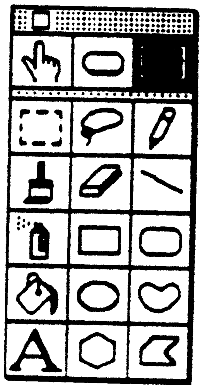
1. **Go to Card 3 of the practice stack.**

   You can use this card for your "field work."

2. **Choose the Field tool from the Tools menu.**

3. **Hold down the Command key and drag to create a new field.**

   Make the field about an inch wide and a quarter inch high.



The Field tool

4. **Move the field to the left of center on the card.**

   The location isn't too important now; later you can adjust it.

5. **Double-click the field to see its Info box.**

   Notice that this field is card field 1.

6. **Name this field `Amount` and select "shadow" as the field's style.**

7. **Click the Font button to see the Text Style dialog box.**

8. **Change the font size to 14.**

   This setting will make the numbers in the fields easier to read. The Line Height setting automatically changes to 18.

9. **Click OK.**

   The Field Info box closes; the first field should still be selected.

Copying and naming the other fields

Instead of creating new fields from scratch, you can just make copies. Be sure you make the copies in the order specified so you can keep track of which field is which.

1. **Position the pointer in the middle of the selected field, hold down the Option key, and drag to duplicate the field.**

   When you copy the field, HyperCard automatically identifies the copy as card field 2. When you release the mouse button, the second field is automatically selected.

2. **Position the copied field below the first one.**

   You can drag the field by its center the same way you would a button to get it in the right position.

3. **Double-click the field to see its Info box.**

4. **Name the field `Rate`**

   All other settings are the same as the first field, which is just what you want.

5. **Click OK.**

   The Info box closes.

6. **Repeat the Option-drag procedure on card field 2 to create a third field.**

7. **Position the new field 3 on the right side of the card, with some space in the middle between it and the first two fields.**

   You can adjust the spacing in a moment after you've made all the fields.

8. **Double-click the field to see its Info box and name it** `Interest`

9. **Click OK.**

10. **Repeat the Option-drag procedure with field 3 to make a new field 4 below it; name this field** `Total`

11. **Repeat the Option-drag with field 4 to make a new field 5 below field 4; name field 5** `Monthly`

    Your screen should now look roughly like that in Figure 4-1. If you want to adjust the position of your fields, go ahead. Leave some room above each field so you can type a label. Don't be too concerned with precise placement—the important thing is the scripting practice coming up after the next section.

Card field "Amount"

Card field "Rate"

Card field "Interest"

Card field "Total"

Card field "Monthly"

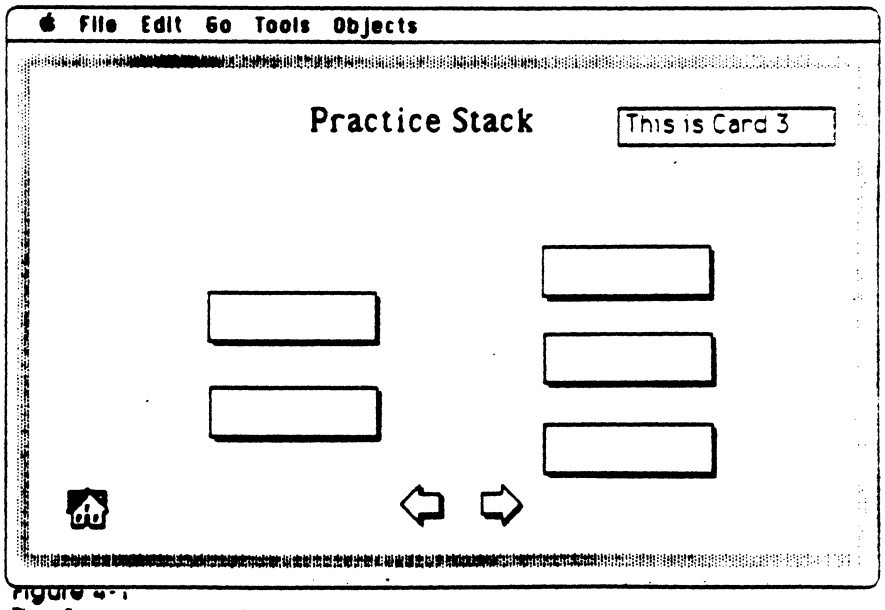
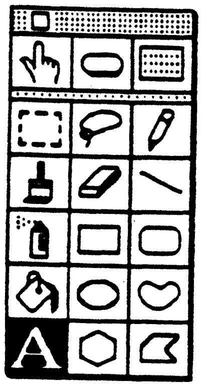


Figure 4-1
The five new card fields

Callouts will be lined up correctly in production.

Labeling the fields on the card

Next, put Paint text labels above each field to help you identify their contents. Follow these steps:


The Paint Text tool

1. **Choose the Paint Text tool from the Tools menu.**
2. **Choose Text Style from the Edit menu.**
3. **Select Geneva 12 and click OK.**
4. **Click just above the first new field to place the insertion point.**
5. **Type "Amount" (don't include quotation marks).**
6. **Click above card field 2, which should be the one below card field 1.**
7. **Type "Interest Rate" (don't include quotation marks).**
8. **Add labels above the other fields as follows:**

field 3	**Interest**
field 4	**Total Amount**
field 5	**Monthly Payments**

Don't worry about getting the text lined up exactly—just get the labels close enough for practice. The card should look something like the one in Figure 4-2.

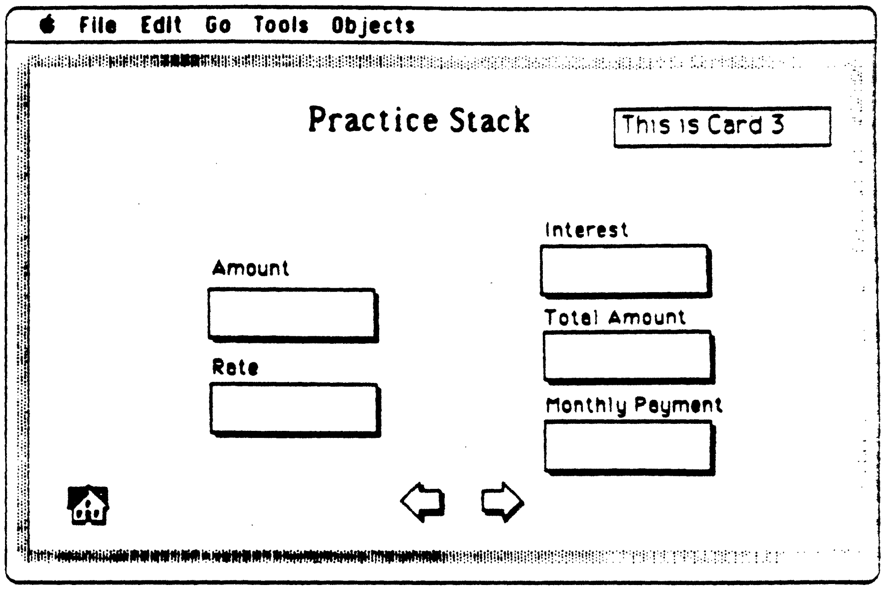


Figure 4-2
Paint text labels above the five fields

A calculating handler

Next, you'll create a Calculate button and put a handler into its script to make use of the fields for the interest calculation:

1. **Switch to the Button tool.**

2. **Command-drag to create a new button.**

   You can put the button anywhere you like on the card. In the middle with fields on each side is one possibility if you have room, but its placement won't affect its operation.

3. **Double-click the button with the Button tool.**

   The Button Info box appears.

4. **Name the button** `Calculate` **and choose "Show name," "Auto hilite," and "round rect."**

6. **Click the Script button and type these lines between the existing lines:**

```
set numberFormat to 0.00
get card field "Amount"
multiply it by card field "Rate"
divide it by 100
put it into card field "Interest"
add card field "Amount" to it
put it into card field "Total"
divide it by 12
put it into card field "Monthly"
```

7. **Check your typing carefully, and then click OK.**

The first thing this handler does is change the number format of HyperCard to "dollars and cents" (two places to the right of the decimal point). The number format is a HyperCard property aptly named `numberFormat`. The `set` command gives this property the value `0.00` (zeros, not letters), which specifies the standard dollars-and-cents format.

The `get` command fetches a value from a container—in this case, field 1, named "Amount"—and puts it into `it`. You first encountered the word `it` in the last chapter; the `answer` command also uses `it` to store the label of a clicked button. (The button's label is treated as a value.)

`It` is a sort of ever-present container. Once a value is in `it`, HyperCard can perform arithmetical operations. The result of an operation on `it` always goes back into `it`, replacing what was there before.

In English, the handler would say almost exactly the same thing as it does now in HyperTalk. All the subsequent statement do is perform operations on numbers from card fields using `it` and put the results into other card fields.

The handler as written is not the most elegant way to accomplish the calculation. It's used as an example here because it shows each step of the calculation separately.

Important In HyperTalk you must use "card" or "cd" in front of "field" to specify a card field. If you leave out "card," HyperCard assumes you mean a background field.

Conversely, you must use "background," "bkgnd," or "bg" in front of "button" to specify a background button, otherwise HyperCard assumes you mean a card button.

(The abbreviations "cd" and "bg" are not available in HyperCard versions earlier than 1.2.)

Test the handler

The best way to see how the handler works is to try it by typing some values into the "Amount" and "Interest Rate" fields and clicking the Calculate button.

Let's say you want to know what the interest, total amount, and monthly payments would be for a one-year loan of $8,000 at 16.5 percent annual simple interest.

1. **Choose the Browse tool.**
2. **Click inside field 1 to set the insertion point.**
3. **Type `8000` for the amount $8,000.**

   Don't type the comma or the dollar sign—they will cause an error.
4. **Click in field 2 and type `16.5` for the interest rate.**

   Don't include a percent symbol.
5. **Click the Calculate button.**

Almost instantly, you should see numbers appear in fields 3, 4, and 5. Those numbers should be

Interest	1320.00
Total Amount	9320.00
Monthly Payments	776.67

❖ *Something else happened?* If you got different values, no values at all, or an alert box, check the script. Make sure you haven't left out a line and that the handler is free of typing errors. Check that you have the fields labelled correctly (look at their Info boxes to verify their names). Also be sure you haven't put a comma or dollar sign into field 1 or a percent symbol into field 2.

Try some other values for amount and interest. (You'll have to select and type over the numbers already in fields 1 and 2.) Then click the Calculate button to see the new results.

The handler with comments

The following version of the handler shows *comments* that describe the action of the handler's statements. **Comments** are text lines typed into a script that are not part of the instructions. In HyperTalk, a comment must be preceded by two hyphens (--); the double hyphen indicates to HyperCard that the text following is a comment and should be ignored.

You do not have to type these comments into your own script; they are shown for example only.

```
on mouseUp
  set numberFormat to 0.00                -- Dollars and cents.
  get card field "Amount"                 -- The value in "Amount" gets put into It.
  multiply it by card field "Rate"        -- The result of the multiplication
                                          --    remains in It.
  divide it by 100                        -- Because field "Rate" is a percent.
  put it into card field "Interest"       -- The amount of interest.
                                          -- Note: The Put command puts only the VALUE
                                          --    of It into card field 3; It still
                                          --    contains the same value. The Put
                                          --    command doesn't empty It.
  add card field "Amount" to it           -- Interest plus original amount.
  put it into card field "Total"          -- And the total amount is still in It.
  divide it by 12                         -- To get monthly payments for one year.
  put it into card field "Monthly"        -- The final action.
end mouseUp                               -- As always.
```

Comments typed into the script editor would not look as neat as those shown here. In this book, the comments have been formatted for readability.

Although HyperCard ignores comments, other scripters generally appreciate them. Adding complete comments to your scripts is an excellent way to document what your scripts do. Comments not only help other scripters understand what you've done, but also help *you* remember, when you look at old scripts long after you've written them.

Your comments don't have to be as elaborate as those in the example. In fact, the more clean and elegant your handlers are, the fewer comments you're likely to need.

Other containers

Other HyperTalk containers are the Message box, the selection, and variables.

The Message box

You can see the Message box anytime by pressing Command-M. You use the Message box to give a one-line command to HyperCard and to search for text.

The Message box is a single-line container. The `put` command uses the Message box as its destination if you don't specify any other container. For example, typing `card field 5` in the Message box and pressing Return would cause the contents of card field 5 to appear in the Message box; the same thing would happen if `put card field 5` were in a handler.

You can type a HyperCard function into the Message box and press Return to see the value of that function. (See "A Few Words About Functions" near the end of this chapter for an example.)

You can also use the Message box as a calculator by typing numbers and arithmetic operators into it—say, `350 - 62`. The answer, `288`, appears in the Message box when you press Return.

The selection

Anytime you select regular text in a field by dragging across it, the part that appears highlighted is put into a container called the `selection`. The `selection` can be a destination for the `put` command.

Text located using the HyperTalk `find` command is *not* put into `selection`.

Variables

A **variable** is something that can have any value you choose to give it. The values of variables change; by contrast, the values of **constants** are always the same. For example, `pi` is a HyperTalk constant having the value 3.14159265358979323846. You can name variables anything you want. You create a variable simply by naming it and using it with the `put` command. For example, in a handler you might have

You can name variables anything you want. You create a variable simply by naming it and using it with the `put` command. For example, in a handler you might have

```
put 16 into Ham   -- "Ham" is the first variable name.
put 2 into Eggs   -- Likewise. The names are up to you.
put Ham+Eggs      -- Puts 18 into the Message box.
```

The name of a variable must start with a letter and can contain any combination of letters and numbers plus the underscore character (_), up to 29 characters maximum length. Operators or special characters can't be used.

You can use variables to streamline calculations by making them more like formulas. Here's the first part of the handler for the Calculate button, using two variables, `Amt` and `Rte`, to figure the interest rather than using the `get` command and `it`.

```
set numberFormat to 0.00
put card field "Amount" into Amt
put card field "Rate" into Rte
put (Amt*Rate)/100 into card field "Interest"
```

Instead of using the `multiply` and `divide` commands, this version uses the arithmetic symbols `*` and `/` to combine the variables on a single line. (See the section "Syntax Summaries" later in this chapter for more information on arithmetic commands.)

`It` is a variable that's always available. Some HyperTalk commands, such as `answer` and `get`, automatically put a value into `it`.

❖ *Local versus global:* The variables discussed here are **local variables;** that is, they and their values exist only within the handler in which they're created. HyperCard also has **global variables,** whose values are available to all handlers everywhere. Global variables aren't covered in this book. See the *HyperCard Script Language Guide*.

A few words about functions

HyperTalk contains both commands and functions. A **function** produces a value of some sort. You can use names of functions in commands to get values, without having to figure out how to write out the formula as part of your handler. A few examples of built-in HyperTalk functions are

`average` (*list*)	Finds the average of a list of values. The values must be separated by commas.
`compound` (*rate, periods*)	Finds the value of an account bearing compound interest.
`the date`	Gives the current date.
`the diskSpace`	Gives the number of bytes of free space.
`the mouseLoc`	Provides the location of the pointer on the screen.
`the sound`	Gives the name of the sound currently playing, or if no sound is playing, gives `"done"`.

You can type a function into the Message box and get a value when you press Return. Press Command-M to see the Message box and type these functions:

```
the date

the time

the diskSpace

average (17,24,56,52)
```
(The answer should be 37.25.)

You must include the word `the` with functions that require it. Typing `date` by itself into the Message box won't work.

A thorough discussion of HyperTalk functions is beyond this book's scope. The Appendix and the Quick Reference Card contain a list of all HyperTalk built-in functions.

Syntax summaries

This section contains syntax descriptions of the commands you used in this chapter. Use this section for reference as needed.

The arithmetic commands

The arithmetic commands are `add`, `subtract`, `multiply`, and `divide`.

`add` *expression* `to` *destination*

`subtract` *expression* `from` *destination*

`multiply` *destination* `by` *expression*

`divide` *destination* `by` *expression*

In all four commands, *expression* is something having a numerical value. *Destination* is a container.

HyperTalk also contains arithmetic symbols, or **operators,** that perform calculations: + (addition), - (subtraction), * (multiplication), and / (division). For example,

An **operator** is a character or group of characters that cause an operation, such as addition or subtraction, or an evaluation, such as comparison of two things. See Appendix A for a list of HyperTalk operators

```
put 3 into it
add 7 to it
```

does the same thing as

```
put 3 + 7 into it
```

In both cases the result is 10 in `it`.

Get

The syntax of the `get` command is

`get` *expression*

Expression is a description of something having a value; for example,

```
get field 1
get the name of background button 3
get the userLevel       -- Puts the value of the
                        --   user level into It.
get 72+13       -- puts 85 into It.
```

Get puts the value of *expression* into it. In fact, these two commands are identical:

```
get field 1
```

```
put field 1 into it
```

In fact, anything you might want to do with the get command can probably be accomplished just as well with put. For example, the lines

```
get the date
put it
```

do the same thing as

```
put the date
```

Put

The syntax of the put command is

put *expression* [*preposition destination*]

Expression is a description of something having a value; it can be a text string or a number. *Preposition* is either into, before, or after. *Destination* is a container, such as it, a field identifier, or some other container. For example,

```
put 256 into card field 3
```

Into causes anything already in the destination container to be replaced by the expression. Before places the expression at the beginning of what's in the container (if anything), and after puts the expression at the end.

If you don't specify a destination, the expression is put into the Message box.

What you've done in this chapter

In this chapter you practiced using fields as containers for numbers and wrote a handler to perform a calculation. You also learned about other HyperTalk containers, such as the Message box, the `selection`, and variables. You also saw how comments are added to scripts using the double-hyphen (`--`).

Additions to your word list:

Commands

`add`	
`divide`	
`get`	Fetches a value and puts it into the variable `it`.
`multiply`	
`subtract`	

Prepositions

`after`	
`before`	
`into`	

Properties

`numberFormat`	A property of the HyperCard environment. You change it with the `set` command.

Containers

`it`	An all-purpose variable container used as a destination by some commands.
`selection`	A container that automatically holds whatever text might be highlighted by dragging across it to select it.

Chapter 5

Animation

With HyperTalk, you can write commands to change the images on the screen rapidly, creating animation effects. Animation combined with visual effects and sound can turn a presentation, a demonstration, or a training stack into an exciting multimedia production. In this chapter you'll explore two of the ways to animate images.

The first kind of animation involves using HyperTalk commands to manipulate graphics on a single card; the second kind uses different images on a number of cards, which are then shown in rapid succession.

If you took a break after the last chapter, start HyperCard again and go to your scripting practice stack.

Animation on a single card

An amazing facet of HyperCard is that anything you can do in with a menu command you can also do with a HyperTalk command in a handler. You can achieve an animated effect by writing a handler to select a picture and cause it to move.

Make something to animate

The first step is to create a graphic image to animate. The one you'll make next is simple to do using the Paint tools. You might want to tear off the Tools menu and work with it as a palette so you can switch tools more easily.

Drawing a circle

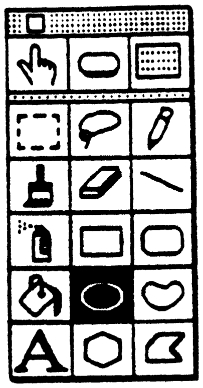
Circles are easy to make with the Paint tools. Follow these steps:

1. **If the Message box is visible on the screen, close it by pressing Command-M or clicking its close box.**

   If you continued on to this chapter from the last chapter, you might still see the Message box. If not, you can just go on to the next step.

2. **Go to Card 4 of the Scripting Stack.**

   Your first animation effect will take place entirely on this card.


The Oval tool

3. **Choose the Oval tool from the Tools menu (or palette).**

   You'll use this tool to draw a circle on the card. First, you'll need to set the line width for the circle and set the Draw Centered option.

4. **Choose Line Size from the Options menu.**

   A small box appears with line width choices.

5. **Click the second width from the left.**

   The Line Size box closes automatically when you make the selection.

6. **Choose Draw Centered from the Options menu.**

   This option causes the circle to be drawn from the starting point outward when you drag, which makes it easy to center the circle on the card.

7. **Position the crossbar pointer near the card's center, hold down the Shift key, and drag until the circle is about three inches in diameter.**

   Holding down the Shift key makes a perfect circle.

8. **Release the mouse button when the circle is the right size. (Release the Shift key also.)**

   If you're not satisfied with your first attempt, you can press Command-Z to undo it and try again.

Make sure that the circle doesn't overlap or crowd any other pictures or decorations that you may have drawn on the card with the Paint tools; erase any other graphics that come too close with the Eraser tool.

Drawing a smaller circle inside the first one

Next, you'll make a smaller circle inside the large one and then position it near the large circle's edge.

1. **Position the crossbar pointer inside the large circle, hold down the Shift key, and drag until the circle is anywhere from three-quarters to an inch in diameter.**

   Your small circle doesn't have to touch the edge of the large one yet—you'll adjust its position next. Don't click anywhere else when you're finished drawing it; go right on to the next step.

2. **Immediately press Command-S.**

   Pressing Command-S selects the last thing you drew; in this case, the small circle. (You can tell it's selected because it's "shimmering.") The crossbar pointer changes to the Lasso.

✧ *Didn't work?* If the smaller circle didn't get selected, just use the Lasso to encircle it—or switch to the Selection tool, drag across the circle, and then press Command-S to tighten the selection.

3. **Move the Lasso tool pointer to the edge of the selected circle until the tool changes to the arrow pointer.**

4. **With the arrow pointer on the edge of the small circle, hold down the mouse button and drag the circle until it touches the edge of the large circle.**

   See Figure 5-1 for an example of how the graphic should look. It's fine for your version to have the smaller circle in some other location.

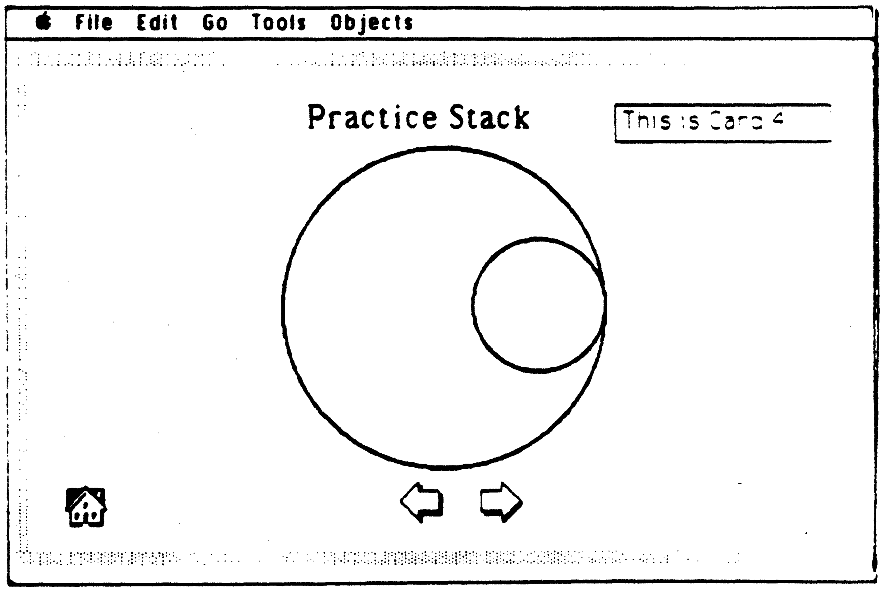
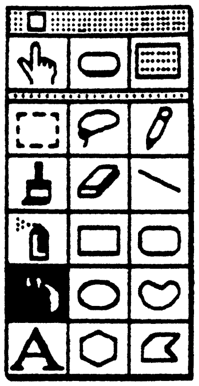


Figure 5-1
The smaller circle inside the large one

Filling in the smaller circle

You can fill the circle with a pattern or with plain black.


The Bucket

1. **Choose the Bucket from the Tools menu or palette.**
2. **Choose a pattern from the Patterns menu, unless you just want to use black.**

   Black is automatically selected.

3. **Click inside the smaller circle.**

   The small circle should fill with the pattern you chose, or with black if you didn't choose a pattern.

❖ *Everything changed?* If the paint "leaked" out and filled other areas besides the small circle, just press Command-Z to undo it. Then you'll have to inspect the small circle using the FatBits option for "holes" where the paint could leak through. Close any gaps using the Pencil, and then try using the Bucket again. See the *HyperCard User's Guide* for details on using the Paint tools.

The finished graphic should look approximately like that in Figure 5-2.

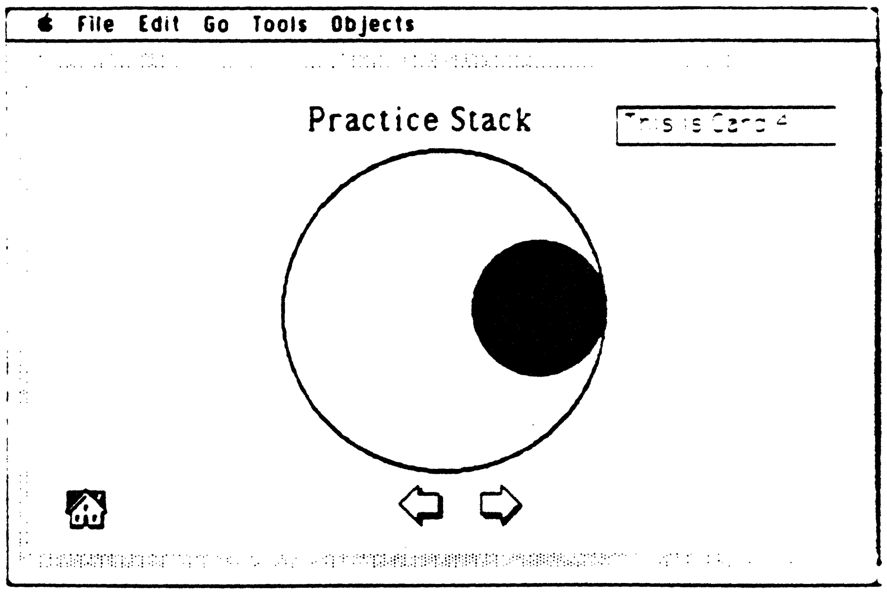


Figure 5-2
The finished graphic image

Write a handler to use tools and menu commands

To cause this graphic to rotate, you could select it and choose Rotate Left or Rotate Right from the Paint menu. However, that action would rotate the image 90 degrees only once. To make it spin, you'd have to continue to choose a Rotate command repeatedly. You can have HyperCard perform this action with a HyperTalk handler.

You can select the image from a handler by using the `choose` command to choose the Selection tool and then the `drag` command to drag across the image. You'll do that in a moment; first, you need to know the starting point and ending point for dragging.

Finding the starting point

If you were going to drag across the graphic to select it, you would position the pointer above and to the left of the image. That point would be your starting point.

1. **Press Command-M to see the Message box.**

2. **Choose the Button tool.**

   You'll make a button shortly; changing to the tool now allows you to use the arrow pointer, which is somewhat easier to position, for the next step.

3. **Position the pointer to the upper left of the graphic image, as you would if you were preparing to drag across it.**

   Be sure that the pointer is higher than the top of the image and farther to the left than the left edge of the image.

4. **Letting go of the mouse and leaving the pointer where it is, type these words into the Message box:**

   ```
   the mouseLoc
   ```

   You must include the word `the`.

5. **Press Return.**

You should see two number in the Message box. The numbers represent the horizontal and vertical position of the pointer on the screen as measured from the top-left corner of the card window. The distances are measured in **pixels**; the value of the top-left corner of the screen is `0,0`.

Pixel is short for *picture element*, which is the smallest dot that you can draw on the screen.

`The mouseLoc` is a HyperCard function that tells you the current position of the pointer. As you learned in the last chapter, you can type HyperCard functions into the Message box to get their values.

6. **Make a note of these two numbers; you'll need to put them into your animation handler.**

Finding the ending point

You use similar steps to find the ending point for dragging:

1. **Position the pointer to the lower right of the graphic image.**

   Make sure the pointer is lower and farther right than the image.

2. **With the pointer where it is, type into the Message box:**

   ```
   the mouseLoc
   ```

   Again, be sure to include `the`.

3. **Press Return.**

4. **Make a note of the new numbers in the Message box. These numbers will also go into your handler.**

Making a button and completing the handler

Next, create a button to hold the handler that will cause the animation:

1. **Create a new button and name it `Spin`**

   Choose the usual settings in the Info box.

2. **Click the Script button in the Info box to go to the script editor.**

3. **Type the following lines between the existing lines, substituting the numbers you made note of in the previous sections for the ones shown here:**

```
choose select tool
drag from 125,73 to 361,281 with commandKey
repeat for 16
doMenu "rotate right"
end repeat
choose browse tool
```

   Be sure to put in the numbers you got using `the mouseLoc` in the Message box: the first number goes first (the starting point for the drag), and the second number last (the ending point). Specifying `with commandKey` has the same effect as dragging with the Selection tool while holding down the Command key: the selection is tightened to the perimeter of the image.

   `DoMenu` lets you choose any command from an available HyperCard menu. The command name must be inside quotation marks.

4. **Press Tab to format the handler.**

   Here's how the handler should look:

```
on mouseUp
  choose select tool
  drag from 125,73 to 361,281 with commandKey
  repeat for 16
    doMenu "rotate right"
  end repeat
  choose browse tool
end mouseUp
```

   If you have any extra blank lines, you can select them and delete them. HyperCard just skips them, however, when the handler is read.

5. **Click OK.**

   The script editor closes.

Trying it out

Switch to the Browse tool and click the Spin button.

You should see the graphic image turn through four complete rotations; that's because the `repeat` statement specifies 16 repetitions of the 90-degree Rotate Right command. If you had not specified a number, the image would just keep turning "forever."

❖ *By the way:* You can press Command-period (.) to stop a handler from running—a useful feature with "runaway" handlers.

In the Clip Art stack that came with HyperCard you'll find a picture of an old car with a button labeled "Drive the car." When you click the button, the car rolls forward and back. The script for that button contains a handler that selects the image and drags it back and forth, finally putting it back where it started. It's another example of animation using the Paint tools in a handler. Feel free to look at the button's script on your own.

Important If you use Command-period to stop this kind of animation, you could end up making the animation unusable. If an image were left somewhere in mid-drag, the handler might not be able to select it again, or might select only part of it.

Repeat structures

`Repeat` is a keyword that tells HyperCard perform a command or series of commands over and over again without your having to type them out. Like the `if` structures, `repeat` structures must be inside handlers to work. They come in several varieties:

`repeat [forever]`

`repeat [for]` *number* `[times]`

`repeat until` *condition*

`repeat while` *condition*

`repeat with` *variable* `=` *startValue* `to` *finishValue*

When using any of these structures, you would follow the `repeat` line with a statement or list of statements making up the commands you wanted to have repeated. At the end, you must include `end repeat`.

The `repeat [for]` *number* `[times]` version lets you specify how many times HyperCard will go through the loop; you replace *number* with the number of repetitions you want. You used this version in the animation.

The prepositions `until` and `while` specify different ways of looking at a situation. The *condition* is something like a property or a value. For example, you could use `repeat until the mouseClick`, which means "keep going until someone clicks the mouse button," or you could have just as easily use `repeat while the mouseClick is false`, which means the same thing.

The `repeat with` form allows repetition to continue until the value of a variable changes from the starting value to the ending value that you specify. For example,

```
repeat with count = 1 to 100 -- "count" is the variable.
  doMenu "rotate right"
end repeat
```

HyperCard adds 1 to the value of `count` after each rotation. This structure in the handler would cause the image to rotate 90 degrees 100 times, or 25 complete rotations. (In this case, you could get the same effect with `repeat for 100 times`; however, there could be cases where the variable would not simply be counting the number of times through the loop.)

Animation using several cards

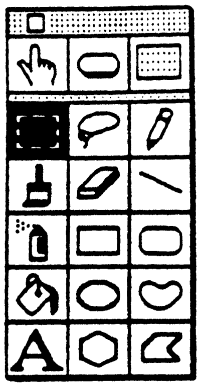
Instead of changing the image on a single card, this next technique involves putting different images on sequential cards and then showing the cards rapidly.

Set up the cards

You'll use the same image in this version of animation. To get set up, you'll copy, paste, and turn the image on each of four cards.

Copying and pasting the image once

Follow these steps:


The Selection tool

1. **Go to Card 4, if you aren't there already.**
2. **Choose the Selection tool from the Tools menu or palette.**
3. **Drag to select the symbol on Card 4.**
4. **Press Command-S to tighten the selection**

   This step makes sure that you copy only the image you want, and not anything outside it.
5. **Press Command-C to copy the image.**
6. **Go to Card 5.**

   You can press the Right Arrow key or you can switch to the Browse tool and click the right-arrow button.

❖ *By the way:* If pressing the Right Arrow key doesn't work, try Option–Right Arrow. (You might have Text Arrows selected on the User Preferences card.)

7. **When Card 5 is on the screen, press Command-V to paste the image.**

   Card 5 will be the first card of the four cards needed for this type of animation.

Adding more cards

You have only one more card in the stack at this point, and you need three more to use for animation:

Press Command-N two times.

You should see "This is Card 6" appear in the label field the first time you press the keys, and "This is Card 7" the second time. HyperCard inserts each new card immediately after the card you were on. The card that used to be Card 6 is now Card 8. Figure 5-3 illustrates the addition of new cards.

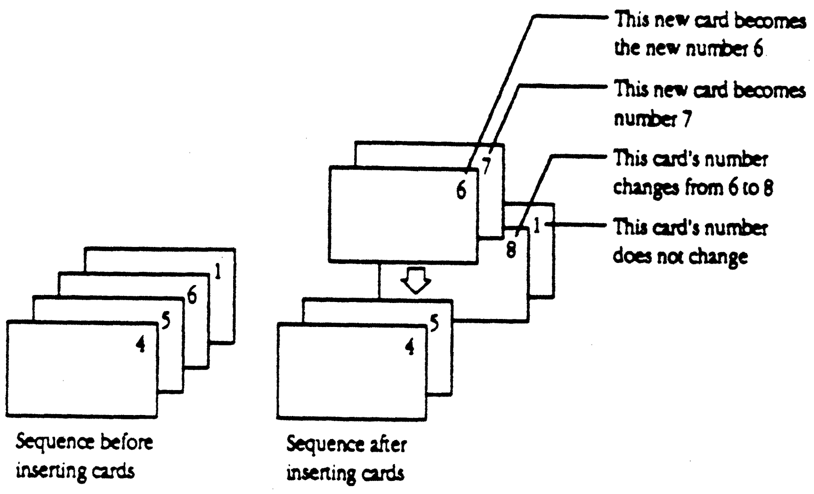

This new card becomes the new number 6
This new card becomes number 7
This card's number changes from 6 to 8
This card's number does not change
1
6
5
4
7
6
8
1
5
4
Sequence before inserting cards
Sequence after inserting cards

Figure 5-3
New cards inserted after the current card

Copying and pasting the image twice more

Put rotated copies of the graphic image onto the remaining cards following these steps:

1. **Go back to Card 5.**

   Press the Left Arrow key or click the left-arrow button.

2. **Select the image and tighten the selection with Command-S if it is not still selected.**

   If it's still shimmering, you can just go on to the next step.

3. **Press Command-C to copy the image.**

4. **Press the Right Arrow key to go to Card 6.**

   If pressing the Right Arrow key doesn't work, try Option–Right Arrow. (You probably have Text Arrows checked on the User Preferences card.)

5. **Press Command-V to paste the image on Card 6.**

6. **With the image still selected, choose Rotate Right from the Paint menu.**

   Each image needs to be rotated 90 degrees from the previous one to create the animation effect.

7. **Press Command-C to copy the rotated image.**

8. **Go to Card 7 and press Command-V to paste the image.**

9. **Choose Rotate Right from the Paint menu again.**

10. **Repeat steps 7, 8, and 9, but going to Card 8, to paste and rotate the image for the last time.**

When you're finished putting the images on the cards, go back to Card 5, where you'll create a button and write a handler to perform the animation.

Write a handler to show the cards

The handler for the animation will again go into a button script.

1. **Create a new button and name it** `Spin 2`

   Select the usual settings.

2. **Click Script to see the script editor.**

3. **Type these lines between the existing ones:**

```
repeat for 10
go to card 5
show 3 cards
end repeat
go to card 5
```

4. **Press Tab to check the formatting of the handler.**

   Here's how the complete handler should look:

```
on mouseUp
  repeat for 10
    go to card 5
    show 3 cards
  end repeat
  go to card 5
end mouseUp
```

5. Click OK.

6. Try the button.

Choose the Browse tool and click the Spin 2 button. The image spins ten times and then stops. You'll see the Spin 2 button flash by and the number in the card identification field flash by each time you get to Card 5 (you'll fix this shortly). When the handler is finished, you should be on Card 5.

Notice that the animation is faster going card-to-card than it was before on a single card. That's because in the first case HyperCard must redraw the image each time through the loop, which takes some time.

Here's a commented version of the handler explaining what each line does:

```
on mouseUp
  repeat for 10         -- The number of times to loop.
    go to card 5        -- Always start here.
    show 3 cards        -- Shows cards 6, 7, and 8.
  end repeat
  go to card 5          -- Back to the starting card.
end mouseUp
```

Another way to control the spin

You can add a "contingency plan" to your repeat structure to give you another way to stop the image from spinning.

1. Go to the script editor for the button.

Use Command-Option-click.

2. Click to place the insertion point in front of the first occurrence of `go to card 5`.

That's the line just after `repeat for 10`.

3. Type this line and press Return:

```
  if the mouse is down then exit repeat
```

4. Click OK.

5. Try the Spin 2 button again, and this time click the mouse button before the ten cycles have completed.

Now when you start the image spinning you can click anywhere to stop it before it spins ten times.

`Exit` is another keyword. Used with `repeat`, `exit` jumps to the end of the `repeat` structure, ending the loop when a certain condition is met (such as, in this case, the mouse being pressed).

`Exit` allows you to have two controlling conditions with `repeat`. You can use `repeat for`, `repeat while`, `repeat until`, or `repeat with` and specify one condition, and you can also have an `if` statement specifying a second condition and ending with `exit repeat`. For example,

```
repeat until i = 100   -- "i" is a variable
  if the mouse is down then exit repeat
  show all cards
  add 1 to i
end repeat
```

This structure in a handler would cause HyperCard to continue cycling through all cards in a stack until either the variable `i` has the value 100, or someone clicks the mouse button.

You could also use just plain `repeat`, which is the same as `repeat forever`, to start an animation running indefinitely, to be stopped only when some interested person clicked the mouse

Some finishing touches

Every time Card 5 appears during the animation, you see the Spin 2 button flash by and the numbers change in the label field, which detract from the effect. Use the `hide` and `show` commands to remove the button and field temporarily while the animation is running:

1. Go to the script editor for the Spin 2 button.

Use Command-Option-click, or Shift–double-click with the Button tool.

2. Click in front of `repeat` on the second line.

You want to add new statements before the `repeat` structure.

3. Type these statements, pressing Return after each line:

```
hide field 1
hide me
```

`Me` is the object containing the handler (the button itself) and `field 1` is background field 1 (the label field).

4. Click to place the insertion point in front of `end mouseUp`.

You want to add the next new statements after the `repeat` structure.

5. Type these statements, pressing Return after each line:

```
show field 1
show me
```

The entire handler looks like this:

```
on mouseUp
  hide field 1
  hide me
  repeat for 10
    go to card 5
    show 3 cards
  end repeat
  go to card 5
  show field 1
  show me
end mouseUp
```

6.Click OK.

7. Try the Spin 2 button.

Switch to the Browse tool and click the Spin 2 button. If there are no other graphics on cards 5 through 8, the effect should now be cleaner, showing only the turning symbol.

Syntax summaries

Refer to these descriptions of the syntax of commands you used in this chapter as you need to.

The syntax of the Choose command

The `choose` command's general structure is as follows:

```
choose toolName tool
```

ToolName is any one of the HyperCard tools from the Tools menu. You must always use `tool` after the name. Here are the HyperTalk names for the tools that you can use:

```
browse      field       reg[ular] poly[gon]
brush       lasso       round rect[angle]
bucket      line        select
button      oval        spray
curve       pencil      text
eraser      rect[angle]
```

The only tool you can't use is the Polygon tool.

You can use the `choose` command only with the user level set to Painting, Authoring, or Scripting. You can set and reset the `userLevel` property inside a handler with the `set` command, if you don't want to change the user level permanently in a stack.

The syntax of the DoMenu command

The `doMenu` command's structure is simple:

```
doMenu menuItem
```

MenuItem can be the name of an accessory in the Apple menu or the name of a menu command. *Menuitem* can also be the name of a container holding a command name.

❖ *By the way:* Include three typed periods if that's how a particular command is shown in the menu; for instance, `"card info..."`. You must *type* the three periods; don't use the ellipsis character (Option-semicolon).

The syntax of the Drag command

The `drag` command's syntax is

```
drag from start to finish [with key[, key2[, key3]]]
```

Start and *finish* are the points on the screen where the pointer starts to drag and where it ends up. The points are expressed as coordinates: two numbers representing horizontal and vertical placement in pixels, separated by commas. As mentioned earlier, the top left corner of the card window is `0,0`.

You can determine the coordiantes of the pointer's position using the function `the mouseLoc`.

Key, key2, and *key3* are one or more of the following HyperTalk key names, which must be separated by commas after `with`: `shiftKey`, `optionKey`, or `commandKey`. Including more than one key has the same effect as holding down more than one key while dragging.

The syntax of the Show Cards command

Here are the general forms of `show cards`:

```
show [all] cards
show number cards
```

Number is the number of cards you want to show if you don't want to show all of them. The cards are shown in sequence.

What you've done in this chapter

You learned two ways to cause animation effects: by using Paint tools and menu commands in a script and by using the show cards command with a sequence of cards. You experimented with the `repeat` structure, a way of performing a set of commands over and over again. You also learned how to use the functions `mouseLoc` in the Message box to find the screen coordinates of the pointer.

Commands

`choose`	This command chooses a tool just as though it had been chosen from the Tools menu.
`doMenu`	Performs a menu command just as though you had chosen it from the menu with the mouse.
`drag`	Does the same thing as dragging using the mouse.
`show cards`	A command. The cards to be shown (all or some number) must be in sequence.

Keywords

`commandKey`	The HyperTalk name for the Command key.
`end repeat`	The last statement of a `repeat` structure.
`exit repeat`	An alternative way out of a `repeat` structure.
`repeat`	A keyword; it begins the `repeat` structure.

Functions

`the mouse`	Has as a value either `up` or `down`, corresponding to the state of the mouse button.
`the mouseClick`	Either `true` (the mouse button has been clicked) or `false` (it hasn't).
`the mouseLoc`	Gives the location of the pointer on the screen in horizontal and vertical coordinates.

Miscellaneous

`from`	A preposition; used with the `drag` command and some other commands.

Chapter 6

Stacks You Can Build

This chapter describes two stacks you could build and script on your own, starting with materials already available in HyperCard.

This chapter is different from previous chapters in that you don't have to try to build these hypothetical stacks as you go, although you can if you like. The development of the stacks is discussed in a general way rather than detailed step by step.

If the example stacks in this chapter don't appeal to you, you are free to experiment. Browse through the Idea Stacks folder to look for possibilities. For example, each card in the Stack Ideas stack comes with prewritten handlers in its background script. You can create a new stack from each of these "seed" cards using the New Stack command and copying the background (the handlers are copied automatically). You could then add to and modify the scripts (and, for that matter, the appearance of the stack) to suit yourself.

❖ *By the way:* The *HyperCard Stack Design Guidelines,* available through Addison-Wesley publishing company, describes graphic, text, and instructional design principles as they apply to stacks.

A travel records stack

Suppose you wanted to computerize your records of vacations travels or business trips so that you could update them easily. You could create a stack in which to keep the important information.

For the first card of this stack you could use a map of your country. Transparent buttons placed over each state or province would allow you to click a particular state to go to a card specific for that state. From each of the state-specific cards, you could have other buttons to take you to cards for the cities or other localities you've visited. Finally, the city cards would have fields to contain information on accommodations and restaurants, clients visited (for a business stack), or points of interest (for a vacation travel stack).

Each time you visit a new state, you would add a new card for that state and a new button to the country map to go to that card. Each time you visit a new city or other location, you would add a new button on that city's state card, create a new city card, and fill in the information.

To build such a stack, you could use three backgrounds: one for the country map, one for the states or provinces, and one for the cities or locations. The different backgrounds would allow you to have different common fields for each of the three levels. The next sections describe how you might create this stack.

Creating the stack

You don't have to try to create this stack as you go. If you want to do so anyway for the practice, go ahead; be aware, however, that not all steps are written out.

The first thing you need is a map of some sort as a starting point. The Card Ideas stack contains some maps of the United States. (See Figure 6-1.)

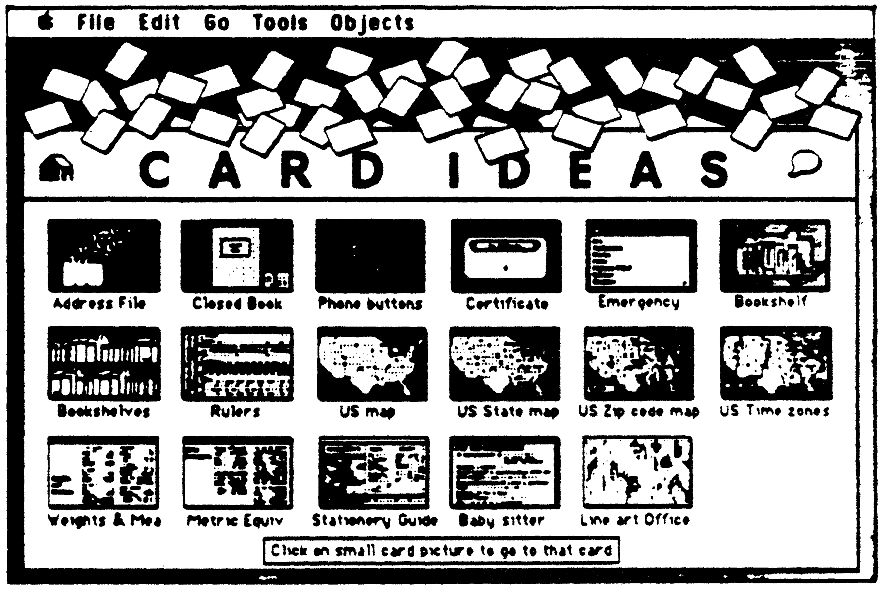


Figure 6-1
First card of the Card Ideas stack

You would click one of the small U.S. maps to go to a card with the same map on it; the one labeled "US State map," with state abbreviations, is a reasonable choice (Figure 6-2).

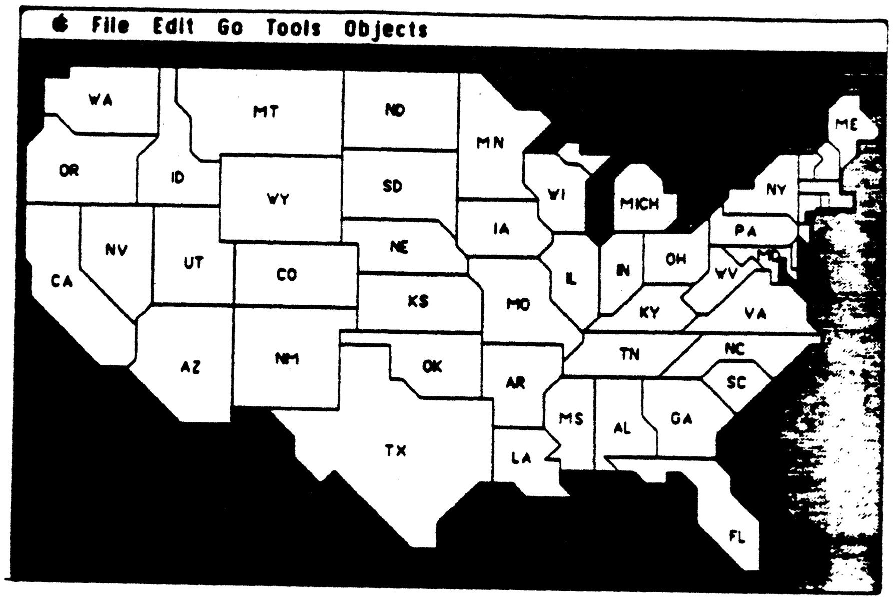


Figure 6-2
A U.S. map showing state abbreviations

Check to see what elements of this card are background elements by pressing command-B. Everything is in the background on this card, which means that if you copied the background you would get everything you see.

To create a stack with this card as your first card, you would choose New Stack from the File menu. In the resulting dialog box, you'd keep "Copy current background" checked and name the stack—something like "My Travel Stack."

Once you click New, you'll be in the new stack, even though the card on the screen won't have changed. You can then name this first background by choosing Bkgnd Info from the Objects menu and typing a name—for instance, "Map Background."

◆ *Stack building tip:* It's a good idea to name backgrounds because you might for some reason want to create different backgrounds that look very similar. You can confirm which one is which in that case by checking their names.

You can also name this first card in the Card Info box; for example, "Main Map." You can then use the card's name in any handlers you write.

As with your practice stack, you can add a Home button to the background of this first card. You can copy a Home button, complete with script, from the Button Ideas stack or from any stack with a working Home button. Make sure you are working in the background before you paste the button.

You could also add a title to the stack (either regular text in a field or as Paint text), such as "My Travel Stack." It's also a good idea to add a line or two of instructions somewhere, in case anyone besides yourself uses the stack; for example, "Click any state."

Because this stack uses buttons for each individual state or city to take you to the card you want, it's probably not necessary to add arrow buttons for traveling. Rather than moving card-by-card, you'll want to jump around.

The second background

The next information level would be the state level. To create a new, blank background for the state cards in this same stack, simply choose New Background from the Objects menu. HyperCard will create a new card with the new, blank background. This card is the second card in the stack.

You could give this background the name "States Background" using the Background Info box

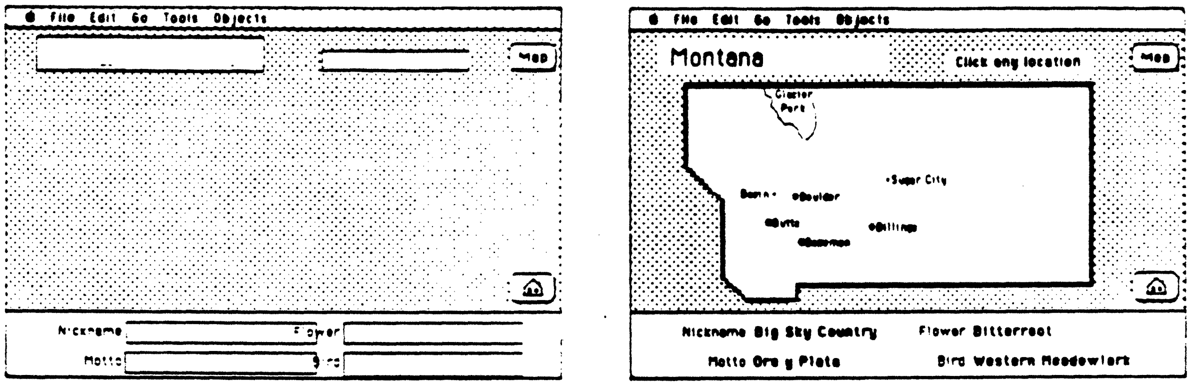
On each of the state cards, you want a map of the state, its name, and perhaps some fields for relevant information. You'd probably want to create a background field to hold the states' names, because putting the name in a field lets you search for the name with the Find command.

You can create other background fields for other information that you want to have on every card. Give each card the same name as the state it represents, both in the name field and in the Card Info box. Add some instructional Paint text to the background, such as "Click any location." (Paint text in the background will appear on every card.)

Finally, you can add a background button to take you to the Home card and one to take you back to the main map card. This latter button's script needs a `mouseUp` handler with the statement `go to card "main map"` (or whatever name the card has). You could also add `visual effect iris close` or some other visual effect before the `go` statement.

Each state card can have as its card picture (not in the background) an image of the state. One quick way to make a rough state outline is to copy the small state image from the main map using the Lasso or Selection tool. (You must be in the background to copy a background picture.) You then go to the appropriate state card, paste the image, and then stretch it to a larger size by using Command-drag or Shift-Command-drag. You can refine the rough outline further if you want. Another way to add a state outline would be to find a clip-art stack with pictures or maps in it and copy them with the Import Paint command.

Figure 6-3 shows a background layout for a hypothetical state card and a finished version.



Nickname Big Sky Country
Flower Bitterroot
Motto Oro y Plata
Bird Western Meadowlark

Figure 6-3
One possible layout for state cards; the background, showing fields, on the left, entire card on the right

The third background

The next information level would be the cities, towns, or localities level, for which you can create another background by choosing New Background again; you'll get another blank card. You can name this third background "Local Background."

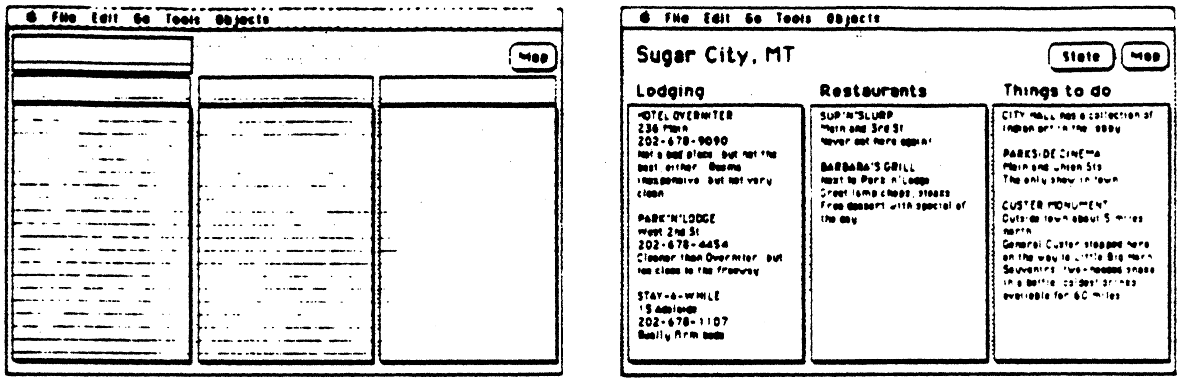
Again, make background fields for information such as the locality name, accommodations information, client addresses, restaurants, and so on—whatever information you want. When filling in the information, if you happen to have more information than will fit into a particular field, you could add a second card for the locality, including a card button or buttons to get to the second card and back again. You could also use scrolling fields , which allow you to add as much text as you want, regardless of the size of the field on the screen.

❖ *Stack building tip:* It's best not to use scrolling fields as a way of putting an enormous amount of text into one card. If you expect to have a great deal of text to fit into a small amount of space, consider using several cards to hold the text rather than putting it into a scrolling field.

Put the fields' permanent labels in Paint text in the background above each field—or you could create separate fields for the labels. Remember that you can search for text in fields with the Find command, but you can't search for Paint text.

Add a background button to go back to the main map; you can copy it (and its handler, automatically) from the state card. Also include a *card* button to take you back to the state card. (You wouldn't want to put this button into the background, because the destination for the button's handler would have to be different for different localities.) You could add visual effects to these buttons' handlers also.

Figure 6-4 shows a background layout for the locality cards and a finished version for a fictional city.



Sugar City, MT

Lodging

HOTEL OVERWINTER
236 Main
202-678-9090
Not a bad place, but not the best, either. Rooms inexpensive, but not very clean

PARK'N'LODGE
West 2nd St
202-678-4454
Cleaner than Overwinter, but too close to the freeway

STAY-A-WHILE
1 S Adelaide
202-678-1107
Really firm beds

Restaurants

SUP'N'SLURP
Main and 3rd St
Never eat here again!

BARBARA'S GRILL
Next to Park n'Lodge
Great lamb chops, steaks
Free dessert with special of the day

Things to do

CITY HALL has a collection of Indian art in the lobby

PARKSIDE CINEMA
Main and Union Sts
The only show in town

CUSTER MONUMENT
Outside town about 5 miles north
General Custer stopped here on the way to Little Big Horn
Souvenirs: two-headed snake in a bottle, coldest drinks available for 60 miles

Figure 6-4
A possible layout for locality cards: the background is on the left, the entire card on the right.

How the stack would work

Here's a summary of how you would use this stack:

1. For each state you visit, create a transparent card button over the image of the state on the main map. For some states, the button may be difficult to size correctly because of the state's shape. Try to center the button over the state abbreviation.

   On the eastern seaboard, where many small states are close together, you might want to have a transparent button over the whole area on the main map and create a second map card for just that area. On the second map, enlarge the area by selecting and stretching it. If you copy the same background as the main map, cover the background map before you paste the copied graphic.

Important Always think twice before you erase or change any graphics in the background. Any change you make in the background will happen on *every* card sharing the background.

You can cover background graphics on a card by using the Brush and painting with a pattern or by using the Command key with the Eraser to "erase" with white (opaque) on the card layer.

2. Create a new card using the second background for each state. Name the card the same as the state name. Then write a handler to link the transparent button on the main map to the respective card using the `go` command. The `zoom open` visual effect works well here.
3. Add cities or locations (such as national parks) to each state's map after you've visited them. Create a transparent button over each location.
4. Add a new card using the third background for each location. Name the card the same as the location. Link the button on the state card to the respective card.
5. Fill in the information for each place you visit in the appropriate fields.
6. When you plan to visit a locality again, or when you want information for some other purpose, use your stack to get to the information quickly. You can print each locality card if you need a "hard copy" for some reason.

You could also include a `mouseUp` handler at the background or card level of the main map that would use the `answer` command to respond with an alert box when someone clicks a state that doesn't yet have a button. Use a message such as "That state hasn't been visited yet." An OK button will appear automatically with the message—or you can add any button label you like.

A flash card stack

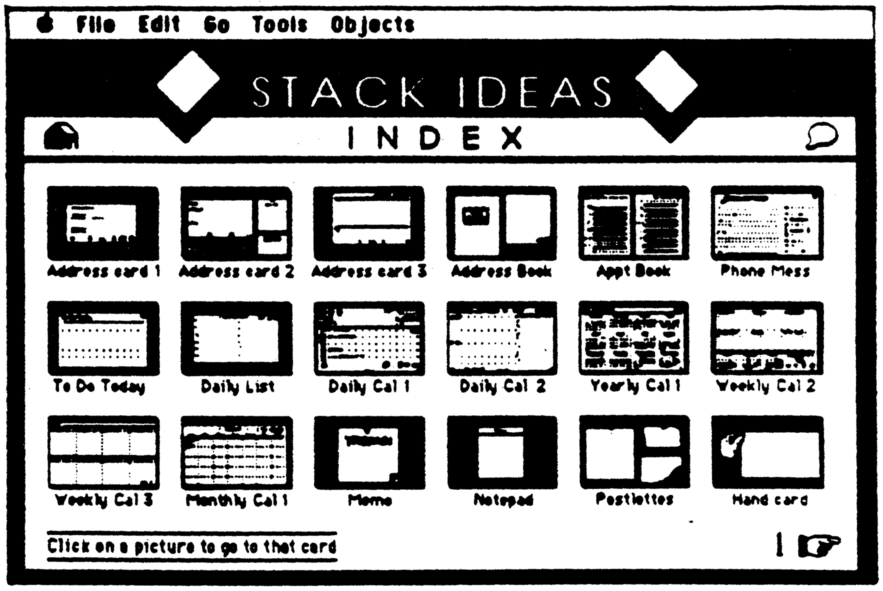
Flash cards are useful study aids for questions and answers, for vocabulary drill, or for any fact-memorization task. The idea is that a word or a question is put on one side of a card, and the answer on the other side. You try to match each answer as you go through the cards.

A stack that acted like a set of flash cards could have the answers contained in a hidden field, which would be shown when you type the correct answer into another field and click a button, or when you give up and ask for the answer by clicking another button.

A flash card stack would most likely need only a single background. You might want to have the option of moving sequentially through all cards, picking a card at random, or sorting cards into a new random order (like shuffling). You would probably want to be able to tell where you are in the stack by seeing the number of the current card as well as the total number of cards.

Creating the stack

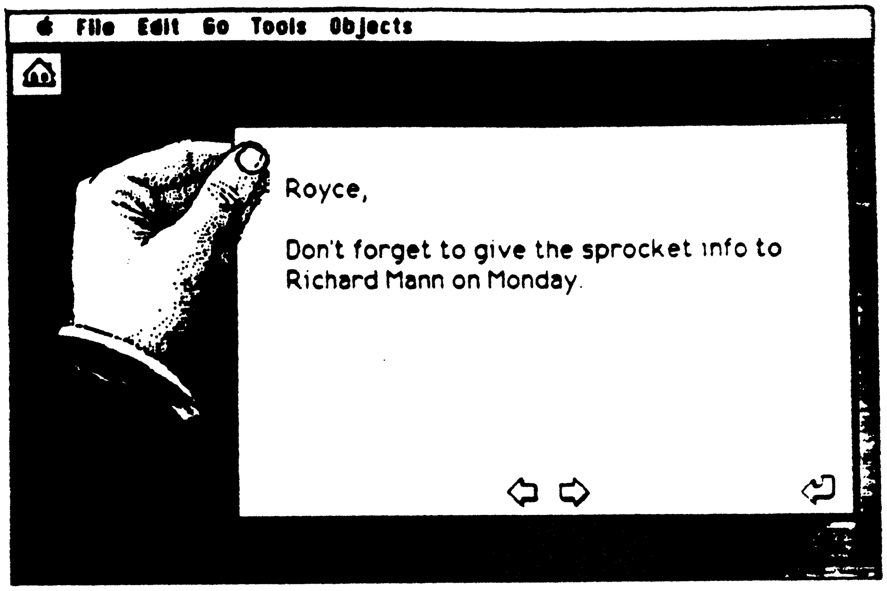
You can start by going to the Stack Ideas stack to look for some possible starting points for building your stack (Figure 6-5).



Click on a picture to go to that card

Figure 6-5
First card of the Stack Ideas stack

Clicking any of the images takes you to a full-size card. These cards contain prewritten handlers for navigation buttons as well as other ready-made scripts, often in their backgrounds. The card that looks like a hand holding a note card might be a good choice for the flash card stack (Figure 6-6).



Royce,

Don't forget to give the sprocket info to Richard Mann on Monday.

Figure 6-6
A card with a card on it

You can check the background by pressing Command-B; you might also want to look at the fields on the card; you'll see there's one large transparent field in the background. You can change the size of this field and duplicate it to make more fields if you want.

Again, to create a new stack from this card you would choose New Stack from the File menu. You'd copy the current background and name the stack "My Test Stack," or some such.

Features of the stack

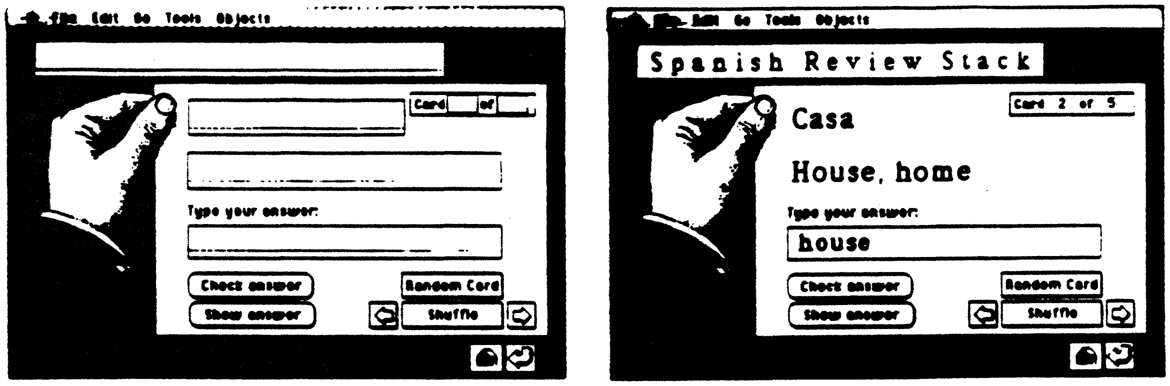
Here is a description of the features needed to make a flash card stack for Spanish-to-English vocabulary:

1. For each card you go to, you should be able to see the number of the card and the total number of cards in the stack. You could make a field for each number and write a handler to put the correct numbers into the fields automatically.
2. The Spanish word for which you are being tested needs to be in a field. A background field would be best.
3. The insertion point should be placed automatically into another, blank background field where you would type your answer.

4. The correct answer, in a third background field, should be hidden when you go to a new card.
5. A button should be available to check whether the English word you type is correct.
6. A button should be available for "giving up" and showing the hidden field.
7. Something should happen if the guess is wrong—a message and a chance to try again.
8. It would be nice to be able to choose random cards and to sort the cards in random order.

The idea of this flash card stack is that the words or information on the cards changes, but the shape and appearance of the cards don't. You can put essentially all the fields, buttons, and labels into the background. The test words and answers are typed into background fields; every card can have different text.

Figure 6-7 shows a completed version of this stack, a vocabulary stack for Spanish. The card on the left shows the background, including fields. The card on the right shows the entire image, complete with a word, an answer, and the correct answer showing.



Spanish Review Stack

Casa

House, home

house

Figure 6-7
One possible layout for a flash card stack; background on the left, entire image on the right

Notice that the preexisting arrow buttons and Home button have been moved and changed to rectangle buttons. You can move any button anywhere you want it in HyperCard and change its style as you like; the button will still act the same way.

The single field on the original card in Stack Ideas has been duplicated to create a total of five fields, all in the background: one holds the word in Spanish; the second is where you type your answer or guess; the third has the correct answer, which is hidden until you get the right answer or give up; and four and five are the small fields that tell you the card number and the total number of cards.

The scripts

This section shows the scripts for the objects the handlers that accomplish the flash card stack's actions. Notice that the handlers have been placed where they work most effectively.

Some objects have **empty** scripts—that is, their scripts contain no handlers—and so they are not listed here. The card level script is empty and so are the scripts for all fields.

The stack script

The following handlers are placed at the stack level so that they are availabe to every card in the stack automatically.

```
on openStack
  hide message box
end openStack
```

This handler came with the stack.

```
on openCard
  put the number of this card into field "cardNumber"
  put the number of last card into field "totalNumber"
  click at the loc of field "guess"
end openCard
```

This handler uses the `click` command to set the insertion point into the field, so you're ready to type your answer immediately when the card opens.

```
on closeCard
  hide field "answer"
  put empty into field "guess"
end closeCard
```

It's better to hide the Answer field and blank out the Guess field when leaving a card (closeCard) rather than when going to it (openCard) because otherwise you would see a "ghost" of the answer when the new card appeared, spoiling the test.

The background script

The background script contains this handler:

```
on openBackground
  push recent card
end openBackground
```

(This script came with the stack.)

The openBackground handler contains the statement push recent card. The "recent card" is whatever card you were on just before you came to this stack, if any. This statement keeps track of that card by "pushing" it onto a memory stack (different from a HyperCard stack). The return-arrow button in the lower-right corner of the card contains the reverse command, pop card. When you click the return-arrow button, the card that had been pushed is now "popped" out of the memory stack, and you go back to it. You use the push and pop card commands to have HyperCard keep track of where you've been so you can get back automatically.

This handler could just as easily have been at the stack level. In a stack with multiple backgrounds, it probably ought to be at the stack level.

The button scripts

The scripts for the Home button and the right-arrow and left-arrow buttons are what you would expect them to be from writing the handlers in the earlier chapters.

The return-arrow button script contains this handler, as mentioned in the preceding section:

```
on mouseUp
  pop card
end mouseUp
```

The "Check answer" button script contains this handler:

```
on mouseUp
  if field "guess" is in field "answer" then
    show field "answer"
  else
    beep
    answer "Oops.  Select the word and try again."
  end if
end mouseUp
```

The fields were given names in their Info boxes. The `if` statement uses the phrase `is in` to compare the string of characters typed into the Guess field with any string of characters in the Answer field, allowing you to include several meanings for any word. In the sample shown previously in Figure 6-7, *home* would also be a correct answer.

The "Show answer" button has this handler:

```
on mouseUp
  show field "answer"
end mouseUp
```

The "Random Card" button has this handler:

```
on mouseUp
  visual effect scroll down
  go to any card
end mouseUp
```

The "Shuffle" button script contains this handler:

```
on mouseUp
  sort numeric by random(the number of cards)
end mouseUp
```

The `sort` command puts all the cards in a stack in order, or in no particular order. It can sort in ascending or descending order, alphabetically (text) or numerically, and you can specify such things as card name, card number, a field name, a word in a field, and so on as the characteristic being sorted by. Here are some examples:

```
sort ascending text by last word of field "Name"

sort descending numeric by field 2
```

In the example stack, the `random()` function causes a sort into random order. The parentheses must contain a number—in this case, it's the number of cards in the stack. You could use the `random()` function to pick a number from 1 to 10 by putting 10 inside the parentheses—for instance, as part of a game.

How the stack would work

Once the stack is set up, its use is simple.

1. Add a new card for every new word.
2. Type the word into field 1 (which is named Word).
3. Use the Message box to show the hidden Answer field and then type the correct answer or answers into the field.
4. For review purposes, use the buttons to go through the stack and to shuffle its order. Type your answer for each word into the Guess field and click "Check answer." If you can't make a guess, click "Show answer."

❖ *By the way:* You can sort the cards into alphabetical order from the Message box with the statement

```
sort ascending text by field "word"
```

You could copy and modify the stack for other languages or subjects. You could even reverse the action of the stack—that is, make it English-to-Spanish rather than Spanish-to-English—simply by changing the handlers. Can you see how?

Where to go from here

Now that you're an experienced scripter, you can go on to other sources to learn more about HyperTalk and the possibilities of using HyperCard. Many people have written books on HyperCard and scripting that you might find helpful. The *HyperCard Script Language Guide* contains complete descriptions of HyperTalk elements. The HyperCard Help system is also a good reference to consult while you're working.

Some of your most valuable scripting information is likely to come from your own experimentation and experience. Think of ways you can change existing stacks to suit yourself—and then look at those stacks' scripts to see how they work and how you might modify them. Use your practice stack as a place to test handlers and as a repository for buttons with prewritten handlers and other scripts that you can copy and paste when you want them. Talk to other HyperCard users and scripters, and exchange tips and shortcuts. Most of all, enjoy having a new tool to enhance your creativity.

Appendix

HyperTalk Summary

This appendix contains syntax statements for all HyperTalk built-in commands and functions; lists of keywords, properties, constants, and system messages; a table of operators and their order of precedence; keyboard shortcuts for seeing scripts; and synonyms and abbreviations.

The Quick Reference Card contains the commands and functions, operators list, script editor commands, and keyboard shortcuts.

Syntax statement notation

Syntax statements show the most general form of a command or function, with all elements in the correct order. The syntax statements in this book use the following typographic conventions: Words or phrases in `this kind of type` are Hypertalk language elements that you type literally, exactly as shown. Square brackets ([ ]) enclose optional elements that may be included if you need them. (Don't type the square brackets.) In some cases, optional elements change what the command does; in other cases they are helper words that have no effect except to make the command more readable. Words in *italic* are placeholders describing general elements, not specific names; you must replace them in an actual command. For example, *effectName* stands for any of the HyperTalk visual effect names: `barn door`, `checkerboard`, `zoom`, and so on.

It doesn't matter whether you use uppercase or lowercase letters in HyperTalk, but names that are formed from two words are shown in small letters with a capital in the middle (`likeThis`) merely to make them more readable. The HyperTalk prepositions `of` and `in` are interchangeable—the syntax statements use the one that sounds more natural.

Commands

The following list includes all HyperTalk commands up to and including version 1.2.2. A full description of the action of these commands is beyond this appendix's scope. The HyperCard Help system contains a HyperTalk reference section explaining the use of the commands. The *HyperCard Script Language Guide* also contains complete descriptions of HyperTalk commands, functions, and so on. Many other books on HyperCard and scripting are also available.

```
add expression to destination
answer "question" [with "reply" [or "reply2" [or "reply3"]]]
arrowKey keyName
ask [password] question [with defaultAnswer]
beep number
choose toolName tool
click at location [with key[, key2[, key3]]]
close file fileName
close printing
convert container to format [and format]
delete chunk [of container]
dial expression [with modem [modemCommands]]
divide destination by expression
doMenu menuItem
drag from start to finish [with key[, key2[, key3]]]
edit script of object
enterKey
find [chars] expression [in field fieldDesignator]
find [word] expression [in field fieldDesignator]
find string expression [in field fieldDesignator]
find whole expression [in field fieldDesignator]
functionKey keyNumber
get expression
go [to] [stack] "stackName"
go [to] bkgndDescriptor [of [stack] "stackName"]
go [to] cardDescriptor [of bkgndDescriptor] [of [stack] "stackName"]
help
hide menuBar
hide windowName
hide object
hide picture
lock screen
multiply destination by expression
open [document with] application
open file fileName
open printing [with dialog]
```

```
play "voice" [tempo tempoValue] ["notes"]
play stop
pop card [preposition destination]
print card
print expression cards
print cardDescriptor
print document with application
push cardDescriptor
put expression [preposition destination]
read from file fileName until character
read from file fileName for numberOfCharacters
reset paint
returnKey
select object
select [preposition] expression of field
sekect [preposition] expression of msg
select [preposition] text of field
select empty
set [the] property [of object] to value
show [all] cards
show number cards
show menuBar
show windowName [at h,v]
show object [at h,v]
show picture
sort [direction] [style] by expression
subtract expression from destination
tabKey
type expression [with key[, key2[, key3]]]
unlock screen [with effectName]
visual [effect] effectName [speed] [to image]
wait [for] number [seconds]
wait until condition
wait while condition
write source to file fileName
```

Functions

The following list includes all HyperTalk functions up to and including version 1.2.2.

When using functions in HyperTalk statements you must either use the word `the` before the function name or add parentheses after it (both forms are shown in the list that follows). The parentheses are used to enclose values on which the function operates, called **parameters.** If the function takes several parameters (for example, the `average` function), the parameters must be separated by commas. See the *HyperCard Script Language Guide* for a more complete discussion of functions and parameters.

Factor is a single value, such as the number 5 or a container holding a value; *expression* can be a single factor or a combination of several factors and operators that results in a value, such as `(2+3)` or `(2+(field 1))`.

The result or use of a function is shown on the right side of the page.

`the abs of` *factor*	Absolute value
`abs(`*expression*`)`	
`annuity(`*rate, periods*`)`	Calculates an annuity
`the atan of` *factor*	Arc tangent—radians
`atan(`*expression*`)`	
`average(`*list*`)`	Calculates an average
`the charToNum of` *factor*	Returns the ASCII value of a character
`charToNum(`*expression*`)`	
`the clickH`	Gives horizontal coordinate of where the user last clicked
`the clickLoc`	Tells where the user last clicked as a pair of coordinates (h,v)
`clickLoc()`	
`the clickV`	Gives vertical coordinate of where the user last clicked
`the commandKey`	Condition of the Command key: `up` or `down`
`commandKey()`	
`compound(`*rate, periods*`)`	Calculates compound interest
`the cos of` *factor*	Cosine—radians
`cos(`*expression*`)`	
`the [`*modifier*`] date`	Current date set in the Macintosh: long or short
`the diskSpace`	Amount of free space on the current disk
`diskSpace()`	
`the exp of` *factor*	Mathematical exponential
`exp(`*expression*`)`	
`the exp1 of` *factor*	1 less than mathematical exponential: `exp()-1`
`exp1(`*expression*`)`	
`the exp2 of` *factor*	The value of 2 raised to the power of *factor*
`exp2(`*expression*`)`	
`the foundText`	Returns characters found by the `find` command

`the foundChunk`	Returns a description of where the text is found
`the foundLine`	Tells which line the found text is in
`the foundField`	Tells which field the found text is in
`the length of` *factor*	Number of characters in a text string
`length(`***expression***`)`	
`the ln of` *factor*	Natural logarithm—base-*e*
`ln(`***expression***`)`	
`the ln1 of` *factor*	1 plus the natural logarithm: `ln(1+`*factor*`)`
`ln1(`***expression***`)`	
`the log2 of` *factor*	Base-2 logarithm
`log2(`***expression***`)`	
`max(`***list***`)`	Returns the highest number value of a list
`min(`***list***`)`	Returns the lowest number value of a list
`the mouse`	Condition of the mouse button: `up` or `down`
`mouse()`	
`the mouseClick`	Returns `true` if the mouse button is clicked
`mouseClick()`	
`the mouseH`	Horizontal position of the pointer on the screen
`mouseH()`	
`the mouseLoc`	Horizontal and vertical coordinates of the pointer
`mouseLoc()`	
`the mouseV`	Vertical position of the pointer
`mouseV()`	
`[the] number of` *objects*	Number of buttons/fields on current card or bg
`[the] number of` ***chunks*** `in` *factor*	Number of characters, words, lines, and so on in text string
`[the] number of cards of` ***background***	Number of cards in specified background
`the numToChar of` *factor*	Returns the character corresponding to an ASCII value
`numToChar(`***expression***`)`	
`offset(`*string1, string2*`)`	Gives number of characters between the beginnings of two strings
`the optionKey`	Condition of the Option key: `up` or `down`
`optionKey()`	
`the param of` *factor*	Returns the value of a parameter in a list
`param(`***expression***`)`	
`the paramCount`	The total number of parameters
`paramCount()`	
`the params`	The entire list of parameters
`params()`	
`the random of` *factor*	Gives a random integer from 1 to the value of *factor*
`random(`***expression***`)`	
`the result`	Returns a text string if `find` or `go` is unsuccessful
`result()`	

the round of *factor*	Rounds to nearest integer: an odd integer plus 0.5 rounds up; an even integer, down
round(***expression***)	
the screenRect	The rectangle of the screen in which the menu bar is displayed: left, top, right, bottom coordinates
screenRect()	
the seconds	Number of seconds between midnight January 1, 1904, and the current time in your Macintosh
seconds()	
the selectedText	Returns the text currently selected
the selectedChunk	Describes the location of the selected text
the selectedLine	Tells which line the selected text is in
the selectedField	Tells which field the selected text is in
the shiftKey	Condition of the Shift key: up or down
shiftKey()	
the sin of *factor*	Sine—radians
sin(*expression*)	
the sound	Name of sound resource currently playing, or "done" if none is playing
sound()	
the sqrt of *factor*	Square root of a positive number—a negative number gives the result NAN(001) meaning "not a number"
sqrt(*expression*)	
the tan of *factor*	Tangent—radians
tan(*expression*)	
the target	Identifies the original recipient of a message
target()	
the ticks	Number of ticks (1/60 second) since the Macintosh was turned on or restarted
ticks()	
the [*modifier*] time	Gives time as a text string: long, short, abbreviated
time()	
the tool	Name of currently chosen tool
tool()	
the trunc of *factor*	The integer part of a number in *function*
trunc(***expression***)	
the value of *factor*	Gives the value of a string as an expression
value(***expression***)	
the [long] version [of HyperCard]	Returns the version number of HyperCard
version()	
the version of *stackDescriptor*	Tells version of HyperCard used to create, compact, change since compacted, and make latest changes, plus the date modified in seconds since January 1, 1904

Keywords

The following list of HyperTalk keywords includes their syntax, where appropriate, or a comment on their use. Keywords are predefined; you can't redefined them—for instance, you can't use a keyword as a name of a variable.

Only `send` can be used in the Message box.

`do`	`do` *expression*
`else`	`-- used with "if" structures`
`end`	`end` *functionName*
	`end` *messageName*
	`end if`
	`end repeat`
`exit`	`exit` *functionName*
	`exit` *messageName*
	`exit repeat`
	`exit to HyperCard`
`function`	`function` *functionName* [*parameterList*]
`global`	`global` *variableList*
`if`	`-- begins "if" structures`
`next`	`next repeat`
`on`	`on` *messageName*
`pass`	`pass` *functionName*
	`pass` *messageName*
`repeat`	`-- begins "repeat" structures`
`return`	`return` *expression*
`send`	`send "`*messageName* [*parameterList*]`"` [`to` *object*]
`then`	`-- used in "if" structures`

Properties

This section lists properties of the HyperCard environment and of objects up to and including HyperCard version 1.2.2.

Global properties

```
blindTyping        lockMessages       textArrows
cursor             lockRecent         userLevel
dragSpeed          lockScreen         userModify
edit Bkgnd         numberFormat
language           powerKeys
```

Window properties

loc[ation]	the height of	the top of
rect[angle]	the left of	the width of
the bottomRight of	the right of	
the bottom of	the topLeft of	

Painting properties

brush	multiple	textFont
centered	multiSpace	textHeight
filled	pattern	textSize
grid	polySides	textStyle
lineSize	textAlign	

Stack properties

cantDelete	freesize	script
cantModify	name	size

Background properties

cantDelete	name	script
ID	number	showPict

Card properties

cantDelete	name	script
ID	number	showPict

Field properties

autoTab	number	textHeight
the bottomRight of	rect[angle]	textSize
the bottom of	the right of	textStyle
the height of	script	the topLeft of
ID	scroll	the top of
the left of	showLines	visible
loc[ation]	style	wideMargins
lockText	textAlign	the width of
name	textFont	

Button properties

autoHilite	loc[ation]	textAlign
the bottomRight of	name	textFont
the bottom of	number	textHeight
the height of	rect[angle]	textStyle
hilite	the right of	the topLeft of
icon	textSize	the top of
script	showName	visible
ID	style	the width of

the left of

Constants

Constants are named values that never change. You can't use the name of a constant as a variable name.

down	The value of the key functions for Command, Option, and Shift keys and for the mouse button when pressed
empty	The string containing nothing (the **null** string)—same as ""
false	The opposite of true
formFeed	The form feed character, ASCII 12
lineFeed	The line feed character, ASCII 10
pi	The value of pi to 20 decimal places
quote	The double quotation mark character
return	The return character, ASCII 13
space	The space character, ASCII 32—same as " "
tab	The horizontal tab character, ASCII 9
true	The opposite of false
up	The value of the key functions for Command, Option, and Shift keys and for the mouse button when not currently pressed
zero..ten	The numbers 0 through 10

System messages

These messages are sent to the objects specified to inform them of system events. Some messages are accompanied by a variable (*var*), the nature of which depends on the message. For example, the `arrowKey` variable can be `left`, `right`, `up`, or `down`.

Messages sent to a button

newButton	mouseStillDown	mouseWithin
deleteButton	mouseUp	mouseLeave
mouseDown	mouseEnter	

Messages sent to a field

newField	mouseDown	mouseEnter
deleteField	mouseStillDown	mouseWithin
openField	mouseUp	mouseLeave
closeField	tabKey	

Messages sent to the current card

`newCard`	`enterKey`	`newStack`
`deleteCard`	`tabKey`	`deleteStack`
`openCard`	`arrowKey` ***var***	`openStack`
`closeCard`	`functionKey` ***var***	`closeStack`
`mouseDown`	`controlKey` ***var***	`help`
`mouseStillDown`	`doMenu` ***var***	`suspend`
`mouseUp`	`newBackground`	`resume`
`startUp`	`deleteBackground`	`quit`
`idle`	`openBackground`	`hide` ***var***
`returnKey`	`closeBackground`	`show` ***var***

Operators

The table below shows the order of precedence of HyperTalk operators. The order of precedence determines which operation HyperCard performs first when evaluating an expression. Operators are evaluated from left to right, except for exponentiation, which is from right to left. Parentheses force evaluation in a certain order; for example, `2*3+5` yields `11`, but `2*(3+5)` yields `16`.

Order	Operators	Type of operator
1	`( )`	Grouping
2	`-`	Minus sign for numbers
	`not`	Logical negation for Boolean values
3	`^`	Exponentiation for numbers
4	`*  /  div  mod`	Multiplication and division for numbers
5	`+ -`	Addition and subtraction for numbers
6	`&  &&`	Concatenation of text
7	`> < <= >= ≤ ≥`	Comparison for numbers or text
	`is in  contains`	Comparison for text
	`is not in`	Comparison for text
8	`=  is  is not  <>  ≠`	Comparison for numbers or text
9	`and`	Logical for Boolean values
10	`or`	Logical for Boolean values

Shortcuts for seeing scripts

These shortcuts were introduced with HyperCard version 1.2.

Key combination	Effect
Command-Option	Display buttons; click a button with keys down to edit its script
Shift-Command-Option	Display fields; click a field with keys down to edit its script
Command-Option-C	Edit script of current card
Command-Option-B	Edit script of current background
Command-Option-S	Edit script of current stack

Synonyms and abbreviations

These synonyms and abbreviations include those introduced with HyperCard version 1.2.

Term	Synonym or abbreviation
abbreviated	abbr
	abbrev
background	bg
	bkgnd
backgrounds	bgs
	bkgnds
button	btn
buttons	btns
card	cd
cards	cds
character	char
characters	chars
commandKey	cmdKey
field	fld
fields	flds
gray	grey
location	loc

message box	message msg box msg
middle	mid
picture	pict
polygon	poly
previous	prev
rectangle	rect
regular polygon	reg poly
round rectangle	round rect
second (time unit)	sec or secs or seconds
second (ordinal)	sec or secs or seconds
spray can	spray
ticks	tick

Glossary

algorithm: A step-by-step procedure for solving a problem or accomplishing a task. Writing HyperTalk handlers or programs in other languages often begins with figuring out a suitable algorithm for a task.

ASCII: Acronym for *American Standard Code for Information Interchange,* pronounced "ASK-ee." A standard that assigns a unique number to each text character and control character. ASCII code is used for representing text inside a computer and for transmitting information between computers and other devices.

background: A "holding area" where you can place elements that you want a group of cards to have in common. A background is an object and thus has a script; you can place handlers in its script that you want to be accessible to all cards in a group.

background picture: The part of the screen image that is common to all cards sharing a background; that is, the part that's not card-specific.

card picture: The part of the screen image that is specific to the card; that is, the part that's not on the background level.

container: A place where you can store a value, such as HyperCard fields, the Message box, and **variables.**

comments: Descriptive lines of text in a script or program that are not intended as instructions for the computer but rather are explanations for people to read. Comment lines are set off from instructions by symbols called **delimiters,** which vary from language to language. In HyperTalk, two hyphens (--) in front of a line marks it as a comment.

constant: An entity having a fixed, unchanging value. HyperTalk contains a number of constants, such as `true`, `false`, `up`, `down`, and `pi`. Compare **variable.**

delimiter: A character used to mark the beginning or end of something, that is, to define limits. For example, double quotation marks act as delimiters for **literals.** Comments in HyperTalk are set off with two hyphens at the beginning of the comment and a return character at the end.

empty: (adj.) Said of scripts that contain no handlers. Every HyperCard object has a script, even if the script is empty.

ex-command or **external command:** See **XCMD.**

global variable: A variable that is valid for all handlers in which it is declared. You declare a global variable by preceding its name by the keyword `global`. Compare with **local variable.** See also **variable.**

handler: A set of HyperTalk instructions specific to a **message.** A handler must begin with the keyword `on` and end with the keyword `end`. Both keywords must be followed by the name of the message.

keyword: A HyperTalk word having a predefined meaning that you cannot change. Some examples of keywords are `end`, `if`, `on`, `repeat`, and `send`.

literal: (n.) Something you want taken literally. In HyperTalk you use quotation marks (" ") to set off a string of characters as a literal, such as the name of an object or a group of words you want treated as a text string.

local variable: A variable that is valid only within the handler in which it is used. Compare with **global variable.** See also **variable.**

loop: A section of a handler that is repeated until a limit or condition is met, such as in a `repeat` structure. See **loop.**

message: A string of characters sent to an object. You can write handlers in the object's script containing instructions for HyperCard to carry out when the message is received. Messages can come from the system, from the Message box, or from other handlers. See also **handler, object hierarchy.**

message-passing hierarchy: See **object hierarchy.**

metasymbol: See **syntax.**

nested: (adj.) Said of similar structures occurring one inside the other; for example, an `if` structure may itself contain an `if` structure, and that one may contain another, and so on.

null: (adj.) Having no value at all, not even zero. The HyperTalk constant `empty` is defined as a string containing nothing, that is, a *null string*. A string containing `zero` would not be `empty`.

object: Any HyperCard element that has a script associated with it and that can receive and send messages. Objects are stacks, backgrounds, cards, fields, and buttons.

object hierarchy: The order in which a message is passed between objects. For example, a message that goes first to a button, such as `mouseUp`, would go next to the card, then the background, then the stack, and finally to HyperCard itself, unless intercepted and acted upon by a handler.

operator: A character or group of characters that cause a particular calculation or comparison to occur. Operators operate on **values.** For example, the plus sign (+) is an arithmetic operator that adds numerical values.

parameters: Values that accompany or are acted upon by a function. Parameters in HyperTalk are separated by commas.

pixel: Short for *picture element;* the smallest dot you can draw on the screen. The position of the pointer is often represented by a set of two numbers separated by commas. These numbers are horizontal and vertical distances of the pointer from the top and left edges of the card window, measured in pixels. The top-left corner of the screen has the coordinates `0,0`.

properties: Characteristics of objects or of HyperCard as a whole. For example, setting the user level to Scripting changes the `userLevel` property of HyperCard to the value 5. Properties are often selected as options in dialog boxes or on palettes, or they can be set from within handlers.

script: A collection of HyperTalk instructions associated with a HyperCard object. You use the object's script editor to add to and revise its script. Every object has a script, even though some scripts are *empty;* that is, they contain nothing. See also **message handler, object.**

script editor: A large dialog box containing a window in which you can type and edit a script. You get to the script editor by clicking the Script button in an object's Info box (the user level must be set to Scripting). The top line of the script editor box identifies the object to which the script belongs. You use keystroke commands to eidt text in the script editor. See also **message handler, object, script.**

syntax: A description of the way in which language elements fit together to form meaningful phrases. A syntax statement for a command shows the command in its most generalized form, including placeholders (sometimes called *metasymbols*) for elements you must fill in as well as optional elements.

tick: One-sixtieth (1/60) of a second. The `wait` command assumes a value in ticks unless you specify seconds by adding `secs` or `seconds`.

values: Information on which HyperCard operates. Values in HyperCard are essentially strings of characters—they are not formally separated into types. For example, a numeral could be interpreted as being a number or as being text, depending on what you do with it in a handler.

variable: An entity having a changing value. In HyperTalk, a container you can create to hold some value (either numbers or text) simply by using a name in a statement. Compare with **constant.**See also **container, global variable, local variable.**

XCMD: Short for *external command.* A command written in a computer language other than HyperTalk but made availabe to HyperCard to extend its built-in command set. Similarly, and *XFCN,* or *external function,* is a function written in another language.

THE APPLE PUBLISHING SYSTEM

This Apple® manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word. Proof pages were created on the Apple LaserWriter® printers; final pages were printed on a Varityper® VT600™. Line art was created using Adobe Illustrator™ and typeset on a Linotronic® 300. Stack illustrations were created with HyperCard® software. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated.

Text type and display type are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.

Reviewers: This section is being designed to fit on a detachable card in the back of the book -- similar to the one in the user's guide.
- JL

HyperTalk™ Quick Reference Card

The information in this Quick Reference Card pertains to HyperCard version 1.2.2.

Script editor command summary

Key combination	Effect
Command-A	Select entire script
Command-C	Copy selection to Clipboard
Command-F	Find text (same as Find button)
Command-G	Find next occurrence of same text
Command-H	Find current selection
Command-P	Print selection or (if no selection) entire script (same as Print button)
Command-period	Close script without saving changes (same as Cancel button)
Command-V	Paste Clipboard contents at insertion point
Command-X	Cut selection to Clipboard
Enter	Close script and save changes (same as OK button)
Option-Return	Wrap line without return character ("soft" return—symbolized by ¬ in scripts. Don't use a "soft" return inside quotation marks.)
Return	return character—indicates end of HyperTalk statement
Tab	Format script

Shortcuts for seeing scripts

Key combination	Effect
Command-Option	Display buttons; click a button with keys down to edit its script
Shift-Command-Option	Display fields; click a field with keys down to edit its script
Command-Option-C	Edit script of current card
Command-Option-B	Edit script of current background
Command-Option-S	Edit script of current stack

Commands

In the statements listed below, square brackets ([ ]) enclose optional elements. (Don't type the square brackets.) Words in *italic* are placeholders describing general elements, not specific names; you must replace them in an actual command. It doesn't matter whether you use uppercase or lowercase letters in HyperTalk; names formed from two words are shown with an embedded capital letter (`likeThis`) merely to make them more readable. The HyperTalk prepositions `of` and `in` are interchangeable.

add *expression* to *destination*
answer "*question*" [with "*reply*" [or "*reply2*" [or "*reply3*"]]]
arrowKey *keyName*
ask [password] *question* [with *defaultAnswer*]
beep *number*
choose *toolName* tool
click at *location* [with *key*[, *key2*[, *key3*]]]
close file *fileName*
close printing
convert *container* to *format* [and *format*]
delete *chunk* [of *container*]
dial *expression* [with modem [*modemCommands*]]
divide *destination* by *expression*
doMenu *menuItem*
drag from *start* to *finish* [with *key*[, *key2*[, *key3*]]]
edit script of *object*
enterKey
find [chars] *expression* [in field *fieldDesignator*]
find [word] *expression* [in field *fieldDesignator*]
find string *expression* [in field *fieldDesignator*]
find whole *expression* [in field *fieldDesignator*]

```
functionKey keyNumber
get expression
go [to] [stack] "stackName"
go [to] bkgndDescriptor [of [stack] "stackName"]
go [to] cardDescriptor [of bkgndDescriptor] [of [stack] "stackName"]
help
hide menuBar
hide windowName
hide object
hide picture
lock screen
multiply destination by expression
open [document with] application
open file fileName
open printing [with dialog]
play "voice" [tempo tempoValue] ["notes"]
play stop
pop card [preposition destination]
print card
print expression cards
print cardDescriptor
print document with application
push cardDescriptor
put expression [preposition destination]
read from file fileName until character
read from file fileName for numberOfCharacters
reset paint
returnKey
select object
select [preposition] expression of field
select [preposition] text of field
select empty
set [the] property [of object] to value
show [all] cards
show number cards
show menuBar
show windowName [at h,v]
show object [at h,v]
show picture
sort [direction] [style] by expression
subtract expression from destination
tabKey
type expression [with key[,key2[,key3]]]
unlock screen [with effectName]
visual [effect] effectName [speed] [to image]
```

```
wait [for] number [seconds]
wait until condition
wait while condition
write source to file fileName
```

Functions

In the statements listed below, square brackets ([ ]) enclose optional elements. (Don't type the square brackets.) Words in *italic* are placeholders describing general elements, not specific names; you must replace them in an actual command. It doesn't matter whether you use uppercase or lowercase letters in HyperTalk; names formed from two words are shown with an embedded capital letter (`likeThis`) merely to make them more readable. The HyperTalk prepositions `of` and `in` are interchangeable.

When using functions in HyperTalk statements you must either use the word `the` before the function name or add parentheses after it. Both forms are shown in the list that follows. *Factor* is a single value, such as the number 5 or a container holding a value; *expression* can be a single factor or a combination of several factors and operators that results in a value, such as `(2+3)` or `(2+(field 1)`. Parameters in a list must be separated by commas.

`the abs of` *factor*	Absolute value
`abs(`*expression*`)`	
`annuity(`*rate, periods*`)`	Calculates an annuity
`the atan of` *factor*	Arc tangent—radians
`atan(`*expression*`)`	
`average(`*list*`)`	Calculates an average
`the charToNum of` *factor*	Returns the ASCII value of a character
`charToNum(`*expression*`)`	
`the clickH`	Gives horizontal coordinate of where the user last clicked
`the clickLoc`	Tells where the user last clicked as a pair of coordinates (h,v)
`clickLoc()`	
`the clickV`	Gives vertical coordinate of where the user last clicked
`the commandKey`	Condition of the Command key: `up` or `down`
`commandKey()`	
`compound(`*rate, periods*`)`	Calculates compound interest
`the cos of` *factor*	Cosine—radians
`cos(`*expression*`)`	
`the [`*modifier*`] date`	Current date set in the Macintosh: long or short
`the diskSpace`	Amount of free space on the current disk
`diskSpace()`	
`the exp of` *factor*	Mathematical exponential
`exp(`*expression*`)`	
`the exp1 of` *factor*	1 less than mathematical exponential: `exp()-1`

exp1(*expression*)	
the exp2 of *factor*	The value of 2 raised to the power of *factor*
exp2(*expression*)	
the foundText	Returns characters found by the find command
the foundChunk	Returns a description of where the text is found
the foundLine	Tells which line the found text is in
the foundField	Tells which field the found text is in
the length of *factor*	Number of characters in a text string
length(*expression*)	
the ln of *factor*	Natural logarithm—base-*e*
ln(*expression*)	
the ln1 of *factor*	1 plus the natural logarithm: ln(1+*factor*)
ln1(*expression*)	
the log2 of *factor*	Base-2 logarithm
log2(*expression*)	
max(*list*)	Returns the highest number value of a list
min(*list*)	Returns the lowest number value of a list
the mouse	Condition of the mouse button: up or down
mouse()	
the mouseClick	Returns true if the mouse button is clicked
mouseClick()	
the mouseH	Horizontal position of the pointer on the screen
mouseH()	
the mouseLoc	Horizontal and vertical coordinates of the pointer
mouseLoc()	
the mouseV	Vertical position of the pointer
mouseV()	
[the] number of *objects*	Number of buttons/fields on current card or bg
[the] number of *chunks* in *factor*	Number of characters, words, lines, and so on in text string
[the] number of cards of *background*	Number of cards in specified background
the numToChar of *factor*	Returns the character corresponding to an ASCII value
numToChar(*expression*)	
offset(*string1*, *string2*)	Gves number of characters between the beginnings of two strings
the optionKey	Condition of the Option key: up or down
optionKey()	
the param of *factor*	Returns the value of a parameter in a list
param(*expression*)	
the paramCount	The total number of parameters
paramCount()	
the params	The entire list of parameters
params()	
the random of *factor*	Gives a random integer from 1 to the value of *factor*

`random(`*expression*`)`	
`the result`	Returns a text string if `find` or `go` is unsuccessful
`result()`	
`the round of` *factor*	Rounds to nearest integer: an odd integer plus 0.5 rounds up; an even integer, down
`round(`*expression*`)`	
`the screenRect`	The rectangle of the screen in which the menu bar is displayed: left, top, right, bottom coordinates
`screenRect()`	
`the seconds`	Number of seconds between midnight January 1, 1904, and the current time in your Macintosh
`seconds()`	
`the selectedText`	Returns the text currently selected
`the selectedChunk`	Describes the location of the selected text
`the selectedLine`	Tells which line the selected text is in
`the selectedField`	Tells which field the selected text is in
`the shiftKey`	Condition of the Shift key: `up` or `down`
`shiftKey()`	
`the sin of` *factor*	Sine—radians
`sin(`*expression*`)`	
`the sound`	Name of sound resource currently playing, or `"done"` if none is playing
`sound()`	
`the sqrt of` *factor*	Square root of a positive number—a negative number gives the result `NAN(001)` meaning "not a number"
`sqrt(`*expression*`)`	
`the tan of` *factor*	Tangent—radians
`tan(`*expression*`)`	
`the target`	Identifies the original recipient of a message
`target()`	
`the ticks`	Number of ticks (1/60 second) since the Macintosh was turned on or restarted
`ticks()`	
`the [`*modifier*`] time`	Gives time as a text string: long, short, abbreviated
`time()`	
`the tool`	Name of currently chosen tool
`tool()`	
`the trunc of` *factor*	The integer part of a number in ***function***
`trunc(`*expression*`)`	
`the value of` *factor*	Gives the value of a string as an expression
`value(`*expression*`)`	
`the [long] version [of HyperCard]`	Returns the version number of HyperCard
`version()`	

`the version of` *stackDescriptor*	Tells version of HyperCard used to create, compact, change since compacted, and make latest changes, plus the date modified in seconds since January 1, 1904

Operator precedence

Order	Operators	Type of operator
1	`( )`	Grouping
2	`-`	Minus sign for numbers
	`not`	Logical negation for Boolean values
3	`^`	Exponentiation for numbers
4	`* / div mod`	Multiplication and division for numbers
5	`+ -`	Addition and subtraction for numbers
6	`& &&`	Concatenation of text
7	`> < <= >= ≤ ≥`	Comparison for numbers or text
	`is in contains`	Comparison for text
	`is not in`	Comparison for text
8	`= is is not <> ≠`	Comparison for numbers or text
9	`and`	Logical for Boolean values
10	`or`	Logical for Boolean values