

# HyperActivity

## Part 1: Start Scripting

*by Chris Pelkie, Senior Technical Consultant, Cornell Computer Services Manager, Mac Ed Center for Faculty Courseware Development, 215 CCC, Cornell University, Ithaca, New York, 14853-2601. AppleLink A24.*

**HyperActivity is one of the regular HyperCard features in Nibble Mac Magazine. Nibble Mac is published eight times a year, and an 8-issue subscription costs \$19.95. from MicroSPARC Inc., 52 Domino Drive, Concord, MA 01742, (617) 371-1660. This article is © 1987 by MicroSPARC Inc., but may be reprinted in User Group newsletters and on BBS's as long as this paragraph is included.**

Trying to describe HyperCard to someone who hasn't seen it brings to mind the tale of the three blind men trying to describe an elephant — it depends on which end of it you grab hold of first! Apple Computer calls HyperCard “system software,” but Bill Atkinson, the developer of HyperCard, QuickDraw, and MacPaint, calls it a “software erector kit.” I think of it as an “information presentation system” because it allows you to create associations easily within a set of information, including text, graphics, sound, music, even videodiscs and other external devices.

While the name HyperCard might imply that a special piece of hardware is involved (like a printer card or a video card), in fact, this program is entirely software. The only hardware required is a Macintosh Plus, Macintosh SE, or Macintosh II, preferably with two disk drives. The basic unit of information in HyperCard is a card, which is one screenful of text, graphics, and buttons; a group of related cards make up a stack. A stack is a HyperCard document and, like any other document, is displayed as an icon by the Finder. Though not required, having a hard disk is very desirable, since HyperCard can take advantage of as much stackware as you can fit on a hard disk.

Related cards generally share a consistent appearance, like library catalog cards or pages in a spiral notebook. For example, a teacher may combine illustrations and maps with a set of lecture notes. She can establish cross-references to definitions of important terms or link close-up views to larger maps. Her students can browse through this information network in any order that makes sense to them, rather than in a strictly sequential order. Even more exciting, a student can add his own associations to this set of information, similar to highlighting important passages in a book, but in a much more dynamic and personal way. This ability to jump around in a collection of information is more like the way many of us browse through a library, a dictionary, or encyclopedia — we follow interesting trains of thought and investigate side paths before returning to the mainstream. But no one has been able to use a computer this way before. The prefix hyper is used in its traditional sense of over, above, beyond, which implies that you can travel through hyperspace from card to stack to graphic without following a straight line, that you can get from point A to point C without passing through point B. In fact, everyone who sees HyperCard tends to get a little hyper in the modern sense of overexcited as they start to envision its potential.

This brief introduction to HyperCard only scratches the surface of its many features. In this article, I want to introduce you to one of its most remarkable features — the scripting capability — by describing the built-in HyperTalk language. Working closely with Bill Atkinson, Apple's Dan Winkler has created a new way for “the rest of us” to teach our Mac how to perform some clever feats. For jargon lovers, HyperTalk is a high-level, object-oriented, interpreted computer programming language. I hope that's the last time I have to call it that! If you aren't a programmer, don't turn the page — writing a HyperTalk script may be the most exciting thing you have ever done on your Macintosh! Scripts are almost English-like, but HyperTalk gives you access to Mac features previously reserved only for those who have read the programmer's manual Inside Macintosh at least twice.

## **HYPERTALK SCRIPTS**

Let's start by creating simple scripts that perform flashy effects and ease into more sophisticated tasks later. Open your copy of HyperCard, and choose New Stack from the File menu. To start with a clean white card, uncheck the Copy current background button by clicking it. Name your stack “Script Stack” and save it to the disk.

A HyperTalk script is a set of instructions attached to a button, field, card, background, or stack. Think of each of these objects as actors in a play, each with his own script to perform. As in a play, each actor speaks only his lines and makes his moves on cue. So how do we cue a HyperCard object to perform? There are a number of cues that HyperCard understands, but the most common and the most familiar to any Mac user is the button click.

Think about the hundreds of OK buttons you've clicked in your Mac career. When does a button do what it is designed to do? The precise answer is: When you press the mouse button down while the cursor is on top of the button, then release the mouse button while the cursor is still on top of the button. Remember, if you press the mouse button down on a button but drag the mouse away before releasing it, that button will not be activated. So the essential cue a clicked-on button sends to HyperCard is “the mouse went up.” The first command in a script is frequently on mouseUp, followed by the set of instructions you want performed, and terminated by end mouseUp. (Peculiar capitalization is not required, but it's typical of people who have stared at Pascal code for too long. If you want people to think you are a knowledgeable Mac hacker, just capitalize compound words in the middle instead of using spaces.)

With that background, let's write a script that's attached to a button. First, create a new button by choosing New Button from the Objects menu. Double-click on the new button, and change its name to “My Button” by typing in the appropriate place in the dialog box. Click the Script button in the lower left area of the dialog box.

You are now in the script editor dialog box. Note that HyperCard has already filled in the on mouseUp and end mouseUp commands for you, and the blinking insertion point is conveniently placed right between them, waiting for your instructions.

Type go Home. Your script should look like this (you don't need to type the comments at this time):

```
on mouseUp --when the user releases the mouse button
go Home --go to the first card of the Home stack
end mouseUp --return control to user
```

Note that each command in a script appears on its own line. You can add new lines by pressing the Return key at the end of the previous line.

Next, click on the OK button, or press the Enter key to leave the script editor. Now choose the browse tool (the hand with the pointing index finger) from the Tools menu. When you return from the script editor, you are still in the button editing mode but to try out a script, you must select the browse tool, so you can click on the button.

Ready? Click your new button with the browse tool's index finger. One of three things will happen:

1. If you typed the script correctly, you are now looking at the Home Card.

Congratulations! Press the key at the top left of the keyboard (either the Escape key or the tilde (~) key) to return to Script Stack.

2. If you accidentally typed in something that HyperTalk couldn't understand, you'll see a dialog box complaining about the problem. If this happens, click the Script button to return to the script editor and check your spelling and typing. Generally, the insertion point will be blinking near the spot where HyperTalk got confused, so look there first for the problem.

3. You may have typed something that is neither correct as far as you are concerned nor an error as far as HyperTalk is concerned. The symptom for this problem is that nothing happens when you press My Button. For instance, if you had pressed the script editor's OK button before you added the command go Home, the script will be perfectly legal but do absolutely nothing. To check your script, choose the button tool (in the middle of the top row in the Tools menu), double-click My Button, and then click the Script button in the Properties dialog box. Another common mistake is to edit a script, and then click the Cancel button. Of course, none of your edits will be recorded if you do this, but since you are sure you just made those changes, you can't understand why they don't work! I've found that this is particularly maddening at two o'clock in the morning.

As we get to more advanced scripts, refer back to the above set of cases if you encounter problems. Those three basic situations are the most common: your script works as planned, your script has errors and can't be understood, or your script either doesn't do anything or does something you don't want it to do. In all but the first case, you must chase the bugs out. We'll discuss debugging tips as we need them.

Right now, let's add the promised flash to this script, so you can amuse your friends and amaze your neighbors. First, return to the script editor. Here's a trick to speed things up: choose the button tool from the Tools menu, then hold down the Shift key while double-clicking the button tool on My Button to go straight into the button script. Click after the words on mouseUp, and press the Return key once. Type visual effect dissolve to gray.

Your script should now look like this:

```
on mouseUp
visual effect dissolve to gray
go Home
end mouseUp
```

Press the Enter key or click the OK button to leave the script editor.

Now here's another keyboard shortcut: press the Command key and the Tab key together to reselect the browse tool (or just select browse tool from the Tools menu). Try out your new button script by clicking on My Button. See the difference? When you see the Home Card, press Escape or ~ to return to your stack.

There are a number of fancy visual effects like dissolve that you can use whenever you write scripts that move from one card to another. Some of the others include iris open, barn door close, checkerboard, and wipe up. The complete set is documented in the Help Stack and can be found by choosing Help from the Go menu, clicking the tab labeled HyperTalk on the bottom of the Help card, clicking on the word Commands, then clicking on the command visual effect. If you want to take a look at the Help stack right now, go ahead. When you are finished, simply click the Exit tab in the bottom right of the Help card to return to your Script Stack.

To try another visual effect, go back into the script of My Button and change the effect type from dissolve to one of the others. The added effect to gray can be removed if you like. The correct syntax of the visual effect command is listed in the HyperTalk help as follows:

```
visual [effect] <type of effect> [speed]
```

That means that the word visual is required, although "effect" is optional. An effect type is required, but modifiers like to gray or slow are optional. This way of describing a command's syntax is used throughout the HyperTalk command reference stack. Remember, if a command word is not surrounded by square brackets, it's a required part of the HyperTalk command, but words in square brackets are optional. Words in angle brackets are place holders for specific values that must be substituted in place of the words (don't type the angle brackets in the actual script command). Allowable substitutes are usually listed in the Help stack. In many cases, the replacement value can even be a computed value.

## LEAVING A TRAIL

Return to the stack ScriptStack if you aren't there already (just press ~ until you see the card with New Button on it.) Choose New Card from the Edit menu, select the paint brush tool from the Tools menu, and then make a doodle on the new blank card. The doodle will serve as a visual cue when you return to this card later. I'll refer to this card as the doodle card and the card containing My Button as the button card. By the way, when you choose the New Card command, the new card is added right after the card you are viewing.

Let's add some more commands to our button script. Choose Prev (previous) from the Go menu to return to the button card. Select the button tool, and then click once on My Button. You should see the "marching ants," indicating that the button is selected. Put the mouse in the center of the button, and drag the button to a new location. You can also resize a selected button by placing the mouse arrow tip in one of the four corners of the button and dragging. Try resizing your button. If you end up relocating the button, try again but be sure the arrow tip is very close to a button corner before you press down the mouse button.

Another important capability of HyperCard is that you can copy a button and paste a copy onto any other card in any stack along with all the button's properties, including its script. For example, with your button still selected, choose Copy Button from the Edit menu, then choose Paste Button from the Edit menu. Since the copy is superimposed on the original, drag the copy away to make sure that there are really two buttons.

Double-click the copy and change its name by typing in the Button Properties dialog box. Let's name it "Flash Button." Type the new name then click the Script button in the dialog box to modify this button's script. When you enter the script editor, you'll see that Flash Button and My Button have identical scripts. Edit the third line of the script to read go Next rather than go Home (shortcut: double-click on the word Home and then type Next). Click OK to leave the script editor, choose the browse tool, and test the action of Flash Button. You should be looking at the card with your paint doodle on it, since it's the next card in Script Stack. If not, follow the debugging steps presented earlier to check Flash Button's script for

mistakes. Press Escape or ~ to return to the button card and open the script editor dialog box for Flash Button. Add these lines just before end mouseUp:

```
push recent card
```

wait 2 seconds  
pop card

Exit the script editor and try the Flash Button now. The doodle card will appear for two seconds, and then return automatically to the button card.

The function of the command wait 2 seconds is obvious, but push recent card and pop card need some explanation. Think of a card game which uses a deck of playing cards. When one player discards a card to the top of the deck, that card becomes the first one that the next player can draw. In HyperTalk, adding a card to this imaginary deck of cards is called “pushing” a card. When you remove the top card, the action is called “popping” a card. You may push any number of cards before you begin popping them off in the reverse order. Pushing a card has no visible effect, but when HyperCard pops a card, it automatically displays that card after removing it from this imaginary deck.

These commands are useful when you write scripts that allow a HyperCard user to browse through different cards, and then return along the same path. Although it's impossible to know ahead of time exactly which card the user will be looking at when he jumps to another card, you still want to provide a trail so he can find his way back. In Flash Button's script, we go to another card and push the recent card, which instructs HyperCard to add the last card we were looking at (the button card) to our imaginary card deck. After we look at the doodle card for two seconds, HyperCard pops the top card off the deck and takes us back to that card (the button card).

Push and pop don't have to occur within the same script. You can push a card on the stack and go to another card. That card may have a button whose script contains a pop card command. In fact, it's generally a good, user-friendly practice to provide a Return button in your stacks (the arrowhead that hooks down and to the left), and push and pop is one technique you can use. This Return button is the browsing equivalent of an Undo command, which lets the users of your stack back up and explore another path whenever they like.

## **GETTING USER FEEDBACK**

There will be many occasions when you want to allow the user to spend more time looking at a card. It isn't very nice to show them something interesting and then whisk them away before they're done admiring your artwork. Instead of using the wait command with a fixed delay, you can write a command that will pause until the user gives the go ahead cue.

To do this, return to the button card, and enter the Flash Button script editor. Change the script so that wait 2 seconds is replaced by repeat until the mouse is “down”. Press Return to start a new line, and add the command put the long time and press Return.

You'll notice that the final command end mouseUp does not line up at the left margin of the window. That indicates that you haven't entered a valid script. Whenever you start a “repeat” command, you must also add the line end repeat after the instructions you want repeated. If you think of end repeat as a close parenthesis that matches the open parenthesis of repeat, you probably won't forget to add it in your own scripts. Your script should now look like Example 1.

## EXAMPLE 1

```
on mouseUp
  visual effect dissolve to gray --you may have entered your own effect
  go Next      --show the next card in this stack
  push recent card  --remember the card we just left
  repeat until the mouse is down --keep doing until mouse button pressed
    put the long time --put the current time (h:m:s) into message window
  end repeat    --don't forget this command!
  pop card      --go to the first card on top of our return deck
end mouseUp
```

See how the commands are indented? On mouseUp and end mouseUp should line up, as do repeat and end repeat. The command put the long time is indented within the repeat subsection which indicates that the command will be repeated until the user presses the mouse button to end the repeat loop. When the loop ends, the rest of the script will be performed.

Although the explanatory comments in the script are optional, you should get in the habit of adding comments to your scripts to explain commands that may not be obvious. A comment is preceded by two hyphens (--). HyperCard will ignore comments, so they don't cause syntax errors. Reading a script a month from now will be much easier if you've commented it. Also, if you plan to share scripts with others, they will understand your scripts much more quickly if you comment them.

Test your modified button script and make sure that everything works.

HyperTalk understands a number of variations of the repeat command. But for beginning script writers, the basic concept to learn is that a loop is a set of instructions that are performed repeatedly before going on to the instructions that follow the loop. If this is an unfamiliar concept, think of the actions you perform when setting the dinner table. The sequence of instructions (the place setting script) might read:

```
on tableSetting
  take place settings to dining room
  repeat until all plates are in position
    put one plate in position
  end repeat
  return to kitchen for glasses
  --and so on
end tableSetting
```

For an average family, you could write repeat 4 times, but then you'd have to rewrite the script for guests. Specifying a fixed value is sometimes called hard-wiring and would not be desirable in this case. There are better ways to design a script in HyperTalk if you don't know in advance exactly how many times to repeat a loop.

One good way is to ask the user to type a number, and then use that number in the script. We used another technique above when we repeated the loop “until” the user signaled us that all the plates were done. This is called an infinite loop, since it will keep repeating forever if the user doesn't click. If you ever suspect you're caught in an infinite loop, you can terminate any HyperCard script by pressing Command-Period.

Let's write a simple script that asks you how many times you want the Mac to beep. To begin, return to the button card and choose New Button from the Objects menu to create another button. Double-click on the button and change its name to “Beeper.”

Then click on the Script button and enter the script shown in Example 2.

## EXAMPLE 2

```
on mouseUp
ask "How many beeps" --displays a dialog box asking for user input
--the number the user types is placed in a container called "it"
put it into beepCount --put value of "it" into the container "beepCount"
repeat beepCount --repeat the number of times equal to beepCount
beep --beep the Mac speaker
wait 30 ticks --one half second pause between beeps
end repeat
end mouseUp
```

## ASKING FOR IT

There are couple of new HyperTalk features in this script. First, the command ask followed by a quoted phrase is the fast and easy way to display a dialog box to the user. The quoted phrase will be displayed as the message in the dialog box. A space is provided for the user to type the requested information, and an OK and a Cancel button are provided automatically.

When you enter your answer to the request and click on OK, the answer is stored temporarily in a special HyperTalk container called “it.” This is a special object called a variable in traditional programming languages. You created another variable or container called beepCount in your script. Containers are similar to mailboxes. A mailbox has a name on the outside, like Smith or Jones, but the contents of the mailbox are neither Smith nor Jones, but letters or bills. Likewise, we can refer to a container by its name (it, beepCount), but the contents of that container may be numbers or text.

In this script, when the user provides a response, say 5, the value 5 is first automatically placed in the container called “it” by HyperTalk. Since “it” is used for lots of things, it's usually good practice to store the value of “it” in a container that you create. To send a letter to Smith, we address it to his mailbox; to put a value into a container, we call the container by name. It's a good idea to give your containers names that mean something to you. Since you asked for the number of times to beep, our container is called beepCount. The command put it into beepCount does two things: it creates a new container called beepCount and, at the same time, stores a copy of the current value of “it” in beepCount. In this short script, the value of “it” doesn't change, but you will be writing longer scripts where “it” may change several times in one script. The script then beeps the speaker for the number of times stored in beepCount. So you didn't have to hard-wire the number of beeps—we get that information whenever this script is performed. Try the script a couple of times and give different numbers each time to check that the script works as advertised.



We've learned about buttons, containers, loops, scripts that allow the user to have some input and control, and a handy way to return to the last card we visited. Next time, we'll look at some more HyperTalk commands and system messages and learn how to use HyperCard fields to display information. We'll also encounter two more important programming concepts, conditional statements and modular design.