HYPERTALK TUTORIAL MODULES

KEVIN G. CHRISTMAS

B.Ed., University of Lethbridge, 1986

A One-Credit Project
Submitted to the Faculty of Education
of The University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF EDUCATION

LETHBRIDGE, ALBERTA

June, 1993

TABLE OF CONTENTS

Introduction

This package was intended to be used as support material for a teacher developing his or her own unit in HyperTalk. It is assumed that the students already know the basics of HyperCard, and are now ready to start learning how to script in HyperTalk. The modules (tutorials) have been linked together in alphabetical order in an endless loop, but at any point one can return to the Main Menu, and then go to any other command. Conceptually it is set up as a wheel, with the Main Menu at the center as the hub of the wheel, and spokes going to each module around the wheel. As well, you can go around the wheel in alphabetical or reverse alphabetical order. To install the tutorial modules, simply copy the contents of both accompanying disks into a folder on your hard drive called "HyperTalk Modules." You will also need a copy of HyperCard version 2.1 or newer to run the modules.

Much of this manual has been prepared using the information which is presented in the modules. All of the pages pertaining to commands are hard copies of what has been presented in a more dynamic method as HyperCard stacks. This is the theory, while the modules model the command in action, and show the student how the command acts when it is used within a script. The modules can be used as on-line tutorials or as classroom demonstrations, which ever the teacher feels more confident with. At any rate, they were intended to be available to students to browse through as needed.

The actual proposed unit is to be set up in twenty classes lasting eighty minutes each. In a five credit Computer Processing 10 or 20 course (Alberta Education Curriculum, 1985) this is one method of scheduling the classes. Although the intention is to have the material presented in twenty, eighty minute classes, there is no real need for this to be rigid. The material can be modified to suit a number of different scenarios. As well, the progression suggested in this package is also not intended to be rigid, If a teacher wishes to pick and choose, this program will allow that.

Unit Outline

As stated above, this package was intended to be used as the basis for a twenty class unit consisting of eighty minute classes. Each class would consist of a short introduction of what has been presented on the previous day, and an introduction to the new material. The modules can be used as a whole class presentation with the instructor describing what is happening and supplementing the material.. The last part of the class will consist of work time intended to give students the time

to explore the new material in a structured manner. There should be some kind of assignment associated with each days instruction.

Of the twenty classes, the first ten are intended to be used as teaching lessons, with the second ten used for project working time. As HyperCard can be extremely open ended, students will need quite a bit of time to plan and develop their projects. There are a total of forty-six modules presented in the package, but only twenty-nine modules presented in a formal manner to the students as part of the unit. These twenty-nine modules deal with, for the most part, card manipulation and presentation. The other seventeen are optional because they deal with more sophisticated material and require students to have a more thorough understanding of programming. Additionally, some programming structure must accompany the materials. For example, a logical time to present If-Then-Else loops is when students cover the ask and answer commands. They can then start to analyze input into the computer and have the computer react accordingly. Because the expectations of structure change slightly form teacher to teacher, it is left up to the individual teacher to stress the structure that he or she wants.

As well, it is expected that in each class, students have some kind of activity that they are expected to complete during the latter part of the class. Classes

Lesson #1

This lesson deals with the basics of getting around in HyperCard using HyperTalk. Some students will have explored and gotten around HyperCard quite well, and not even know that the go command exists. The first three commands that can be taught are: go, set and put.

The go command is a good command to start with because it allows students who have the ability to make new cards and put things on those cards to create buttons to navigate through these cards in what ever order they want to. They can even go to a different stack with the go command.

The set command should be introduced here because you can look at another aspect of HyperCard, the user level. The set command can be used to set the user level to scripting, which is the top level. Students can learn that they can restrict access to their stack just by setting the user level to a lower level.

The put command allows the students to put text into containers like fields. Students can practice creating buttons which put relevant text into a field.

The assignment for this class could be for students to develop a HyperCard stack

which upon opening, will put some kind of message to the teacher into the screen in a field. It should then go to another card and set the user level to a lower level. The scripts which do the work should be printed and handed in to the teacher for marking, along with the stack. This can be done now or later as this stack can be use to build upon throughout the unit and handed in at the end of the presentation part, before the project.

Lesson #2

Lesson two looks at the visual effect and wait commands. The visual effect command can be particularly exciting to new scriptors as they can start to add a bit of jazz to their HyperCard stacks.

The wait command can be useful to slow proceedings down. Students can use the wait command to give users time to read text that has been put into a field, for example.

The visual effect command has numerous possibilities. Students may experiment with the different visual effects that are at their disposal, and choose the one or tow that they like the most.

The assignment for this class could be to take the stack that the student created in the last class, and add a wait command to give the user more time to read material that is presented. Students can also add visual effects to the go commands as well.

Lesson #3

Lesson three looks at the mathematical functions built into HyperCard. Each of the functions, addition, subtraction, division and multiplication are presented here.

The assignment for this lesson could be to modify the students existing stack so that they use each of the four mathematical functions, but they must give the user time to see what is going on, using the wait command, and they must tell the user what is going on using the put command. They can make narration fields and use put commands to tell the user what function is happening and the result.

Lesson #4

Now we will look at having receiving input from the keyboard and how it can be managed. The ask and answer commands allow the student to have a dialog box come up on the screen to ask a question. The input can be either in the form of a

mouse click or typing in some kind of information. The get command can be introduced her as well, as it is just another way to allow HyperCard to get information from a specific place. The different between the ask and answer commands and the get command is that the get command retrieves information form somewhere in the stack and the other two get information from the keyboard or mouse.

As well, this would be an excellent time to look at If-Then-Else loop and repeat loop structures. Students can then analyze the input from the keyboard or from a field. The concept of variables can be introduced at this point as well.

A good assignment for this lesson would be to have students use the ask and answer commands to get specific information form the user. They should then be able to put his information into containers (either variables or fields) and then manipulate this information. You may have them create a math game where the computer asks the user a mathematical question and then the user has to input the answer. Students can then analyze the input with an If-Then-Else loop or a repeat loop and have the computer react accordingly.

Lesson #5

Since the amount of information presented in lesson #4 was quite extensive, lesson #5 is less demanding. There are only two commands introduced here, the show and hide commands. These can be useful when creating games where the answer is hidden and then shows up after the answer has been made.

An assignment for this lesson may be to develop a stack which asks the user to answer a question and then puts this input into a container, like a field. Once the user inputs the answer, the stack analyses the input and then tells the user if the answer was correct by showing the correct answer in a previously hidden field. This will reinforce the previous lesson's content and support the new lesson's content.

Lesson #6

Now it may be time for sound. The simplest way to make the computer make some kind of sound is to use the beep command to have the computer exhibit the system beep. Students can use this to indicate that a process has been completed. As well, students can learn how to play a sound resource. At this point, it would be a good idea to introduce them to the ways in which sound can be put into a computer. There are a number of places that students can find sounds to be placed into HyperCard stacks. As well, the audio capabilities build into the newer versions of HyperCard can be explored and students can create their own sound

resources and save them into their own stacks.

An assignment for this lesson could be as simple as having students record a sound and create a stack to play the sound back. If you want to get fancy, you can have students make a stack which will ask the user to input some information and then analyze it. The computer can then return an appropriate audio response to the user. Students will likely have plenty of fun with this one, and it may be time to have them construct some kind of mini-project using all of the commands that they have learned so far. This will reinforce what they have already learned.

Lesson #7

Lesson #7 starts to introduce the students to the database qualities of HyperCard. Although HyperCard makes a poor database program, there are a number of features that do find things and sort things. The examples in the modules are very basic, and deal only with single cards. These commands can be expanded to deal with a number of cards. The find command can look throughout a stack to find the information that the user is looking for. The sort command can be used to shift the order of cards within a stack so that they are in alphabetical order. These are very low level database type procedures, but they work in HyperCard, although they are very slow. The do command would be good to introduce at this time because it will give students the ability to develop buttons which do menu commands like "New Card."

An assignment for this lesson may be to develop a database stack with a number of cards filled with different information. The data from an old database assignment could be the basis for this assignment. Students can set up a HyperCard database which allows the user to find specific information using the ask command. The student may also have a button built into the database which sorts the cards according to a specific criterion. For example, alphabetically according to a specific field on each card. This script would have to use a number of concepts that have already been learned in order to accomplish its job. It would also be a good idea to have a button which creates a new card to add to the database. This will help them to make the stack in the first place. All they will have to do is to press a button and a new card will be created.

Lesson #8

This lesson starts to look more closely at the things that one can do to the menu bar. Each of the menus on the menu bar can be manipulated and changed to what ever the programmer wants. Although the delete command does many other delete functions, when it is discussed with other menu commands, it makes sense

to look at its ability to delete specific menu items on specific menus. The delete menu command deletes the whole menu and the disable command allows the programmer to disable menus or specific menu items. They are all ways of restricting access to the menu system built into HyperCard. They quite naturally go together.

An assignment for this lesson would be to have students create a stack which will, at the press of a button, display each of these commands at work. It could quite simply be a single card with four buttons on it, one for deleting a menu item, one for deleting a whole menu, a third for disabling a whole menu, and a fourth disabling a specific menu item.

Lesson #9

This lesson looks at getting the menus back to normal, or adding other menus and menu items to the existing menu system. It can be used if the programmer wants to have a menu which does a number of specific things. For example, in a database stack, the programmer may want to have a sort menu which has a number of different sort options as menu items. This way, instead of having a number of buttons cluttering up the screen, the programmer may have a new menu added to the existing menus. The enable command it the opposite of the disable command and works exactly the same way. The reset menuBar command sets the menu back to its original condition before any modifications took place and the create menu command allows the programmer to create custom menus for his or her stacks.

An assignment for this lesson could take the form of modifying the stack created in the last lesson and restoring the menu to its original condition. It may also be used to introduce the idea of having two buttons that do opposite things in exactly the same place. When one button is pressed, it hides itself and shows the other button. The disable and enable commands lend themselves to this technique quite well. When the disable command disables a menu or menu item, it hides itself and shows enable button. The enable button can enable the same menu or menu item and then hide itself and show the disable button. Its like making an on off switch for a menu or menu item. This is a common and useful technique for a HyperTalk scriptor to learn. Finally, the student should be required to make a new menu, complete with menu items (although they don't have to be functional at this point).

Lesson #10

This lesson looks at the lock and unlock commands. These commands can be very useful when it comes to having a programmers stacks look professional.

When things are happening, for example a script is doing a number of tasks that change the screen while they are being executed, it is very distracting to have the screen jumping around while a script is running. The lock and unlock commands can be used to keep the screen exactly the way it starts, until all modifications are done, then it will change and show the modifications. The example in the lock module is very subtle and the instructor will have to point out exactly what is happening and that the timing of buttons being removed and returned to the screen is what is being looked at.

As well, the mark and unmark commands can be used in our old database stack to mark cards using the find command. Only those cards with the desired information will be marked for later recall. Again, a repeat loop works well in this situation, and can be reviewed. The lock command is useful in this situation so that the user does not have to sit a watch the computer go to each card to be marked. If the screen is locked, the user will not be aware that the script is looking through each card to find the information that they are looking for.

An assignment for these commands could be to modify the old database assignment to have it use the find command to mark cards with specific information on them. The lock and unlock commands can be sued to make the stack look more professional.

Lessons #11 - #20

Major Project:

Since all of the introductory commands have been presented, it is time to tie it all together and have students create a stack or number of stacks which demonstrate the concepts learned so far. One example of a project stack would be a mathematical game which goes through a number of cards asking the user to answer a number of questions. Many of the commands that have been presented can be incorporated into this assignment. Students should be encouraged to be creative and play with the assignment.

Another idea for a project would be a complete database stack with buttons and modified menus to do specific database type activities. This stack too would be required to use many of the commands that have been presented.

A third idea for a project may be some kind of game or tutorial which may interest the student. Hockey tutorials, geography games, and productivity stacks are example of some of the things that can be done.

These are just ideas for projects. There are countless things that can be done at

this point. It would be a good idea to have a number of examples of HyperCard stacks available for students to browse though and get ideas from. It would be a good idea to have some kind of clip art library available to the students. A scanner is a good idea since students can use it to input images that they need for their specific project. Some way to have sound input is almost essential. A device like a MacRecorder or a new Macintosh with sound capabilities will do quite nicely.

Don't rule out the idea of assigning a project in which the only requirement is that students use at least 20 of the 29 commands presented. Have them be creative and have fun with the assignment. Above all, allow students to have fun with their projects and show off their creativity. In this environment, freedom may be more important than tight structure.

Have students print out the scripts to show that they have used the commands that they are required. They need not print out all scripts, just those which demonstrate their ability to use the required number.

Scope & Sequence Chart

DAY **I = Introduce** **R = Review** **W = Work on command** **O = Optional**

COMMAND	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Go	I	R									W	W	W	W	W	W	W	W	W	W
Put	I	R									W	W	W	W	W	W	W	W	W	W
Set	I	R									W	W	W	W	W	W	W	W	W	W
Visual		I	R								W	W	W	W	W	W	W	W	W	W
Wait		I	R								W	W	W	W	W	W	W	W	W	W
Add			I	R							W	W	W	W	W	W	W	W	W	W
Divide			I	R							W	W	W	W	W	W	W	W	W	W
Multiply			I	R							W	W	W	W	W	W	W	W	W	W
Subtract			I	R							W	W	W	W	W	W	W	W	W	W
Get				I	R						W	W	W	W	W	W	W	W	W	W
Answer				I	R						W	W	W	W	W	W	W	W	W	W
Ask				I	R						W	W	W	W	W	W	W	W	W	W
Hide					I	R					W	W	W	W	W	W	W	W	W	W
Show					I	R					W	W	W	W	W	W	W	W	W	W
Beep						I	R				W	W	W	W	W	W	W	W	W	W
Play						I	R				W	W	W	W	W	W	W	W	W	W
Do (doMenu)							I	R			W	W	W	W	W	W	W	W	W	W
Find							I	R			W	W	W	W	W	W	W	W	W	W
Sort							I	R			W	W	W	W	W	W	W	W	W	W
Delete								I	R		W	W	W	W	W	W	W	W	W	W
Delete Menu								I	R		W	W	W	W	W	W	W	W	W	W
Disable								I	R		W	W	W	W	W	W	W	W	W	W
Create Menu									I	R	W	W	W	W	W	W	W	W	W	W
Enable									I	R	W	W	W	W	W	W	W	W	W	W
Reset MenuBar									I	R	W	W	W	W	W	W	W	W	W	W
Lock										I	R	W	W	W	W	W	W	W	W	W
Mark										I	R	W	W	W	W	W	W	W	W	W
Unlock										I	R	W	W	W	W	W	W	W	W	W
Unmark										I	R	W	W	W	W	W	W	W	W	W
Choose Tool	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Click	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Convert	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Create Stack	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Drag	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Edit Script	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Exit	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Exit to HyperCard	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Global	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Pop Card	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Push Card	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Reset Paint	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Reset Printing	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Return	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Select	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Send	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Type	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

Add

Explanation

The add command will add a number to a container. The original number in that container will be lost and the content of the container will be the new number.

The basic syntax of the add command is:

add <number> to <container>

It should be noted that with the add command, there must be a container, and that the container must already contain a number. If there is no container specified, or no number in the container, HyperCard will return an error.

The Script

```
on mouseUp
   put 1 into line 4 of background field "Run Field"
   wait 1 second
   add 5 to line 4 of background field "Run Field"
end mouseUp
```

Script Description

This script initially puts the number 1 into line four of the background field "Run Field". This is to accommodate HyperTalk's need to have a number in the container before the add command is executed. It then waits 1 second merely to allow you to see that the number 1 is in the field. After the wait, it adds 5 to the number in line four of the field, which makes 6. The number six is then displayed in line four of the field.

Answer

Explanation

The answer command is a very useful way to get user input into your stacks. It allows you to create a dialog box, which asks a question and allows the user to pick from up to three specified replies.

The basic syntax of the answer command is:

answer "<question>" with "<reply1>" or "<reply2>" or "<reply3>"

There are three rules that must be kept in mind when using this command.

1) The replies will be inserted into the dialog box from the left to right, as if the second and third replies pushed the others to the left.

2) The extreme right hand reply is always highlighted.

3) If no replies are specified, the default is a single "OK" reply.

The Script

```
on mouseUp
  answer "This is an example of the answer command" with "Reply 1" or ¬
  "Reply 2"  or "Reply 3"
end mouseUp
```

Script Description

In this script the answer command allows the programmer to make the statement, "This is an example of the answer command." The three replies, Reply 1, Reply 2 and Reply 3 are put after the word "with." If there is nothing after the statement ie. no "with" and replies, HyperCard will return only one reply button. This box will contain "OK."

Ask

Explanation

The ask command is a very useful way to allow the user to input strings of information into the computer. Anything that can be typed into the computer can be an answer to an ask command.

The basic syntax for the ask command is:

ask "<question>" with "<reply>"

When this command is used, HyperCard will create a dialog box which asks a question, giving the user the ability to type in an answer. In addition, there are two predetermined buttons on the dialog box. One which says "OK" and one which says "Cancel".

The use of the "with" and "reply " are optional, and if left out, leave the answer area blank.

The "OK" button is automatically highlighted.

The Script

```
on mouseUp
  ask "What is the capital of Canada?" with "Ottawa"
end mouseUp
```

Script Description

In this script the ask command allows the programmer to ask the question, "What is the capital of Canada?" The reply box contains the word "Ottawa", indicating that it is a suggested response. As well, all ask dialog boxes, contain two answer buttons, "OK" and "Cancel." The user simply must click on the appropriate button, or change the answer by typing in a new one, and then pressing the appropriate button.

Beep

Explanation

The beep command allows the programmer to have the computer respond to an action with a system beep.

The basic syntax of the beep command is:

beep <number>

The number is optional and only tells the computer how many times to play the system beep. If there is no number, the computer will play the system beep only once.

The Script

```
on mouseUp
  beep 3
end mouseUp
```

Script Description

This simple script will play the system beep three times when activated. If the system beep is set to the Clink-Klank or boing, or any other system beep, that beep will play.

It is a good practice to have this as an alert when something in a script does not go as planned. For example if the user presses the "Cancel" button on an ask or answer button, it will alert the user that they have done something that was not really expected.

Choose Tool

Explanation

The choose tool command allows the programmer to choose any paint tool from within a script. When activated, it will act exactly as if it were chosen from the tools menu itself. The programmer may then have it perform its special function under the control of the script.

The basic syntax of the Choose Tool command is:

choose <tool name> tool

Each of the tools in the tool menu has a name that can be used with this command.

The Script

```
on mouseUp
  choose spray can tool
  set dragSpeed to 150
  drag from 300,100 to 400, 200
  wait 1 second
  choose eraser tool
  drag from 300,100 to 400, 200
  choose browse tool
end mouseUp
```

Script Description

Although this is a rather long script, it has only three specific tasks. The initial choose tool command selects the spray can tool. The next two lines of the script set the speed of the movement on the screen and actually draws the diagonal spray painted line on the action window. The wait command tells the computer to wait for 1 second and then the next choose command selects the eraser tool. The sixth line erases the spray painted line previously made and the last line selects the browse tool so that the everything is returned to the way it started.

Click

Explanation

The click command simulates the user clicking the mouse button at a specific point. The point is specified by a set of coordinates indicating a screen position. These coordinates follow the click command in the syntax.

The syntax of the click command is:

click at <point> with "Key List"

the key list may include modifier keys like optionKey or commandKey. Each modifier key in the key list must be separated by commas.

The Script

```
on mouseUp
  click at 380,150
end mouseUp
```

Script Description

Although it's transparent, this simple script simulated a mouse click at the coordinates 380,150 to press the button called "Beep". The script for the beep button is:

```
on mouseUp
  beep
end mouseUp
```

Convert

Explanation

The convert command allows the programmer to convert the various forms of the time and date into other forms of the time and date. For example, the convert command illustrated in the action window shows the result of the long date being converted into seconds. All of the conversions are based on the time elapsed since the Macintosh base date and time of January 1, 1904, at 00:00:00 h (12 midnight).

The basic syntax of the convert command is:

convert <container> to <format>

The seven different formats are:

long date
short date
seconds
abbreviated date
dateItems
long time
short time

The first five of these are shown in the action window. The dateItems format splits the date as follows: year, month, day, hours, minutes, seconds, day of the week (with Sunday being #1)

The long time is given in the format: hours:minutes:seconds AM or PM
For example: 12:34:23 PM

The short time is given in the format: hours:minutes AM or PM
For example: 12:34 PM

The Script

```
on mouseUp
  put the long date into datetoday
  put "The long date is:" && datetoday into line 3 of background field ¬
  "Run Field"
  convert datetoday to short date
  put "The short date is:" && datetoday into line 5 of background field ¬
  "Run Field"
  convert datetoday to seconds
  put "The date in seconds is:" && datetoday into line 7 of background field ¬
  "Run Field"
  convert datetoday to abbreviated date
  put "Abbreviated date is:" && datetoday into line 9 of background field ¬
  "Run Field"
end mouseUp
```

Script Description

This script puts the long version of today's date into a container called "datetoday." It then puts "Today is:" and the long date onto the screen. It then proceeds to convert the contents of the container "datetoday" into three other forms of the date. The short date, the date in seconds and the abbreviated date, putting each of these onto it's own line.

Create Menu

Explanation

The create menu command creates a new menu in the menu bar.

The basic syntax of the create menu command is:

create menu <menu name>

This command is useful if you want to create custom menus with specific tasks that are not ordinarily present in HyperCard.

To add menu items to a new menu, the programmer must use the put command to put the menu item into the menu. To do this, the following syntax must be followed:

put <itemName> after menu <menu Name>

This will place a new menu item on the list of a menu. To add more items, simply put more items onto the list.

The Script

```
on mouseUp
  show menubar
  create menu "Test Menu"
  put "Item #1" after menu "Test menu"
  put "Item #2" after menu "Test menu"
  put "Item #3" after menu "Test menu"
  put "Item #4" after menu "Test menu"
end mouseUp
```

Script Description

This script shows the menu bar first, and then creates the new menu, "Test Menu". It then creates four menu items to go into this menu, Item #1, Item #2, Item #3, and Item #4.

Create Stack

Explanation

The create stack command creates a stack with the name given in the command. There are not dialog boxes as would happen if the New Stack menu command were chosen.

The basic syntax of the create stack command is:

create stack <stack name>

The Script

```
on mouseUp
  create stack "Test stack"
end mouseUp
```

Script Description

This script leaves the current stack and creates a new stack called "Test Stack." The actual script that is run by pressing the "Run Script" button is slightly more complicated for the demonstration, but the create stack command works exactly as shown.

Delete

Explanation

The delete command can be one of the most powerful text manipulation devices available to the HyperTalk programmer. It can be used to delete any chunk of text in any field or any container. You can delete any component of a field which can be specified. For example, you can delete the fourth word in line six of card field "Text Field." You can also delete line six of card field "Text Field."

The basic syntax of the delete command is:

delete <specific chunk of text>

A specific chunk of text would be the item number, line number, field name and card name of the text.

Additionally, the delete command can be used to delete menu items, both custom and normal HyperCard menu items.

The basic syntax of the menu version of the delete command is:

delete menuItem <Item Name> from menu <Menu Name>

The Script

```
on mouseUp
  delete the first word in line 1 of background field "Run Field"
  wait 4 seconds
  delete the last word in line 9 of background field "Run Field"
  wait 4 seconds
  delete fourth word of line 4 of background field "Run Field"
  wait 4 seconds
  delete line 5 of background field "Run Field"
end mouseUp
```

Script Description

This script deletes the first word in line 1, the last word in the paragraph, the fourth word in the fourth line and then the whole fifth line, waiting 4 seconds between each action.

Delete Menu

Explanation

The delete menu command deletes a specific menu from the menu bar. All menu items on that menu are also deleted along with the menu.

The basic syntax of the delete menu command is:

delete menu <menu name>

The delete menu command will delete any menu, both normal HyperCard menus and custom menus.

The Script

```
on mouseUp
  ask "Which menu would you like to delete?"
  put it into chosenmenu
  delete menu chosenmenu
end mouseUp
```

Script Description

This script asks the user which menu that he/she would like to delete and puts the response into an container called "chosenmenu." It then deletes the menu "chosenmenu."

Disable

Explanation

The disable command will disable the menu or the menu item indicated.

The basic syntax of the disable command is:

disable <menu item> of menu <menu name>

If the whole menu is to be disabled, the "menu item" and "of" can be deleted from the command.

The Script

```
on mouseUp
  disable menu go
end mouseUp
```

Script Description

This script simply disables the Go menu so that nothing can be accessed.

Divide

Explanation

The divide command will divide a container by a number. The original number in that container will be lost and the content of the container will be the quotient.

The basic syntax of the add command is:

divide <container> by <number>

It should be noted that with the divide command, there must be a container, and that the container must already contain a number. If there is no container specified, or no number in the container, HyperCard will return an error.

The Script

```
on mouseUp
  put 12 into line 4 of background field "Run Field"
  wait 1 second
  divide line 4 of background field "Run Field" by 3
end mouseUp
```

Script Description

This script initially puts the number 12 into line four of the background field "Run Field". This is to accommodate HyperTalk's need to have a number in the container before the divide command is executed. It then waits 1 second merely to allow you to see that the number 12 is in the field. After the wait, it divides the number in line four of the field by 3, which makes 4. The number 4 is then displayed in line four of the field.

Do

Explanation

The do command can be used to call a menu command.

The basic syntax of the do command is:

doMenu <Menu Item>

The menu item in this case must be exactly as it looks in the menu itself. For example, doMenu "Open Stack...". The three dots at the end of the command are in the menu itself, so they also have to be in the command in a script.

The other form of the do command is relatively uncommon, and most programmers will never use it at all. One of its uses it to help when you are programming very long and complex scripts. Sometimes, scripts can become so long that they can not be executed because the maximum number of characters in a script may not exceed 32 000. If you have a scripts that exceeds this, you may use the do command to extend the limit.

The basic syntax of the do command in this form is:

do <string>

This string may be the name of a field, or any addressable component of HyperCard. For example, the command do card field "Instructions" will look at the card field "Instructions" and execute the text in the field as if it were part of the script.

The Script

```
on mouseUp
  do background field "Instructions"
end mouseUp
```

Script Description

This script tells HyperCard to go to background field "Instructions" and execute the text within it as if it were a script.

Drag

Explanation

The drag command acts just like pressing the mouse button at a specific place on the screen and dragging it to a new position on the screen before letting the mouse button up. The coordinates used with this command are based on the fact that there are 512 pixels across the screen and 342 pixels down the screen. The top left hand corner is the 0,0 coordinate and the bottom right hand corner is the 512,342 coordinate.

The basic syntax for the drag command is:

drag from <coordinate> to <coordinate>

The Script

```
on mouseUp
  choose spray can tool
  drag from 300,100 to 400, 200
  wait 1 second
  choose eraser tool
  drag from 300,100 to 400, 200
  choose browse tool
end mouseUp
```

Script Description

In this script, the initial choose tool command selects the spray can tool. The next line uses the drag command to draw the diagonal spray painted line on the action window from 300,100 to 400, 200. In the next two lines the wait command tells the computer to wait for 1 second and then the eraser tool is selected. The fifth line again uses the drag command to erase the spray painted line previously made. The last line selects the browse tool so that the everything is returned to the way it started.

Edit Script

Explanation

The edit script command is likely one that you will never ever use, but it is included here so that you know that it exists. What is does is allows the user to edit the script of an object, from within a script.

The basic syntax for the edict script command is:

edit script of <object>

The object may be any HyperCard based object from a background, to a button, to a field. Anything that has a script and can be identified by name can be modified.

The Script

```
on mouseUp
  edit script of bg button "Play Sound"
end mouseUp
```

Script Description

This script allows the user to go in physically change the script of the background button "Play Sound," which would have played the sound floop before it was modified.

Enable

Explanation

The enable command will allow the menu or the menu item indicated to be used if it has previously been disabled.

The basic syntax of the enable command is:

enable <menu item> of menu <menu name>

If the whole menu is to be enabled, the "menu item" and "of" can be deleted from the command.

The Script

```
on mouseUp
  enable menu go
end mouseUp
```

Script Description

This script simply enables a previously disabled menu so that it can now be used.

Exit

Explanation

The exit command is used to leave a handler before the end is reached. For example, if you only want a certain list of commands executed when a specific condition is present, then you can use the exit command to exit the script if that condition is not present.

The basic syntax of the exit command is:

exit <handler name>

The Script

```
on mouseUp
  repeat 3
    if bg field "Run Field" is not EMPTY
    then
      put "The repeat loop has been exited on the second loop"
      into line 3 of bg field "Run Field"
      exit repeat
    else
      put "This is the first time through the repeat loop" into line 3 of bg field ¬
      "Run Field"
    end if
  end repeat
end mouseUp
```

Script Description

This script is meant to be run through a repeat loop three times, checking to see if the field called "Run Field" is EMPTY each time it goes through the loop. The first time through, the field "Run Field" is empty so HyperTalk executes the else portion of the script, which puts a message into the field.

The second dime through the loop, the field "Run Field" is not EMPTY, so HyperTalk exits the loop without going through the third time.

Exit to HyperCard

Explanation

The exit to HyperCard command quits all levels of handlers immediately and stops HyperTalk completely

The basic syntax of the exit to HyperCard command is:

exit to HyperCard

This command can be used if you have a number of things happening that the user may not be aware of. It is a quick and easy way to shut down what is happening without leaving the stack that you are in.

Find

Explanation

The find command can be used to have HyperTalk look for many different types of text on the cards that you create.

The simplest and most basic syntax of the find command is:

find <text>

The word text represents what ever text that you want found.

There are a number of modifiers that can be used with the find command. They are: whole, string, word, and chars. Each of these results in a slightly different interpretation of the find command.

The most efficient use of the find command may be to use an ask command to find out what information the user is looking for and then using that information in the find command. For example: if you create a find button in a stack, you may want to use the answer command to have the user type in the information what will be used in the find command and put it into a container. You must then have HyperTalk find matches with the information in the container.

The Script

```
on mouseUp
  ask "What would you like to find?"
  put it into response
  find response
  if the result is EMPTY then
    put "The word " & response & " has been found" into line 3 of bg field ¬
    "Run Field"
  else
    put "The word " & response & " was not found" into line 3 of bg field ¬
    "Run Field"
  end if
end mouseUp
```

Script Description

The script here first uses an ask command to find out the comparison information for the search. Then it just looks throughout the stack until it finds that word, drawing a rectangle around a hit. The action window indicates that a hit was made. A click of the mouse anywhere will make the rectangle disappear. If the word was not found, no word is marked and the action window indicates that there was no hit.

Get

Explanation

The get command retrieves data from one of many sources and saves the information to a special variable called "it." The get command is often used with the put command to put the data retrieved by the get command into a more stable container, like a field. The "it" variable is very extensively used by HyperTalk so it could be erased in another step which puts data into "it."

The basic syntax of the get command is:

get <source expression>

Where the sources expression is a field.

The Script

```
on mouseUp
  get line 3 of bg field "Explanation"
  put it
end mouseUp
```

Script Description

In this script, HyperTalk looks into the background field called Explanation, and gets the information from the third line to store in the variable "it." It is important to note that the third line starts after the second return command, making it the third paragraph in the field. The script then puts the contents of the variable "it" into the message box with chows up at the bottom of the screen.

Global

Explanation

The global command is used to specify variables which will be available to other scripts within the stack or even in another stack. Unless variables are declared a global, they are considered by HyperTalk as local which means that they may only be used within a single script. If they are global, they may be carried to other scripts. Variables must be identified before they are used in order for them to become global.

The basic syntax of the global command is:

global <variable name>, <variable name>, ...

You can name more than one global variable in a command line.

Go

Explanation

The go command can be used in two ways. It can allow the user to go to any card in the stack that it is in or to any card of any other stack that is available.

The basic syntax of the go command is:

go card <card name> or go stack <stack name>

Sometimes when a stack is getting too large, the programmer may want to create a new stack and then transfer to that stack. This can be accomplished with the go command. The go command also helps to navigate throughout any stack.

The Script

```
on mouseUp
  go stack "Main Menu"
  go stack "Go"
end mouseUp
```

Script Description

This script goes to the stacks called "Main Menu" and then returns to this stack.

Hide

Explanation

The hide command can be used to hide any object. Some examples are: buttons, fields, and the menubar. The opposite of the hide command is the show command, and it can be used to make an object show up again.

The basic syntax of the hide command is:

hide <object>

The Script

```
on mouseUp
  hide bg field "title"
  wait 5 seconds
  show bg field "title"
end mouseUp
```

Script Description

In this script, the background field "title" is hidden for a period of 5 seconds and is then shown again.

Lock

Explanation

The lock command is very useful when the programmer is trying to have a number of things happen on the screen and doesn't want the user to see them. Technically, it is the lock screen command, and it freezes the screen exactly the way that it looks when the command is given. Any changes to the screen will happen in the background and when the unlock command is given, the screen will show the changes.

The basic syntax of the lock command is:

lock screen

The Script

```
on mouseUp
  hide card field "Show Script"
  hide card field "Run Script"
  hide card field "Describe Script"
  hide card field "Action Window"
  hide card field "To Menu"
  lock screen
  show card field "Show Script"
  show card field "Run Script"
  show card field "Describe Script"
  show card field "Action Window"
  show card field "To Menu"
  unlock screen
end mouseUp
```

Script Description

In this script, all of the fields at the bottom of this field are hidden one by one, and then shown all at once using the lock and unlock commands. You can actually see them disappear from left to right. The screen is then locked and all of the fields are shown while it is locked. Unlocking the screen makes them all show up at the same time.

Mark

Explanation

The mark command is used to mark cards that have specific similarities that can be identified by HyperTalk. For example, if you had a stack which contained all of your recipes, and you wanted to find all of those recipes which contained cherries. You could use the mark card command to mark all of the recipes containing cherries in the "ingredients" field and then browse through the marked cards to find a recipe you like.

The basic syntax of the mark command is:

mark <card identifier> by finding <text>

The Script

Although there is no script to demonstrate here because this stack only has one card, we can look at an example of how the command could be used. Let us consider the example to the left about the cherries. A script to do this task may look as follows:

```
on mouseUp
  mark cards where field "ingredients" contains "Cherries"
end mouseUp
```

Script Description

This script would look through all cards in a stack and mark each one which has cherries in the field called "ingredients."

Multiply

Explanation

The divide command will divide a container by a number. The original number in that container will be lost and the content of the container will be the quotient.

The basic syntax of the add command is:

divide <container> by <number>

It should be noted that with the divide command, there must be a container, and that the container must already contain a number. If there is no container specified, or no number in the container, HyperCard will return an error.

The Script

```
on mouseUp
  put 12 into line 4 of background field "Run Field"
  wait 1 second
  multiply line 4 of background field "Run Field" by 3
end mouseUp
```

Script Description

This script initially puts the number 12 into line four of the background field "Run Field". This is to accommodate HyperTalk's need to have a number in the container before the divide command is executed. It then waits 1 second merely to allow you to see that the number 12 is in the field. After the wait, it multiplies the number in line four of the field by 3, which makes 36. The number 36 is then displayed in line four of the field.

Play

Explanation

The play command is one of the most exciting commands that most new users find in HyperTalk. One of the things about HyperTalk that really interests people is HyperTalk's ability to play sound resources that can be installed into stacks. The play command allows the programmer to play any sound that has been installed into a stack.

The basic syntax of the play command is:

play "sound name"

The Script

```
on mouseUp
  play "boooing"
end mouseUp
```

Script Description

This script plays the sound called "boooing" that has been installed into this stack.

Pop Card

Explanation

The pop card command retrieves a card perviously saved with the push card command. An example of a use for these commands would be if you have some kind of help information that is always available and want HyperTalk to return to the exact card that you left, after getting help. You would use push card, and then when returning, you would use pop card.

The basic syntax of the push card command is:

pop card

The Script

```
on mouseUp
  push card
  go stack "Main Menu"
  pop card
end mouseUp
```

Script Description

In this script, HyperTalk saves the card ID information in memory and then goes to the Main Menu stack. To return to this stack, all that is needed is a pop card command.

Push Card

Explanation

The push card command saves the current card's ID information in memory for future retrieval with the pop card command. These two commands are generally used in conjunction with each other, so you would not use a push card without needing a pop card at some other time. An example of a use for these commands would be if you have some kind of help information that is always available and want HyperTalk to return to the exact card that you left, after getting help. You would use push card, and then when returning, you would use pop card.

The basic syntax of the push card command is:

push card

The Script

```
on mouseUp
  push card
  go stack "Main Menu"
  pop card
end mouseUp
```

Script Description

In this script, HyperTalk saves the card ID information in memory and then goes to the Main Menu stack. To return to this stack, all that is needed is a pop card command.

Put

Explanation

The put command can be used in a couple of interesting ways. First, it can be used to put the contents of a container (a variable) into the message box. This can be useful to track what is happening with a variable as a script is executed. The second, and likely more important use is to put the contents of a container (a variable) into another container like a field or variable. Both of these examples will be demonstrated by the Run Script button.

The basic syntax of the put command is:

put <information> into <destination>

In this command, the information can be in the form of a variable or even text in quotation marks like, "Today is the first day of the rest of your life." The destination can be any field, either hidden or showing. If the "into" and "destination" are omitted in the command, HyperTalk automatically puts the information to the message box.

The Script

```
on mouseUp
  put "Here is an example of text being put into a variable, then a field." into ¬
  testvar
  put testvar into line 3 of bg field "Run Field"
  put "Here is an example of text being put into the message box."
end mouseUp
```

Script Description

In this example there are three forms of the put command. The first shows what happens when a variable is put into the command line. The sentence is put into the variable called "testvar". The second form shows what happens when the variable is put into line 3 of the background field "Run Field." The final form shows information being put directly into the message box.

Reset MenuBar

Explanation

Any time any modifications have been done to the menubar using scripts, there is a simple way to return the menu setting to the default HyperCard settings. This command is reset menubar. It will put all settings that have been changed, back to the way they were before the scripts which modified them effected them.

The basic syntax of the reset menubar command is:

reset menubar

The Script

```
on mouseUp
  show menubar
  disable menu go
  create menu "Test Menu"
  reset menubar
end mouseUp
```

Script Description

The first thing that this script does is shoe the user the menubar. Then it does two modifications to the menu. First it disables the go menu and then it creates a new menu called "Test Menu." Next it resets the menu to its original settings and finally, it hides the menubar again.

Reset Paint

Explanation

The reset paint command simply resets the painting parameters to their default settings.

The basic syntax of the reset paint command is:

reset paint

Any changes that were made on the paint palette will be changed back to their original default settings. For example, if you had changes the font and size of the text tool, it would change back to Geneva 12 point.

The Script

```
on mouseUp
  reset paint
end mouseUp
```

Script Description

This short script will change all of the paint setting back to their defaults.

Reset Printing

Explanation

The reset printing command simply resets the printing parameters to their default settings.

The basic syntax of the reset paint command is:

reset print

Any changes that were made to the print settings will be changed back to their original default settings. For example, plain Geneva 10 point, aligned to the left, with line height of 13 and margins of 0, 0, 0, & 0.

The Script

```
on mouseUp
  reset print
end mouseUp
```

Script Description

This short script will change all of the print setting back to their defaults.

Return

Explanation

The return command can be used when using a put command, to have a string of text break and start on a new line.

The basic syntax of the return command is:

put <text> & return & <text>...

This command gives the programmer more control over where the text is put into a field, and how it looks.

The Script

```
on mouseUp
  put "There is no" & return & "script" & return & "for this command" into ¬
  line 3 of background field "Run Field"
end mouseUp
```

Script Description

This script uses the ampersand (&) to list things to be placed in a field using the put command. In addition, you will see the return command imbedded into the string. This command will have the next part of the string start on a new line. Therefore, "The is no" is on the third line, "script" is on the next line, and "for this command" is on the following line.

Select

Explanation

The select command has two specific functions. You can select a paint tool such as the eraser or text in a field. For example, the programmer can select the third line of the field "Test Field." Alternately, the second word can be selected or the last word.

The basic syntax of the select command is:

select <tool> or select <location> of <field>

The first of these is self explanatory, while the second needs some more explanation.

In the second example of basic syntax, the location of the text within the field can be specified. An example would be:

select word 2 of line 3 of card field "Test"

You can also use the words before and after to give the command more information. For example:

select before first line of card field "Test"

This will put the cursor at the start of the field.

select after the last line of card field "Test"

This will put the cursor at the end of the field.

The Script

```
on mouseUp
  select word 2 of line 3 of background field "Explanation"
end mouseUp
```

Script Description

This script selects the word "basic" from the third line of background field "Explanation." It is important to note that lines in fields are separated by returns, therefore, the third line in the field is after the second return in the field

Send

Explanation

With the send command you can send a message to another object from within a script.

The basic syntax of the send command is:

send <message> to <object>

Using this method allows the HyperTalk programmer to use the send command very much like a procedural call in other languages. Control returns to the script that sent the message after the object has completed its task.

The Script

```
on mouseUp
  send mouseUp to card button "Boooing"
end mouseUp
```

Script Description

This script sends the mouseUp command to the card button "Boooing," causing the button to run its script.

Set

Explanation

The set command can be used to set the properties of numerous objects. One that is used extensively in these stacks is the lockscreen property. You were likely unaware of the fact that the lockscreen property has been switched from "true" to "false" using the set command.

The basic syntax of the set command is:

set <property> of <object> to <value>

The properties that you can change are too numerous to list. Please use other resources for more information.

The Script

```
on mouseUp
  set the textstyle of bg field "Title" to plain
  set the textstyle of bg field "Title" to italic
  set the textstyle of bg field "Title" to outline
  set the textstyle of bg field "Title" to bold
  set the textsize of bg field "Title" to 10
  set the textsize of bg field "Title" to 18
end mouseUp
```

Script Description

This script changes the style and size of the text in background field "Title" a total of 6 times, all in the same script.

Show

Explanation

The show command can be used to show any object which as previously been hidden. Some examples are: buttons, fields, and the menubar. The opposite of the show command is the hide command, and it can be used to make an object hide.

The basic syntax of the shoe command is:

show <object>

The Script

```
on mouseUp
  hide bg field "title"
  wait 3 seconds
  show bg field "title"
end mouseUp
```

Script Description

In this script, the background field "title" is hidden for a period of 3 seconds and is then shown again.

Sort

Explanation

The sort command can be used in a number of ways but because of the way in which this tutorial is set up, there is only one use that can be shown here. The sort command can be used to sort cards in a stack and can also sort data within a field on a card. Here we will deal with sorting data within a field.

The basic syntax of the sort command is:

sort <container> <parameter>

a container us usually a field and the parameter is either ascending or descending

The Script

```
on mouseUp
  sort background field "Run Field" ascending
end mouseUp
```

Script Description

This script just sorts the contents of background field "Run Field" alphabetically by the first word in each line.

Subtract

Explanation

The subtract command will subtract a number from a container. The original number in that container will be lost and the content of the container will be the new number.

The basic syntax of the add command is:

add <number> to <container>

It should be noted that with the subtract command, there must be a container, and that the container must already contain a number. If there is no container specified, or no number in the container, HyperCard will return an error.

The Script

```
on mouseUp
  put 16 into line 4 of background field "Run Field"
  subtract 7 from line 4 of background field "Run Field"
end mouseUp
```

Script Description

This script puts 16 into line 4 of background field "Run Field" so that there is a number in the field at the start. It then subtracts 7 from line 4 of background field "Run Field." Finally, it shows the result in line 4 of background field "Run Field."

Type

Explanation

The type command is used relatively little in HyperTalk programming because it is very much like the put command except that it is much more limited. One good situation for the use of the type command is if the programmer wants to simulate someone typing something into a field. The type command will do very well here.

The basic syntax of the type command is:

type <text>

The Script

```
on mouseUp
  select before first line of background field "Run Field"
  type "Hello there. I am using the type " & "command to put this into this field."
end mouseUp
```

Script Description

This script first has to put the cursor into the field that we want, by selecting before first line of background field "Run Field." Then it is a simple matter to use the type command to have the computer type in the text.

Unlock

Explanation

The unlock command is very useful when the programmer is trying to have a number of things happen on the screen and doesn't want the user to see them. Technically, it is used with the lock screen command, which it freezes the screen exactly the way that it looks when the command is given. Any changes to the screen will happen in the background and when the unlock command is given, the screen will show the changes.

The basic syntax of the lock command is:

unlock screen

The Script

```
on mouseUp
  hide card field "Show Script"
  hide card field "Run Script"
  hide card field "Describe Script"
  hide card field "Action Window"
  hide card field "To Menu"
  lock screen
  show card field "Show Script"
  show card field "Run Script"
  show card field "Describe Script"
  show card field "Action Window"
  show card field "To Menu"
  unlock screen
end mouseUp
```

Script Description

In this script, all of the fields at the bottom of this field are hidden one by one, and then shown all at once using the lock and unlock commands. You can actually see them disappear from left to right. The screen is then locked and all of the fields are shown while it is locked. Unlocking the screen makes them all show up at the same time.

Unmark

Explanation

The unmark command is used to unmark cards that have been previously marked by HyperTalk. For example, if you had a stack which contained all of your recipes, and you wanted to find all of those recipes which contained cherries. You could use the mark card command to mark all of the recipes containing cherries in the "ingredients" field. If you then found out that the person that you were making this recipe for was allergic to nuts, you could use the unmark command to find all of those marked cards which do not contain nuts.

The basic syntax of the unmark command is:

unmark <card identifier> by finding <text>

The Script

Although there is no script to demonstrate here because this stack only has one card, we can look at an example of how the command could be used. Let us consider the example to the left about the cherries and that we have already marked all cards containing cherries. A script to unmark those which contain nuts may look as follows:

```
on mouseUp
  unmark cards where field "ingredients" contains "nuts"
end mouseUp
```

Script Description

This script would look through all of the marked cards in a stack and unmark each one which has nuts in the field called "ingredients."

Visual

Explanation

The visual command is a very interesting way to go from one place to another in HyperCard. It allows the programmer to change screens with one of a number of visual effects.

The basic syntax of the visual command is:

visual effect <effect name> <speed>

The visual effects that you can program are: wipe, scroll, zoom, iris, barn door, stretch, shrink, dissolve, checkerboard, venetian blinds and plain. The first seven of these effects require the programmer to also define the directions that they are to go. As well, iris, barn door, and zoom require the programmer to define open or close.

The speed can be set at very slow, slow, normal, fast and very fast.

The Script

```
on mouseUp
  visual dissolve to black
  visual dissolve to white
  go stack "Main Menu"
  visual checkerboard
  go stack "Visual"
end mouseUp
```

Script Description

This script goes to the Main Menu stack but with a twist. When it goes, it fades to black and then fades back to white. It then returns to this stack with a checkerboard visual effect.

Wait

Explanation

The wait command is a very simple way to suspend the actions of a script for a specific length of time.

The basic syntax of the wait command is:

wait <number> seconds

The Script

```
on mouseUp
  set cursor to watch
  wait for 5 seconds
  set cursor to hand
end mouseUp
```

Script Description

This script sets the cursor to the hand and back again after 5 seconds so that you can see that it is working.

References & Resources

Alberta Education (1985). Business Education Curriculum Guide: Computer Processing 10-20-30.

Apple Computer, Inc. (1990). Apple HyperCard Script Language Guide: The HyperTalk Language. Addison Wesley, Don Mills, Ontario.

Apple Computer, Inc. (1987). HyperCard Stackware. Cupertino, California.

Apple Computer, Inc. (1989). HyperTalk Beginner's Guide: An Introduction to Scripting. Cupertino, California.

Goodman, Danny (1988). Danny Goodman's HyperCard Developer's Guide. Bantam Books, Toronto, Ontario.

Goodman, Danny (1988). The HyperCard Handbook 1.2 Upgrade Kit. Bantam Books, Toronto, Ontario.

Michel, Steven L. (1989). HyperCard: The Complete Reference. McGraw-Hill, Berkeley, California.

Shafer, Dan (1988). HyperTalk Programming. Hayden Books, Indianapolis, Indiana.

Shafer, Dan (1991). The Complete Book of HyperTalk 2. Addison Wesley, Don Mills, Ontario.

Vaughn, Tay (1988). Using HyperCard: From Home to HyperTalk. Que Corporation, Carmel, Indiana.