

# Main Topics for HyperTalk Reference

1. Click an item to get a list of topics:

HyperTalk basics

Editing scripts

System messages

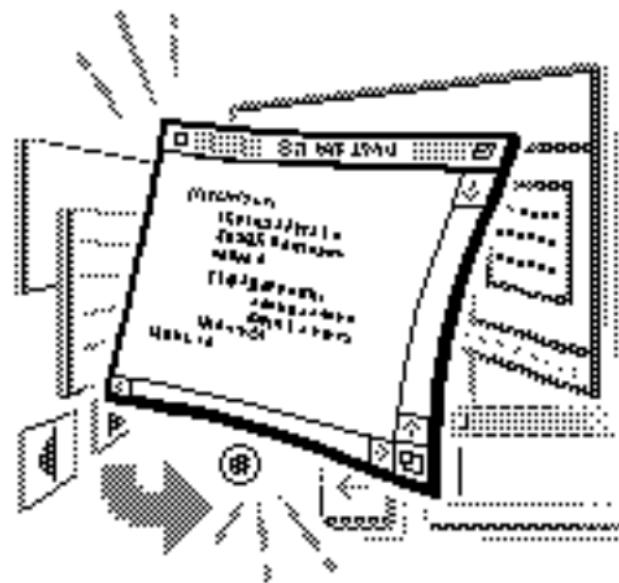
Keywords

Commands

Functions

Properties

Operators and constants



# HyperTalk



Go Audio Help



Find Topic



Overview of Help



# Set the user level to scripting

Card 1 of 1

Before you can write your own scripts, you must set your user level to Scripting (5).

Go to the Preferences card of your Home stack and set your user level there if you want it set to Scripting every time you start HyperCard.

Your current user level is: 5

Click Set to Scripting if it is less than 5.

**Set to Scripting**

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)

 [Find Topic](#)



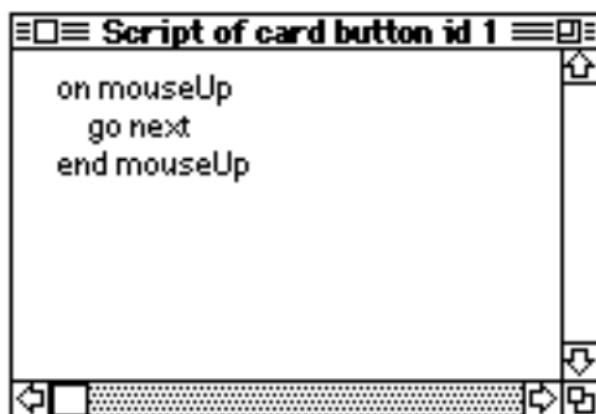
[Main Topics](#)



HyperTalk® is HyperCard's script language. You use it to write English-like statements that respond to events (such as when the user clicks a button or goes to a new card).

In HyperTalk, responding to an event is called **handling** the event. As a scripter, you write a specific handler for each event that you want your stack to handle. A collection of handlers is called a script.

The following graphic shows a handler for the `mouseUp` event. It's part of the script of card button id 1.



```
on mouseUp
  go next
end mouseUp
```

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(870f5d5e9c0d57485634be3ecf52f3ca\_img.jpg\) Related Topics](#)

## What are messages?

A message is simply an announcement that an event has occurred: The user has clicked the mouse, requested the answer to  $4 * 3$ , added a new card to a stack, and so on.

To understand messages, think of mailing a letter to a friend. You write a message and place it inside an envelope. Then you address the envelope and send it to your friend.

HyperCard does the same thing:

1. It determines the content of a message (what just happened).
2. It decides where to send the message.



--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)

### What's the content of a message?

The message itself is just HyperCard's name for the event that occurred. HyperCard acts as a translator: it "watches" the stack and translates events into message names.

To see HyperCard translate events into message names, move the pointer over the Example Button and click.

#### Example Button

HyperTalk translations:

- mouseEnter**
- mouseWithin**
- mouseLeave**
- mouseDown**
- mouseStillDown**
- mouseUp**

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



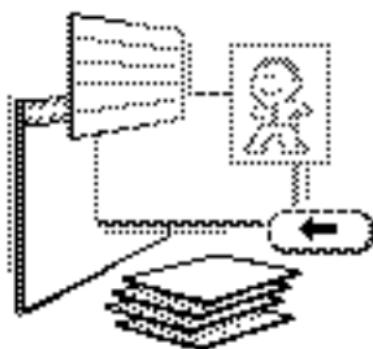
[Main Topics](#)



### Where does HyperCard send the message?

HyperCard determines what object the user has acted on and uses this as the "address" for the message. HyperCard then sends the message to one of its objects:

- a button
- a field
- a card
- a background
- a stack



Once HyperCard knows both the content of the message (the message name) and the destination of the message (where to send it), it sends the message to the object.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Handling messages

As a scripter, you write message handlers to respond to messages. When an object receives a message, HyperCard searches the object's script for a handler with the same name (a handler starts with the word `on` followed by the name of a message). If HyperCard finds a match, it runs any HyperTalk statements in the handler until it hits an `end` statement.

For example, if you want a button to respond to a `mouseUp` message, you would add a `mouseUp` message handler to its script.

```
on mouseUp  
    play "boing"  
end mouseUp
```

**Boing**

Click Related Topics for more information about opening a script window.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



What happens if an object that receives a message doesn't handle it?

In this case, HyperCard passes the message to other objects and searches their scripts for a message handler that matches the current message. The order in which HyperCard passes a message to objects is called the *message-passing order* or the *message-passing path*.

Initially, HyperCard sends messages to a specific button or field or to the current card. If a button or field doesn't handle the message, it goes on to the current card. From the current card, the message goes to the following objects, in order:

- the current background
- the current stack
- the stack script of the Home stack
- HyperCard itself

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



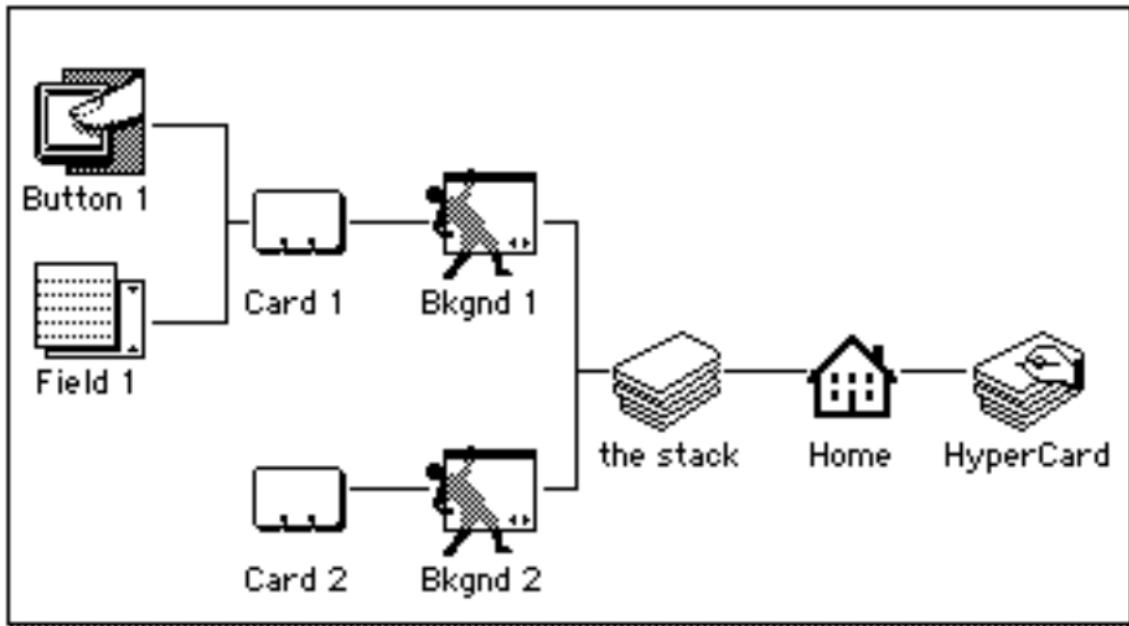
To see the message-passing order in action, click each of the following buttons:

**Message Box**

**Field 1**

**Button 1**

**Dynamic Path**



----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

[Related Topics](#)

[Find Topic](#)



[Main Topics](#)



## Writing message handlers

Card 1 of 2

When you write a handler for a message, you specify a sequence of statements for HyperCard to run. Each message handler has the following form, with the italicized words as placeholders:

```
on messageName
  statements
end messageName
```

When it runs the message handler, HyperCard sends each line of the handler as a message itself (so handlers can call other handlers).

**Important:** The message name does *not* have to be one of HyperCard's built-in system messages or commands.

For example, if you wanted a new command called `doubleBeep` that would beep twice, you would write a handler for it as follows:

```
on doubleBeep
  beep
  beep
end doubleBeep
```

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Of course, HyperCard doesn't know about the doubleBeep command, so it will never be sent automatically in response to an event. But you can send doubleBeep from the Message box or use it as a statement in other handlers.

For example, the script of a card button might contain the following handler for the system message mouseUp:

```
on mouseUp  
    doubleBeep  
end mouseUp
```

If the script also contains the doubleBeep handler (or if the script of any object later in the message-passing path contains it), the mouseUp message will send the doubleBeep message, and doubleBeep will send two beep commands. Because beep is a built-in command, HyperCard beeps twice.

If HyperCard can't find the doubleBeep message, it will complain (with a dialog box) that it "can't understand" the message.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Using parameter variables

Card 1 of 1

A handler can receive values (called parameters) and use them as it runs. You represent each value with a **parameter variable**. A parameter variable always follows the handler name in a comma-separated list.

For example, when running the following mouseUp handler, HyperCard calls sayMessage with two values, "red" and "apple". It then binds these values to the parameter variables color and fruit in the sayMessage handler.

```
on mouseUp
    sayMessage "red", "apple"
end mouseUp

on sayMessage color, fruit
    put "I want a" && color && fruit
end sayMessage
```

You can use the variables color and fruit anywhere inside the handler. When HyperCard sees them, it uses the values currently bound to them. (The variables remain bound only while the handler runs.)

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Writing function handlers

Card 1 of 1

When you write a handler for a function, you specify statements that compute and return a value to the handler that calls the function. Each function handler has the following form, where the italicized words are placeholders:

```
function functionName
  statements
end functionName
```

HyperCard has many built-in functions, but you can also write your own:

```
on mouseUp
  put square(5) into the Message box
end mouseUp

function square x
  return (x * x)
end square
```

The function `square` receives a number through its parameter variable, `x`. It then returns the value of `x * x` to the handler that called it (`mouseUp`) using the `return` keyword.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Building blocks

- basic sources of value
- operator
- expression
- constant
- local variable
- global variable
- it
- button
- field
- message box
- selection
- chunk expression
- property



HyperTalk scripts use a number of basic building blocks as sources of value.

Click one for more information about it.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## Refer by

- name
- number
- id
- part
- ordinal
- position
- me
- target

You can refer to HyperCard objects and other elements from a script in any of several ways.

Click a term in the "Refer by" list for more information about that term.

Click Tips for a list of synonyms that you can use for referring to objects.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



In HyperTalk scripts, two hyphens (--) indicate a comment. HyperTalk ignores all commented lines when executing a script.

For example:

```
on mouseUp
    -- Displays a dialog box
    answer "What?" with "No" or "Yes"
    put it -- the user's reply
end mouseUp
```

- To comment several lines at once, drag to select the lines you want to comment, and then choose Comment from the Script menu (or press ⌘--).
- To “uncomment” several lines at once, drag to select the lines you want to uncomment, and then choose Uncomment from the Script menu (or press ⌘-=).

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



There are several ways to close a script window. If you close a window in which you've made changes, HyperCard asks you if you want to save the changes before it closes the window.

## To close a script window:

- Click the window's close box.
- Choose Close from the Edit menu (or press ⌘-W).
- Hold down ⌘-Option and click.

### To close a script window without saving your changes:

Press ⌘-. (period).

### To close a script window and save your changes:

Press Enter.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

HyperCard provides some useful tools for troubleshooting your scripts. These debugging tools let you step through a handler line by line as it runs, inspect the values of variables, and trace the flow of messages.

To debug a handler, follow these steps:

1. Open the script that contains the message or function handler you want to debug.

2. Click to place the insertion point on the line with the `on` or `function` statement that defines the handler.

Or, depending on where you want to start, place the insertion point on any statement of the handler that is not a comment.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



3. Choose Set Checkpoint from the Script menu (or press ⌘D) to set a temporary checkpoint.

A checkpoint tells HyperTalk where you want to start watching the script as it runs.

4. Press Enter to save the script and close the script editor window.
5. Now, perform the action in HyperCard that will run the script (for example, click a button).

When HyperCard hits the checkpoint, it opens the script and puts a box around the current statement. HyperCard also displays the Debugger menu.

HyperCard does not enter the script editor itself: All of HyperCard's normal menus remain in the menu bar. (Because the script is still running, HyperCard must preserve the current context.) But *only the Debugger menu is active*.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



6. Choose commands from the Debugger menu to proceed.

Click a command for more information about it.

:  
----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



In HyperCard, you can have more than one script window open at once.

## To move between the windows that contain scripts:

- Click to activate the script window in which you want to work.
- Choose an open script window from the Go menu.

## To move between all open windows (including windows that contain stacks):

- Choose Next Window from the Go menu (or press ⌘-L).

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## Objects

- A button
- A field
- A card
- A background
- A stack

Click any object in the list at the left to see how to open its script window.

You can have more than one script window open at a time, and you can leave the script window open while you work on your stack.

As you write handlers in a script window, place each HyperTalk statement on a single line. Press Option-Return to break a statement across more than one line. Press Tab to reformat the script.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Once you have made changes to a script, you can save it in two ways:

- To save a script without closing its script window, choose Save Script from the File menu (or press ⌘-S).
- To save a script and close its script window, press Enter.

When you close an unsaved script's window by any method other than pressing Enter, HyperCard asks if you want to save the changes.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Text in the script windows appears in 9-point Monaco as the default. You can change the default font and size of the text in your script window using two properties:

the `scriptTextFont`  
the `scriptTextSize`

These two properties affect all script windows—you can't set the font and style of each script window separately.

For example, to set the font and size of the script windows to 10-point Geneva, type the following messages into the Message box, and press Return:

```
set the scriptTextFont to "Geneva"
```

```
set the scriptTextSize to 10
```

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Commands sent automatically

Card 1 of 1

HyperCard sends some commands automatically in response to events, just as it sends system messages. But unlike system messages, if the command passes all the way to HyperCard, HyperCard performs a built-in action. So if you trap the command, the action won't happen.

HyperCard sends `choose` and `doMenu` as messages when the user chooses a tool or an item from one of HyperCard's menus. It sends `close` as a message when the user clicks the close box of a window.

HyperCard sends `help` as a message when the user chooses Help from the Go menu (or presses ⌘-?).

HyperCard sends `arrowKey`, `commandKeyDown`, `controlKey`, `enterInField`, `enterKey`, `functionKey`, `keyDown`, `returnInField`, `returnKey`, and `tabKey` as messages when the user presses a key. It automatically sends `errorDialog` and `appleEvent` messages.

[Click Related Topics](#) for more information about each command.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Events

- HyperCard startup
- Resume
- New stack
- New stack in a new window
- New background
- New card
- Delete stack
- Delete background
- Delete card
- Cut card
- Paste card

For some events, HyperCard automatically sends a sequence of system messages.

For example, when you close an existing stack by opening another one, HyperCard sends the following system messages in order: `closeCard`, `closeBackground`, `closeStack`, `openStack`, `openBackground`, `openCard`.

Click an event to see the order in which HyperCard sends multiple system messages in response to it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## Messages sent to

Buttons  
Fields  
The current card

HyperCard automatically sends system messages and a few commands to one of three objects: a specific button, a specific field, or the current card. If none of these objects handles the message, it continues along the message-passing order until it reaches HyperCard itself.

The commands HyperCard sends automatically deal with keyboard events and menu events.

Click an object to see the messages it can receive from HyperCard.

---- End of Topic ----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



# appleEvent

Card 1 of 2

appleEvent *class, id, sender*

Handler:

```
on appleEvent class, id, sender  
    statements  
end appleEvent
```

HyperCard sends the `appleEvent` message to the current card when it receives an Apple event from another program. It sends the message at idle time as soon as all pending handlers have finished running.

*class* is the general category of the event (such as `aevt` or `misc`), *id* is the actual event received (such as `odoc`, `pdoc`, `dosc`, or `eval`), and *sender* is the name of the application or process that sent the event.

Because Apple event commands are usually generated by other processes, you may want to check to make sure that they are not destructive. Click Examples to see a handler, written for the Home stack, that intercepts an incoming Apple event.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



You'll see lots of `appleEvent` messages in the message watcher when HyperCard is being controlled by AppleScript from another application.

Apple events generated internally when HyperCard runs AppleScript don't produce `appleEvent` messages.

Use the `request` command within an `appleEvent` handler to obtain additional information about the event.

You don't have to define an `appleEvent` handler to enable HyperCard to handle Apple events. Use an `appleEvent` handler only if you need to get a peek at the incoming Apple events or to handle them yourself.

The `appleEvent` message occurs only under System version 7.0 and later.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## closeBackground

closeBackground

Handler:

```
on closeBackground  
  statements  
end closeBackground
```

HyperCard sends the `closeBackground` message to the current card when a user (or handler) quits HyperCard, goes to a card whose background is different from the background of the current card, and when a background or stack is deleted.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## closeCard

closeCard

Handler:

```
on closeCard  
  statements  
end closeCard
```

HyperCard sends the closeCard message to a card when a user (or script) goes to another card; deletes a card, background, or stack; or quits HyperCard.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**closeField**

`closeField`

Handler:

```
on closeField
  statements
end closeField
```

HyperCard sends the `closeField` message to an unlocked field when, after editing, a user (or handler) performs an action that closes (that is, removes the insertion point from) the field.

HyperCard sends `closeField` only when the text actually changes.

The following actions close a field, saving any changes that were made to the text:

- Clicking outside the field
- Moving the insertion point to the next field with the Tab key
- Pressing the Enter key
- Pressing ⌘-Shift-Z to revert the field to the last saved version
- Going to another card
- Quitting HyperCard

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## closeStack

closeStack

Handler:

```
on closeStack  
  statements  
end closeStack
```

HyperCard sends the `closeStack` message to the current card when a user (or script) opens a different stack in the current window, closes the current window, deletes the stack, suspends HyperCard to launch an application, or quits HyperCard.

Note: If you have more than one stack open at a time, HyperCard sends `suspendStack`, not `closeStack`, when the stack becomes inactive.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## deleteBackground

deleteBackground

Handler:

```
on deleteBackground  
    statements  
end deleteBackground
```

HyperCard sends the `deleteBackground` message to the card that is being deleted if no other cards in the stack share its background. HyperCard sends the message just before the card disappears.

Note: You cannot stop a background from being deleted by trapping the `deleteBackground` message. Instead, you must handle the `doMenu` message or set the `cantDelete` property for the background.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## deleteButton

deleteButton

Handler:

```
on deleteButton  
    statements  
end deleteButton
```

HyperCard sends the `deleteButton` message to a button that is being deleted just before the button disappears.

Note: You cannot stop a button from being deleted by trapping the `deleteButton` message. Instead, you must handle the `doMenu` message.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## deleteCard

Card 1 of 1

deleteCard

Handler:

```
on deleteCard  
    statements  
end deleteCard
```

HyperCard sends the `deleteCard` message to a card that is being deleted just before the card disappears.

Note: You cannot stop a card from being deleted by trapping the `deleteCard` message. Instead, you must handle the `doMenu` message or set the `cantDelete` property for the card.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## deleteField

deleteField

Handler:

```
on deleteField  
    statements  
end deleteField
```

HyperCard sends the `deleteField` message to a field that is being deleted just before the field disappears.

Note: You cannot stop a field from being deleted by trapping the `deleteField` message. Instead, you must handle the `doMenu` message.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## deleteStack

deleteStack

Handler:

```
on deleteStack  
  statements  
end deleteStack
```

HyperCard sends the `deleteStack` message to the current card of the stack that is being deleted.

Note: You cannot stop a stack from being deleted by trapping the `deleteStack` message. Instead, you must handle the `doMenu` message or set the `canDelete` property for the stack.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## errorDialog

errorDialog *errorMessageText*

Handler:

```
on errorDialog whatText
  statements
end errorDialog
```

HyperCard sends the `errorDialog` message and its text to the current card if it encounters an error when the `lockErrorDialogs` property is set `true`.

(In such a case, the ordinary error dialog box is not displayed.)

*errorMessageText* is the contents of the error dialog box that would be displayed if the `lockErrorDialogs` property were `false`.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## exitField

exitField

Handler:

```
on exitField  
  statements  
end exitField
```

HyperCard sends the `exitField` message to an unlocked field when, after clicking in a field or tabbing to it, a user (or handler) removes the insertion point from the field without changing any of its text.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**idle****idle****Handler:****on idle***statements***end idle**

HyperCard repeatedly sends the `idle` message to the current card when no other events are occurring (that is, when all handlers have finished running and HyperCard itself isn't sending other messages).

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# mouseDoubleClick

Card 1 of 2

mouseDoubleClick

Handler:

```
on mouseDoubleClick  
  statements  
end mouseDoubleClick
```

HyperCard sends this message to a button, field, or card after the mouse is clicked twice at the same place. When the message is sent, it's the only one sent during the second click; there's no mouseDown, mouseStillDown, or mouseUp message sent after the first mouseUp.

For HyperCard to send this message, the following conditions must all be true:

- (a) The downstroke of a second click follows the downstroke of a previous click within the double-click speed set in the Mouse control panel; and
- (b) the second click occurs within four pixels of the first; and
- (c) the second click occurs within the same object as the first.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

If the user clicks repeatedly at the same location faster than the double-click speed set in the Mouse control panel, HyperCard treats each odd-numbered click as a first click and each even-numbered click as a second click.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## mouseDown

mouseDown

Handler:

```
on mouseDown  
    statements  
end mouseDown
```

HyperCard sends the mouseDown message to a button or to a locked field when the user presses the mouse button down and the pointer is inside the rectangle of the button or field.

HyperCard sends mouseDown to the current card when the user presses the mouse button down and the pointer is not in the rectangle of a button or field.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## mouseEnter

mouseEnter

Handler:

```
on mouseEnter  
    statements  
end mouseEnter
```

HyperCard sends the mouseEnter message to a button or field just after the pointer moves within its rectangle.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## mouseLeave

mouseLeave

Handler:

```
on mouseLeave  
  statements  
end mouseLeave
```

HyperCard sends the mouseLeave message to a button or field just after the pointer moves outside its rectangle.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## mouseStillDown

mouseStillDown

Handler:

```
on mouseStillDown  
    statements  
end mouseStillDown
```

HyperCard repeatedly sends the `mouseStillDown` message to a button or to a locked field while the user holds the mouse button down and the pointer is inside the rectangle of the button or field.

HyperCard sends `mouseStillDown` to the current card when the user holds the mouse button down and the pointer is not in the rectangle of a button or field.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# mouseUp

mouseUp

Handler:

on mouseUp

statements

end mouseUp

HyperCard sends the mouseUp message to a button or locked field when the user releases the mouse button and the pointer is inside the rectangle of the same button or field it was in when the user pressed the mouse button.

HyperCard sends mouseUp to the current card when the user both presses *and* releases the mouse button while the pointer is not in the rectangle of a button or field.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## mouseWithin

mouseWithin

Handler:

```
on mouseWithin  
  statements  
end mouseWithin
```

HyperCard sends the mouseWithin message to a button or field repeatedly while the pointer is inside its rectangle.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## moveWindow

moveWindow

Handler:

on moveWindow

*statements*

end moveWindow

HyperCard sends the moveWindow message to the current card when the user or a script moves the window. In many cases, HyperCard will send a sizeWindow message immediately before a moveWindow message.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## newBackground

newBackground

Handler:

```
on newBackground  
  statements  
end newBackground
```

HyperCard sends the newBackground message to the current card (in this case, the first card of the new background) just after HyperCard creates the background.

Although the new background will not have a script with which to respond to the message, any other object along the message-passing path can handle it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## newButton

newButton

Handler:

```
on newButton  
    statements  
end newButton
```

HyperCard sends the newButton message to a button just after HyperCard creates it.

Although the new button will not have a script with which to respond to the message, any other object along the message-passing path can handle it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## newCard

newCard

Handler:

```
on newCard  
    statements  
end newCard
```

HyperCard sends the newCard message to the current card (in this case, the new one) just after HyperCard creates it.

Although the new card will not have a script with which to respond to the message, any other object along the message-passing path can handle it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## newField

newField

Handler:

```
on newField  
    statements  
end newField
```

HyperCard sends the newField message to a field just after HyperCard creates it.

Although the new field will not have a script with which to respond to the message, any other object along the message-passing path can handle it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



newStack

Handler:

on newStack

*statements*

end newStack

HyperCard sends the newStack message to the current card (in this case, the first card of the new stack) just after HyperCard creates the stack.

Although the new stack will not have a script with which to respond to the message, any other object along the message-passing path can handle it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# openBackground

openBackground

Handler:

```
on openBackground  
    statements  
end openBackground
```

HyperCard sends the `openBackground` message to the current card just after a user (or script) goes to a card whose background differs from the background of the most recent card.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## openCard

openCard

Handler:

```
on openCard  
    statements  
end openCard
```

HyperCard sends the `openCard` message to the current card just after a user (or handler) goes to the card.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# openField

Card 1 of 1

openField

Handler:

```
on openField  
  statements  
end openField
```

HyperCard sends the `openField` message to an unlocked field when a user (or handler) first opens it for text editing. A user opens a field for editing either by clicking in the field or by tabbing from the previous field.

Once opened for editing, a field no longer receives `openField` messages.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# openStack

Card 1 of 1

openStack

Handler:

on openStack

statements

end openStack

HyperCard sends the openStack message when a user (or handler) goes to a card in a stack different from that of the most recent card.

Note: If you have more than one stack open at a time, HyperCard sends resumeStack, not openStack, when the stack becomes active.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**quit**

quit

Handler:

on quit

statements

end quit

HyperCard sends the quit message to the current card when the user chooses Quit HyperCard from the File menu (or presses ⌘-Q), just before HyperCard quits.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

resume

Handler:

on resume

statements

end resume

HyperCard sends the resume message to the current card when HyperCard resumes running after the user quits an application launched from HyperCard.

Note: HyperCard does not send this message when it's running under MultiFinder.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## resumeStack

resumeStack

Handler:

```
on resumeStack  
  statements  
end resumeStack
```

HyperCard sends the resumeStack message to the current card when the stack's window becomes active after being inactive (for example, when the user clicks in a window).

HyperCard sends resumeStack only when it is displaying more than one stack.

**Important:** HyperCard does not send resumeStack when the user switches to HyperCard from another layer under MultiFinder. (Click Related Topics for more information about the property the suspended.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## sizeWindow

sizeWindow

Handler:

```
on sizeWindow
  statements
end sizeWindow
```

HyperCard sends the `sizeWindow` message to the current card when the user or a script resizes the window. In many cases, HyperCard will send a `moveWindow` message immediately after a `sizeWindow` message.

Here are the ways a window can be resized:

- The user resizes the window using the size box.
- The user resizes the window using the Scroll window.
- The user clicks the zoom box, and the zoom changes the window size.
- A handler sets the `rect` of the card window to a new size.
- A handler resizes all the cards in a stack by setting the `rect` of the card to a new size.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

## startUp

startUp

Handler:

```
on startUp  
    statements  
end startUp
```

HyperCard sends the `startUp` message to the first card displayed when HyperCard is first started.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# suspend

suspend

Handler:

```
on suspend  
  statements  
end suspend
```

HyperCard sends the `suspend` message to the current card when a user (or handler) launches an application from HyperCard with the `open` command, just before the application is launched.

Note: HyperCard does not send this message when it's running under MultiFinder.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## suspendStack

suspendStack

Handler:

```
on suspendStack  
    statements  
end suspendStack
```

HyperCard sends the `suspendStack` message to the current card when the stack's window becomes inactive (for example, when the user clicks another card window).

HyperCard sends `suspendStack` only when it is displaying more than one stack.

**Important:** HyperCard does not send `suspendStack` when the user switches to another layer under MultiFinder. (Click [Related Topics](#) for more information about the `the suspended`.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**do**

do *expression* [as *scriptLanguage*]

The do keyword forces HyperCard to evaluate *expression* and to send the result as a message to the current card.

The value of *expression* can contain more than one line. For example, if you have a series of statements in a card field called Example, HyperCard will apply do to each line:

```
do card field "Example"
```

When you use the as *scriptLanguage* form, HyperCard executes the script in *expression* using the OSA-compliant scripting component named in *scriptLanguage*:

```
do field 1 as AppleScript  
do theScript as UserTalk
```

You can also send do from the Message box.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

exit repeat  
exit functionName  
exit messageName  
exit to HyperCard

The `exit` keyword interrupts the current flow of control.

`Exit repeat` sends control to the end of a `repeat` structure, ending execution of the loop regardless of the state of the controlling conditions.

The `exit functionName` and `exit messageName` forms stop the current message or function handler. Control returns to any pending statements from another handler, if any.

`Exit to HyperCard` terminates all running or pending handlers and cancels all pending messages.

Using `exit` to leave a function handler sets the value of the function to empty.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## function

```
function functionName [parameterList]
  statements
end functionName
```

The function keyword defines a new function handler of the specified name. You call a function by placing parentheses after its name, enclosing any parameters within the parentheses:

```
get deleteSpaces(" hello ")
```

The optional parameterList lets a function handler receive values sent along with the function call.

When a function is called, HyperCard evaluates each item within the parenthetical list following the function's name. When the handler begins to execute, HyperCard assigns each value to a parameter variable in the parameterList.

Use the return keyword within the function definition to have the function return a value to the handler that called it. If you don't use return, the function evaluates to empty.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## global

global variableList

The **global** keyword makes a variable and its contents available to any handler in HyperCard. Changing the value of a global variable in any handler changes its value everywhere.

Note: You must use the **global** keyword in **each handler** to declare the global variables you want to use.

Global variables are not saved between sessions of HyperCard. Global variables are also lost under System 6's single Finder when a user (or handler) suspends HyperCard by launching another application with the open command.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## if (multiple-statement)

```
if trueOrFalse then  
  statements  
[else if trueOrFalse then  
  statements]  
[else  
  statements]  
end if
```

The multiple-statement `if` structure tests the specified condition and executes **one or more** statements if the condition is true. You use the optional `else if` or `else` form to run alternative blocks of code in case the condition following `if` is false.

Because each part of a complex `if` structure may contain more than one statement, you must have an `end if` statement at the end of the structure.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## if (single-statement)

```
if trueOrFalse then statements ~
[else statements]
```

```
if trueOrFalse
then statements
[else if trueOrFalse
then statements]
[else statements]
```

The single-statement `if` structure tests for a condition and executes **one** statement if the condition is true. You use the optional `else if` or `else` form to run other blocks of code in case the condition following `if` is false.

Because each part of a simple `if` structure is limited to one statement, you don't need an `end if` statement.

You can send a one-line `if` structure from the Message box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**on**

```
on messageName [parameterList]  
  statements  
end messageName
```

The **on** keyword defines a new message handler of the specified name.

The optional parameterList lets the handler receive values sent along with a message. HyperCard assigns each value to a parameter variable in the parameterList.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## pass

pass functionName

pass messageName

The `pass` keyword ends execution of the current handler and sends the entire message that initiated execution of the handler to the next object in the message-passing order.

(Ordinarily, once a message is handled, it does *not* continue along the message-passing order.)

In general, a stack should pass any system messages that it handles so that other stacks later in the message-passing order also get a chance to handle the message.

For example, a `mouseWithin` handler in your Home stack won't ever run if you also have a `mouseWithin` handler without a `pass` statement in a stack before Home in the message-passing path.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## repeat

```
repeat [forever]
  statements
end repeat
```

The statements in a repeat forever structure repeat continuously.

If HyperCard executes an exit repeat statement in the loop, it continues running the handler starting from the first statement after end repeat.

If HyperCard executes a next repeat statement, it returns immediately to the beginning of the repeat loop.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## repeat for

```
repeat [for] pos/integer [times]  
  statements  
end repeat
```

The statements in a repeat for structure repeat for a specified number of times.

If HyperCard executes an exit repeat statement in the loop, it continues running the handler starting from the first statement after end repeat.

If HyperCard executes a next repeat statement, it returns immediately to the beginning of the repeat loop.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## repeat until

```
repeat until trueOrFalse
  statements
end repeat
```

The statements in a repeat until structure repeat as long as the condition following the word until is false. HyperCard checks the condition before the first and any subsequent iterations of the loop.

If HyperCard executes an exit repeat statement in the loop, it continues running the handler starting from the first statement after end repeat.

If HyperCard executes a next repeat statement, it returns immediately to the beginning of the repeat loop.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## repeat while

```
repeat while trueOrFalse
  statements
end repeat
```

The statements in a repeat while structure repeat as long as the condition following the word `while` is true. HyperCard checks the condition before the first and any subsequent iterations of the loop.

If HyperCard executes an `exit repeat` statement in the loop, it continues running the handler starting from the first statement after `end repeat`.

If HyperCard executes a `next repeat` statement, it returns immediately to the beginning of the repeat loop.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## repeat with

```
repeat with variableName = integer1 to integer2
  statements
end repeat
```

```
repeat with variableName = integer1 down to integer2
  statements
end repeat
```

The statements in a repeat with structure repeat until a variable with an initial value of integer1 is greater than (or, in the case of down to, less than) the number integer2.

The value of the variable increases (or decreases) by 1 during each iteration of the repeat loop.

If HyperCard executes an exit repeat statement in the loop, it continues running the handler starting from the first statement after end repeat. If HyperCard executes a next repeat statement, it returns immediately to the beginning of the repeat loop and increases (or decreases) the value of the variable.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## return

return *expression*

The return keyword ends execution of a handler and, in function handlers, returns the value of *expression* to the handler that called the function.

If return appears in a message handler (as opposed to a function handler), it ends execution of the handler and places the value of the expression into the HyperTalk function the result.

The value of the result as set by a return statement is valid only immediately after the return statement executes; each new statement resets the result to empty.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**send**

```
send "messageName [parameterList]"-  
[to object]
```

```
send "messageName [parameterList]"-  
to HyperCard
```

```
send expression to { program -  
program | program id programID | -  
this program } [without reply]
```

In the first two forms, the `send` keyword sends a message directly to a particular object or to HyperCard. For example, you can send a message to an object already passed by in the message-passing order (from a stack back to the

current card), or you can bypass handlers later in the message-passing order that might otherwise handle the message.

You can send messages to any object in the current stack, and you can send messages to another stack (but not to objects within another stack).

**Important:** If you send a message to a card other than the current card, HyperCard doesn't go to the card or open it.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



For example, if you send a message to another stack, and the handler refers to a field that's specific to that stack, you'll get a script error.

HyperCard evaluates any parameters before it sends the message, even though the entire message is in quotation marks. (You don't need quotation marks if the message is a single word.)

When an object receives a message from `send`, HyperCard sets the value of the `target` to the name of the object.

If the object doesn't handle the message, the message continues along the message-passing path from that point.

If you send a message directly to HyperCard, you ensure that no other objects will handle the message. For example,

`send "doMenu next" to HyperCard`  
always takes you to the next card.

You can type `send` as a message in the Message box.

--- More ---

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

The third form sends a do script Apple event from HyperCard to another running application:

```
send expression to { program ~  
    program | program id programID | ~  
    this program } [without reply]
```

where *program* is the path name to the target program in the form

*zone:computer:program*, and *programID* is the signature of a program on the same computer. *this program* denotes HyperCard.

*expression* is any valid expression or any sequence of commands in the scripting language supported by the target program. If the target program is HyperCard, the scripting language is HyperTalk.

By default, HyperCard waits for a reply from the target program before continuing; but you can specify *without reply* if you don't want to wait for one.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Any reply from the target program goes into the result.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(f405b1e8bd04ee8aed9e864a3e5b89df\_img.jpg\) Related Topics](#)[Find Topic](#)[Main Topics](#)

**add**

add number to [chunk of] container

Note: The container or chunk referred to must contain a number.

The add command adds the value of number to the number in a container or chunk and replaces the contents of the container or chunk with the result.

(You can use the is a operator to see if the container is a number.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



answer *text*

answer *text* with *reply1*

answer *text* with *reply1* or *reply2*

answer *text* with *reply1* or *reply2* -  
or *reply3*

answer file *text* [of type *fileType*]

answer program *text* [of type -  
*processType*]

The form answer *text* displays a  
**question** (*text*) in a dialog box. The  
dialog box contains from one to three  
buttons, each representing a different  
reply the user can select.

If you use answer without specifying any replies, HyperTalk displays an OK button as the default. Otherwise, the last reply you specify becomes the default button. (Pressing Enter or Return chooses the default button.) Answer returns the name of the button clicked by the user in the local variable *it*.

Answer automatically sizes the dialog box to fit the size of the text (up to 13 lines). The total length of the text cannot exceed 254 characters.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics

The form Answer file *text* displays a directory dialog box that you use to select a file.

answer file *text* of type *fileType* displays only files of a specified type: application, picture, painting or paint, stack, or text. (Click the *fileType* placeholder for more information about file types.)

Answer file returns the full pathname of the selected file in the local variable *it*. It returns empty if the user clicks Cancel.

Answer file also sets the result to Cancel if the user clicks Cancel. Check the value of the result in the statement immediately following answer file.

The form Answer program *text* produces a dialog box of all System 7-friendly processes currently running on the local machine and on any networked computers to which the local machine has access.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

A System 7-friendly process is a program or entity that's capable of exchanging information with another process. The name of the selected program is placed in the container it.

*text* is a quoted prompting string that appears at the top of the dialog box. If you provide the null string for *text* (that is, ""), the system puts Choose a program to link to: at the top of the dialog box.

Use the form `answer program text` of type `processType` to see only certain types of processes (spell checkers, word processors, spreadsheets, and so on).

An application's default process type is its creator. So to see only copies of HyperCard, you'd use `WILD`.

For more details, see "PPC Toolbox" in *Inside Macintosh V.6* or *Inside Macintosh: Interapplication Communications*.

---- End of Topic ----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



◀



Home



## arrowKey

arrowKey *direction*

If the global property `textArrows` is `false`, the `arrowKey` command navigates through cards:

`arrowKey left` = go to previous card

`arrowKey right` = go to next card

`arrowKey up` = go forward through recent cards

`arrowKey down` = go backward through recent cards

If the global property `textArrows` is `true`, the `arrowKey` command navigates through cards unless the insertion point is in a field. Then `arrowKey` moves the insertion point within the field.

HyperCard sends the `arrowKey` command to the current card when an arrow key is pressed. The value passed to the parameter variable *direction* is `left`, `right`, `up`, or `down`, depending on which arrow key is pressed.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics

To handle the arrowKey message, use the following form:

```
on arrowKey whichKey  
    statements  
end arrowKey
```

In the above form, the parameter variable whichKey is set to a direction.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**ask**

`ask text1 [with text2]`

`ask password [clear] text1 [with text2]  
]`

`ask file text1 [with fileName]`

The `ask` command displays a **question** (text1) in a dialog box along with a text box where the user can type a reply. The `ask` dialog automatically provides the OK and Cancel buttons.

You can supply a **default reply** using the `with text2` option. The default text appears highlighted in the text box.

Ask returns the text entered by the user, if any, in the local variable `it`. If the user clicks Cancel, ask places empty in `it`.

Ask also sets the HyperTalk function the result to empty if the user clicks OK or to Cancel if the user clicks Cancel. (So you can use the value of the result to determine whether the user provides an empty string or clicks Cancel.) You must check the result in the statement immediately after the `ask` command.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

Note: `ask` automatically sizes the dialog box to fit the size of the text (up to thirteen lines). The total length of the text cannot exceed 254 characters for both the prompt and the default reply.

`Ask password` displays a bullet (\*) for each character the user types and encrypts the reply as a number. You can save this number in a field to compare with future passwords. (The `ask password clear` form does not encrypt the reply, but it does display a bullet for each typed character.)

Note: `ask password` is different from setting the password of the stack with the Protect Stack dialog. With `ask password`, your handler must set and check any passwords.

`Ask file` displays a directory dialog box in which you type the name of a file. The `with filename` option provides a default name that appears in the text box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## beep

beep

beep *pos/integer*

The beep command sounds  
the Macintosh system beep.

If you specify a *pos/integer*, your  
Macintosh beeps that many times.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## choose

```
choose tool pos/integer
choose toolName tool
```

where *pos/integer* is between 1 and 18.

The choose command chooses the tool with the specified number or name from the Tools palette.

HyperCard sends the choose command to the current card when you choose a tool from the Tools menu.

HyperCard passes the word tool to the first parameter variable and the tool number to the second parameter variable. You can handle the choose command as follows:

```
on choose what, toolNumber
  statements
end choose
```

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



click at *point*

click at *point* with *key1*

click at *point* with *key1*, *key2*

click at *point* with *key1*, *key2*, *key3*

The click at command clicks the mouse from within scripts. It acts exactly as if the user had clicked the mouse on the screen.

If *point* is within the rectangle of a button, a locked field, or anywhere else on the card, HyperCard sends the mouseDown, mouseStillDown, and mouseUp messages to the object.

If *point* is within the rectangle of an unlocked field, HyperCard sets the insertion point in the field.

The with *key* options specify combinations of the commandKey, the optionKey, and the shiftKey, just as if the user were holding down the key or keys while clicking the mouse.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



`close printing`

`close file fileName`

`close [document { in | with } ] -  
application`

`close { externalWindow | card window }`

The `close printing` command ends a print job previously begun with an `open printing` command.

The `close file` command closes a disk file previously opened with the `open file` command. Usually, you open files to import or export text. Always use `close file` when you're finished.

If you try to close an unopened file, the result gets `File not open`.

HyperCard automatically closes all open files when

- it runs an `exit to HyperCard` statement,
- the user presses `⌘-.` (`⌘-period`), or
- the user quits HyperCard.

You must provide the full path name of the file if it's not at the same directory level as HyperCard.

[--- More ---](#)

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

The `close [document {in|with}] application` command closes the named running document, application, or desk accessory. (The words `in` and `with` are synonymous.)

**Note:** This command works only with Apple event-aware applications running under System 7 on the same Macintosh as HyperCard.

If the document or application isn't running, the result is set to No such document or No such application as appropriate.

The form `close application` sends a quit Apple event, while the form `close document {in|with} application` sends a clos Apple event.

All Apple event-aware applications support the quit Apple event, but they don't all support clos.

The `close externalWindow` command closes an external window—a palette or other window displayed by an `external command` or `external function`—and removes it from the window list.

--- More ---

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**close** (cont'd)

Thus you can't show an external window once you've closed it; you'll have to create a new one using its external command or external function.

The `close card window` command closes the frontmost stack if at least two stacks are open.

HyperCard also sends the `close` command to the current card when the user clicks the close box of the card window. You can handle the message as follows:

```
on close  
statements  
end close
```

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## commandKeyDown

Card 1 of 1

commandKeyDown *char*

The commandKeyDown command provides a way of sending a ⌘-key event from a handler. It acts exactly as if you had pressed ⌘ at the same time as the specified character.

HyperCard sends the commandKeyDown command to a field (if the insertion point is set) or to the current card when the user presses a ⌘-key combination. The value passed to the parameter variable *char* corresponds to the key pressed.

You can handle the commandKeyDown message as follows:

```
on commandKeyDown theKey  
    statements  
end commandKeyDown
```

Note: commandKeyDown is not sent for characters typed using the type command.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## controlKey

`controlKey pos/integer`

The `controlKey` command has no built-in effect. HyperCard sends the `controlKey` command to the current card when a combination of the Control key and another key is pressed.

You can handle the `controlKey` message as follows:

```
on controlKey theKeyNumber
  statements
end controlKey
```

HyperCard passes the following numbers for each control key combination:

<b>Value</b>	<b>Key Pressed</b>
1	a, home
2	b, Enter
3	c
4	d, end
5	e, help
6	f
7	g
8	h, delete
9	i, tab
10	j

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



```
convert {value|container}[from -  
formatName [and formatName]] -  
to formatName [and formatName]
```

The convert command changes a value expressed as a valid date, time, or date and time format to another format.

You use and formatName in combination with the first formatName to convert a value to any two formats (often the date and time).

The form convert value to formatName returns the converted value in the local variable it. The form convert container to formatName converts a value in a chunk or container (including variables) and places the result in that chunk or container.

You use the form from format in situations where you don't want HyperCard to do the conversion automatically.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



There are four types of date formats:

- seconds, a positive integer equal to the number of seconds since 12:00 midnight on January 1, 1904
- dateItems, a comma-delimited list of seven positive integers equal to the following values:

year, month, day, hour, minute, second, dayNumber

where dayNumber 1 = Sunday and  
7 = Saturday.

- date, which has one of three formats:

[dayName,] monthName, day, year  
month/day/year  
month-day-year

where dayName = Sunday, Sun, Monday, Mon, Tuesday, Tue, Wednesday, Wed, Thursday, Thur, Friday, Fri, Saturday, or Sat

--- More ---

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



monthName = January, Jan,  
February, Feb, March, Mar, April,  
Apr, May, June, Jun, July, Jul,  
August, Aug, September, Sep,  
October, Oct, November, Nov,  
December, Or Dec

- time, which has the following format:

hour:minute[:second] [timeOfDay]

where timeOfDay = am Or pm

You can precede the format names date and time with an optional *adjective*, producing the following formats:

abbrev date Fri, Jun 15, 1990

long date Friday, June 15, 1990

short date 6/15/90

abbrev time 3:30 PM

long time 3:30:00 PM

short time 3:30 PM

--- More ---

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



Note: HyperCard can handle dates from 1/1/1000 to 12/31/9999 in all formats. It handles dates from 1/1/1 to 12/31/9999 only in the date|items or seconds format. If you try to convert an invalid date (such as "Friday, May 50, 1990"), HyperCard sets the HyperTalk function the result to "Invalid date."

When System 7.1 is running, HyperCard uses the date and time settings from the Date & Time Control Panel.

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## copy template

```
copy template templateName ~  
    to stack
```

The `copy template` command makes a copy of printing report template *templateName* from the current stack and moves it into the stack *stack*.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



create menu *menuName*

create stack *fileName* [with *bknd*] ~  
[in a new window]

The `create menu` command makes a new menu and adds it to the menu bar. HyperCard displays an error message if you try to create a menu that already exists.

Use the `put` command to add menu items to the new menu. (Click [Related Topics](#) for more information about `put`.)

The `create stack` command creates a new stack from within a handler without presenting the New Stack dialog box. The cards in the new stack are the same size as the cards in the current stack.

HyperCard sets the function `the result` to "Couldn't create stack." if it can't create the stack; otherwise, it sets `the result` to `empty`, goes to the new stack, and sends a `newStack` message to the only card in that stack (that is, the current card).

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## debug

`debug checkPoint`

The `debug checkPoint` command sets a permanent checkpoint in a HyperTalk handler. When HyperCard runs a handler, the `debug checkpoint` command causes HyperCard to enter the debugger; it pauses execution of the handler and opens a script editor window with a box around the line with the checkpoint.

This command works only when the `userLevel` property is set to 5.

Click Related Topics for information about debugging scripts.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
delete chunk of container
delete [menu/item of] menu
delete [menu/item from] menu
delete {button | field | part}
```

The delete command removes text from a container, menu items from a menu, menus from the menu bar, and buttons or fields from the current card or background.

When you use the form `delete part`, `deleteButton` or `deleteField` is sent to the object that's being deleted.

You can't use this command to delete a *part* anywhere except on the current card.

Note: Using `delete` to delete a line is not the same as putting `empty` into the line: `delete` removes the final return character as well as the text, while putting `empty` into the line just removes the text.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# dial

dial *pos/integer*

dial *pos/integer* with modem →  
[modemCommands]

The dial command generates touch-tone sounds for the digits in *pos/integer* through the Macintosh speaker. To dial the phone from HyperCard, you must either hold the handset up to the speaker of your Macintosh or use a device that feeds Macintosh audio output to the telephone.

If you use the with modem option, HyperCard sets up calls using the modem connected to the modem port. For more information about using modems with HyperCard, see the Phone Dialer stack. See your modem manual for valid *modemCommands*.

Note: If you include a hyphen in the number, place the entire expression in quotation marks. Otherwise, HyperCard performs a subtraction before dialing the number.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)

## disable

disable menu  
disable menu/item of menu  
disable button

The disable command dims menu items, entire menus, and buttons. It's a shortcut for setting the enabled property of a menu item, menu, or button to false.

Users cannot choose dimmed menu items.

Disabled buttons don't receive mouseDown, mouseStillDown, mouseUp, or mouseDoubleClick messages when you click them.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



divide [*chunk of*] *container* by *number*

The `divide` command divides the number in the container or chunk by *number* and puts the result into the container.

Dividing by 0 places `INF` (for infinity) into the chunk or container. Division is carried out to a precision of up to 19 decimal places.

Note: The container or chunk referred to must contain a number.

If the result is put into a field or the Message box, it is displayed according to the global property `numberFormat`.

(You can use the `is a` operator to see if the container is a number.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

doMenu /itemName [without dialog] ~  
[with *keys* ]

doMenu /itemName, menuName ~  
[without dialog] [with *keys* ]

The doMenu command performs the action specified by the item name and menu name just as if the user chose the item directly from the menu.

without dialog bypasses the dialog box that would normally appear after the commands Delete Stack and Convert Stack, and, when a background field is selected, after Cut Field and Clear Field.

with *keys* chooses the named menu command with the shift, option, and/or ⌘ keys pressed.

To determine from a script which keys were specified, look at param(6) of the original command.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



HyperCard sends the `doMenu` command as a message to the current card when the user selects a menu item. `/itemname` is the exact name of the menu item selected, and `menuname` is the exact name of the menu that contains the menu item. To handle the `doMenu` message, use this form:

```
on doMenu theItem, theMenu  
    statements  
end doMenu
```

Note: A `doMenu` handler can override a `menuMessage`.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## drag

```
drag from point to point
drag from point to point with key1
drag from point to point -
    with key1, key2
drag from point to point -
    with key1, key2, key3
```

The drag command simulates the user dragging the mouse manually (except that you must use the with shiftKey option in order to select text in a field).

The with *key* options specify combinations of the commandKey, the optionKey, and the shiftKey, simulating the user holding down the key or keys while dragging.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## edit script

edit [the] script of object

The `edit script` command opens the script of an object with the HyperCard script editor.

When run as a statement in a handler, `edit script` suspends execution of the handler until the user closes the script editor or activates the card window.

Note: Even though HyperCard itself is an object (it can receive messages), it does not have a script. The following statement yields an error:

`edit the script of HyperCard`

For this command to work, the `userLevel` property must be set to 5.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## enable

```
enable menu
enable menu/item of menu
enable button
```

The `enable` command activates menu items, entire menus, or buttons that are inactive (dimmed). It's a shortcut for setting the `enabled` property to true.

The `enable` command enables only items in HyperCard's menus that are currently available to the user. For example, the following command will not enable the Button Info command in the Objects menu unless a button is currently selected:

```
enable menuitem 1 of menu "Objects"
```

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## enterInField

enterInField

HyperCard sends the enterInField command to a field when the user presses Enter and the insertion point is in the field.

This command saves the results of any changes the user or a handler makes to a field and closes the field.

HyperCard sends a closeField message if the user made changes to the text or sends exitField if the user did not make any changes).

You can handle the enterInField message as follows:

```
on enterInField  
    statements  
end enterInField
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## enterKey

enterKey

The `enterKey` command sends a statement typed into the Message box to the current card.

HyperCard sends the `enterKey` command to the current card when the user presses the Enter key unless the insertion point is in a field, in which case HyperCard sends `enterInField` instead.

You can handle the `enterKey` message as follows:

```
on enterKey  
    statements  
end enterKey
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## export paint

export paint to file *fileName*

The `export paint` command saves a Paint image of the current card to the specified file. `Export paint` works only when one of the Paint tools is chosen.

This command has the same effect as the `Export Paint` menu item that appears in the `File` menu (when a Paint tool is chosen), except that it avoids the dialog box that prompts the user for a file name.

If `export paint` succeeds, HyperCard sets the function `the result` to empty; if the command fails (if, for example, you use `export paint` when the Browse tool is chosen), HyperCard sets the result to "Couldn't export paint."

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**find**

```
find [international] text ~
    [in field] [of marked cards]
find chars [international] text ~
    [in field] [of marked cards]

find word [international] text ~
    [in field] [of marked cards]
find whole [international] text ~
    [in field] [of marked cards]
find string [international] text ~
    [in field] [of marked cards]
```

The **find** command searches for a text string in **all** the card and background fields (visible or not)

of the current stack.

International considers diphthongs and diacritical marks as it searches. In field searches only a specific card or background field. Of marked cards restricts its search to marked cards.

When **find** succeeds, a box appears around found text (or the first part of the text if the targets are discontinuous); the result is set to empty. When **find** fails, the result is set to the string "Not found".

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)

**find** (cont'd)

The commands `find`, `find chars`, and `find word` treat each word of the search string as a separate search item. For example,

```
find "this command"
```

is treated as `find "this"` **and** `find "command"`. The search succeeds if HyperCard finds all the words of the search string **anywhere** on the card (or anywhere in the specified field). The words do not have to be in order.

The commands `find whole` and `find string` treat spaces as part of the search string. For example,

```
find whole "this command"
```

succeeds only if HyperCard finds that exact string, including the space. The words must appear in order for `find whole` to succeed.

To prevent `find` from searching in a particular field, card, or background, see `dontSearch` in Related Topics.

[--- More ---](#)

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Here are the forms of `find` types:

<b>Command</b>	<b>Matches</b>
<code>find</code>	Whole or partial strings starting from the beginning of a word.
<code>find chars</code>	Partial strings anywhere within a word.
<code>find word</code>	Whole words only.
<code>find whole</code>	Whole or partial strings, including spaces, starting from the beginning of a word.
<code>find string</code>	Partial strings anywhere, including spaces (ignores word boundaries).

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## functionKey

functionKey *pos/integer*

Note: *pos/integer* must yield a number between 1 and 15.

HyperCard sends the functionKey command to the current card when the user presses one of the function keys on the Apple Extended Keyboard.

The functionKey command performs an undo, cut, copy, or paste for the values 1 through 4. Integer values 5 through 15 have no built-in effect.

You can handle the functionKey message as follows:

```
on functionKey whichKey  
    statements  
end functionKey
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**get**`get expression``get [the] property [of object]`

The `get` command puts the value of any expression or property into the local variable `it`.

That is,

`get expression`

is the same as

`put expression into it`

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(d18b807949d717b59192f9152e737c03\_img.jpg\) Related Topics](#)[Find Topic](#)[Main Topics](#)

**go**

`go [to] ordinal`

`go [to] position`

`go [to] card [of bkgnd] [of stack]`

`go [to] bkgnd [of stack]`

`go [to] stack`

`go [to] [card of] [bkgnd of] stack -  
[in a new window] [without dialog]`

The go command takes the user to a card in a stack. The ordinal and position forms take the user to a card in the current stack.

If you name a stack (or background) without specifying a card, HyperCard goes to the first card of the stack (or background).

HyperCard puts empty into the function the result when the go command succeeds; it puts "No such card." or "No such stack." into the result when it can't go to the card or stack.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics

**go (cont'd)**

The `in a new window` option tells HyperCard to open a stack in another window when it goes to the stack.

The `without dialog` option tells HyperCard to go to another stack directly based on the search paths that are specified on the Search Paths card of the user's Home stack. If HyperCard can't find the stack, it places "No such stack" into the result.

If you don't use `without dialog`, the result is set to `Cancel` if the user clicks the "Where Is" dialog box's `Cancel` button.

Note: The options `in a new window` and `without dialog` take effect only if the `go` command explicitly specifies a stack other than the current stack.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## help

help

The `help` command goes to the first card of the HyperCard Help stack.

HyperCard sends the `help` command to the current card when the user chooses Help from the Go menu (or presses `⌘-?`).

Note: The command `go help` is equivalent to `go stack "help"`, while the `help` command tries to execute `go stack "HyperCard Help"`.

You can handle the `help` message as follows:

```
on help  
    statements  
end help
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## hide

hide menuBar  
hide titleBar  
hide groups

hide card picture  
hide background picture  
hide picture of card  
hide picture of bkgnd

hide field  
hide button  
hide window  
hide externalWindow

The `hide` command removes HyperCard objects and elements from view.

`hide menuBar` removes the menu bar from the top of the screen.

`hide titleBar` removes the title bar on the card window.

Use both of these commands with care: hiding the menu bar, or the title bar of a window, may confuse your users.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**hide (cont'd)**

The `hide groups` command removes the two-pixel gray underline displayed for all text that has the "group" text style. The underline appears only after a `show groups` command.

For graphics, the `hide` command removes the card or background picture from view. It's the same as setting the `showPict` property of the card or background to false.

For buttons, fields, and windows, the `hide` command is equivalent to setting the `visible` property to false.

If you hide the frontmost document window, the next document window becomes active. `Hide` does not remove an external window from the window list (from memory); use the `close` command to dispose of the window.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## import paint

```
import paint from file fileName
```

The `import paint` command loads a Paint image to the current card from a file. `import paint` only works when one of the Paint tools is chosen.

This command has the same effect as the `Import Paint` menu item that appears in the `File` menu (when a Paint tool is chosen), except that it avoids the dialog box that prompts the user for a file name.

If `import paint` succeeds, HyperCard sets the function `the result` to empty; if the command fails (if, for example, you use `import paint` when the Browse tool is chosen), HyperCard sets the result to "Couldn't import paint."

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## keyDown

keyDown *char*

The keyDown command simulates a key press from within a handler. It acts exactly as if the user pressed a character from the keyboard.

HyperCard sends the keyDown command to a field (if the insertion point is set) or to the current card when the user presses a key. The value passed to the parameter variable *char* corresponds to the key pressed.

You can handle the keyDown message as follows:

```
on keyDown theKey  
    statements  
end keyDown
```

Note: keyDown is not sent for characters typed using the type command.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# lock

Card 1 of 1

lock {messages|recent|screen}

lock error dialogs

Lock messages has the same effect as set lockMessages to true: it prevents HyperCard from sending open, close, suspend, and resume system messages along the message-passing path.

Lock recent has the same effect as set lockRecent to true: it prevents HyperCard from keeping a visual record of cards visited by the user (or a handler) in the Recent Card dialog box.

Lock screen has the same effect as setting the property lockScreen to true: it prevents HyperCard from updating the screen.

Lock error dialogs prevents error dialog boxes from appearing; instead, the message `errorDialog errorMessage` is sent to the current card.

Locking is automatically unlocked at idle time.

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



# mark

Card 1 of 1

mark all cards

mark card

mark cards where trueOrFalse

mark cards by finding ~

[international] text [in field]

mark cards by finding chars ~

[international] text [in field]

mark cards by finding word ~

[international] text [in field]

mark cards by finding whole ~

[international] text [in field]

mark cards by finding string ~

[international] text [in field]

The mark command sets the marked property of the specified cards to true. You can operate on the set of marked cards with commands such as print, go, show, and sort.

Mark cards where evaluates the given expression for every card in the stack. If its value is true, the card is marked.

Mark cards by finding marks cards using the same mechanism as the find command. It marks cards very quickly.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## multiply

`multiply [chunk of] container -  
by number;`

where container or chunk must contain a number. (You can use the `is a` operator to see if the container is a number.)

The `multiply` command multiplies the number in the container or chunk by number and puts the result into the container or chunk. The result is calculated to a precision of up to 19 decimal places.

The result is displayed in a field or the Message box according to the global property `numberFormat`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



open *application*

open *filename* with *application*

The open command launches another application program or opens a document with another application from within HyperCard. You must provide the full path names for the files if they're not at the same directory level as HyperCard.

Under the Finder in System 6, HyperCard sends the suspend system message to the current card before turning over control to the application.

If HyperCard can't find the document or application, it displays a directory dialog box and asks the user to find it.

HyperCard also sets the result to Cancel if the user clicks Cancel in the dialog box. Otherwise, it sets the result to empty.

If HyperCard has problems opening the specified application (for example, there's not enough memory), it sets the result to "Couldn't open that application."

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## open file

open file *filename*

The `open file` command opens the specified file so that you can read data from it and write to it. Usually, the file is an ASCII text file opened to allow importing or exporting text. If the file doesn't exist, HyperCard creates it.

Use the `close file` command to close files after you've opened them.

Note: You must provide the full path name of the file if it's not at the same directory level as HyperCard.

If HyperCard has problems opening the specified file, it sets the result as follows:

Nonexistent file that can't be created:  
Can't create that file.

Existing file already open:  
File is already open.

Other error opening file:  
Can't open that file.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## open printing

open printing [with dialog]

The open printing command begins a print job. It uses the current settings from the Print Stack dialog box.

If you specify the `with dialog` option, HyperCard displays the Print Stack dialog box, and the user can choose new settings. HyperCard sets the result to `Cancel` if the user clicks `Cancel`; otherwise, it sets the result to empty.

You must use the close printing command to end a print job begun with open printing.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## open report printing

open report printing

open report printing with dialog

open report printing -

with template text

where text is the name of a report template in the current stack.

The open report printing command begins the process of printing a stack (or part of a stack) as a report. It uses the current settings from the Print Report dialog box. You must use the close printing command to end a job begun with open report printing.

If you specify the with dialog option, HyperCard displays the Print Report dialog box and the user can choose new settings. If you specify the with template option, HyperCard prints the stack with the named report template.

HyperCard sets the function the result to Cancel if the user clicks Cancel in the dialog box, to no such report template if you specify a template that doesn't exist, or to empty in all other cases.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# play

`play stop`

`play sound`

`play sound notes`

`play sound tempo pos/integer`

`play sound tempo pos/integer notes`

The `play` command plays a sound or a series of notes using a sound through the speaker of the Macintosh (or through the audio jack if it's in use).

`Play stop` stops the current sound immediately; otherwise, the sound plays until it's done and stops by itself.

**Important:** HyperCard continues to run handlers and perform other actions while a sound plays. Use the command `wait until the sound is done` to stop a handler until the sound is done playing.

The text string `notes` is an unlimited sequence of words in which each word represents one note. A note has the following NAOD format (Name, Accidental, Octave, and Duration):

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics

Name one of these letters:  
a, b, c, d, e, f, g, r  
(where r = rest)

Accidental one of these characters:  
#, b

Octave a positive integer  
(4 is middle C)

Duration one of the letters:  
w, h, q, e, s, t, x  
where  
w = whole note  
h = half note  
q = quarter note

e = eighth note  
s = 16th note  
t = 32nd note  
x = 64th note

You don't have to specify the accidental, octave, or duration of a note. Initially, the accidental defaults to none. The octave and duration default to **the same values as the previous note**, or to 4 (octave) and q (duration) for the first note.

--- More ---

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



Specify the note `r` to get a rest.  
For example:

```
re --eighth note rest
```

A duration followed by a period (.) means a dotted note. A duration followed by the number 3 means one note of a triplet.

HyperCard can also play digitized sounds stored as 'snd' resources.

Use `play` with the name of the resource as the sound. The resource must appear

in the current stack, a stack being used, or the Home stack.

HyperCard requires more RAM to play large digitized sounds—about 22K for every second the sound plays.

If HyperCard can't find the sound or load it into memory, the result gets `Couldn't load sound`. If the volume is set to 0, if an XCMD is using the sound channel, or if HyperCard is running in the background, the result gets `Sound is off`.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# pop

pop card

pop card { into | after | before } ~  
[chunk of] container

The `pop card` command retrieves the identification (full card ID and stack path name) of a card previously saved with the `push card` command.

If you don't provide a container to hold the card information, `pop` goes directly to the popped card.

If you do specify a container, `pop` puts the card's identification into the container, and you don't go anywhere. You can then check the card ID or stack to decide whether you want to return to that card.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# print

Card 1 of 3

print card [from *point1* to *point2*]  
print all cards  
print marked cards  
print *pos/integer* cards  
print *card* [from *point1* to *point2*]  
  
print {*field* | *button* }  
print *fileName* with *application*  
print *expression*

The print command prints card images, the contents of fields and buttons, documents from other programs, or the value of any HyperTalk expression.

Print card prints an image of the current card. The option from *point1* to *point2* specifies a rectangular area of the card.

Print all cards prints the image of every card in the stack.

Print marked cards prints a subset of the cards in the stack based on each card's marked property (as reflected in its Card Info dialog box). You can mark cards with the mark command.

--- More ---

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



Print *pos/integer* cards prints a range of consecutive cards starting from the current card.

Print *card* prints the card specified by the card expression. The option from *point1* to *point2* specifies a rectangular area of the card. HyperCard sets the function the result to "No such card." if the specified card doesn't exist; otherwise, the result returns empty.

Print *button* prints the contents of the specified button.

Print */field* prints the contents of the specified field, preserving the fonts, sizes, and styles of text used in the field.

Print file with application prints a document using another program. HyperCard launches the application; the application tries to print the document. (The user might see a Print dialog box.)

If HyperCard can't find either the document or the application, it displays a directory dialog box and asks the user to find it.

--- More ---

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



HyperCard also sets the result to Cancel if the user clicks Cancel in the dialog box. Otherwise, it sets the result to empty.

If HyperCard has problems opening the application (for example, there's not enough memory), it sets the result to "Couldn't open that application."

After printing, the application program quits, and control returns to HyperCard.

Finally, print *expression* prints the value of any HyperTalk expression. You can print the values of local and global variables, fields, *chunk* expressions, the current selection, the contents of the Message box, and the result of any function or property.

Expressions are printed using the settings in printMargins, printTextAlign, printTextFont, printTextSize, printTextHeight, and printTextStyle.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# push

`push card`  
`push card [of stack]`  
`push bkgnd [of stack]`  
`push stack`

The `push` command saves the identification of a card in HyperCard's memory. If you specify a background or stack, HyperCard stores the location of the **first** card of the background or stack. In all cases, HyperCard saves the full card ID and the path name for the stack.

Each call to `push` saves a card ID in last-in, first-out order. You can return to saved locations using the `pop` command.

Note: The word `stack` that forms part of the stack identifier `stack` is optional with the `push` command.

There is a limit of twenty pushes.

If you `pop` more than you `push`, you go to the Home stack.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**put**

**put *expression***

**put *expression preposition* ~**  
 [chunk of] *container*

**put *menu/itemList preposition* ~**  
 [menu/item of] *menu* ~  
 [with menuMessages *messageList*]

The **put** command evaluates an expression and places the value it extracts into, after, or before the contents of a container.

The container can be a button, a field, a variable, the Message box, the selection, a *chunk* expression, or a menu. If you don't specify a container, HyperCard puts the value into the Message box, showing it if it's hidden.

Use **put with into** to replace the contents of a container or menu, with **before** to place the value at the beginning of its contents, and **with after** to append the value to the end of its contents.

--- More ---

**Examples**

**Demo Script**

**Tips**



**Related Topics**



**Find Topic**



**Main Topics**



The lines of text that you put into a pop-up button become the items of the menu that appears when you click the button.

Before you can add items to a menu, the menu must already exist.

The *menu/itemList* is a comma-separated list of the items you want to add to the menu. Use the single character "-" to get a gray line. You can put up to 64 items into a menu with a single put statement.

The optional *messageList* is a comma-separated list of *message* names that HyperCard sends to the current card when the user chooses one of the menu items. The number of items in the *messageList* must equal the number of items in *menu/itemList*.

To skip a message name, use an empty item—a null between two commas:

"myMessage 1,,myMessage 3"

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



read from file *fileName* [at [-] ~ *integer*] {for *pos/integer* |until *char*}

The read command reads data from a file. (You must have already opened the file with the open file command.) Read places the data into the local variable *it*.

Reading of a newly opened file starts from the beginning of the file, or optionally at character *integer*. If you use the form at -*integer*, reading starts at *integer* characters from the end of

the file. Subsequent reads continue from the last point read.

Read continues until it has read the specified number of characters or it reaches the named ASCII character (which can be specified as a constant: colon, comma, end, eof, formfeed, quote, return, space, or tab).

All characters count as data, including return characters at the end of lines, spaces, and tab characters.

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



# reply

`reply expression [with keyword aeKeyword ]`

`reply error expression`

*Expression* is any text. *AeKeyword* is an Apple event keyword (a 4-character string).

The `reply` command answers an incoming Apple event.

If you don't specify a keyword, *expression* becomes the direct parameter of the `reply`.

You use `reply error expression` to notify the Apple event sender that an error has occurred.

You can use `reply error expression` to define your own error messages. This form is equivalent to `reply expression with keyword "errs"`. The Apple event keyword "errn" sets the error number.

The `reply` command sets the result to `No current Apple event` when there is no current Apple event to handle.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**reply (cont'd)**

Use `reply` only if you're handling Apple events yourself. If HyperCard handles an Apple event, it will provide the appropriate information in the reply.

See the documentation for the sending program to know which `reply` keywords it expects. (`errs` and `errn` are standard keywords.)

In AppleScript, you may need to reply with both an error string and an error number to trigger an error clause.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**request**

*request expression* from|of program ~  
*program*

*request expression* from|of ~  
*program id* *programID*

*request expression* from|of ~  
*this program*

*request appleEvent* data|class|id ~  
*sender|return id|sender id*

*request appleEvent* data ~  
*with keyword* *aeKeyword*

*Expression* yields an expression  
understandable to the target program.

*From* and *of* are interchangeable.

*Program* yields a valid program path  
name in the form

*zone:targetComputer:targetProgram*

where *targetProgram* is the name of a  
program running on computer

*targetComputer* in network zone *zone*.

*ProgramID* is the application's signature.  
*AeKeyword* is an Apple event keyword.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



The `request` command sends an "evaluate expression" Apple event from HyperCard to another application.

You can use this command to send an expression to any program that understands the standard `eval` Apple event.

The expression you use must be understandable to the target program. For example, if the target program is another HyperCard, the expression can be any valid HyperTalk expression.

When the target program executes the statement, the result of the request (the value of the expression) goes into the local variable `i.t.`

If the target program reports an error, HyperCard sets the result with an error message.

You use the `request appleEvent` forms to examine the data and attributes of an incoming Apple event.

--- More ---

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**request** (cont'd)

You can omit the *zone* parameter from the program path name when the target computer is in the same zone as the source computer.

You can omit the *targetComputer* parameter if the target program is running on the same computer as HyperCard.

You can pass the user selection from the answer program command as the *program* parameter.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## reset

Card 1 of 1

```
reset menuBar  
reset printing  
reset paint
```

Reset menuBar restores the menubar to HyperCard's standard menus. Reset printing restores the default values for the printing properties, as follows:

```
printMargins: 0,0,0,0  
printTextAlign: left  
printTextFont: Geneva  
printTextHeight:13  
printTextSize: 10  
printTextStyle: plain
```

Reset paint reinstates the default values for all the painting properties, as follows:

brush: 8	textAlign: left
centered: false	textFont: Geneva
filled: false	textHeight: 16
grid: false	textSize: 12
lineSize: 1	textStyle: plain
multiple: false	
multiSpace: 1	
pattern: 12	
polySides: 4	

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## returnInField

### returnInField

The `returnInField` command places a return character at the position of the insertion point in a field.

If the `autoTab` property of the field is true and the insertion point is on the last line of the field and the field is any type except scrolling, the `returnInField` command does not insert a return character but instead sends the `tabKey` command to the field.

HyperCard automatically sends the `returnInField` command to a field when the user presses Return and the insertion point is in the field.

You can handle the `returnInField` message as follows:

```
on returnInField  
    statements  
end returnInField
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## returnKey

returnKey

The returnKey command sends a statement typed into the Message box to the current card.

HyperCard sends the returnKey command to the current card when the user presses the Return key unless the insertion point is in a field, in which case HyperCard sends returnInField instead.

You can handle the returnKey message as follows:

```
on returnKey  
    statements  
end returnKey
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## run

run

send "run" to *object*

The run command executes an OSA script.

Used in any other way, run does nothing.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
save [this] stack as [stack] fileName
save stack fileName -
    as [stack] fileName
```

The `save` command saves a copy of a stack. It's the same as choosing `Save A Copy` from the `File` menu, but it doesn't display a directory dialog box. Use `save` when you don't want a dialog box to interrupt a handler.

The form `save [this] stack` saves a copy of the current stack.

If the specified stack already exists, HyperCard sets the value of the result to "Couldn't duplicate stack." You can test whether HyperCard saved the stack successfully as follows:

```
save this stack as "My Copy"
if the result is not empty then ...
```

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**select**

```
select empty
select button
select field
```

```
select text of container
select before text of container
select after text of container
select chunk of container
select before chunk of container
select after chunk of container
```

**Note:** container cannot be a variable.

The **select** command selects buttons, fields, or text.

Select button selects a button as if you had chosen the Button tool and clicked it. Select field selects a field as if you had chosen the Field tool and clicked it.

Note: You can't use **select** to select hidden buttons or fields, and the user level must be set to Authoring or Scripting for **select** to work.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Select empty removes the current selection. Use select empty instead of click at to deselect text or objects.

Select text applies to all the text in a field or in the Message box. You can select all the text or place the insertion point before the first character or after the last character of text.

Select chunk applies to a specified range of text in the Message box or in a field, to one or more lines in a list field, or to a line (that is, a "menu" item) in a popup button.

In the Message box or in a field you can select the entire range of text or place the insertion point before the first character or after the last character of the range.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] *property* to *expression*  
set [the] *property* of *object* ~  
to *expression*

set [the] *property* of *window* ~  
to *expression*

set [the] *property* of [*menu/item* of] ~  
*menu* to *expression*

set [the] *property* of *chunk* ~  
of *field* to *expression*

Note: *expression* must yield a valid  
setting for the specified property.

The **set** command changes the state  
of a specified property. If the *object* or  
element to which the property belongs is  
not specified, the property must be a  
global property or painting property.

You can use the Info dialog box of an  
object to set many of its properties.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# show

Card 1 of 2

show menuBar

show titleBar

show groups

show all cards

show marked cards

show pos/integer cards

show card picture

show background picture

show picture of card

show picture of bkgnd

show field [at point]

show button [at point]

show window [at point]

show externalWindow [at point]

The show command displays HyperCard objects and elements.

Show menuBar displays the menu bar at the top of the screen (unless the screen is locked). Show titleBar displays the title bar on the card window if it's been hidden. (Normally, the title bar is visible.)

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



Show groups displays a two-pixel gray underline for all text that has the group text style. (By default, the group text style is invisible.) The underline appears for grouped text in every field (in all stacks). Use the hide groups command to remove the underline.

The show card forms display the specified cards in the current stack in turn, beginning with the next card or the next marked card.

The other forms of the show command display the card or background picture, a window, or an object at a specified location on the screen. If the point is not given, the window or object is displayed at its previous location.

Showing a window makes it the frontmost window. With external windows, an external command or external function must first create a window before show will work on it. Show does **not** create windows.

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



```
sort [sortDirection] [sortStyle] ~  
    by expression  
sort [this] stack [sortDirection] ~  
    [sortStyle] by expression  
sort [marked] cards [of this stack] ~  
    [sortDirection] [sortStyle] ~  
    by expression  
  
sort bkgnd [sortDirection] ~  
    [sortStyle] by expression  
sort [marked] cards of bkgnd ~  
    [sortDirection] [sortStyle] ~  
    by expression
```

```
sort [chunks of] container ~  
    [sortDirection] [sortStyle] ~  
    [by expression]
```

where *chunks* are limited to either lines or items. (See next card for details.)

The first five forms of the `sort` command order all the cards in a stack or background by the value of *expression*, evaluated for each card in the stack or background.

--- More ---

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(b68c29742f83f84c941b74ac2119f380\_img.jpg\) Related Topics](#)[Find Topic](#)[Main Topics](#)

**sort** (cont'd)

The last form of the `sort` command (by `expression`) sorts lines or items of a container by any expression. If you don't specify, `sort` orders by lines. Before `expression` is evaluated for each line or item of the container, the local variable `each` is set to the contents of the chunk. (Click the Examples button to see syntax examples using `each`.)

For all forms of the `sort` command, the default sort direction is ascending, and the default sort style is `text`.

Sort direction `ascending` orders the sort elements—the value of the expression on each card or the lines or items in the container—from lower to higher values.

Sort direction `descending` orders the sort elements from higher to lower values.

Sort style `text` compares the sort elements based on their ASCII values:

"1" < "101" < "2" < "a" < "ab" < "b"

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



Note that neither case nor diacritical marks matter with the sort style `text`:

```
"apple" = "APPLE" = "äpple"
```

Sort style `numeric` correctly sorts numbers. With sort style `text`,

```
"1" < "100" < "17" < "2"
```

The sort style `numeric` correctly sorts these values as:

```
"1" < "2" < "17" < "100"
```

The sort style `dateTime` orders the sort elements by their date or time format. (See the `convert` command for valid date and time formats.)

The sort style `international` correctly sorts non-English text containing diacritical marks and special characters based on the international resource installed in the current stack, the Home stack, HyperCard itself, or the System file.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## start using

start using *stack*

The `start using` command inserts the specified stack into the message-passing order between the current stack and the Home stack (or between the current stack and any other stacks being used).

Handlers in the stack script of the newly inserted stack can intercept messages as the messages move through the message-passing order.

HyperCard records the full path name of the stacks being used as lines in the global property `the stacksInUse`. You can add up to 16 stacks.

The first stack in use is on line 1 of `the stacksInUse`, the second stack is on line 2 of `the stacksInUse`, and so on. The order of items in `the StacksInUse` determines the message-passing order: from the current stack to line 1 of `the StacksInUse`, to line 2, and so on, to the Home stack.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## stop using

stop using *stack*

The `stop using` command removes a stack from the message-passing order. Handlers in the stack script of the removed stack will no longer be available for use to the current stack.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## subtract

subtract number from [chunk of] ~  
container

where *container* or *chunk* must contain a number.

The subtract command subtracts number from the specified container (or chunk) and puts the result into the container (or chunk), replacing what was there.

(You can use the `is a` operator to see if the container is a number.)

For example, if you say

subtract 3 from theTotal

and `theTotal` previously held 7, it will now hold 4.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## tabKey

### tabKey

The `tabKey` command opens the first unlocked field on the current card or background (placing the insertion point in the field) and selects its entire contents.

If a field is already open for editing, `tabKey` closes it and opens the next editable field, selecting its contents. (A field is editable only if it is unlocked and visible.)

The `tabKey` command opens fields in the following order: from the lowest to the highest numbered background field, then from the lowest to the highest numbered card field.

HyperCard sends the `tabKey` command to a field or the current card when the user presses the Tab key. You can handle the `tabKey` message as follows:

```
on tabKey  
  statements  
end tabKey
```

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



## type

type *text*

type *text* with commandKey

The type command acts exactly as if the user had typed *text* from the keyboard.

The text appears in the Message box when it's visible (or also when it's hidden if blindTyping is set to true).

To type text into a field or to add paint text, a handler must first set the insertion point (using either the select or click at command).

To perform a menu command, use the form type *text* with commandKey. For example, if you have a graphic on the Clipboard, you can paste it with the command

type "V" with commandKey

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Topics](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**unlock**

`unlock {screen|messages|recent}`  
`unlock error dialogs`  
`unlock screen with effect [speed] ~`  
`[to image]`  
`unlock screen with visual [effect] ~`  
`effect [speed] [to image]`

Unlock screen lets HyperCard update the screen after a lock screen command. (Click the placeholders *effect*, *speed*, and *image* to see their possible replacements.)

Optionally, you can add a single visual effect. You can lock the screen, perform

actions on the card, and then unlock the screen with a visual effect.

Unlock messages lets open, close, suspend, and resume messages traverse the message-passing path.

Unlock recent lets HyperCard keep a visual record of visited cards in the Recent Cards dialog box.

Unlock error dialogs lets HyperCard show error dialog boxes when an error occurs.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



unmark all cards

unmark card

unmark cards where trueOrFalse

unmark cards by finding

[chars|word|whole|string] -

[international] -

text [in field] [of marked cards]

The unmark command sets the marked property of the specified cards to false. Use it to deselect sets of cards that you have marked.

Unmark cards where visits each card in the stack and evaluates the expression. If its value is true, HyperCard unmarks the card. Unmark cards by finding unmarks cards using the same mechanism as the find command.

It unmarks cards very quickly.

----- End of Topic -----

**Examples**

**Demo Script**

**Tips**



**Related Topics**



Find Topic



Main Topics



visual [effect] *effect* [*speed*] ~  
[to *image*]

The `visual` command specifies a visual effect for HyperCard to use as it moves from one card to another. (Click the placeholders *effect*, *speed*, and *image* to see their possible replacements.) `Visual` must be followed by a `go` command to have any effect.

The optional *speed* parameter tells HyperCard to perform the visual effect faster or slower than normal.

The `to image` option changes the screen to white, gray, black, inverse, or the image of the destination card before applying the visual effect.

HyperCard uses the visual effect `plain` as its default effect. The default image is `card`.

Note: Only the effects `push`, `scroll`, `shrink`, `stretch`, and `zoom` work with the command `go this card`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
wait [for] pos/integer [ticks]
wait [for] pos/integer seconds
wait until trueOrFalse
wait while trueOrFalse
```

The **wait** command causes HyperCard to pause before executing the rest of a **handler**, either for a specific length of time, until a specified condition becomes true, or while a specified condition remains true.

If you do not specify **seconds** as the unit of time, HyperCard uses ticks. (One tick equals one-sixtieth of a second.)

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
write text to file fileName ~  
[at { eof|end|[-]integer }]
```

The `write` command copies text to a file. You must have already opened the file with the `open file` command, and you should close it when writing is completed with the `close file` command.

The first `write` command after opening a file begins at the start of a file unless you use the `at integer` option, with integer as the character position within the file where writing is to begin.

The `-integer` option begins writing at integer characters from the file's end.

**Warning:** HyperCard **does not ask** if you want to write over existing text.

Subsequent `write` commands append text to the file's contents after the last character written until you close the file.

Note: You must provide the full path name of the file if it's not at the same directory level as HyperCard.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**abs**

the abs of *number*

Value returned: number equal to the absolute value of *number*

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# annuity

`annuity(number1, number2)`

where number1 represents the **interest rate** (expressed as a decimal) and number2 represents the number of **periods** over which you receive annuity payments.

Value returned: the total cost of an annuity now that will pay you one unit per period over the specified number of periods, or a number equal to  

$$(1 - (1 + \text{rate})^{-\text{periods}}) / \text{rate}$$

An interest rate involves a certain percentage (expressed as a decimal) per some unit of time—usually per year. You must use the same unit of time to measure the number of periods.

For example, if you have a yearly percentage rate but your annuity pays you monthly, use `rate / 12`, and be sure to express the number of periods as months (2 years = 24 months).

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**atan**

the atan of *number*

Value returned: number equal to the trigonometric arc tangent of *number* expressed in radians

Note: There are  $2 * \pi$  radians in 360 degrees.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## average

`average(numberList)`

Value returned: number equal to the arithmetic average of the comma-separated list of numbers

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics



## charToNum

the `charToNum` of `char`

Value returned: positive integer equal to  
the ASCII value of the `char`

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the clickChunk

Value returned: character chunk expression equal to the word the user clicked or the longest range of characters with the text style Group around the character that the user clicked (if the text has been grouped)

The text style Group allows the clickChunk to extend beyond word boundaries.

The chunk expression returned has the following form:

char pos/integer to pos/integer -  
of container

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## clickH and clickV

the clickH

the clickV

Values returned:

clickH returns an integer equal to the number of horizontal pixels from the left side of the card to the place the mouse was last clicked.

clickV returns an integer equal to the number of vertical pixels from the top of the card to the place the mouse was last clicked.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the clickLine

Value returned: line chunk expression equal to the line that the user clicked or, if the text has been grouped, the first line of the longest range of lines with the text style Group around the character that the user clicked

The text style Group allows the clickLine to extend beyond one line.

The chunk expression returned has the following form:

line pos/integer of container

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the clickLoc

Value returned: *point* equal to the place  
on the screen where the user most  
recently clicked relative to the top left  
corner of the current card

HyperCard does **not** reset the clickLoc  
at idle, nor does it reset when a handler  
is running, unless you use the wait  
command:

wait until the mouseClick

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## clickText

the clickText

Value returned: text string equal to the word the user clicked or the longest contiguous string of characters with the text style Group around the character that the user clicked

The text style Group allows the clickText to extend beyond word boundaries.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## commandKey

the commandKey

the cmdKey

Value returned: either of the constants  
up or down, depending on whether the  
⌘ key is up or down

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## compound

`compound(number1, number2)`

where *number1* represents the **interest rate** (expressed as a decimal) and *number2* represents the number of **periods** over which the interest is compounded.

Value returned: value of one unit of principal invested at the interest rate and compounded over the specified number of periods, or a number equal to  $(1 + \text{rate})^{\text{periods}}$

An interest rate involves a certain percentage (expressed as a decimal) per some unit of time—usually per year. You must use the same unit of time to measure the number of periods.

For example, if you have a yearly interest rate that is compounded monthly, you must convert the yearly rate to the interest per month ( $\text{rate} / 12$ ) and be sure to express the number of periods as months (2 years = 24 months).

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**cos**

the cos of *number*

Value returned: number equal to  
the trigonometric cosine of *number*  
expressed in radians

Note: There are  $2 * \pi$  radians in 360  
degrees.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**date**

the [*adjective*] date

date()

Value returned: text string representing the current date set in your Macintosh, in the following date formats:

the date -- 12/25/93

date() -- 12/25/93

the abbrev date -- Sat, Dec 25, 1993

the short date -- 12/25/93

the long date -- Saturday,

December 25, 1993

the English date -- Saturday,  
December 25, 1993

When HyperCard is running under System 7.1, the long, abbrev, and short forms agree with the formats set in the Date & Time Control Panel.

The form the English date returns the date in the form *day, month dayNumber, fullYearNumber* using english weekdays and months no matter what language the system is localized for and no matter what the settings are on the Date & Time Control Panel.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## destination

the destination

Value returned: the full path name of the stack that HyperCard is in the process of going to (as in `Inside:HyperCard:Home`)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the diskSpace [of disk *diskName*]

Value returned: a positive integer equal to the number of bytes of free space on the disk that contains the current stack or of the disk whose name appears in *diskName* (assuming *diskName* is a mounted volume).

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## **exp, exp1, exp2**

Card 1 of 1

the `exp` of number

the `exp1` of number

the `exp2` of number

Values returned:

`exp` returns a number equal to the constant `e` raised to the power of number.

`exp1` returns a number equal to 1 less than `e` raised to the power of number.

`exp2` returns a number equal to 2 raised to the power of number.

Note:  $e \approx 2.7182818$

----- End of Topic -----

**Examples**

**Demo Script**

**Tips**



**Related Topics**



**Find Topic**



**Main Topics**



## foundChunk

the foundChunk

Value returned: a character chunk expression that indicates where the most recent `find` command located its target string. If nothing was found, it returns empty. The chunk expression returned has the following form:

`char pos/integer to pos/integer -  
of container`

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## foundField

the foundField

Value returned: a field expression that identifies the field in which the most recent `find` command located its target string. If nothing was found, it returns empty. The field expression returned has one of the following forms:

card field *n*

bkgnd field *n*

where *n* is the number of the field.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## foundLine

the foundLine

Value returned: a line expression equal to the line of a field where the most recent `find` command located its target string. If nothing was found, it returns empty.

The line expression has the following form:

line / of card field *n*  
line / of bkgnd field *n*

where */* is the line number and *n* is the number of the field.

Note: A return character determines a line, not the line wrap. A line that wraps and is displayed as two lines is treated as one line by HyperTalk.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## foundText

the foundText

Value returned: a string equal to the characters enclosed in the box after the most recent `find` command has located its target string. If nothing was found, it returns empty.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## heapSpace

the heapSpace

Value returned: a number equal to the amount of working memory HyperCard has left (the number of bytes remaining in HyperCard's application heap)

The memory that HyperCard can use is divided into two parts—the heap and the stack. StackSpace returns the amount of memory that's available in the stack portion. HeapSpace returns the amount available in the heap portion.

The available memory determines whether the user can use the Paint tools, whether HyperCard can open a stack in a new window, and other performance-related factors.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# length

the length of *expression*

Value returned: the number of characters  
in the value of the expression

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## ln, ln1, log2

the ln of *number*

the ln1 of *number*

the log2 of *number*

Values returned:

ln returns a number equal to the base-e or natural logarithm of *number*.

ln1 returns a number equal to the natural logarithm of  $1 + \text{number}$ .

ln2 returns a number equal to the base-2 logarithm of *number*.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**max**

`max(numberList)`

Value returned: the highest-valued number from the comma-separated list of numbers

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## menus

the menus

Value returned: a return-separated list of the names of the menus in the current menu bar

If HyperCard is running under System 7, the list includes the System 7 menus (such as System Help and Application).

Note: The string Apple is a synonym for the ⚡ menu.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## min

`min(numberList)`

Value returned: the number with the lowest value from the comma-separated list of numbers

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**mouse**

the mouse

Value returned: one of the constants up or down indicating whether the mouse button is up or down

Note: Use the mouse to return the current state of the mouse button and the mouseClick to return whether the mouse has been clicked in the current handler.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## mouseClick

the mouseClick

Value returned: the constant `true` if the mouse has been clicked sometime during the running of the current handler, or `false` if it hasn't.

If the mouse button is down, the `mouseClick` waits until the mouse button is up before returning `true`.

Note: Use the `mouse` to return the current state of the mouse button and the `mouseClick` to return whether the mouse has been clicked in the current handler.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the mouseH

the mouseV

Values returned:

mouseH returns an integer equal to the number of pixels from the left of the card to the current location of the mouse pointer.

mouseV returns an integer equal to the number of pixels from the top of the card to the current location of the mouse pointer.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**mouseLoc**

the mouseLoc

Value returned: a *point* equal to the current position of the mouse pointer relative to the current card

----- End of Topic -----

*Examples*

*Demo Script*

*Tips*



*Related Topics*



Find Topic



Main Topics



**number**

the number of [card] {buttons|parts}  
 the number of bkgnd {buttons|parts}  
 the number of card fields  
 the number of [bkgnd] fields

the number of marked cards  
 the number of cards [in bkgnd] -  
 [of this stack]  
 the number of bkgnds [of this stack]

the number of windows  
 the number of menus  
 the number of menu items of menu  
 the number of chunks in expression

Values returned: A non-negative integer equal to one of the following:

- total number of buttons or fields on the current card or background
- total number of parts (buttons and fields combined) on the current card or background
- total number of marked cards, cards in a specific background, cards in an entire stack, or backgrounds in a stack

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**number** (cont'd)

- number of all the windows (including built-in palettes and external windows) in HyperCard
- number of menus in the menu bar or the number of menu items in a specified menu
- total number of characters, words, items, or lines in the value of any HyperTalk expression (treated as text)

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## numToChar

the numToChar of pos/integer

Value returned: the character whose  
ASCII equivalent equals pos/integer

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## offset

`offset(text1, text2)`

Value returned: the number 0 if text1 does not appear in text2; otherwise, a positive integer equal to the number of characters from the first character of text2 to the first character of text1 within text2

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## optionKey

the optionKey

Value returned: one of the constants up or down indicating whether the Option key is up or down

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## param

the param of `pos/integer`

Value returned: the value (as opposed to the name) of a parameter variable in the current handler, or empty if the parameter variable doesn't exist

The param of 0 is the name of the message itself.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## paramCount

the paramCount

Value returned: a positive integer equal to the total number of parameters, passed to the current handler.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## params

the params

Value returned: a text string equal to the entire parameter list, including the message name, passed to the currently executing handler.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## programs

the programs [of machine "zone:Mac"]

Value returned: a return-delimited list of applications currently running on the same machine as HyperCard

If you use the optional form of machine "zone:Mac", you get a list of Apple event-aware programs running on a remote machine.

This function requires System 7 to work.  
(See Demo Script.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## random

the random of [pos/integer](#)

Value returned: a random integer  
between 1 and [pos/integer](#).

Random returns values for integers  
up to  $2^{31} - 2$ .

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the result

Values returned:

- the value set by a `return` keyword during the execution of a message
- empty if most commands succeed; otherwise, a message string

In handlers, it's usually best to test a command with an expression such as

`if the result is not empty then ...`

That way, the handler doesn't rely on the specific value of a string.

[Click Related Topics](#) for more information about the value returned by the `result` for each command.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## round

the round of *number*

Value returned: the integer nearest to  
*number*

Odd integers plus exactly 0.5 round up;  
even integers (or 0) plus exactly 0.5 round  
down.

If *number* is negative, HyperCard  
internally removes the negative sign,  
rounds its absolute value, then puts  
the negative sign back on.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## screenRect

the screenRect

Value returned: a rectangle equal to the dimensions of the screen in pixels

If there's more than one monitor, the screenRect returns the dimensions of the monitor displaying the current stack as offsets from the top-left corner of the screen that contains the menu bar.

If the card window appears on more than one monitor, the screenRect returns the dimensions of the screen that shows the most area from the card window.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## seconds

the seconds

the secs

Value returned: an integer equal to  
the number of seconds between  
midnight, January 1, 1904, and the current  
time set in your computer

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## selectedButton

the selectedButton of ~  
[background | card] family /intExpr

Value returned: the domain and number  
(for example, card button 3) of the  
highlighted button in the specified button  
family on the current card or background

If you don't specify card (or cd) or  
background (or bg or bkgnd), the family  
is assumed to be on the card layer.

If no button in the specified family is  
highlighted, selectedButton returns  
empty.

If the specified family doesn't exist, you  
get an error dialog box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## selectedChunk

the selectedChunk

Value returned: a character chunk expression that indicates the range of characters currently highlighted

If nothing is highlighted, the selectedChunk returns empty. The chunk expression returned has the following form:

char pos/integer to pos/integer -  
of container

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## selectedField

the selectedField

Value returned: a field expression  
that indicates the field in which a range of  
characters is currently highlighted. If  
nothing is highlighted, it returns empty.  
The field expression returned has one  
of the following forms:

card field *n*

bkgnd field *n*

where *n* is the number of the field.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## selectedLine

the selectedLine

the selectedLine of *listField*

the selectedLine of *popupButton*

Values returned:

- for the selectedLine, a line expression that indicates the line of a field or the Message box in which a range of characters is currently highlighted

- for the selectedLine of *listField*, a line expression that indicates the lines of a *list field* in which the characters are currently highlighted

- for the selectedLine of *popupButton*, a line expression that evaluates to the line number in a popup button's contents indicating the current selection

If nothing is selected, the selectedLine returns empty.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



If a field isn't currently a list field but it was in the past, and a selection had been made when it was a list field, the **selectedLine** returns the most recent selection.

The line expression returned has the following forms:

line / of {card|bkgnd} field n  
line / of {card|bkgnd} button n  
line x to y of {card|bkgnd} field n

where / is the line number and n is the number of a popup button or a field, and x to y is a range of lines in a list field.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## selectedLoc

the selectedLoc

Value returned: a *point* equal to the left  
and bottom offsets of the insertion point  
or the current selection in a field

It returns *empty* if there is  
no selection or if the insertion point  
is in the Message box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## selectedText

the `selectedText`

the `selectedText` of *listField*

the `selectedText` of *popupButton*

Value returned: a string equal to  
the range of characters currently selected.  
If nothing is highlighted, it returns empty.

If a field isn't currently a list field but it  
was in the past, and a selection had been  
made when it was a list field, the  
`selectedText` returns the most recent  
selection.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## shiftKey

the shiftKey

Value returned: one of the constants up or down indicating whether the Shift key is up or down

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**sin**

the sin of *number*

Value returned: a number equal to the trigonometric sine of *number* expressed in radians

Note: There are  $2 * \pi$  radians in 360 degrees.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# sound

the sound

Value returned: a text string equal to the name of the sound resource currently playing (such as "boing") or the string "done" if no sound is currently playing.

You use the sound function to synchronize sounds with other actions, because scripts continue to run while sounds are playing.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**sqrt**

the `sqrt` of *number*

Value returned: a number equal to the square root of *number*. If *number* is negative, `sqrt` returns `NAN(001)`, which means "not a number."

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

the stacks

Value returned: a return-separated list of the full path names for all the open stacks, in front-to-back order

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the stackSpace

stackSpace( )

put the stackSpace into howMuch

Value returned: an integer representing the free space, in bytes, in HyperCard's memory stack.

The memory that HyperCard can use is divided into two parts—the heap and the stack. HeapSpace returns the amount of memory that's available in the heap portion. StackSpace returns the amount that's available in the stack portion.

The memory in HyperCard's stack determines, for example, the number of times you can call a recursive handler.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**sum**

```
sum(numberList)
```

Value returned: the sum of a comma-delimited list of items

*numberList* evaluates to a comma-delimited list of items, including any container holding such a list.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## systemVersion

the systemVersion

Value returned: a decimal string representing the running version of system software

Use this function, for example, to determine if a particular command or handler will run correctly under the current version of the system software.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**tan**

the tan of *number*

Value returned: a number equal to  
the trigonometric tangent of *number*  
expressed in radians

Note: There are  $2 * \pi$  radians in 360  
degrees.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## target

the [*adjective*] target

Value returned: a text string that identifies the object that originally receives a message sent by HyperCard or by the send keyword. If an object does not have a name, the target returns its ID.

The adjectives abbreviated, long, and short return various forms of an object's name as with the property the name. (Click Related Topics for more information about name.)

Note: the abbreviated target is the same as the target.

HyperTalk distinguishes between the target and target: the target always returns a string that identifies an object, but the single word target is a container.

If the target is a button or field, target refers to the contents of that button or field. If not, target returns an error.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## ticks

the ticks

Value returned: an integer equal to the number of ticks since the Macintosh was turned on or restarted. (One tick equals one-sixtieth of a second.)

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**time**

the [*adjective*] time

time()

Value returned: A text string representing the current time set in your Macintosh, in one of the following time formats:

the time -- 9:14 AM

time() -- 9:14 AM

the abbrev time -- 9:14 AM

the short time -- 9:14 AM

the long time -- 9:14:42 AM

the English time -- 9:14:42 AM

When HyperCard is running under System 7.1, the long, abbrev, and short forms agree with the formats set in the Date & Time Control Panel.

The form the English time returns the time in the form *hh :mm :ss: AM|PM* no matter what language the system is localized for and no matter what the settings are on the Date & Time Control Panel.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**tool**

the tool

Value returned: a text string equal to the name of the currently chosen tool (that is, the `toolName`)

(Click `toolName` for a list of all the tool names.)

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**trunc**

the trunc of *number*

Value returned: an integer equal to the integer part of *number*. Any fractional part is disregarded, regardless of the number's sign.

Note: `trunc` returns correct values only for real numbers in the range -2,147,483,648 through 2,147,483,647 (the maximum long integer value).

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## value

the value of *expression*

Value returned: the value of the expression after HyperTalk evaluates it

There is no limit to the number of characters that the `value` can have as its argument.

When the argument of `value` is a multitoken literal expression, the expression evaluates to itself:

```
put value("HyperCard 2.2")
```

yields HyperCard 2.2

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## windows

the windows

Value returned: a return-separated list of the names of all the windows (including built-in palettes) in front-to-back order

If the longWindowTitles is true, the windows returns full path names for windows that contain stacks.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



get the address

This read-only property tells you where on the AppleTalk network you are, in the form `zone:computer:program`. If `zone` is an asterisk (\*), either your system is not on a network or the network has just one zone.

So if you're running HyperCard on a computer named Quille on a zone called HyperText, the statement

yields

`HyperText:Quille:HyperCard`

If you're running HyperCard on an unnamed computer that's not on a network, you get

`*::HyperCard`

This property requires System 7.0 or later.

put the address

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

[Related Topics](#)



Find Topic



Main Topics

```
set [the] autoHilite of button -  
to trueOrFalse
```

The `autoHilite` property sets or returns whether a button highlights automatically in response to a `mouseDown` event.

With check boxes and radio buttons, `autoHilite` determines whether the button can change from deselected to selected and vice versa.

The `autoHilite` property corresponds to the Auto Hilite check box in a Button Info dialog box.

Automatic highlighting occurs if `autoHilite` is set to `true`.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] autoSelect of field ->  
    to trueOrFalse
```

The `autoSelect` property sets or returns whether lines in a locked field highlight automatically in response to a `mouseDown` event and to dragging or shift-clicking. (Such fields are called [list fields](#).)

The `autoSelect` property corresponds to the Auto Select check box in a Field Info dialog box.

Automatic highlighting occurs if `autoSelect`, `lockText`, and `dontWrap` are all set to `true`.

`AutoSelect` is set to `false` when `dontWrap` is set to `false`; `dontWrap` is set to `true` when `autoSelect` is set to `true`.

For `autoSelect` to affect more than one line in a locked field, that field's `multipleLines` property must also be set to `true`.

[--- More ---](#)

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



To learn which lines are selected, get the `selectedLine` of `field`. To learn the contents of those lines, get the `selectedText` of `field`. To preselect, use `select line x [to y] of field`.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] autoTab of field to ~  
trueOrFalse
```

The `autoTab` property returns or sets whether HyperCard inserts a return character when the insertion point is on the last line of a field (`autoTab` is false) or moves the insertion point to the next editable field on the card (`autoTab` is true).

Note: Auto tabbing doesn't work with scrolling fields.

The `autoTab` property corresponds to the Auto Tab check box in a Field Info dialog box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

```
set [the] blindTyping to trueOrFalse
```

The `blindTyping` global property returns or sets whether you can type into the Message box and send messages from it even when it isn't visible.

HyperCard determines the default setting for `blindTyping` at startup (and when HyperCard resumes after being suspended) from the Blind Typing option on the Preferences card of the Home stack.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## bottom

the bottom of *card*

```
set [the] bottom of button to integer
set [the] bottom of field to integer
set [the] bottom of window to integer
get [the] bottom of menuBar
```

The `bottom` property returns or sets an integer equal to item 4 of an element's rectangle. If you set the bottom of an element, it moves vertically; its size remains the same. You can't set the bottom of cards: use the `rect`, `height`, and `width` properties to resize the cards in a stack.

You can't set the bottom of the menu bar.

HyperCard determines the bottom of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

HyperCard determines the bottom of the card window relative to the top-left corner of the screen with the menu bar.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## bottomRight

the bottomRight of card

```
set [the] bottomRight of button to point
set [the] bottomRight of field to point
get [the] bottomRight of menuBar
```

The bottomRight property returns or sets a point equal to items 3 and 4 of an element's rectangle. If you set the bottomRight of an element, it moves; the element's size remains the same.

You can't set the bottomRight of cards: use the rect, height, and width properties to resize the cards in a stack.

HyperCard determines the bottomRight of buttons, fields, and built-in windows relative to the top-left corner of the current card. It determines the bottomRight of the card window relative to the top-left corner of the screen with the menu bar.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



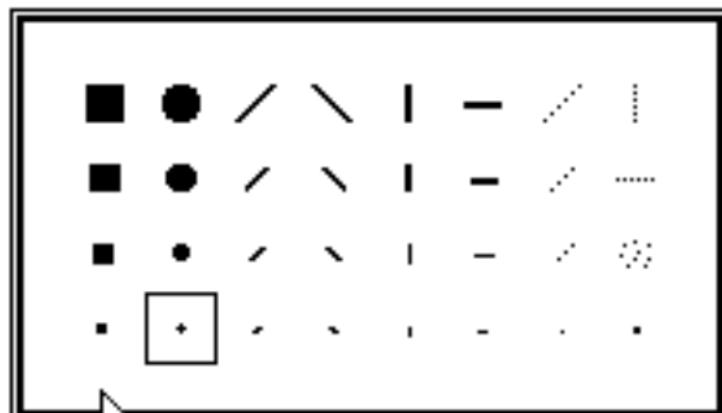
## brush

set [the] brush to pos/integer

where pos/integer is a positive integer in the range 1 through 32.

The brush property returns or sets the current brush shape used by the Brush tool.

The value of the brush property represents a brush shape from the Brush Shape dialog box. The default brush is 8.



Press and hold the mouse over this illustration to see the brush numbers.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] cantAbort of stack to trueOrFalse
```

The `cantAbort` property returns or sets whether users can type ⌘-. (⌘-period) to stop any running handlers. It corresponds to the Can't Abort check box in the Protect Stack dialog box. Use `cantAbort` to prevent users from canceling certain critical operations that would leave a stack in a confusing or dangerous state.

**Warning:** Use `cantAbort` with caution. Once `cantAbort` is set to true, there's no way to halt an errant handler. Set `cantAbort` to true, and then immediately set it to false when you no longer need it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



```
set [the] cantDelete of card ->  
    to trueOrFalse  
set [the] cantDelete of bkgnd ->  
    to trueOrFalse  
set [the] cantDelete of stack ->  
    to trueOrFalse
```

The `cantDelete` property returns or sets whether a user can delete a specified card, background, or stack. It corresponds to the Can't Delete check box in the Card Info, Background Info, and Protect Stack dialog boxes.

The default value is `false` (meaning that a card, background, or stack can be deleted).

Note: Setting the `cantDelete` of a stack to `true` doesn't prevent the user from deleting the stack by dragging it to the Trash.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## cantModify

```
set [the] cantModify of stack -  
    to trueOrFalse
```

The `cantModify` property returns or sets whether a stack can be changed in any way. It corresponds to the Can't Modify check box in the Protect Stack dialog box.

Setting the `cantModify` of a stack to true selects both the Can't Modify Stack check box and the Can't Delete Stack check box in the Protect Stack dialog box. When `cantModify` is true, a padlock appears in the menu bar.

The default value is `false` (meaning that the card, background, or stack can be modified).

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] cantPeek of stack -  
    to trueOrFalse
```

The `cantPeek` property returns or sets whether users can press ⌘-Option or ⌘-Shift-Option to peek at the location of buttons and fields and use the ⌘-Option shortcuts for accessing scripts. It corresponds to the Can't Peek option in the Protect Stack dialog box.

The default value is `false`, meaning that you can peek at fields and buttons and use the ⌘-Option shortcuts for accessing (or peeking at) scripts.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] centered to trueOrFalse

The centered property returns or sets whether HyperCard draws shapes from the center rather than from a corner. It corresponds to the Centered command in the Options menu (which appears when you select a Paint tool).

The centered property affects the Line, Rectangle, Rounded Rectangle, and Oval tools. Its default value is false.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## checkMark

```
set [the] checkMark of menu/item of menu to trueOrFalse
```

The `checkMark` property returns or sets whether a check-mark character appears in front of a menu item.

It uses the `✓` character, `numToChar(18)`, as the default check-mark character. Click Related Topics for information about the `markChar` property, which lets you use characters other than the check mark.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## commandChar

```
set [the] commandChar of menu/tem ~  
    of menu to char
```

The `commandChar` property returns or sets the character that you'd press with the ⌘ key as the keyboard shortcut (commonly called the ⌘-key equivalent) for a menu item.

If the menu item has no ⌘-key equivalent, the property returns `empty`. Otherwise, it returns the character.

If more than one menu item use the same command character, the menu item on the menu farther to the right takes precedence.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**cursor**

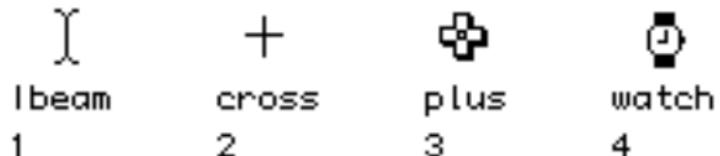
set [the] cursor to cursor

The `cursor` property sets the image that appears as the pointer on the screen.

**Note:** `cursor` is a set-only property. You cannot get the current value of the `cursor` from a script.

HyperCard provides a number of built-in cursors that you can use. (HyperCard automatically resets the cursor to the image for the current tool on idle—when no other action is happening.)

The built-in cursors are:



The `busy` cursor rotates an eighth of a turn each time you call it.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



## debugger

set [the] debugger to *debuggerName*

The debugger property returns or sets the name of the current HyperTalk debugger. The default value of this property is ScriptEditor, the name of HyperCard's built-in editor and debugger. (The built-in editor and debugger are integrated.)

Click Related Topics for more information about debugging a script.

Because HyperCard's debugger is actually an external command (XCMD), you can replace it with your own or third-party debuggers.

If HyperCard can't find a debugger with the name provided, it uses its built-in debugger.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics



```
set [the] dialingTime to numberOfTicks
```

The `dialingTime` property sets or returns the total length of time (in ticks) that HyperCard leaves the serial port open while dialing a modem.

The default value for `numberOfTicks` is 180, where 60 ticks = 1 second.

**Note:** `dialingTime` is not reset to its default value at idle time.

HyperCard maintains the value that you set for this property for the duration of the HyperCard session.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] dialingVolume to integer
```

The `dialingVolume` property sets or returns the volume of the *dialing tones* generated through the computer speaker by the `dial` command.

integer evaluates to an integer in the range 0 through 7, where 0 is extremely low but does not shut off the sound entirely.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## don'tSearch

```
set [the] don'tSearch of field to trueOrFalse
set [the] don'tSearch of card to trueOrFalse
set [the] don'tSearch of background to trueOrFalse
```

The `don'tSearch` property returns or sets whether HyperCard's `find` command will look for matches in a field, card, or background. It corresponds to the `Don't Search` check box in the `Field Info`, `Card Info`, and `Background Info` dialog boxes.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## don'tWrap

```
set [the] don'tWrap of field -  
    to trueOrFalse
```

The `don'tWrap` property returns or sets whether a field wraps text that is longer than the width of the field or instead truncates the viewable text at the right edge of the field. (In HyperTalk, a return character determines a line.)

This property corresponds to the Don't Wrap option in a Field Info dialog box.

Truncated viewable text isn't lost. If you set the `don'tWrap` to true, the "missing" text appears in the field.

This property is set to true when `autoSelect` is set to true; and it sets `autoSelect` to false when it is set to false.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#) [Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] dragSpeed to pos/ninteger

The dragSpeed property returns or sets how many pixels per second the pointer will move when manipulated by all subsequent drag commands. Use 0 to drag as fast as possible.

The dragSpeed affects all of the paint tools except the Bucket and Text tools.

On idle, HyperCard resets the dragSpeed to 0.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] editBkgnd to trueOrFalse

The editBkgnd property returns or sets the layer where new painting or new buttons and fields will appear—in the card layer or in the background layer.

It corresponds to the Background command in the Edit menu, and it's available only when the user level is Painting (3) or higher.

The default setting is `false`, meaning that new images and parts will appear on the card layer.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] enabled of menu|menu/item to trueOrFalse
set [the] enabled of button to trueOrFalse
```

The `enabled` property returns or sets whether a menu item, menu, or button is active or inactive (dimmed). Users cannot choose dimmed elements.

If you set the `enabled` of a menu to false, all items on the menu become inactive.

The `enabled` property won't enable items in HyperCard's menus unless they're currently available to the user.

For example, the following code won't enable the Button Info command unless a button is selected:

```
set the enabled of menuItem 1 of ~
menu "Objects" to true
```

The default state is `true`, meaning that the element is enabled.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## environment

get the environment

This read-only property returns development when the fully enabled version of HyperCard is running or player when HyperCard Player or a standalone stack is running.

(A standalone stack is an application that you create by choosing "Application" from the pop-up File Type menu in the Save a Copy dialog box.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## family

set [the] family of button to integer

where integer resolves to a value between 0 and 15, and 0 means "no family."

This property sets or retrieves the button family for a given button.

Setting the hilite of any button in a family to true sets the hilite of all other same-family buttons to false.

The default family value for a new button is 0.

Card and button families are distinct; so there can be a family *n* for card buttons and another family *n* for background buttons.

You can also assign a button to a family by using the Family pop-up menu in the button's Get Info dialog box.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] filled to *trueOrFalse*

The `filled` property returns or sets whether HyperCard fills shapes with the current pattern on the Patterns palette as you draw them. It corresponds to the Filled command in the Options menu (which appears when you select a Paint tool).

The default value is `false`.

The `filled` property affects the Rectangle, Rounded Rectangle, Oval, Curve, Regular Polygon, and Polygon tools.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] fixedLineHeight of field to trueOrFalse
```

The `fixedLineHeight` property returns or sets whether a field has uniform line height or varies the line height of each line according to largest font size that appears in the line.

It corresponds to the Fixed Line Height check box in a Field Info dialog box.

Setting the `fixedLineHeight` property to false sets the `showLines` property to false. Setting the `fixedLineHeight` to true has no effect on the `showLines`.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the freeSize of stack

The freeSize property returns the amount of free space, in bytes, in the specified stack. (Free space is created in a stack each time you add or delete an object or graphics.)

To set the freeSize property to 0, choose Compact Stack from the File menu or use the command

```
doMenu "Compact Stack"
```

from a handler or the Message box.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(6c39e3430e6576afe55d769096e6f4ad\_img.jpg\) Related Topics](#)[Find Topic](#)[Main Topics](#)

# grid

set [the] grid to trueOrFalse

The `grid` property returns or sets whether HyperCard constrains the movement of many Paint tools to eight-pixel intervals. It corresponds to the Grid command in the Options menu (which appears when you select a Paint tool).

The default value is `false`.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] height of button to integer
set [the] height of field to integer
set [the] height of card to integer
set [the] height of window to integer
get [the] height of menuBar
```

The `height` property returns or sets an integer equal to the height in pixels of an object or window. Setting the height of a button, field, or window resizes it.

HyperCard maintains the location (center coordinate) of the object, expanding or shrinking it on both sides evenly.

Setting the height of a card resizes all the cards in a stack. HyperCard forces the integer specifying the height to be an even number greater than 64 pixels.

You can't set the height of the menu bar.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] hilite of button -  
to trueOrFalse
```

The `hilite` property returns or sets whether a button is highlighted (shown in inverse video).

With check boxes and radio buttons, `hilite` determines whether the button is selected.

If a button belongs to a `family` and its `hilite` property is set to `true` from a script, the `hilite` property of each of the other buttons in the family is set to `false`.

(Click **Tips** to see a list of synonyms you can use for `hilite`.)

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**icon**

```
set [the] icon of button to pos/integer
set [the] icon of button to text
```

where text is the name of an icon.

The `icon` property returns or sets the icon displayed by a button. Setting it is the same as choosing an icon from the Button Info dialog box. HyperCard sets the function the result to "Can't find that icon." if it can't find the icon (otherwise the result is empty).

The value of the `icon` property is an integer corresponding to the ID number of an available icon resource. If a button has no icon, the `icon` property is 0. For an icon to be displayed on a button, its resource must be available in the current stack, one of the stacks currently being used, the Home stack, or HyperCard itself.



Click the goldfish bowl to see simple icon animations.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the ID of *button*

the ID of *field*

the [adjective] ID of *card*

the ID of *bkgnd*

the ID of *menu*

the ID [of HyperCard]

the ID of *window*

The ID property returns the permanent ID number of any button, field, card, background, window, or menu in the current stack.

The ID of HyperCard is WILD.

The ID of a standalone application is its creator code.

All objects except stacks have IDs. You can't change the ID number of an object, window, or menu.

If an object doesn't have a name, HyperCard returns the ID instead.

The adjectives abbreviated, long, and short return various forms of a card's ID. (Click Demo Script.)

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics



## itemDelimiter

```
set [the] itemDelimiter to char
```

The `itemDelimiter` property sets or retrieves what character HyperCard uses to separate items in a list.

The default delimiter is `comma`.

HyperCard resets `itemDelimiter` to the default delimiter at idle.

This property has no effect on comma-delimited structures such as `dateItems`, `location (loc)`, or `rectangles (rect)`.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] language to *resourceName*

where *resourceName* is a language supported by HyperCard.

The language property returns or sets the language in which HyperCard displays scripts. The default setting is English, and it's always available.

To use other languages, a script translator resource must exist in the current stack, any stack later in the message-passing order, or in HyperCard itself.

Contrast this property with the scriptingLanguage of *object*, which describes an object's scripting system.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



**left**

the **left** of *card*

```
set [the] left of button to integer
set [the] left of field to integer
set [the] left of window to integer
get [the] left of menuBar
```

The **left** property returns or sets an integer equal to item 1 of an object's rectangle. If you set the **left** of an object, it moves horizontally. The size of the object remains the same. You can't set the **left** of cards: use the **rect**, **height**, and **width** properties to resize the cards in a stack.

You can't set the **left** of the menu bar.

HyperCard determines the **left** of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

HyperCard determines the **left** of the card window relative to the top-left corner of the screen with the menu bar.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## lineSize

set [the] lineSize to *pos\integer*

where pos\integer is 1, 2, 3, 4, 6, or 8.

The lineSize property returns or sets the thickness, in pixels, of lines drawn by the Paint tools. It corresponds to the line size you select in the Line Size dialog box. (The Line Size dialog box appears when you choose Line Size from the Options menu.)

The default value is 1.

This property affects the Line, Rectangle, Rounded Rectangle, Oval, Curve, Regular Polygon, and Polygon tools.



Press and hold the mouse over this illustration to see the line sizes.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## location

```
set [the] loc of button to point
set [the] loc of field to point
set [the] loc of window to point
set [the] loc of externalWindow -
    to point
```

The `location` (or `loc`) property returns or sets the center point of a button or field and the top-left corner of a window. Setting the location of button, field, or window moves it to the new location.

HyperCard determines the location of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

For a window displaying a stack, HyperCard determines the location of the card window relative to the top-left corner of the screen with the menu bar.

HyperCard adjusts the horizontal offset of the card window to the closest multiple of 16 to the number specified.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] lockErrorDialogs to  
    trueOrFalse
```

The `lockErrorDialogs` property returns or sets whether HyperCard, on encountering an error, presents an error dialog box.

This property defaults to false at idle time, meaning that error dialogs usually appear.

When this property is set to `true`, HyperCard, on encountering an error, does not display an error dialog box; instead, it sends the message `errorDialog errorMessageText` to the current card.

`errorMessageText` contains the text of the error dialog box.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] lockMessages to trueOrFalse

The lockMessages property returns or sets whether HyperCard sends certain messages automatically.

The messages affected are closeCard, closeBackground, closeStack, openCard, openBackground, openStack, resumeStack, and suspendStack.

Setting lockMessages to true is useful when you want to go to a card to retrieve or save information, but you don't want to stay there. (The handler will run faster with lockMessages set to true.)

The default setting is false, meaning that HyperCard does send the messages. HyperCard sets lockMessages to false on idle.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)

set [the] lockRecent to trueOrFalse

The lockRecent property returns or sets whether HyperCard displays miniature pictures for the last 42 cards visited by the user (or a handler) in the Recent card dialog box. (LockRecent does not affect the trail of cards you can go back to.)

The default setting is false, meaning HyperCard does display miniature pictures of the cards visited.

Setting lockRecent to true speeds up scripts that go to cards.

HyperCard sets lockRecent to false on idle.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] lockScreen to trueOrFalse

The lockScreen property returns or sets whether HyperCard updates the screen when you go to another card. You can use lockScreen to prevent the user from seeing cards as a handler goes to them.

The default setting is false. HyperCard sets lockScreen to false on idle.

Setting lockScreen to true speeds up scripts that go to cards momentarily before returning to the source card. (HyperCard runs faster when it doesn't have to redraw the screen.)

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] lockText of field ->  
    to trueOrFalse
```

The `lockText` property returns or sets whether the user can edit the text within a specified field. It corresponds to the Lock Text check box in a Field Info dialog box. The default value is false (meaning the field is unlocked).

When a field is locked, it can receive the system messages `mouseDown`, `mouseDoubleClick`, `mouseStillDown`, and `mouseUp` when the user clicks it.

Before a field can act as a list field, its `lockText` property must be set to true.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## longWindowTitles

Card 1 of 1

set the longWindowTitles  
to trueOrFalse

The longWindowTitles returns or sets whether HyperCard displays the full path name of a stack in the title bar of all windows that contain stacks. Its default value is false.

**WARNING:** Commands or properties that reference stack windows by name won't work once you set the longWindowTitles to true because the name becomes the full path name of a stack.

For example, compare the following two commands:

```
show window "Home"  
show window "My HD:HyperCard:Home"
```

The first works only when the longWindowTitles is false, the second only when it's true. Check the value of the longWindowTitles before you use such commands or properties, or force it to be false in your stack (which takes away the feature).

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] markChar of menu/item of ~  
menu to char
```

The `markChar` property returns or sets the character used to mark a menu item. If the item has no mark, the `markChar` returns empty. Otherwise, it returns the character.

Setting the `markChar` of a menu item to a character also marks the item, that is, sets its `checkMark` property to true.

The default character used to mark menu items is the check mark, a character equal to `numToChar(18)`.

It prints in the Chicago font: ✓

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] marked of card -  
to trueOrFalse
```

The `marked` property returns or sets whether a card is marked. It corresponds to the Card Marked check box in a Card Info dialog box.

You can operate on the set of marked cards with commands such as `print`, `go`, `show`, and `sort`.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## menuMessage

```
set [the] menuMessage of menu/tem ~
of menu to ~
"messageName [parameterList]"
```

The `menuMessage` property returns or sets the `message` sent to the current card when the user chooses a menu item from a menu.

The `menuMessage` property returns empty if the menu item has no associated message.

HyperCard's default menu items have no associated messages sent to the current card unless they have been explicitly set with this property.

A `doMenu` handler can override a `menuMessage`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] messageWatcher to resourceName
```

The `messageWatcher` property returns or sets the name of the external command (or XCMD) that displays the Message Watcher window for tracing scripts. The name of HyperCard's message watcher is `MessageWatcher`.

Third-party developers might supply other message watchers that you can install into HyperCard.

Click **Tips** for information about the properties of the message watcher.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] multiple to trueOrFalse

The `multiple` property returns or sets whether HyperCard draws multiple images when the user (or a handler) drags with a Paint tool. It corresponds to the Draw Multiple command in the Options menu (which appears when you select a Paint tool).

The `multiSpace` property affects the number of multiple shapes drawn.

The `multiple` property affects the Line, Rectangle, Rounded Rectangle, Oval, and Regular Polygon tools. Its default value is false.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)

```
set [the] multipleLines of field to trueOrFalse
```

The `multipleLines` property returns or sets whether a user can extend the range of highlighted lines in a `list` *field*.

When the `multipleLines` is true, the user can extend highlighted lines either by shift-clicking any point before or after the insertion point in a field or by dragging through text.

`multipleLines` appears as the "Multiple Lines" option in the Field Info dialog box, where it is disabled if `autoSelect` is set to false.

To learn which lines are selected, get the `selectedLine` of *field*:

To learn the contents of those lines, get the `selectedText` of *field*:

To preselect lines, use

```
select line x [to y] of field
```

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] multiSpace to pos/integer

where pos/integer is a number from 1 to 100.

The multiSpace property returns or sets the minimum amount of space, in pixels, between the edges of multiple shapes drawn when the multiple property is true.

The multiSpace property affects the Line, Rectangle, Rounded Rectangle, Oval, and Regular Polygon tools. Its default value is 1.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

**name**

the [*adjective*] name of *object*

the [long] name of HyperCard

set [the] name of *object* to *text*

set [the] name of [*menu/item* of] *menu* to *text*

get [the] [English] name of [*menu/item* of] *menu*

The `name` property returns or sets the name of an object, menu item, or menu. If the element doesn't have a name, the `name` returns the ID of the object instead.

the long name of HyperCard returns the full path to HyperCard:

```
myDisk:Desktop Folder:HyperCard
Folder:HyperCard
```

You can use the adjective English to determine the names of menus and menu items if you're using a localized version of HyperCard:

```
if the English name of menuItem 5 of menu "Edit" is "Paste"...
```

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



The adjectives abbreviated, long, and short return various forms of an object's name (click Demo Script to see examples).

Note: the abbreviated name is the same as the name.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## number (property)

the number of *button*  
 the number of *field*  
 the number of *card*  
 the number of *bkgnd*  
 the number of *window*

The `number` property returns the number of a button, field, card, background, or window. (You cannot set a number.)

The number of a button or field determines whether it's on top of other buttons and fields within the same layer (background or card) that contains it. Card objects are always on top of background objects.

To change the number of a button or field, select it and choose Send Farther and Bring Closer from the Objects menu, or change its `partNumber` property.

The number of a window reflects its front to back order (similar to the number of a button or field). You can change a window's number by bringing it to the front (for example, by clicking it or by using the `show` command), or by covering it with other windows.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**number (property) (cont'd)**

The number of a card is its position within a stack. The number of a background is the order in which the background was created. To change the number of a card or background, you must use the `sort` command or cut cards and paste them into different positions within the stack.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## numberFormat

set [the] numberFormat to text  
where text is a valid number format.

The `numberFormat` property returns or sets the precision with which the results of mathematical operations are displayed in fields and the Message box. The following symbols specify the number format:

- 0 Use one zero for each digit you want to appear.
- . Use a period to indicate the position of the decimal point, if any.

# Use to indicate where you want trailing digits to appear if they have a value other than zero.

HyperCard sets the default number format to "0.#####" on idle.

**Important:** `numberFormat` takes effect only when you perform a mathematical operation on a number.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



get [the] [long|short] owner of card  
get [the] owner of window

This read-only property tells you the name of the background to which a specified card belongs, or it tells you the creator of a window.

Long owner of card returns the full path name of the background.

Short owner of card returns the short name of the background.

Owner of card returns "bkgnd" plus the leaf name of the background.

The window form returns HyperCard if it's a stack or built-in window, or the name of an XCMD if it's an external window.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## partNumber

```
set [the] partNumber of part ->  
    to integer
```

where integer is in the range 1 through  
the sum of buttons and fields in the  
current card or background

The partNumber property retrieves or  
sets the ordinal position of a button or  
field (that is, of a part) among the total  
number of all buttons and fields within  
the same domain (card or background).

Changing a part's partNumber moves it  
closer to or farther away from the front.

For example, the order of buttons and  
fields on a card is as follows:

button 1  
button 2  
field 1  
field 2  
button 3

The partNumber of field 1 is 3.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



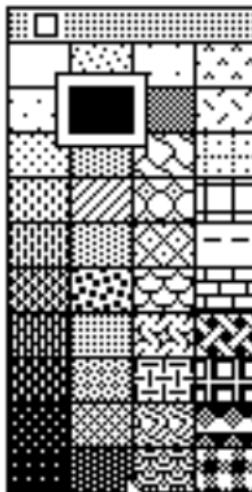
## pattern

set [the] pattern to *pos/integer*

where *pos/integer* is an integer in the range 1 through 40.

The pattern property returns or sets the current pattern used to fill shapes or to paint with the Brush tool.

If you edit one of the patterns, HyperCard saves the new pattern with the stack.



Press and hold the mouse button over a pattern to see its number.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] polySides to *pos/integer*

where *pos/integer* is 0 or a number from 3 to 50.

The `polySides` property returns or sets the number of sides of a polygon created by the Regular Polygon tool. Set `polySides` to 0 to draw a circle.

(You can also select one of six standard polygons by choosing `Polygon Sides` from the Options menu.)

If you set `polySides` to a number lower than 3 (other than 0) or higher than 50, it automatically reverts to 3 or 50.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

set [the] powerKeys to *trueOrFalse*

The powerKeys property returns or sets whether you can use keyboard shortcuts for painting actions.

HyperCard determines the default setting for powerKeys at startup (and when HyperCard resumes after being suspended) from the Power Keys option on the Preferences card of the Home stack.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## printMargins

set [the] printMargins to rectangle

The printMargins property returns or sets the value of the default margin spacing used by the print command.

The default value is 0,0,0,0.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## printTextAlign

set the printTextAlign to *alignment*

The printTextAlign property returns or sets the value of the default alignment used by the print command. The default value is left.

The printTextAlign property does not affect printing of report items.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics



## printTextFont

set the printTextFont to *font*

The `printTextFont` property returns or sets the value of the default text font used by the `print` command. The default value is Geneva.

When you print a report, HyperCard determines the default font for a report item as follows:

- If the report item is the text of a card or background field, HyperCard uses the text font of the field.

- If the report item is generated from any other HyperTalk expression, HyperCard uses the value of `printTextFont`.

In either case, the user can override the default font by selecting a font in the Item Info dialog box of the report item.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## printTextHeight

```
set the printTextHeight to pos/integer
```

The `printTextHeight` property returns or sets the value of the default text height (or line spacing) used by the `print` command. The default value is 13.

When you print a report, HyperCard determines the default text height for a report item as follows:

- If the report item is the text of a card or background field, HyperCard uses the text height of the field.
- If the report item is generated from any other HyperTalk expression, HyperCard uses the value of `printTextHeight`.

In either case, the user can override the default height by selecting a height in the Item Info dialog box of the report item.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)

## printTextSize

set the printTextSize to *pos/integer*

The `printTextSize` property returns or sets the value of the default text size (or point size) used by the `print` command. The default value is 10.

When you print a report, HyperCard determines the default text size for a report item as follows:

- If the report item is the text of a card or background field, HyperCard uses the text size of the field.

- If the report item is generated from any other HyperTalk expression, HyperCard uses the value of the `printTextSize`.

In either case, the user can override the default size by selecting a size in the Item Info dialog box of the report item.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## printTextStyle

```
set the printTextStyle to textStyleList
```

The `printTextStyle` property returns or sets the value of the default text style used by the `print` command when you print an expression. The default value is `plain`.

When you print a report, HyperCard determines the default text style for a report item as follows:

- If the report item is the text of a card or background field, HyperCard uses the text style of the field.
- If the report item is generated from any other HyperTalk expression, HyperCard uses the value of the `printTextStyle`.

In either case, the user can override the default style by selecting a style in the Item Info dialog box of the report item.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



Find Topic



Main Topics



```
set [the] rect of button to rectangle
set [the] rect of field to rectangle
set [the] rect of card to rectangle
set [the] rect of window to rectangle
get [the] rect of menuBar
```

The `rectangle` property returns or sets the rectangular coordinates of buttons, fields, cards, and windows, and it returns the coordinates of the menu bar. Setting the `rectangle` of a card changes the size of all the cards in a stack; setting the `rectangle` of the `card window` resizes the window, not the card.

HyperCard determines the rectangle of buttons, fields, cards, and its built-in windows relative to the top-left corner of the current card. HyperCard determines the `rectangle` of the card window relative to the top-left corner of the screen with the menu bar.

Note: You can't set the `rect` of an inactive card window or of the menu bar.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

the reportTemplates of *stack*

The reportTemplates property returns a return-separated list of the names of all report templates in the specified stack.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



**right**

the right of *card*

```
set [the] right of button to integer
set [the] right of field to integer
set [the] right of window to integer
get [the] right of menuBar
```

The right property returns or sets an integer equal to item 3 of an element's rectangle. If you set the right of an object, it moves horizontally; the object's size remains the same. You can't set the right of cards: use the rect, height, or width properties to resize the cards in a stack.

You can't set the right of the menu bar.

HyperCard determines the right of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

HyperCard determines the right of the card window relative to the top-left corner of the screen with the menu bar.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] script of object to text

The `script` property returns or sets a text string equal to the `script` of the specified object in the current stack or in the stack `script` of another stack.

When you set the `script` property with the `set` command, you replace the existing script entirely.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] scriptEditor to resourceName
```

The `scriptEditor` property returns or sets the name of the current script editor.

The default value of this property is `ScriptEditor`, the name of HyperCard's built-in editor and debugger. (The built-in editor and debugger are integrated.)

Because HyperCard's script editor is actually an external command (XCMD), you can replace it with your own or third-party script editors.

If HyperCard can't find a script editor with the specified name, it uses its built-in script editor.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] scriptingLanguage to  
[of object] to languageName
```

where *languageName* is HyperTalk or the name of any OSA-compliant scripting system such as AppleScript.

The `scriptingLanguage` property sets or retrieves the scripting system of the object or (when you don't use `of object`) of the message box.

*languageName* must be present in the computer's system resources.

The message box and each individual object can all respond to different scripting systems.

You can also set the `scriptingLanguage` property for an object by choosing from the Scripting Language pop-up menu at the top of each object's script editor window.

The default *languageName* is HyperTalk.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## scriptTextFont

```
set [the] scriptTextFont to font
```

The `scriptTextFont` property returns or sets the font used to display scripts in all the script editor windows.

HyperCard uses Monaco as the default font.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## scriptTextSize

```
set [the] scriptTextSize to fontSize/integer
```

The `scriptTextSize` property returns or sets the size of font used to display scripts in all the script editor windows. HyperCard uses 9 as the default size.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] scroll of field to integer
set the scroll of [the] card window to point
```

For scrolling fields, the `scroll` property returns or sets which lines of text currently appear (as indicated by the scroll bar). The `integer` represents the number of pixels that have scrolled above the top of the field's rectangle. For example, the number of lines scrolled in card field 1 equals the scroll of card field 1 divided by the `textHeight` of card field 1.

For fields, HyperCard pins the scroll to a number between 0 and the maximum value for the field.

For the `card window`, the `scroll` property returns or sets a point that specifies the current horizontal and vertical offsets of the portion of the card currently visible in the card window. It affects the card image only when the size of the window is smaller than the size of the card.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] sharedHilite of button to trueOrFalse
```

where button is a background button only.

The sharedHilite property returns or sets whether a background button shares its hilite property with every card in the background. The default value for new buttons is true.

Set sharedHilite to false if you want the hilite property for the background button maintained independently for each card.

For example, you might have one check box button in the background for marking and unmarking cards. You would set its sharedHilite to false so that its highlighting can be different on each card.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] sharedText of field to trueOrFalse
```

where field is a background field only.

The `sharedText` property returns or sets whether a background field displays the same text on every card of the background. The default value for new fields is false, meaning the text in the field can be different on each card.

Set `sharedText` to true if you want the same text to appear on each card of the background. To enter text into a background field with `sharedText` set to true, choose Background from the Edit menu, and type in the field. You can add shared text only while you're in the background layer.

HyperCard does not discard either card-specific text or shared text—it will display the appropriate text when you set the `sharedText` to true or false.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



Find Topic



Main Topics



```
set [the] showLines of field -  
to trueOrFalse
```

The `showLines` property returns or sets whether the text baselines of a field appear. It corresponds to the Show Lines check box in a Field Info dialog box.

The default value is false (meaning that the baselines are invisible).

Setting the `showLines` property to true sets the `fixedLineHeight` property to true. Setting the `showLines` to false has no effect on the `fixedLineHeight`.

Note: The `showLines` property has no effect for scrolling fields.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] showName of button to trueOrFalse
```

The `showName` property returns or sets whether HyperCard displays the name of a button (if the button has one) within the button's rectangle. It corresponds to the Show Name check box in a Button Info dialog box.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] showPict of card ->  
    to trueOrFalse  
set [the] showPict of bkgnd ->  
    to trueOrFalse
```

The `showPict` property returns or sets whether HyperCard shows any graphics or paint text for a card and background. The default value is true, meaning that all graphics and paint appear.

When a card picture or background picture is hidden and you try to use a Paint tool on it or paste a picture onto it, a dialog box appears asking if you want to make the picture visible. Clicking OK shows the picture; clicking the Cancel button cancels the action.

If you draw on a hidden picture from a handler, you do not get the dialog box, and whatever you draw will appear after you set `showPict` to true.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## size

the size of *stack*

The `size` property returns the size of the specified stack in bytes.

The `size` property can't be changed with the `set` command. It's changed only by adding and deleting objects and graphics from a stack. You must compact the stack to recover the space occupied by the deleted objects and graphics.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)

 [Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the `stacksInUse`

The `stacksInUse` property returns a return-separated list of stacks that have been inserted into the message-passing path via the `start using` command. Each stack appears in the order it will receive messages. The `stacksInUse` contains the full path names of the stacks being used.

HyperCard can use up to 16 stacks.

Note: You can't compact a stack that's being used.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## style

```
set [the] the style of button -  
    to btnStyle  
set [the] the style of field -  
    to fldStyle
```

The *style* property returns or sets the style of button or field. It corresponds to the items in the Style pop-up menu in a Button Info or a Field Info dialog box.

A button can have one of the following styles: transparent, opaque, rectangle, roundRect, checkBox, popup, oval, default, shadow, standard, or radioButton.

A field can have one of the following styles: transparent, opaque, rectangle, shadow, or scrolling.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



# suspended

the suspended

The suspended property returns whether HyperCard is currently running in the background under MultiFinder® or under System 7.X. You can switch to another program while a handler is running, and scripts will continue to run in the background.

Use the suspended property in a [handler](#) to alter the handler's behavior if it's running in the background—for example, to avoid displaying ask or answer dialog boxes.

HyperCard gives time to MultiFinder (and thus to other programs) as follows:

- After it executes each HyperTalk statement in a handler
- Whenever it rotates the busy cursor (during compacting, sorting, and printing)
- During the execution of the show cards command and the wait command

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## textAlign

```
set [the] textAlign to alignment
set [the] textAlign of field -
    to alignment
set [the] textAlign of button -
    to alignment
```

The `textAlign` property returns or sets the way Paint text, text in a field, or a button name aligns within its rectangle. It can align `left`, `right`, or `center`. The default value is `left` for fields and paint text, and `center` for buttons.

You can set the `textAlign` for radio buttons and check boxes, but the text always displays aligned `left`.

You can't set the `textAlign` for a chunk of text.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] textArrows to trueOrFalse

The `textArrows` property returns or sets whether the arrow keys move the insertion point in a field or move you through stacks.

When the `textArrows` property is false, the Right Arrow and Left Arrow keys always take you to the next and previous cards in the stack, and the Up Arrow and Down Arrow keys take you forward and backward through the cards you've already viewed.

When the `textArrows` property is true, the arrow keys move the text insertion point around in a field that you're editing or in the Message box; if you're not editing, they move you through cards.

HyperCard determines the default setting for `textArrows` at startup (and when HyperCard resumes after being suspended) from the Arrow Keys in Text option on the Preferences card of the Home stack.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## textFont

```
set [the] textFont to text
set [the] textFont of [chunk of] -
  field to text
set [the] textFont of button to text
set [the] textFont of the message box-
  to text
```

where text is the name of a font available from the current stack.

The `textFont` property returns or sets the current font of the Paint Text tool, text in a field, a button name, or the Message box.

The default value is Geneva for fields, Paint text, and the Message box; the default value is Chicago for buttons.

**Note:** if a chunk of a field contains a mixture of fonts, HyperCard returns mixed when you ask for the `textFont`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## textHeight

```
set [the] textHeight to pos/integer
set [the] textHeight of field to pos/integer
```

The `textHeight` property returns or sets the space, in pixels, between the baselines of Paint text or text in fields. (TextHeight doesn't apply to buttons because buttons can display only one line of text.)

The default value is the value of the `textSize` property plus one-third of that value.

You can't set the `textHeight` to a value less than the `textSize`.

The `textHeight` property affects a field only when the field's `fixedLineHeight` property is true.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] textSize to pos/integer
set [the] textSize of [chunk of] -
    field to pos/integer
set [the] textSize of button -
    to pos/integer
set [the] textSize of the message box -
    to pos/integer
```

The `textSize` property returns or sets the size, in pixels, of the font for Paint text, text in a field, a button name, or the Message box. The default value is 12.

**Note:** if a chunk of a field contains a mixture of sizes, HyperCard returns mixed when you ask for the `textSize`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## textStyle

```
set [the] textStyle to textStyle/elList
set [the] textStyle of [chunk of] ~
    field to textStyle/elList
set [the] textStyle of button ~
    to textStyle/elList
set the textStyle of menu/item ~
    of menu to textStyle/elList
set [the] textStyle of ~
the message box to textStyle/elList
```

The `textStyle` property returns or sets the styles in which Paint text, field text, button names, menu items, or text in the Message box appear.

The style can be a single text style or a comma-separated list of styles. The available styles are plain, bold, italic, underline, outline, shadow, condense, extend, and group. (To see group text, issue the command `show groups`.)

The default text style is plain.

**Note:** if a chunk of a field contains a mixture of styles, HyperCard returns mixed when you ask for the `textStyle`.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)

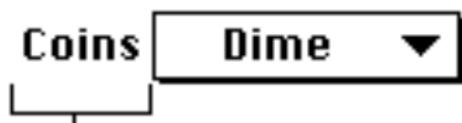


[Main Topics](#)



```
set [the] titleWidth of button to integer
```

The `titleWidth` property retrieves or sets the width in pixels of a pop-up button's title area. (The pop-up button's name appears as the menu title to the left of the collapsed button.)



`titleWidth` set to 42

You can also change the space allocated to the title by dragging the left side of the button's content area with the Button tool or by changing the Title Width setting in the Button Info dialog box.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)

**top**

the top of *card*

```
set [the] top of button to Integer
set [the] top of field to Integer
set [the] top of window to Integer
set [the] top of menuBar
```

The `top` property returns or sets an integer equal to item 2 of an element's rectangle. If you set the `top` of an object, it moves vertically; the object's size remains the same. You can't set the `top` of cards; use the `rect`, `height`, and `width` properties to resize the cards in a stack.

You can't set the `top` of the menu bar.

HyperCard determines the top of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

HyperCard determines the top of the card window relative to the top-left corner of the screen with the menu bar.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



the `topLeft` of `card`

```
set [the] topLeft of button to point
set [the] topLeft of field to point
set [the] topLeft of window to point
get [the] topLeft of menuBar
```

The `topLeft` property returns or sets a `point` equal to items 1 and 2 of an element's rectangle. If you set the `topLeft` of an object, it moves; the object's size remains the same. You can't set the `topLeft` of cards: use the `rect`, `height`, and `width` properties to resize the cards in a stack.

You can't set the `topLeft` of the menu bar.

HyperCard determines the `topLeft` of buttons, fields, and HyperCard's built-in windows relative to the top-left corner of the current card.

HyperCard determines the `topLeft` of the card window relative to the top-left corner of the screen with the menu bar.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## traceDelay

set the traceDelay to pos/integer

where pos/integer specifies ticks.

The traceDelay property returns or sets the number of ticks HyperCard pauses between each statement as it traces a handler while in the debugger.

Its default value is 0.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



set [the] userLevel to pos/integer

where pos/integer is 1, 2, 3, 4, or 5.

The userLevel property returns or sets the user level as follows:

- 1 = Browsing
- 2 = Typing
- 3 = Painting
- 4 = Authoring
- 5 = Scripting

HyperCard determines the default setting for userLevel from the Preferences card of the Home stack at startup and when HyperCard resumes after being suspended.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



## userModify

set [the] userModify to trueOrFalse

The userModify property returns or sets whether the user can temporarily type into fields, use the Paint tools, and move or delete objects in a locked stack.

HyperCard discards any changes made by the user or a handler when it leaves the card, although a handler can record the changes and save them to another stack or file.

The userModify is set to false when the user changes stacks or quits HyperCard.

Note: userModify has no effect on an unlocked stack.

---- End of Topic ----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
set [the] variableWatcher to resourceName
```

The `variableWatcher` property returns or sets the name of the external command (or XCMD) that displays the Variable Watcher window for inspecting the values of local and global variables.

The name of HyperCard's variable watcher is `VariableWatcher`.

To change the value of a variable using HyperCard's Variable Watcher, click to select a variable. Its value appears in the bottom panel. Edit the value, and press Enter to save it.

Third-party developers can supply other variable watchers that you can install into HyperCard.

Click [Tips](#) for information about the properties of the Variable Watcher.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

the version [of HyperCard]

the long version [of HyperCard]

the version of *stack*

The `version` property returns the version number of the HyperCard application that is currently running. You can't set the `version`.

The `long version` returns an 8-digit number in the form `xxyyzzrr` as follows:

`xx` The major revision number

`yy` The minor revision number

`zz` The software state, where

`80 = final`

`60 = beta`

`40 = alpha`

`20 = development`

`rr` The release number

For example, `02206044` is version 2.2 beta release 44, and `02208000` is version 2.2 final.

--- More ---

[Examples](#)

[Demo Script](#)

[Tips](#)

[Related Topics](#)



Find Topic



Main Topics



The `version` of `stack` returns a comma-separated list of five numbers in the format of Macintosh version resources:

- Item 1 is the version of HyperCard that created the stack.
- Item 2 is the version that last compacted the stack.
- Item 3 is the oldest version to change the stack since the last Compact Stack.

- Item 4 is the version that has most recently changed the stack.
- Item 5 is the number of seconds from 12:00 midnight 1/1/04 to the last time that the stack was changed.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] visible of button ->  
    to trueOrFalse  
set [the] visible of field ->  
    to trueOrFalse  
set [the] visible of window ->  
    to trueOrFalse  
set [the] visible of externalWindow ->  
    to trueOrFalse  
set [the] visible of menuBar ->  
    to trueOrFalse
```

The **visible** property returns or sets whether a button, field, window, or the menu bar is visible on the screen.

Setting the **visible** of a window to true makes it the frontmost window.

With external windows, an external command or external function must first create a window before the **visible** will work on it. Setting the **visible** of a window to true (showing it) will not create the window. Similarly, setting the **visible** of a window to false (hiding it) doesn't remove it from the window list (from memory); use the **close** command to dispose of a window.

----- End of Topic -----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

```
set [the] wideMargins of field ~  
to trueOrFalse
```

The `wideMargins` property returns or sets whether HyperCard adds extra space between the edges of a field and its text (to make it easier to read). It corresponds to the *Wide Margins* check box in a Field Info dialog box.

The default value is false.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

## width

```
set [the] width of button to integer
set [the] width of field to integer
set [the] width of card to integer
set [the] width of window to integer
get [the] width of menuBar
```

The `width` property returns or sets an integer equal to the width in pixels of the specified object or window. Setting the width of a button, field, or card window resizes it.

Setting the width of a card resizes all the cards in a stack.

Note: HyperCard restricts the width of a card to 32-pixel increments beginning from 64, the smallest width. For example, setting the width of a card to 420 results in a width of 416.

HyperCard maintains the location (center coordinate) of the object, expanding or shrinking it on both sides evenly.

----- End of Topic -----

[Examples](#)

[Demo Script](#)

[Tips](#)



[Related Topics](#)



[Find Topic](#)



[Main Topics](#)



```
get [the] zoomed of window
set [the] zoomed of window to trueOrFalse
```

When the zoomed is true, window is full sized and centered on the current display. The zoomed becomes false when the user drags the window from its original position and/or resizes it.

window is any window with a zoom box. If window is straddling two displays, setting the zoomed to true opens window to its full size and centers it on the display holding more than half of the window's image.

Clicking the zoom box toggles the zoomed between true and false.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[!\[\]\(22c56851cc9e8b6d2e4ebf7ecfe0cd32\_img.jpg\) Related Topics](#)[Find Topic](#)[Main Topics](#)

## Operators

-  
+  
\*  
/  
^

div  
mod

Click an operator for a description of it.

(Click Tips for more information about operator precedence.)

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## Operators

=, is  
<, >  
<=, >=, contains  
is in  
is not in  
is within  
is not within

Click an operator for a description of it.

(Click Tips for more information about operator precedence.)

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## Operators

not

and

or

Click an operator for a description of it.

(Click Tips for more information about operator precedence.)

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## Operators

&

&&

Click an operator for a description of it.

(Click Tips for more information about operator precedence.)

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## Operators

is a, is an  
there is a, there is an  
there is no  
there is not a

Click an operator for a description of it.

(Click Tips for more information about operator precedence.)

----- End of Topic -----

Examples

Demo Script

Tips



Related Topics



Find Topic



Main Topics



## Constants

- colon
- comma
- down
- empty
- false
- formFeed
- lineFeed
- pi
- quote
- return
- space
- tab
- true



Click a constant for a description of it.

---- End of Topic ----

[Examples](#)[Demo Script](#)[Tips](#)[Related Topics](#)[Find Topic](#)[Main Topics](#)

# Find Topic

To find topics related to a subject, type one or more words into the box and click Find. Or click one of the Sample Subjects.

**Subject:**

## Sample Subjects:

all topics

new features

&

&&

\*

+

-

## Topics found: 1

markChar

Go Back

## Related Topics

Topics related to:  
scriptTextFont

- script
- ✓ scriptEditor
- scriptTextSize
- ✓ Set the font and size of a script



Click a related topic to go to it.

Related topics that you've already visited appear with a "✓".



Return to "scriptTextFont"

# Overview of Help

**Where am I?**

You're currently in the **HyperTalk Reference** stack. This stack describes HyperTalk, HyperCard's script language. By writing your own scripts, you have much more control over HyperCard than is available using just the menu commands. You can make a stack do what you want it to do.

**What can I do?**

**Where can I go?**

If you're new to scripting, you should go through the third chapter of the *HyperCard Reference Manual* for some beginning practice.



Find Topic



Go Back

# Overview of Help

**Where am I?**

Besides using the Main Topics card to locate topics, you can:

**What can I do?**

- Click the **Find Topic** button at the bottom of every card to look for topics matching words that you enter.

**Where can I go?**

- Click the **Examples**, **Demo Scripts**, and **Tips** buttons at the bottom of a topic card to see more information. Click **Related Topics** to jump to other topics in this stack related to the one you're on.
- Click active text (text with a thick gray underline) to see pop-up definitions. This includes HyperTalk placeholders (in italics) such as *card*.



Find Topic



Go Back

# Overview of Help

**Where am I?**

HyperCard has three Help stacks. You can move between them by choosing the one you want from the Help menu:

**What can I do?**

**Where can I go?**

- The **HyperCard Help** stack describes how to browse through and author HyperCard stacks.
- This stack, the **HyperTalk Reference** stack, describes HyperTalk, the language you use to write scripts.
- The **Help Extras** stack contains glossary items, tips, and troubleshooting information used by this stack and by the HyperCard Help stack.



Find Topic



Go Back