PROPERTIES

--> (For Stack, Background, Card, Field, and Button properties, see chapter "Properties II")

A property is a category of feature of an object; that is to say, the value of a property helps to describe the current state of the object, and each property of each object has a set of possible values, one of which it has at all times.

The syntax for referring to objects is "`[the]` propertyName `of` objectRef", unless it is a "global" property in which case you just say "`[the]` propertyName".

The syntax for changing the value of a property is "`set the` propertyName `[of` objectRef`] to` newValue". You cannot "put" into a property (and you cannot "set" a container). -- However, some properties cannot be "set" at all (they are "read-only"); a couple can be "set" but can not be read!

A few properties can be set by a shortcut command, eg "lock screen" instead of "set the lockScreen to true".

A few properties revert to default values "at idle time", ie when all handlers cease. But be warned: there is a bug (or whatever you like to call it) so that during a handler if an event occurs (such as a mouse click) which queues up to trigger another handler, "idle time" will not come. So don't count on this reversion taking place.

These symbols are used to describe properties in what follows:

    • = cannot be set
    ¶ = can only be set (no get!)
    * = reverts to default at idle time
    § = shortcut command equivalent
    ∂ = default

âŒ¥2001   Global

User Abilities:
  blindTyping <boolean> -- can user type into hidden msg? (∂ from Home stack),
  powerKeys <boolean> -- can user employ painting keyboard shortcuts? (∂ from Home stack)
  textArrows <boolean> -- can user employ arrow keys to move insertion point in a field? (If false, arrow keys move between cards even if a field is open.) (∂ from Home stack)
  userModify <boolean> -- if stack is write-protected, can user type and user paint tools anyway? A stack is write-protected if on an unwritable medium, is finder-locked, or its CantModify is true; the UserModify is ignored otherwise. Changes are discarded when leaving the card. (∂ = false, reverts every time a different stack becomes current)
  userLevel <1-5> -- 1 = browsing [restricted File & Edit, Go]; 2 = typing [Font & Style added]; 3 = painting [unrestricted File & Edit, Tools added]; 4 = authoring [Objects added]; 5 = scripting [editing scripts]. (∂ at startup comes from Home stack; reset every time a different stack becomes current, by taking the lower of (1) the last "set UserLevel" call and (2) the Protect Stack dialog setting for the stack; a "set UserLevel" call higher than the current stack's Protect Stack dialog setting is remembered but will not be acted upon while in that stack)

Parsing:
  *itemDelimiter <char> -- ∂ = comma

¶*cursor <nameOrNum> -- included are: watch=4, busy, hand, arrow, ibeam=1, cross=2, plus=3, or none; or add your own as a CURS resource and refer to it by name or number. Successive calls to "busy" rotate the beachball.

*dragSpeed <num> -- bigger is faster (in pixels/sec), except that 0 (= ∂) is fastest. Bucket and Text tools are unaffected

longWindowTitles <boolean> -- ∂ = false (stackName vs. pathName). No window titles will show if you hide them or if the cd window's top is at the screen's top (as with standard windows on a Classic mac)

editBkgnd <boolean> -- toggling is just like selecting Background from the Edit menu

scriptTextFont <name> -- ∂ = monaco
scriptTextSize <#points> -- ∂ = 9
traceDelay <ticksToWait> -- ∂ = 0
debugger <name> -- ∂="ScriptEditor"
scriptEditor <name> -- ∂="ScriptEditor"
variableWatcher <name> -- ∂="VariableWatcher"
messageWatcher <name> -- ∂="MessageWatcher"

*lockScreen <boolean> -- § [un]lock screen. If true, suppresses updating of card drawing (and saves a mess of time); won't prevent msg or dialogs appearing. ∂ = false

*lockMessages <boolean> -- § [un]lock messages. If true, suppresses Open..., Close..., Suspend..., and Resume... system messages. ∂ = false

*lockRecent <boolean> -- § [un]lock recent. If true, suppresses automatic recording of recent cards; can save a mess of time during navigation. ∂ = false

*lockErrorDialogs <boolean> -- § [un]lock errorDialogs. If true, suppresses display of error messages; instead, ErrorDialog message is sent to current cd. ∂ = false

*numberFormat <str> -- 0 = must fill; # = may fill; ∂ = "0.######". What's determined is how results of mathematical operations (not, strings consisting of digits) should subsequently be rendered as strings in fields and msg; no internal accuracy (19 digits) is lost! The Variable Watcher window displays in the default format during idle time, but full accuracy is still present.

printMargins <rect> -- 72 = 1 inch; ∂ = "0,0,0,0"
printTextAlign <left|center|right> -- ∂ = left
printTextFont <name> -- ∂ = geneva
printTextSize <number> -- ∂ = 10
printTextHeight <#points> -- ∂ = 13
printTextStyle <styleList> -- ∂ = plain

NB: the Print... properties are for outputting variables, and for print report header text. § reset printing restores the defaults.

dialingVolume <0-7> -- ∂ = 7; 0 is low but not silent
dialingTime <#ticks> -- wait after dialing until closing serial connection; ∂ = 180

brush <1-32> -- numbered top-to-bottom, then left-to-right, in the Brush Shape dialog; ∂ = 7
linesize <1-8> -- (but 5=6 and 7=8); ∂ = 1
pattern <1-40> -- numbered top-to-bottom, then left-to-right, in the Patterns palette; ∂ = 12
centered, filled, grid, multiple <boolean> -- ∂ = false
multiSpace <1-100> -- number of pixels between multiple images when Multiple is true; ∂ = 1
polySides <3-50, or 0> -- ∂ = 4. "0" means a circle.
textAlign <"left|center|right"> -- ∂ = "left"
textFont <name> -- ∂ = "geneva"
textSize <num> -- ∂ = 12
textHeight <num> -- ∂ = 16
textStyle <commalist> -- ∂ = "plain"

Note: § reset paint restores the defaults for above painting properties

   •stacksInUse <return-list> -- of the "start using" stacks, in hierarchical order
   •suspended <boolean> -- is HC running the background (under MultiFinder / system 7)? This is the only way to find out; no message is sent when HC is backgrounded. Cannot be set, but under system 7 HC can background or foreground itself via DoMenu
   •address <zone:machine:program> -- of HC on the network
   •environment <"development|player"> -- meaning either HC, or else HC Player or a stack saved as an application
   •[long] version -- of HC; the Version is the popular version number (eg "2.2"), but the Long Version returns 4 packed 2-digit numbers (eg "02208000"): major version, minor version, software state (80 = final, 60=beta, 40=alpha, 20=dev), release number
   •ID -- of HC = "WILD", or of a stand-alone whose app signature has been altered
   •name -- of HC = "HyperCard", or of a stand-alone with a different name. Or you can ask for the long name, which returns the pathname of HC on disk
   scriptingLanguage -- in which commands entered in the Message box will be interpreted; ∂ = "HyperTalk". Be careful; if you say in the Message box "set the scriptingLanguage to QuicKeys" (for example) you now cannot say "set the scriptingLanguage to HyperTalk" in the Message box to get back to normal! (The solution is left as an exercise for the reader.)
   language -- for non-English systems (∂ = English). If you have a "translator" resource for another language and set the Language to that language, HyperTalk scripts will appear to be in that language (but they are still "really" in English).

✥2001   Windows

   •name <theName> -- you can refer to the window as "window theName"
   •id <theID> -- you can refer to the window as "window id theID"; IDs are unique and permanent
   •number <theNum> -- you can refer to the window as "window theNum"; reflects current front-to-back order (same as which line of Windows() its name appears in)
   •owner <theOwner> -- either "HyperCard" (for a card window and HC's floating windows) or the name of an XCMD (for an external window)
   rect <theRect> -- with respect to the topLeft of the current card, except that the "rect of cd window" is with respect to 0,0 of the screen holding the menubar. The Rect does not include any title-bar or scroll-bars or borders. Aspects of the Rect are also accessible via left, top, right, bottom, topLeft, botRight, height, width: changing any of the first six moves the whole window without resizing; changing the Height or Width leaves the topLeft unchanged. If the Rect cannot be set, the Height and Width cannot be either, but the others can because they affect only the Loc.
   loc <point> -- same as the TopLeft portion of the Rect
   visible <boolean> -- § show/hide windowRef

   NB on rect: You can change the Rect of the cd window; this can shrink the amount of card visible, but you cannot make the Width or Height of cd window larger than the stack's card size ("the Width|Height of this cd") -- no error results, but the cd window will simply reach the card's height/width and no more. A cd window can, however, be smaller than the stack's card size; viewing of the whole card, piecemeal, is possible via the Scroll pallette. NB: A cd window larger than the screen can make it very difficult to view the rest of the card. (See chapter "Properties II" on the Rect of a card.)
   scroll <point> -- if "the Width|Height of cd window" is smaller than "the Width|Height of this cd", specifies the point of the card which is at the topLeft of the window. The value of the Scroll in relation to the Rect can never be such that "blank space" would show to the right or bottom of the card; trying to set the Scroll such that this would happen will not result in an error but it will have no effect, and changing the Rect such that this would happen will automatically cause a change in the Scroll to compensate. The limiting case is when the cd window is the same size as the card; in this case the Scroll cannot be altered (from "0,0").
   zoomed <boolean> -- "true" means the window is both at full size and centered in the screen

NB on visible: You can turn a cd window invisible, but this can have seemingly weird effects; for example, that window can still be current, will be selected as you cycle through with Next Window, etc., and other buggy-looking things can occur.

Picture Window:
  •pictureWidth, •pictureHeight <numPixels> -- size of original image at normal size
  scale <-5...5> -- 0 is normal; -1 is half size, 1 is double size, etc. (I have not figured out what rule is used for determining what new value of Scroll is defaulted to when you change Scale.)
  NB on rect: Oddly, cannot set •width and •height, although nothing is to stop you from changing these values by altering the rect. (NB: If window has a grow box, you can make Rect larger than image size, but the user cannot: and if the user so much as clicks on the grow box of a window which is larger than the image size, the window snaps down to the image size.)
  globalRect <rect>, globalLoc <point> -- like Rect and Loc, but the latter measure from the topLeft of the current cd window, whereas Global... measure from topLeft of the screen holding the menubar.
  NB -- rect, loc, globalRect, and globalLoc can be set to any of four special screen constants: card, largest, deepest, main. The picture will be centered (and, with ...rect, full-zoomed) on the named screen.
  scroll <point> -- if the Width or Height of the window is smaller than the width or height of the picture image at its present scale, specifies the point of the image which is at the topLeft of the window. (If the window is the same size or larger than the image, the Scroll is "0,0" and trying to set it has no effect -- the image will be in the topLeft corner.) If the image is scaled, "point of the image" means point of the original image: eg, if the Scale is 1, changing the Scroll from 0,0 to 0,1 moves the image up 2 pixels.
  zoomed <boolean>, zoom <"in|out"> -- identical: "true"/"out" means the window is both at full size and centered in the screen; usable only if window is "zoom" style (otherwise, does nothing and returns empty)
  dithering <boolean> -- works only if 32-bit QD is present and the picture was created with bitdepth > 0
  •properties
  NB on visible: No message is sent when the user clicks a card window that was behind your Picture window; so if ensuring visibility is important to you, create the window in the floating layer, or check its Number in an Idle handler. A Picture window cannot be selected with Next Window. Since the Number cannot be set, you cannot use it to bring a Picture window of the document layer to the front; you can accomplish this, however, with show.

Palette Window:
  •buttonCount <num>
  •commands <return-list> -- the command for button n is on line n
  hilitedButton <num> -- hilites that button (without sending its message); 0 to hilite none; but if it is the kind of palette where the buttons do not remain hilite, always -1
  •properties
  cannot set •rect

Variable Watcher:
  hBarLoc, vBarLoc <num> -- ∂ = 98, 99 resp.
  •properties
  you can set •zoomed without error but it remains false

Message Watcher:
  hideIdle, hideUnused <boolean> -- ∂ = false, true resp.
  text <expr> -- complete text
  ¶nextLine <expr> -- text after current text (useful for debugging)
  •properties
  cannot set •rect
  you can set •zoomed without error but it remains false

Msg:
  textFont -- ∂ = geneva
  textSize -- ∂ = 12
  textStyle -- ∂ = plain

cannot set •rect
you can set •zoomed without error but it remains false

cannot set •rect
you can set •zoomed without error but it remains false

âŒ¥2001   Menus

Menubar:
•rect; •loc (same as TopLeft)
visible <boolean> -- §show/hide `menubar`

Menu:
•name <theName> -- you can refer to a menu as "menu theName", or ask for the long name, which returns a valid menuRef, eg <menu "Style">. You can't change a menu name, but you can delete and create whole menus.
•¶number <theNumber> -- you can refer to a menu as "menu theNumber", where TheNumber reflects the current left-to-right order. Yet, amazingly, you can neither get nor set Number as a property!!! Hence you cannot find out the number of a menu known by name without searching through "the menus". (Grrr.)
•ID -- you can refer to a menu as "menu id theID". A permanent unchanging designator reflecting HC's and the System's internal resource numbering.
enabled <boolean> -- § disable/enable. You can set this on any of HC's own menus (whether showing or not), but • not for the system menus. A disabled menu has all of its items disabled; for tear-offs, the whole palette is disabled, even though it can be shown. A script can still make a call to a menuItem which has been disabled by a script (but not a menuItem which HC itself has disabled).

MenuItem:
NB: basic reference form is: `menuitem itemNameOrNum of menu menuNameOrNum`. -- You can set features of most of HC's menus; but • not for the system menus or for Tools, Patterns, or Font. You cannot get anything ¶ for menuitems of Tools or Patterns.
name <theName> -- You can use theName for itemNameOrNum in the above reference format, or ask for the long name, which returns a full valid menuItemRef, eg <menuItem "New Stack..." of menu "File">. To obtain a whole list of names of the menuItems of a menu, just use a menuRef (eg <menu "File">); a return-list is substituted. NB that you can set the name of menuItem 1 of menu 1 ("About..."). § put theName `into|after|before` menuItemRef.
•¶number <theNumber> -- you can use theNumber for itemNameOrNum in the above reference format, where theNumber reflects the current top-to-bottom order. Yet, amazingly, you can neither get nor set Number as a property!!! Hence you cannot find out the number of a menuItem known by name without searching. (Grrr.)
menuMsg -- A menuItem that you create will do nothing unless you either (i) intercept DoMenu, or (ii) give it a menuMsg, or (iii) give it the same name as a standard HC menuItem (in which case it will have HC's default behaviour). A menuMsg can be only one "line", but this is still powerful because this can be either (a) a message you intercept, or (b) a single script line, or (c) a multi-line script in a Do statement. HC's menus do not have any default menuMsg; attaching a menuMsg to one of them overrides its default behaviour, setting the menuMsg to Empty restores it. NB that you can set the menuMsg of menuItem 1 of menu 1 ("About..."). -- § You can add menuMsgs when creating or modifying a menuItem or series of menuItems as part of the put command.
checkMark <boolean> -- whether the item is "checked" (with a markChar). True if MarkChar is not empty, and setting to false sets MarkChar to empty. HC maintains meaningful check-marking in some menus (eg Style) and you won't be able to affect these (though there is no error if you try).
markChar <char> -- the symbol to be used to "check" the item; NB that giving the MarkChar a value does in fact check the item (and sets the CheckMark to true), and making the MarkChar empty unchecks the item (and sets the CheckMark to false). Setting the checkMark to true sets the markChar

to its ∂ = numToChar(18). HC maintains meaningful check-marking in some menus (eg Style) and you won't be able to affect these (though there is no error if you try).

   enabled <boolean> -- §enable/disable menuItemRef. HC maintains meaningful enabling in some menus (eg File, Objects) and you won't be able to affect these (though there is no error if you try). A script can still make a call to a menuItem which has been disabled by a script (but not a menuItem which HC itself has disabled).

   textStyle <style list>

   cmdChar <char> -- command-key equivalent. Nothing stops you from changing any of HC's default command-key equivalents. When it receives a commandKeyDown message, HC searches its menuItems for an enabled, non-deleted equivalent, top to bottom starting with the rightmost menu; if it finds one it sends a doMenu, if not it beeps. (But âŒ˜-Q always quits.)

NB: reset menubar restores all HC's menu defaults.