COMMANDS

--> syntax of built-in "verbs"...

All commands (as opposed to keywords) generate messages, which proceed up the hierarchy from the object whose script generated them, and which must reach HC in order to execute their default behaviour.

Some commands (marked * below) are sent as messages automatically by HC in response to user actions: that is, they are also system messages. Consult chapter "Messages" for info on where and when these are sent. Here is the opening terminological passage from that chapter, explaining the relationship between commands and messages:

A message can be sent (i) by HC in response to events ("system" message), or (ii) by a script ("handler" message), or (iii) from the Message box ("msg" message). A handler message or a msg message can be made up ("user-defined") or within HC's own known vocabulary ("pre-defined"); all system messages are pre-defined. A pre-defined message can be "informational" (it is sent purely to let handlers know what is happening) or a "command" (HC has a built-in action response to it). When a message reaches HC, if its informational, it does nothing; if it is a command, HC obeys it; if it is user-defined, HC complains.

Keypress commands are not re-described below; see chapter "Messages". For commands having to do with talking to other applications, see chapter "Communications".

âŒ¥2001   Containers & Properties

   put expr [into | after | before [chunk of] container] -- default is "into msg", which will also show msg if it is hidden. "Before" and "after" do not cross boundary lines; eg, "before item 2 of myCont" means within item 2, just after the comma that delimits its start. If a chunk number involving only "line" or "item" exceeds what is present, HC will add intermediate delimiters; but, eg, if myCont contains "x", then "put myStr into item 3 of line 3 of myCont" will generate two Returns but not two Commas, and "put myWord into word 3 of myCont" just acts the same as "put myWord after myCont".
   delete chunk of container -- for "word", "item", and "line", a following delimiter, if any, is also deleted, so this is not the same as "put empty into" the same chunk
   add|subtract|multiply|divide number to|from|by|by [chunk of] container -- the answer replaces the contents of the chunk or container. HC complains if either argument is not a number (empty counts as a number). See on put, above, for how HC supplies intermediate delimiters if needed.
   get expression -- same as, "put expression into it"
   set property [of objectRef] to expression -- see chapters "Properties I" and "Properties II"

Property-setting Shortcuts:
   reset paint -- paint properties to defaults; see chapter "Properties I"
   reset printing -- print properties to defaults; see chapter "Properties I"
   enable | disable buttonRef -- sets the Enabled; see chapter "Properties II"
   lock screen; unlock screen [with visualEffect [speed] [to image]] -- sets the Lockscreen (see chapter "Properties I"), but this way of unlocking is more powerful because it can include a visual effect. If no "image" is included, the visual effect moves the locked image out of the way to reveal the new screen; if an "image" is included, the visual effect moves the locked image out of the way to reveal the designated image, but that image then just suddenly changes to the unlocked image, so this is not as effective as with the visual command, where several transformations can be happen on a single go. See below on visual, where "visualEffect", "speed", and "image" are defined.

lock | unlock error dialogs -- sets the ErrorDialogs; see chapter "Properties I"
lock | unlock recent -- sets the LockRecent; see chapter "Properties I"
lock | unlock messages -- sets the LockMessages; see chapter "Properties I"
See also Showing and Hiding (for setting the Visible), Marking (for setting the Marked), and Menus (for setting the Name, MenuMsg, and Enabled), below

âŒ¥2001   Simulating User Actions

Keypress commands are not re-described here; see chapter "Messages".
  select [before | after] [chunk | text] of fieldRef -- sets selection point; the field's lockText need not be false. If it is a "list" field, only the syntax "select line..." will do anything; what follows is about non-list fields. Selecting by item|line|word will not include the delimiter before or after. "Before|after" place the insertion point with no text selected. To select the whole text, use "text" as the chunk; otherwise the field qua object will be selected (ie with the Field Tool). (NB: to deselect all non-list field text, say select empty; to deselect all lines of a list field, say "select line 0 of...".)
  select line num of buttonRef -- for popup-buttons, changes the SelectedLine, and the text appearing in the button "at rest", if room has been allotted for this. If "num" = 0, no text will appear. No effect on non-popups.
  select btnOrFldRef -- choose button or field tool and select the designated object
  *choose <name> tool; choose tool <num> -- tools are numbered right-to-left, then top-to-bottom, in the Tools palette (grr). "Name" can be: browse, button, field, select, brush, bucket, pencil, lasso, eraser, spray, line, text, oval, rectangle, round rect, regular polygon, curve, polygon. It is an error to choose a tool that the user cannot choose at the present UserLevel.
  click at point [with modkey[, modkey[, modkey]]] -- "modKey" can be: shiftKey, optionKey, cmdKey. If the point is one where mouse-click messages would be sent if the user clicked there, a MouseDown, about 9 MouseStillDowns, and a MouseUp are sent; if you ask for two such clicks in a row, a MouseDoubleClick is also sent. However, despite the "modKey" modifier, a handler cannot learn that a modifier key was "down" (grr). The click is "theoretical"; the actual cursor does not move. Clicking outside the card window is legal but has no effect. You cannot script the Polygon tool (grr).
  drag from point to point [with modkey[, modkey[, modkey]]] -- "modKey" can be: shiftKey, optionKey, cmdKey. ShiftKey must be used to select text. The drag is "theoretical"; the actual cursor does not move. You cannot script the Curve tool (grr).
  type string [with cmdKey] -- as if user typed
  *doMenu itemName[, menuName] [without dialog] [with modkey[, modkey[, modkey]]] -- "modKey" can be: shiftKey, optionKey, cmdKey. (The modKey list is carried in param(6).) Same action as choosing the menu with the mouse, except that disabled menuItems are available, and so are deleted menuItems belonging to HC (but not menuItems taken out of action due to low UserLevel). "Without Dialog" suppresses dialogs from Delete Stack, Convert Stack, Cut Field, Clear Field. The ItemName must be exact; capitals don't count, but ellipses do ("...", three periods in a row).
  create stack name [with bgRef] [in new window] -- bypassing dialogs of New Stack menuItem. Errors go into the Result.
  save stackRef as name -- bypassing dialogs of Save A Copy menuItem, but even more powerful: need not be current stack! Errors go into the Result.
  delete buttonOrFieldRef -- the button or field must be on the current card (no error for trying, though)

âŒ¥2001   Showing and Hiding

  picture name[,source,style,visible?,bitDepth,floatingLayer?] -- any of the optional parameters may be

omitted. "Source" = file | resource | clipboard. "Style" = `rect` (no nothing); `shadow` (no nothing); `dialog` (no nothing, nice border); `plain` (titlebar, close box); `roundrect` (titlebar, close box); `windoid` (titlebar, close box); `document` (titlebar, close box, scroll & size); `zoom` (titlebar, close box, scroll & size, zoom). ∂ = "file", "zoom", "true", picture-depth, "true" for rect, shadow, windoid, "false" for others. The purpose of "BitDepth" (1,2,4,16,32) is to let you force a lower depth than the picture actually possesses (you can't force it higher), so as to use less memory at cost of lower resolution, and prevent too high depth from screwing up screen colors; 0 forces no buffering at all, which slows scrolling and zooming (and disables picture properties if memory is really tight).

  palette name[,locPoint] -- you can't create the palette invisibly, but you can use an offscreen loc.
  *close windowRef -- the window must be an XCMD window or the current cd window (when other stacks are open)
  hide `titlebar` | `menubar` | window|btn|fldRef -- sets the Visible (except for titleBar). Also sent as a *message if user hides menubar
  show `titlebar` | `menubar` | window|btn|fldRef [`at` locPoint] -- sets the Visible (except for titleBar), and Loc if "at" is used; if a window, also brings it to front. Also sent as a *message if user shows menubar.
  show|hide `cd|bg` pict; show|hide pict `of` cd|bgRef -- sets the ShowPict. Use the second syntax to affect non-current cd or bg

âŒ¥2001   Finding, Marking, & Sorting

  find [`whole` | `word` | `chars` | `string`] [`international`] str [`in` fldRef] [`of marked cds`] -- the first optional parameter alters how find interprets the presence of multiple words in "str", and when it regards the search as successful, as follows:
a) words regarded as separate strings (only the first gets marked):

(none) -- card has words beginning with every given word

word -- the given words all appear on the card as words

chars -- the given words all appear on the card anywhere
b) spaces regarded as part of a single string:

whole -- str appears between word-boundaries

string -- str appears anywhere
"International" means that diacriticals matter; you cannot make capitalization matter, though (grr). Find will find in invisible fields, which can cause mysterious-looking results, so you may want to prevent this by setting the field's DontSearch. There is no penalty for giving a non-existent fldRef. Errors and failures go into the Result. -- NB: find is fastest if the length of words of str is divisible by 3, and the longer the faster.
  [un]mark cdRef | `all cds` | `cds where` booleanExpr | `cds by finding` ... -- changes the Marked. In the "where" form, the test is carried out on every card of the stack, so "this" in the BooleanExpr refers to each card in turn. Syntax after "finding" is identical to find, above. There is no penalty for giving a non-existent fldRef. Errors and failures go into the Result.
  show|hide groups -- makes grouped text in fields visibly marked, or not

sort [[bgRef]|[`marked cds` [`of` bgRef]] [how] `by` expr -- and...

sort [`lines` | `items of`] container [how] [`by` expr] -- First form default is all the cards of the stack; "bgRef" form sorts all cards of the designated background. "How" = [`ascending` | `descending`] [`text` | `numeric` | `international` | `datetime`]; default is "ascending" & "text". Default chunk of container is "lines". Note that if "by" is used, there is essentially a fifth value-type, Boolean; "false" precedes "true" if ascending (because this is actually a "text" sort!). The "by" expr is evaluated separately for each card or line or item so as to derive the key that will be used to order it; but it can be any expression that yields the required type. It may include the special word each to designate the lines or items taken one at a time. -- As far as I can tell, HC is remarkably good about not disturbing the order and interleaving of other cards when sorting. For example, if just cards 2 and 4 are marked and you sort the marked cards, cards 1 and 3 will remain 1 and 3. But you will also go to the first sorted card (grr).

âŒ¥2001   Dialogs

answer promptStr [`with` btnName1[,btnName2[,btnName3]]] -- last btn is default-type; if no btnNames are given, "OK" will appear. BtnName of response goes into It.

answer file promptStr [`of type` type1[,type2[,...]]] -- types can be `stack`, `text`, `application`, `picture`, `paint`, or any four-letter type code; default is no filtering. Pathname of chosen file goes into It; if user cancels, It is empty and theResult is Cancel. Sorry, no way to select a folder, nor to prevent selector button from reading "Open". Selecting an alias resolves the alias. [Wow, is this wrongly documented in manual and reference stack!]

answer program promptStr [`of type` type1[,type2[,...]]] -- types are four-letter creator codes; default is no filtering. Programs running on macs networked to this one and with Program Sharing on are shown. Net-path of chosen program goes into It; if user cancels, It is empty and the Result is Cancel. Actual linking to a program on another mac is not forged (eg, user must give a password) until you try to communicate with it; see chapter "Communications".

ask promptStr [`with` defaultStr] -- for dialog with one editable text-box. Dialog contains OK (default) and Cancel; edit-box entry goes into It. If user cancels the Result is Cancel. NB that you can enlarge edit-box by including several lines in DefaultStr (if you just want a big box, concatenated Returns would do).

ask password [`clear`] promptStr [`with` defaultStr] -- just like ask, except that text in the edit-box appears as bullets. "Clear" keeps It from being encrypted; the edit-box still shows bullets. An encrypted reply is a ten-digit number.

ask file promptStr [`with` defaultStr] -- the default button will read "Save"; pathname of postulated file goes into It. If user cancels the Result is Cancel.

âŒ¥2001   Menus

See also chapter "Properties I", which describes your power over menus and the results of exercising that power.

create menu name -- menu-name appears leftmost among HC's menus, with no items in it. Will not restore one of HC's standard menus you have deleted.

delete [menuItemRef `of`] menuRef -- deleting HC's menus or menuItems leaves them available via doMenu (and they can be restored), but deleting your own wipes them right out.

enable | disable [menuItemRef `of`] menuRef -- sets the Enabled (see chapter "Properties I")

put itemList `before|into|after` [menuItemRef `of`] menuRef [`with menuMsg` msgList] -- sets Name and MenuMsg (see chapter "Properties I"). Putting into a menuItem that already exists replaces

that menuItem, but only that menuItem even if itemList has several items. For a disabled grey line in a menu, use "-" as a name. NB that giving a menuItem the same name as one of HC's standard menuItems gives it its default behaviour as well; you can therefore provide any desired subset of HC's menuItems in any desired arrangement.

  reset menubar -- restores HC's default menus and removes any other created menus. A problem arises when several stacks each want to display a private menubar, because if a stack is ill-behaved it may leave its menubar present when yours becomes current. The simplest solution is to be defensive and well-behaved yourself: on OpenStack and ResumeStack, use reset menubar and create your menubar anew, and on CloseStack and SuspendStack, use reset menubar. Unfortunately things can get insanely tricky if your stack is a "start using" stack; to see one solution, examine the Home stack script as modified by the Color Tools stack.

âŒ¥2001   Navigating

  go [cdRef | bgRef | markedCdRef] [of] [stackRef [in new window] [without dialog]] -- (I mean to say, you can use a card reference or a stack reference or both, but if you use both you have to connect them by "of"). MarkedCdRefs are restricted to simple ordinal specifications of "marked cd" (ie, unlike cdRef, you can't relate ordination to a bg; grrr). Omitting cdRef sends you to the first cd of whatever you do specify. "Without dialog" prevents SFOpen from appearing if stack isn't in search paths; instead, error is reported in the Result.
  go back | forth | home | cdAdjective -- "back" and "forth" move among recently visited cards. "cdAdjective" is just a shortcut syntax for simple word-ordinal cdRefs in the same stack: you can say "first", "third", "next", etc.
  push whatCard -- you can push any card that you can specify with the first syntax of go, above. Places a specifier for the card on top of a special "push-list".
  pop cd [into | before | after container] -- removes the card specifier currently on top of the "push-list" (see push, above). If no container is specified, also executes a go to that card; if a container is specified, just performs a put. (The second syntax is handy when you want to go in a new window, because you can't do that with pop. Grr.) The bottom-most item in the "push-list" is cd 1 of the Home stack, and cannot be removed.
  show [[howMany] | all | marked] cds -- default is "all". Goes quickly to each of the specified number of cards in turn, starting with the next; no messages are sent, but a knowledge of each card is apparently cached in RAM, so that subsequent navigation will be faster.

âŒ¥2001   Import & Export

Text Files:
  open file pathName -- for text files; a file must be open before you can read or write it. Creates the file if it can't find it.
  read from file pathName [at [-]startNum] [[for numChars]|[until char]] -- you must use either "for" or "until" syntax. "-" lets you start a certain number of chars from the end. "char" can be a literal character or tab, return, space, quote, formFeed, eof, end; the "until" char is included in the read. Nulls are read as spaces. Result goes into It. There used to be a limit of 30K chars, but this is now lifted; however, the text of a field is still limited to 30K.
  write text to file pathName [at [-]startNum|eof] -- "-" lets you start a certain number of chars from the end. If you don't use "at", writing starts from the end of the previous write (from start of file if there was no previous write). If a series of writes contains no use of "at", the file is replaced, but if even one "at" is used, any previous and not overwritten text remains.

close file pathName -- signals done reading or writing; important to do this

Printing:
   print expr -- if expr is a single fieldRef, styled text will be used
   print `all` | `marked` | howmany `cds` -- prints card images, using the current Print Stack and Page Setup settings; if "howmany" used, cards will be sequential starting with the current card
   print `cd` | cdRef [`from` point `to` point] -- prints card image, using the current Print Stack and Page Setup settings, with option to print just a certain rectangle of the card
   open printing [`with dialog`] -- for printing cards not in sequence; all "print" card commands will be collected until "close printing". "With dialog" brings up Print Stack dialog, where user can change settings.
   open report printing [`with dialog`] [`with template` name] -- for printing cards according to a report template; all "print" card commands will be collected until "close printing". "With dialog" brings up Print Report dialog, where user can change settings. Either the command or the user must specify a template, or else one must have been used earlier (and it will be used now by default).
   copy template name `to` stackRef -- for getting a report template from current stack into another
   close printing -- don't forget this after an "open printing"

Paint:
   import paint `from file` name; export paint `to file` name -- just like the corresponding menu items (but without dialog), and only available when they are

Telephone:
   dial num [`with modem` [commandString]] -- default commandString is "ATS0=0DT", and be sure to enclose yours in quotes so HC doesn't try to evaluate it first! If "modem" is used, commands go out the serial port; if not, the speaker makes touchtones.

âŒ¥2001   Special Effects

   visual name [[`very`] `fast` | `slow`][`to` image] -- visual effects pile up within a handler until the next go, when they are executed in sequence. The effect is a way of revealing the next image from the current one. "Image" can be `black`, `white`, `gray`, `inverse` (of card we are going to), `card` (which we are going to); "card" is the default. If you change to a non-card image and do not change to the card with another effect, the image is instantly replaced by the card. Effect names are:

`plain` (the default)
   `zoom open|close` (from the clickloc, but to the center`)`

`iris open|close; barn door open|close`

`push|wipe|scroll right|left|up|down` (push = both cards move; wipe = both cards still; scroll = next card moves)
   `dissolve; checkerboard; venetian blinds`
   `shrink to|from top|center|bottom`
   flash [howMany] -- default is 3
   beep [howMany] -- default is 1
   wait howMany [`ticks` | `secs`] -- default is ticks
   wait `until` | `while` boolean

play resourceName [`tempo` speed] [noteString] -- included resources are `Harpischord, Flute, Boing`; or play stop to kill the currently playing string. 200 is a moderate tempo.
Notes are `abcdefg`[#|b] with octave (4=middle); or you can use numbers if you find that easier, with 60 = middle C; `r` = rest. Durations are `whqestx`[.|3]. Note and duration should be together, followed by space, then next note and duration, etc. Duration and octave remain in effect until altered (inital defaut is `4q`). NB: play name 0 will preload the resource. HC usually continues with actions while the sound plays. Errors go into the Result.
   edit script `of` objectRef
   debug checkpoint -- there are other "debug" commands but they won't *&^*%*&(* document them
   start|stop using stackRef -- insert or remove stack script in hierarchy, just following current stack (see chapter "Messages"). All resources in a "using" stack also become available.
   convert [chunk `of`] expr [[`from` format] [`and` format]] `to` format [`and` format] -- for date-time format conversions; "and" is for handling date and time separately. Formats are: `seconds, long date, short date, abbr date, long time, short time, dateItems` (= yr,mo,day,hr,min,sec,dayNum). Conversion is performed in place, except that conversion of a literal goes into It. Date-time formats respond to itl resources, so calculations and storage should use "seconds" or "dateItems", which are reliable across localisations.
   help -- goes to first cd of "HyperCard Help"