BUILT-IN FUNCTIONS

A built-in function is a calculation on parameters or an information-fetching routine which is evaluated before the line in which it appears is acted upon.

If a built-in function takes no parameters or one parameter, you can call it either with the syntax the funcName or [the] funcName of theParameter, or with the syntax funcName() or funcName(theParameter). But if it takes more than one parameter you can only use the second, ()-form, with the parameters separated by commas.

This double syntax leads to a curious phenomenon about the relationship between your own function handlers and the built-in functions. Your own functions can only be called with the second, ()-form. Your functions can also have the same name as a built-in function, and can thus override a built-in function if encountered in the message hierarchy before the function call reaches HC. Therefore, if you have such a handler, the calling syntax builtInFunc() can access it and override HC's built-in function, but the other calling syntax (if permitted), the builtInFunc, will bypass your handler and go direct to HC.

âŒ¥2001 Mouse

mouse <up | down> -- current
mouseClick <boolean> -- reports whether the mouse has been clicked (regardless of how many times) since either (i) the start of the current handler-chain or (ii) the last call to MouseClick within the current handler-chain. This has the additional (possibly very useful) consequence that simply calling MouseClick during a handler clears all clicks from the event chain (ie, they don't generate a mouseUp); mouse clicks during handlers that are not intercepted in this way do generate mouseUp (one for the lot of them)
mouseH, mouseV <int>; mouseLoc <pt> -- current
clickH, clickV <int>; clickLoc <pt> -- reports position mouse was most recently clicked ever (ie, not just within this handler); the click must be within the card, below the title-bar

âŒ¥2001 Field Text

clickChunk, selectedChunk, foundChunk <"char x to y of card|bkgnd field z"> -- if x > y, insertion pt is after y. ClickChunk refers to the word (or grouped text) in field where the most recent ever mouse-click was; if the click was after all text, x = y = the number of chars of the field + 1
clickText, selectedText, foundText <textString> -- just like using the Value function on the ...Chunk functions above
clickLine, selectedLine, foundLine <"line x of card|bkgnd field z"> -- ClickLine refers to the line in field where the most recent ever mouse-click was; SelectedLine refers only to the first line any part of which is included in the selection
selectedField, foundField <"card|bkgnd field z">
selectedLoc <pt> -- the left bottom (!) of the selected text
NB: these all return Empty if no applicable entity exists. Note that a number of user actions can cause there to be no selection; a useful trick, though, is that pushing a button whose AutoHilite is False will not deselect the selection (though it will lose the Found box).

âŒ¥2001 List Fields

A list field is one whose lockText, dontWrap, and autoSelect (and possibly multipleLines) are true. Clicking (or shift-clicking) automatically selects whole line(s). Clicking elsewhere does not deselect. Two Selected... functions are modified to work with list fields by adding a FieldRef parameter; omitting this parameter reduces the function to a normal field function, which will not see the "selection" in the list field:

selectedLine(fieldRef) <"line x to y of card|bkgnd field z">
selectedText(fieldRef) <textString>

The SelectedField does not apply.

âŒ¥2001 PopUp Buttons

A popup button has "contents" which determine its menu items. Two Selected... functions are modified to work with popup buttons by adding a ButtonRef parameter; omitting this parameter reduces the function to a field function:

selectedLine(buttonRef) <"line x of card|bkgnd button z">
selectedText(buttonRef) <textString>

âŒ¥2001 Radio Buttons

A radio button can be one of a "family", in which case clicking on any member of that family sets its Hilite to true and that of the family's other members to false. To determine which of a family is hilited, use:

selectedButton(`cd|bg family` familyNum) <"card|bkgnd button z">

âŒ¥2001 Keys

commandKey, optionKey, shiftKey <up | down>

âŒ¥2001 Text

length(string) <integer>
offset(findString,inString) <integer> -- returns 0 if InString doesn't contain FindString
charToNum(char) <integer> -- ignores all but first char of string param
numToChar(num) <char>

âŒ¥2001 Math

random(integer) <integer between 1 and given num>
annuity(rate, numPeriods), compound(rate, numPeriods)
average(comma-list), sum(comma-list)
max(comma-list), min(comma-list)
sin(radians), cos(radians), tan(radians), atan(number) <radians>
sqrt(), trunc(), abs()
round() -- returns nearest integer; if an odd and even integer are equally distant (ie the parameter has ".5" at the end), returns the even integer
exp() -- e^x
exp1() -- (e^x)-1
exp2() -- 2^x
ln() -- e^result = x
ln1() -- e^result = x+1
log2() -- 2^result = x

âŒ¥2001 Environment

`[abbr|long]` date <"9/2/94">, <"Wed, 9 Feb 1994">, <"Wednesday, 9 February 1994"> -- the result format depends on itl-resource settings (note how on my British machine, "9/2" means February 9th, and in the others the day precedes the month; on American machines the order is different, and on Swedish machines the comma is not present). Use convert to get a format from which individual items can be extracted
`[long]` time <"10:50 AM">, <"10:50:10 AM"> -- again, use convert to get a reliable format for extracting items or doing calculations
secs -- since 1/1/04 12:00 AM
ticks -- since the last startup/restart; roughly 1/60 sec
sound <"soundName" (or "done" if none playing)
tool <"x tool"> -- x can be browse, button, field, select, brush, bucket, pencil, lasso, eraser, spray, line, text, oval, rectangle, round rect, regular polygon, curve, polygon
menus <return-list> -- the list on my computer starts with "Apple" and ends with "System Help", "Keyboards", "Application"
windows <return-list> -- windowNames in front-to-back order, including invisible windows which themselves include various palettes and windows belonging to HC itself; the list on my computer at this moment goes: Message, Message Watcher, Variable Watcher, Scroll, FatBits, Patterns, Tools, bird (a "picture" window), HyperTalk Reference (the only open stack), although only the last is visible
stacks <return-list> -- full pathNames for all open stacks in front-to-back order; an "open" stack need not have its window visible, but it is not the same as a "start using" stack, for which you need to check to check the global Property, the StacksInUse (and why "stacks" is a function but "stacksInUse" is a property is known only to God)
programs <return-list> -- progNames of sys-7-friendly apps running on the same computer; may include extensions not appearing in the Application menu
programs `of machine` "zone:machine" <return-list> -- of AppleEvent-aware apps running on the other computer
screenRect <rect> -- for the monitor displaying the largest proportion of the current card windows, measured from the top-left of the monitor holding the menubar
diskSpace <#bytes> -- of the disk containing the current stack, or use diskSpace `of disk` "diskName" to check any disk (too bad there's no "disks" function!!!)
heapSpace, stackSpace <#bytes> -- HeapSpace is for general issues of how much free RAM room HC has; StackSpace is not the normal "stack", but some sort of internal structure, and I'm not sure what you would learn from checking it
systemVersion <#.##> -- "7.10" on my machine

âŒ¥2001 Other

destination <pathName> -- usable in handlers for CloseStack, CloseBackground, CloseCard, and SuspendStack to find out where we'll be going after this

number(<chunks> | <objects>) -- <chunks> must name a container too, using `in|of`, and can be `chars, words, items, lines, menuItems`; <objects> can be `[bg|cd] btns|flds|parts, bgs, cds, marked cds, cds of` <background-specifier>, `menus, windows`

value(<expr>) -- forces extra level of evaluation; usually so you can use the name of a container and get the contents of that container evaluated, but also can be used to get quotes off the outside of a literal

param(num); paramCount; params <comma-list> -- used where you don't know how many parameters to expect. Param(0) is the message (handler name); Params lists them all, starting with 0.