MESSAGES

--> the message hierarchy, and what messages HC sends...

A little terminology: 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). Command messages are marked * below; consult chapter "Commands" to find out what HC's built-in response is.

âŒ¥2001　　　　　First Target

   The first place a system message is sent (by HyperCard) depends upon what the message is; it is always a button, field, or card (as described below).

   The first place a message from within a handler is sent is the object containing the handler -- unless the message is sent with the "send" command, in which case, you can have it start at any object in the current stack, or the stack script of another stack. You can also "send" a message direct "to HyperCard", bypassing the Hierarchy.

   The first place message from the message box is sent is the current card -- unless the message is sent with the "send" command, in which case, you can have it start at any object in the current stack, or the stack script of another stack. You can also "send" a message direct "to HyperCard", bypassing the Hierarchy.

âŒ¥2001　　　　　The Hierarchy

Static Path:
   This is the normal order in which HC seeks a handler for a message or function call; it runs along this hierarchy looking for a handler, and stops when it finds one or continues if it finds one and is told to "pass" the message.

　　　button/field
(Only certain system messages come here: mouse events on a button/field; messages with Button or Field in the name; and KeyDown, CommandKeyDown, and TabKey are sent to a field with the insertion point.)
　　　current card
(The recipient for all other system messages.)
　　　current background
　　　current stack
　　　current stack XCMDs
　　　most recently added "start using" stack
　　　most recently added "start using" stack XCMDs
　　　...
　　　first added "start using" stack
　　　first added "start using" stack XCMDs
　　　Home stack
　　　Home stack XCMDs
　　　HyperCard XCMDs
　　　System XCMDs
　　　HyperCard
(At this point, if it's a pre-defined informational message, HC does nothing; if it's a pre-defined command, HC obeys it; otherwise [ie if it's a user-defined message], HC complains.)

Dynamic Path:
   When HC is executing a handler not on the static path of the current card, and if a message gets through the first stack level (including XCMDs) without being handled (or is handled there but "pass"ed), the "dynamic path" may be invoked, meaning that HC tries again, routing the message from the bottom of the other hierarchy. (Caution! The HyperTalk Reference stack states that a message must pass unhandled all the way to HyperCard before the dynamic

path is invoked. This is not true!)

   The rule for whether the dynamic path is invoked has to do with whether two different stacks are involved. If two different stacks are involved, the dynamic path is invoked either if the first path reaches the stack unhandled or if the first path is "pass"ed. If only one stack is involved, the dynamic path is invoked only if the first path reaches the stack totally unhandled.

   Here are examples.

   Send:

(1) A button has a mouseUp handler 'send "MyMessage" to stack "OtherStack"'. Here is the hierarchy, whether the message is unhandled or it is handled but "pass"ed:
        other stack
        other stack XCMDs
        original card
        original background
        original stack
        original stack XCMDs
        (etc.)

(2) A button on cd 1 has a mouseUp handler 'send "MyMessage" to cd 2'. Now things are different from situation (1). If MyMessage is handled at all, including if it is "pass"ed, on cd 2, or its background, or its stack (or stack XCMDs), the message just stops or else continues on the static path. If no handler for it at all is present in any of those places, it is sent to cd 1 (and then on up from there).

   Go:

(1) A button has a mouseUp Handler 'go stack "OtherStack" / MyMessage'. Here is the hierarchy, whether the message is unhandled or it is handled but "pass"ed:
        original button
        original card
        original background
        original stack
        original stack XCMDs
        other stack's card
        other stack's background
        other stack
        other stack XCMDs
        (etc.)

(2) A button on cd 1 has a mouseUp Handler 'go cd 2 / MyMessage'. Now things are different from situation (1). If MyMessage is handled at all, including if it is "pass"ed, on the button, or cd 1, or its background, or its stack (or stack XCMDs), the message just stops or else continues on the static path. If no handler for it at all is present in any of those places, it is sent to cd 2 (and then on up from there).

âŒ¥2001   Button Messages

newButton sent on creation; the button has no script (unless pasted)
deleteButton sent just before the button vanishes

mouseEnter, mouseWithin (sent repeatedly), mouseLeave. With an Oval button, the "boundary" is the oval.

mouseDown, mouseStillDown (sent repeatedly), mouseUp; mouseDoubleClick which is sent only after the three messages for the first click have been sent, and if handlers for these involve dialogs ("answer" etc.) the MouseDoubleClick is never sent. With a Popup button, no MouseDoubleClick is sent, no MouseStillDown message is sent, and MouseUp is sent only if the mouse is released with the "menu". With an Oval button, the "boundary" is the oval. None of these messages is sent if the mouse is over a disabled button.

âŒ¥2001   Field Messages

newField sent on creation; the field has no script (unless pasted)
deleteField sent just before the field vanishes

mouseEnter, mouseWithin (sent repeatedly), mouseLeave

mouseDown, mouseStillDown (sent repeatedly), mouseUp; mouseDoubleClick -- all sent only to locked fields, except that MouseDown can be sent to an unlocked field by âŒ˜-clicking. MouseDoubleClick is sent only after the three messages for the first click have been sent, and if handlers for these involve dialogs ("answer" etc.) the MouseDoubleClick is never sent

openField sent to unlocked field when insertion point comes to be in it by mouseclick or tabkey, but not by HyperTalk e.g. "select text of..."
exitField sent when user leaves field if no text was changed
closeField sent when user leaves field if text was changed

*keyDown, *commandKeyDown
*tabKey, *enterInField, *returnInField -- a KeyDown is sent before each. After a ReturnInField a TabKey will be sent if not intercepted and the insertion point is on the last line of a non-scrolling AutoKey field. After an EnterInField or TabKey the field will be closed and ExitField or CloseField will be sent; in the case of TabKey, the same or another field will then have its text selected, and OpenField will be sent.

âŒ¥2001   Card Messages

These twelve are shown in the order of sending (whichever are appropriate):
closeCard, closeBackground, closeStack
deleteCard, deleteBackground, deleteStack but if you are deleting a stack, DeleteCard and DeleteBackground are not sent
newStack, newBackground, newCard
openStack, openBackground, openCard

startUp sent to first cd to be shown, but before it actually appears (before all Open... messages). Note that the Home stack performs some valuable tasks on StartUp (setting userLevel, fetching search paths, etc.) so it may be useful to Pass StartUp.
quit sent after all Close... messages

moveWindow sent when the card window is moved by user or handler
sizeWindow sent when the card window is resized by user or handler
close sent before all Close... messages when closing this stack's card window, by user clicking in close box or by handler (but not when Close is chosen from the File menu); the window will not be closed unless the message reaches HyperCard. This is a subset of "close" messages generally, so you need to check the parameter to see if it is "card window"
suspendStack sent when moving this stack's card window back from the front (even if screen is locked); no Close... messages are sent
resumeStack sent when bringing this stack's card window to front from behind (even if screen is locked); no Open... messages are sent

mouseDown, mouseStillDown (sent repeatedly), mouseUp; mouseDoubleClick which is sent only after the three messages for the first click have been sent, and if handlers for these involve dialogs ("answer" etc.) the MouseDoubleClick is never sent

*keyDown, *commandKeyDown -- when it receives a commandKeyDown message, HC searches its menuItems for a visible, non-deleted equivalent, top to bottom starting with the rightmost menu; if it finds one it sends a doMenu, if not it beeps
*tabKey, *enterKey, *returnKey, *controlKey, *arrowKey, *functionKey -- a KeyDown is sent before each. If the insertion point is in a field, EnterKey and ReturnKey are not sent (EnterInField and ReturnInField are sent instead); the prior KeyDown goes to the field, and so does TabKey, but ControlKey, FunctionKey and ArrowKey are sent to the card, bypassing the field.   If F1, F2, F3, or F4 is pressed and FunctionKey reaches HyperCard, the effect will be the

sending of a DoMenu message for Undo, Cut, Copy, Paste respectively.

*doMenu sent when a menu item is chosen, via mouse or command key (in the latter case a CommandKeyDown will have preceded). The menu item will not be responded to unless the DoMenu message reaches HyperCard. If a menuItem with a menuMessage was chosen, that menuMessage will then be sent. Some of HC's menu items also have menuMessages which are then sent, and nothing will happen unless those messages reach HyperCard, namely:
*help sent by the Help menu item
*choose sent by any item in the Tools menu (including âŒ˜-TAB(-TAB-TAB) key shortcuts, which send CommandKeyDown first)

*show menubar, *hide menubar sent as a result of pressing âŒ˜-space (a CommandKeyDown precedes); the menubar will not be affected unless the message reaches HyperCard. These are special cases of the show/hide commands, so the parameter must be checked to see if it is menubar if you want to intervene

errorDialog sent if the LockErrorDialogs is true and an error occurs to which HyperCard would otherwise have responded with a dialog box
appleEvent sent if an Apple Event arrives; the event will not be executed unless it reaches HyperCard

openPalette, openPicture; mouseDownInPicture, mouseUpInPicture; closePalette, closePicture (all products of the included Picture and Palette XCMDs) -- ClosePalette and ClosePicture are sent as byproducts when the user clicks in the close box, just before the window disappears (you can handle but not prevent the closing); if a handler closes with "close window...", though, only Close is sent

idle sent repeatedly when Browse tool is current and no handler is executing

[resume, suspend sent under System 6 Finder (not MultiFinder) on launch or and return from another app. Under MultiFinder and System 7, no message is sent in such cases, nor when the user switches apps; if your stack needs to know whether HyperCard is backgrounded it can check the Suspended property in an Idle handler.]

âŒ˜2001   Message Parameters

Your messages can include parameters by using the syntax "MyMessage parameter[,parameter...]" (for a handler message) or "MyFunction(parameter[,parameter...])" (for a function).

Your handler can receive parameters by starting with syntax "on MessageName var1[,var2...]", "function FuncName var1[,var2...]". (The variable names are local and can be anything you like.) Or, you can use the various Param functions.

Some system messages include parameters, as follows:
  errorDialog TextThatWouldHaveAppearedInDialog
  appleEvent Class,ID,Sender
  doMenu MenuItem,MenuName
  show/hide "Menubar"OrWhatever
  close "card window"OrWhatever
  choose "tool",ToolNumber
  open/close-Palette/Picture WindowName,WindowID
  mouseDownInPicture / mouseUpInPicture WindowName,Point
  arrowKey "left"|"right"|"up"|"down"
  keyDown KeyLetter
  commandKeyDown KeyLetter
  functionKey KeyNumber (1-15)
  controlKey KeyNumber 1-127 according to the following mapping, showing key(s) typed with the controlKey down and the KeyNumber code generated: a/Home=1; b=2; c/Enter=3; d/End=4; e/Help=5; f=6; g=7; h/Delete=8; i/Tab=9; j=10; k/PageUp=11; l/PageDown=12; m/Return=13; n=14; o=15; p/Function=16; q=17; r=18; s=19; t=20; u=21; v=22; w=23; x=24; y=25; z=26; Esc/Clear/"["=27; LeftArrow/"\"=28; RightArrow/"]"=29; UpArrow=30; DownArrow/"-"=31; '=39; *=42; +=43; ","=44; "-"=45; "."=46; "/"=47; 0=48; 1=49; 2=50; 3=51; 4=52; 5=53; 6=54; 7=55; 8=56; 9=57; ";"=59; "="=61; ~=96; ForwardDelete=127