COMMUNICATIONS

--> talking to other applications...

âŒ¥2001 Apple Events

SENDING, SAME MAC:

You can send basic Apple Events on the same Mac via the following commands:

open [docPath `with`] appPath -- sends an 'oapp' or 'odoc', as appropriate, telling an application to launch or to open a document. Errors go into the Result. A valuable use is to bring HC to the foreground by telling itself ("this program") to open.
close [docPath `in|with`] appPath -- sends a 'quit' or 'clos', as appropriate, telling an application to close a document or to quit altogether. Unfortunately 'clos' is not a required Apple Event so not every app need be able to handle it. Errors go into the Result. (Do not type the words Close "Finder" into the message box in a misguided attempt to test this command!)
print docPath `with` appPath -- sends a 'pdoc', telling an application to print a document.

SENDING, OTHER MAC:

You can send basic Apple Events to an already running program on the same or a remote Mac via the following commands. If on another Mac, the syntax "[zoneName:]machineName:theProgram") navigates the net. You can specify the program by name or by "ID", meaning its creator signature.

send expression `to  this program | program [ID]` theProgram `[without reply]` -- sends a 'dosc' (doScript). This presumes that the program has some native scripting dialect; what you send should be a command or series of commands in this dialect. Reply or error goes into the Result.
request expression `of|from  this program | program [ID]` theProgram -- sends an 'eval' (evalute). The evaluated expression (ie, the direct param of the reply) goes into It. Errors go into the Result.

RECEIVING AND REPLYING:

Incoming Apple Events trigger an appleEvent system message.You can intercept and deal with such events yourself if you like. This ability was more important under 2.1, when HC could respond automatically only to the basic four Apple Events, plus 'dosc' and 'eval'; it was useful then to be able to define your own Events. Now, however, HC responds automatically to over 150 events, so you may want to check this before interfering. HC won't do its automatic response unless the appleEvent message is passed all the way to HC.
When an Apple Event comes in, class and id (and sender) are passed as parameters to the appleEvent message. If you handle the event yourself and you need more information (so-called "parameters"), you use the request command; to reply to it, you use the reply command, as follows:

request ae data -- fetches the "direct parameter" (the one whose keyword is "----") into It. Errors go into the Result.
request ae data with keyword theKeyword -- fetches the parameter whose keyword is "theKeyword", into It. Errors go into the Result. A special case is the "rtid" keyword, which gives the returnId; you can ask for this with request ae return id. You can fetch the sender's signature ID (as opposed to name) with request ae sender id.
reply replyString [with keyword theKeyword] -- if you omit the "with keyword" option, the reply expression is sent as the "direct parameter". For keyword "errs" you can use the cuter form reply error replyString.

âŒ¥2001 OSA Dialects

If OSA dialect extensions installed themselves onto you Mac during startup (as do AppleScript, QuicKeys, Frontier's UserTalk), you can send commands directly into the system in those dialects. (To find out what dialects are in your system, open a script and examine the pop-up dialect menu at the top of the window.) The "keeper" of the dialect can thus be made to exercise its particular powers on your machine: AppleScript could be made to send some commands to a scriptable app such as Scriptable Text Editor or to take advantage of an 'osax' you happen to have; or, QuicKeys could be made to type an expression in any app whatever. There are two cases: the dialect commands live in a container (or are constructed as an expression), or, the entire script of an object is written in a dialect.

CONTAINER:
You can send expressions and the contents of containers with the do command:

do expressionOrContainer as dialectName -- Sends commands to the "keeper" of the dialect in question. If the container has several lines (a field, perhaps), the whole container will be sent as a script.

SCRIPT:
You can write an object's entire script in an OSA dialect, using the pop-up menu at the top of the script window, or the script property the scriptingLanguage. Note that dialects can't be mixed in one script, though scripts of different objects in one stack can have different dialects.

Scripts in foreign dialects don't work within HC like HyperTalk scripts. The script is not "pretty-printed" until you close the window. At that point it is sent to the "keeper" of the dialect for checking and compilation; so problems can arise if compilation conditions are not right (for example, AppleScript can't compile when there is a reference to a program on a remote computer unless the program is running). Also, foreign dialect scripts can be slow to open (while the script "decompiles"), and if you try to open them when the dialect is not present on the computer or under earlier versions of HC, there can be trouble (like, loss of the script).

How you "talk" to a script in another dialect depends on whether it can receive function messages or not.

If a dialect cannot receive function messages, like QuicKeys, the only way to deal with it is to run the entire script of an object by saying send "run" to theObject (or by letting the message run reach it in the hierarchy). You might think this means there is no good reason to keep such commands in a script (why not just keep them in a container, as above?), but I suspect that there is some speed value, since the script is compiled.

If a dialect can receive function messages, like AppleScript, you can use messages from within HyperTalk to call the functions. For example, if the hyperTalk message MyMessage reaches an AppleScript object script containing a handler "on MyMessage()", the handler will run.

When an AppleScript handler is running, the default application is automatically HyperCard; you don't have to "tell HyperCard" anything. HyperCard is scriptable, so AppleScript can talk to it, either using "do script" or using the various Apple Events that HyperCard recognises. Values can be passed as parameters to messages, or using HyperCard's scripting object type "variable" (meaning a global), or by telling HyperCard to set the contents of a container; but watch out, HyperCard is expecting strings only, whereas AppleScript variables can be of other types and may have to be coerced or otherwise processed.