XCMD and XFCN: The Magic Hooks That Extend HyperTalk

This documentation by Ted Kaehler, 1 August 1987
For HyperCard 1.0.1

Dan Winkler has created an interface that allows powerful new commands to be added to HyperCard "in the field." When a command in a script cannot be found, HyperCard looks for a resource of type XCMD with the same name as the unknown command. Likewise, when a function cannot be found, HyperCard looks for a resource of type XFCN. Whenever a handler (for a command or function) is not found in a stack script, HyperCard immediately looks for an XCMD or XFCN in that stack. The total inheritance order is:

Button (or Field)
Card
Stack
stack XCMD
Home
home XCMD
XCMD in HyperCard application file
HyperCard command

An XCMD or XFCN is a code resource with no header bytes (just like a desk accessory). You can move them from file to file with ResEdit or with any other resource moving tool.

The only thing passed into an XCMD or XFCN is a pointer to a XCmdBlock. It looks like this:

```
XCmdPtr = ^XCmdBlock;
XCmdBlock =
  RECORD
     paramCount: INTEGER; { the number of arguments}
     params: ARRAY[1..16] OF Handle;  { the arguments}
     returnValue: Handle; { the result of this XCMD}
     passFlag: BOOLEAN; { pass the message on?}

     entryPoint: ProcPtr; { call back to HyperCard}
     request:  INTEGER; { what you want to do}
     result: INTEGER; { the answer it gives}
     inArgs: ARRAY[1..8] OF LongInt;  { args XCMD sends HyperCard}
     outArgs: ARRAY[1..4] OF LongInt;  { answer HyperCard sends back}
  END;
```

You read the arguments (they are handles to zero terminated strings), do whatever the purpose of this XCMD is, and optionally store a result into returnValue. All data values going to and from HyperTalk are zero-terminated ASCII strings

Resources of type XCMD are commands, and resources of type XFCN are functions that return a value. If you store a result string into returnValue in a command, the user can get it by asking for "the result" (useful for explaining why there was an error). In a function, you are expected to store the answer into returnValue. If you don't store anything, the result is the empty string.

If passFlag is false (the normal case), this XCMD or XFCN has handled the message and the script resumes execution. If passFlag is true, HyperCard searches the remaining inheritance chain for another handler or XCMD with the same name. This is just like the "pass" control structure in a script.

The file Flash.p is an example XCMD. It takes one argument which is the ASCII characters for a decimal integer. It inverts the screen twice the number of times indicated.

Peek.p is a function (XFCN) that returns the value of any memory location in the machine (purists avert your eyes).

The second part of the XCmdBlock record has to do with calling HyperCard back in the middle of your code to ask a question. If you wanted to manage the call to HyperCard yourself, you would fill inArgs with your arguments, put a request code in request, and JSR to the address in entryPoint. HyperCard returns the values you requested in outArgs and a result code in result.

Dan Winkler has packaged the entire range of calls on HyperCard, so that if you are using Pascal, you can simply call a procedure. Both Peek and Flash use some conversion routines that Dan has kindly supplied. The file XCmdGlue.inc has the glue procedures. Handle is always a handle to a zero-terminated string. If a handle is returned, you are responsible for disposing it. The calls are:

```
PROCEDURE SendCardMessage(msg: Str255);
{Send a HyperCard message (a command with arguments) to the current card.}

FUNCTION EvalExpr(expr: Str255): Handle;
{Evaluate a HyperCard expression and return the answer. The answer is a handle to a zero-terminated string.}
```

```
FUNCTION StringLength(strPtr: Ptr): LongInt;
{Count the characters from where strPtr points until the next zero byte. Does not count the
zero itself. strPtr must be a zero-terminated string.}

FUNCTION StringMatch(pattern: Str255; target: Ptr): Ptr;
{Perform case-insensitive match looking for pattern anywhere in  target, returning a pointer to
first character of the first match,  in target or NIL if no match found. pattern is a Pascal string,
and target is a zero-terminated string.}

PROCEDURE SendHCMessage(msg: Str255);
{Send a HyperCard message (a command with arguments) to HyperCard.}

PROCEDURE ZeroBytes(dstPtr: Ptr; longCount: LongInt);
{Write zeros into memory starting at dstPtr and going for longCount number of bytes.}

FUNCTION PasToZero(str: Str255): Handle;
{Convert a Pascal string to a zero-terminated string. Returns a handle to a new zero-
terminated string. The caller must dispose the handle.}

PROCEDURE ZeroToPas(zeroStr: Ptr; VAR pasStr: Str255);
{Fill the Pascal string with the contents of the zero-terminated string. You create the Pascal
string and pass it in as a VAR parameter. Useful for converting the arguments of any XCMD
to Pascal strings.}

FUNCTION StrToLong(str: Str31): LongInt;
{Convert a string of ASCII decimal digits to an unsigned long integer.}

FUNCTION StrToNum(str: Str31): LongInt;
{Convert a string of ASCII decimal digits to a signed long integer.
Negative sign is allowed.}

FUNCTION StrToBool(str: Str31): BOOLEAN;
{Convert the Pascal strings 'true' and 'false' to booleans.}

FUNCTION StrToExt(str: Str31): Extended;
{Convert a string of ASCII decimal digits to an extended long integer.} VAR x: Extended;

FUNCTION LongToStr(posNum: LongInt): Str31;
{Convert an unsigned long integer to a Pascal string.}
```

```
FUNCTION NumToStr(num: LongInt): Str31;
{Convert a signed long integer to a Pascal string.}

FUNCTION NumToHex(num: LongInt; nDigits: INTEGER): Str31;
{Convert an unsigned long integer to a hexadecimal number and put it
into a Pascal string.}

FUNCTION BoolToStr(bool: BOOLEAN): Str31;
{Convert a BOOLEAN to 'true' or 'false'.}
VAR str: Str31;

FUNCTION ExtToStr(num: Extended): Str31;
{Convert an extended long integer to decimal digits in a string.}

FUNCTION GetGlobal(globName: Str255): Handle;
{Return a handle to a zero-terminated string containing the value of the specified HyperTalk
global variable.}

PROCEDURE SetGlobal(globName: Str255; globValue: Handle);
{Set the value of the specified HyperTalk global variable to be the zero-terminated string in
globValue. The contents of the Handle are copied, so you must still dispose it afterwards.}

FUNCTION GetFieldByName(cardFieldFlag: BOOLEAN; fieldName: Str255): Handle;
{Return a handle to a zero-terminated string containing the value of field fieldName on the
current card. You must dispose the handle.cardFieldFlag set to false indicates background,
instead of card, field.}

FUNCTION GetFieldByNum(cardFieldFlag: BOOLEAN; fieldNum: INTEGER): Handle;
{Return a handle to a zero-terminated string containing the value of field fieldNum on the
current card. You must dispose the handle.}

FUNCTION GetFieldByID(cardFieldFlag: BOOLEAN; fieldID: INTEGER): Handle;
{Return a handle to a zero-terminated string containing the value of the field while ID is
fieldID. You must dispose the handle.}

PROCEDURE SetFieldByName(cardFieldFlag: BOOLEAN; fieldName: Str255; fieldVal:
Handle);
{Set the value of field fieldName to be the zero-terminated string in fieldVal. The contents of
the Handle are copied, so you must still dispose it afterwards.}

PROCEDURE SetFieldByNum(cardFieldFlag: BOOLEAN; fieldNum: INTEGER; fieldVal:
Handle);
```

{Set the value of field fieldNum to be the zero-terminated string in fieldVal. The contents of the Handle are copied, so you must still dispose it afterwards.}

PROCEDURE SetFieldByID(cardFieldFlag:BOOLEAN; fieldID: INTEGER; fieldVal: Handle);
{Set the value of the field whose ID is fieldID to be the zero-terminated string in fieldVal. The contents of the Handle are copied, so you must still dispose it afterwards.}

FUNCTION StringEqual(str1,str2: Str255): BOOLEAN;
{Return true if the two strings have the same characters. Case insensitive compare of the strings.}

PROCEDURE ReturnToPas(zeroStr: Ptr; VAR pasStr: Str255);
{zeroStr points into a zero-terminated string. Collect the characters from there to the next carriage Return and return them in the Pascal string pasStr. If a Return is not found, collect chars until the end of the string.}

PROCEDURE ScanToReturn(VAR scanPtr: Ptr);
{Move the pointer scanPtr along a zero-terminated string until it points at a Return character or a zero byte.}

PROCEDURE ScanToZero(VAR scanPtr: Ptr);
{Move the pointer scanPtr along a zero-terminated string until it points at a zero byte.}

Here are the files you will need:

HyperXCmd.p XCmdGlue.inc
Flash.p (An example command to see how everything is really done.)
Peek.p (An example function to see how everything is really done.)

Here are the typical MPW commands for compiling an XCMD :

```
  pascal -w    PioneerLVP4200.p    link -m ENTRYPOINT -o Video -rt
  XCMD=15 -sn    Main=PioneerLVP4200
∂
  PioneerLVP4200.p.o "{MPW}" Libraries:interface.o
```

"Video" is the stack that MPW will install the XCMD in. If you don't use any of the routines in interface.o, its just:

```
  pascal Flash.p
```

```
link -o HyperCommands -rt XCMD=0 -sn Main=Flash Flash.p.o
```

After executing these, use ResEdit to move the XCMD or XFCN to the proper stack.

For "C" programmers, the author has provided a definition (HyperXCmd.h) and an include file (XCmdGlue.inc.c) and an example (CFlash.c).

Breakpoints do not appear to work in XCMDs, but putting a debugger call in your code does work. In addition, saying:

```
hd 'h'
```

in MacsBug allows you to find your resource in memory by seeing its name and location in the listing.