Talking Back to HyperCard Revisited

Paul R. Potts

March 2025

In April 1989 my article, Understanding Lightspeed C: Talking Back to HyperCard *was published in Washington Apple Pi Journal. See the text and code in Talking Back to HyperCard. In 2025, I revised the code to make it more readable, and created a simple HyperCard demonstration stack to run it. The updated code and stack are available for download. See the links at the end of the article.*

About graphPack

The **graphPack** XCMD adds graph-drawing commands to HyperCard. I originally developed this code to support my work on an instructional stack called **LimCon**, for teaching calculus limits and continuity. If you'd like to read more about LimCon and how it uses XCMDs, see my article Retrospective: the LimCon HyperCard Stack.

For the 1989 article, I created a stripped-down version of **graphPack** designed to demonstrate in "bare-bones" form the use of an XCMD to manage an external window. This stripped-down **graphPack** is the subject of both the original article and this article.

Please note that, as I discuss in the Retrospective on LimCon, the XCMD I describe cannot be used to create and manage a Macintosh window that can be dragged around the screen like other windows. The window will not respond to the events that the Macintosh toolbox normally sends to trigger windows to redraw their content as needed. These limitations were not deal-breakers for the original LimCon project 35 years ago, which ran on Macintosh II systems with 640x480 displays.

HyperCard 2.0 introduced additional support for XCMDs which managed windows, but I wasn't working with **graphPack** any more by that point, so I never attempted to migrate the original XCMD to use the new functionality. That is, as they say, an opportunity for future research!

Revisiting graphPack

For anyone who would like to build and run this code, note that I am using Basilisk II on a MacBook Air M2 running System 7.5.3, HyperCard 2.4.1, THINK

C 5.0.2 and 6.0.1, and MPW 3.2.3. I originally used earlier versions of these tools during the years 1988 to 1990 when I developed this code, but I don't have a good record of exactly which versions I used. I chose these versions because I was able to get them to work under emulation in 2025.

I have included both THINK C 5 and THINK C 6 project files. Both seem to work for me fine and produce identical code. I have also included an MPW Makefile.

In modernizing this code, I first made a number of minor changes to get my original code building. Specifically, there were changes between versions of THINK C involving the definitions of types like **Str31**. When I tried it with MPW, I realized I had to stop relying on the automatic inclusion of **<MacTypes.h>** that THINK C does by default, and instead added the individual Mac headers as needed.

In addition to the changes needed to get it to build, I made the following changes:

Formatting

I originally wrote the code on a compact Mac (with a 9-inch, 512 pixel by 384 pixel screen), which encouraged me to make the source code itself as compact as possible, leaving out whitespace, especially vertical whitespace. I was very sparing with comments inside functions. I even left out spaces between parameters. When I submitted the code to the Washington Apple Pi Journal, I squeezed it down even further to fit the narrow magazine column width, destroying my indentation scheme.

In 2025 I have much more screen real estate to work with, so I reformatted the code for greater readability, limiting myself to a relatively luxurious 75-character maximum line length. I also switched from tabs to spaces, and used BBEdit to clean up trailing whitespace and other issues that are invisible in the THINK C editor, but easily visible using BBEdit's "Show Invisibles" view option.

Modernizing the C Code

While I was at it, I got rid of the old K&R style function definitions in favor of the standard C practice of including types in parameter lists.

I also factored out some small helper functions, and got rid of some unnecessary allocation of pointers and handles by using some local buffers for working with temporary strings.

Note that after rebuilding with THINK C 5, the XCMD resource is now 1302 bytes instead of the original 8810 bytes. That's a pretty dramatic improvement! Some of the savings is due to refactoring to reduce duplicated lines of code, but most of the reduction happened simply because I rebuilt the code with THINK C 5. I think this must be due to improvements to the linker, which seems to be better at stripping out unused code, compared to older versions of THINK C.

The New graphPackStack

As part of my effort to test the **graphPack** XCMD, I created a new HyperCard XCMD called **graphPackStack**. It's not pretty, but here's what it looks like:

Figure 1: "The graphPack Stack"

In the original article, my explanation for how to set up the resources needed by **graphPack** was extremely short, as I was trying to hit a strict word count limit. Here is how I set up the resources in this new HyperCard stack, in more detail. Here's a screen shot of ResEdit showing all the windows:

You will need:

- a PICT resource. This is used to create the window. I used a background image showing an Cartesian coordinate plane. I used resource ID 1, but you can use any resource ID.
- an STR resource. This contains a string that holds the name of the HyperTalk global variable that the XCMD should use to store a pointer to the active window. In this case, it is **myWindowsLoc**. This must match the global definition in the HyperTalk code in the stack! The resource ID must be 100, as the **graphPack** C code always uses 100.
- the XCMD code resource itself. I copied this from the resource file generated by THINK C and pasted it into the stack. It should be called graphPack. The ID is not important, as HyperCard will find the code resource by name, but I used 160.

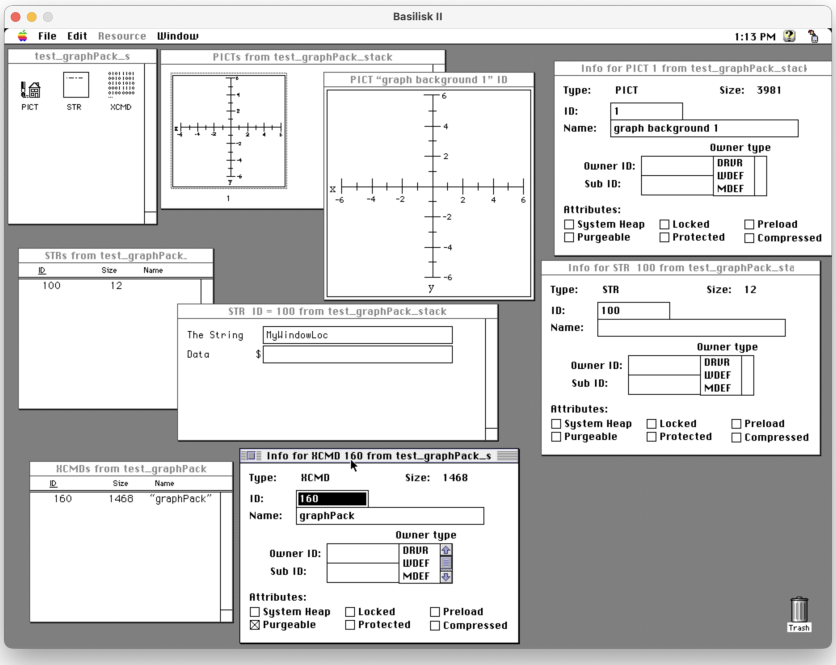

Figure 2: "The graphPack Stack Resources"

The HyperTalk Scripts

A note on testing: these scripts don't do error-checking, and neither does this stripped-down version of **graphPack**. In particular, if you try to use the "Make Window" or "Clean Window" buttons when the window has never been created, or after it has been destroyed with "Trash Window," *you will probably crash your emulated Macintosh computer*, so be cautious. I recommend making regular backups of the disk image file you are using with the emulator, since sometimes, if Basilisk II crashes, it will make the disk image un-bootable.

The Stack Script

The stack script must contain the definition of the global **myWindowLoc**. This just looks like:

```
global myWindowLoc
```

The "Make Window" Button Script

The "Make Window" button has this script:

```
on mouseUp
  -- makeWindow command with PICT resource ID 1
  graphPack 1, 1
end mouseUp
```

The first 1 parameter selects the makeWindow command and the second indicates the resource ID of the PICT resource that the code will load to create the window. If I hit command-M to open up HyperCard's message box, and type in "hide menuBar," and hit the "Make Window" button, I see the window pop up:

Note that I mentioned hiding the menu bar, because if I didn't, the window would be drawn partly underneath the menu bar. This is because its position was originally hard-coded for use with a particular HyperCard stack designed to run full-screen. To get an idea of what I designed the full version of **graphPack** to do, see Retrospective: The LimCon HyperCard Stack.

The "Trash Window" Button Script

I can dispose of the window using the "Trash Window" button. Its script looks like this:

```
on mouseUp
  -- trashWindow command - this command accepts no parameters
  graphPack 9
end mouseUp
```

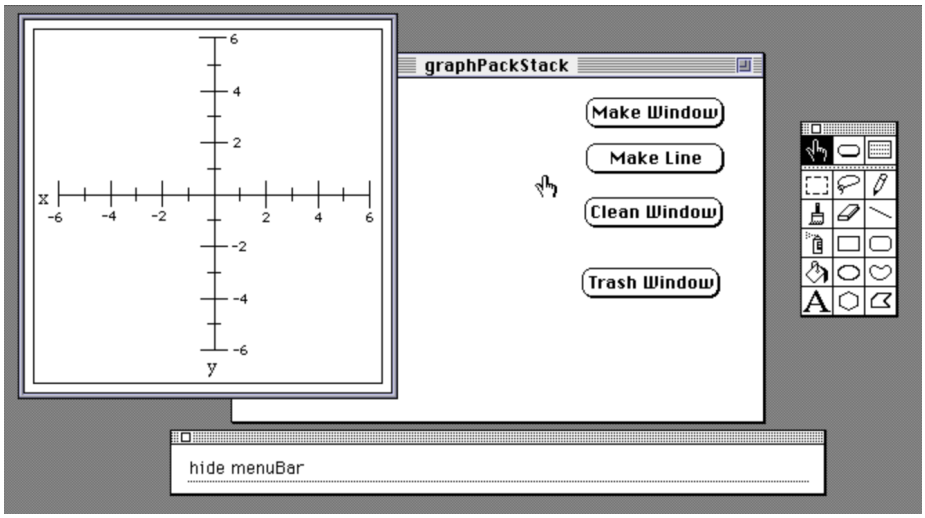

Figure 3: "The graphPack Window"

The "Make Line" Button Script

A very minimal script for testing the "makeLine" command in **graphPack** might look like this:

```
on mouseUp
  -- makeLine command with x1, y1, x2, y2
  graphPack 2, 55, 44, 205, 194
```

The resulting window looks like this:

The "Clean Window" Button Script

And finally, I can clean the window, re-drawing the PICT, with the "Clean Window" button, which has this script:

```
on mouseUp
  -- cleanWindow command with PICT resource ID 1
  graphPack 3, 1
end mouseUp
```

The Revised Source Code

Note that this source code does not fit well into my current HTML template. The code in the PDF version of the article is more readable. You can also download the source code. See "How to Get the Source Code," at the end of this article, below.

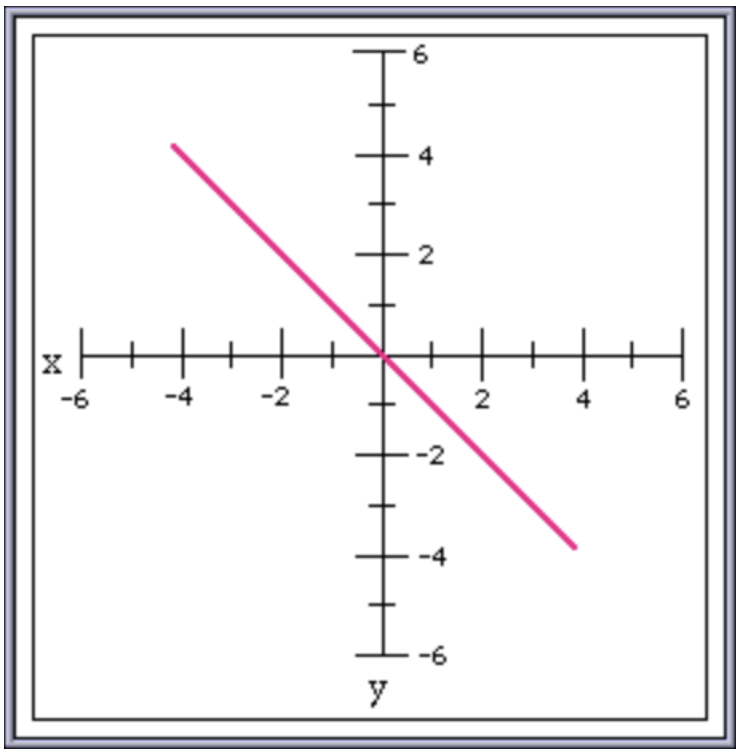

Figure 4: “A Line in the Window”

Source code file 1 of 3: graphPack.h

```
/*****************************************************************************/
/* File: graphPack.h                                                         */
/*                                                                           */
/* Supplies the prototype for our XCMD entry function Dispatchommand()       */
/*                                                                           */
/* The original code is from 1988. Updated 2025.                             */
/*****************************************************************************/

void SelectCommand(XCmdPtr paramPtr);
```

(Yes, that's really all that is needed in the header file, since there is only one function in **graphPack** that has external linkage.)

Source code file 2 of 3: graphPack.c

```
/*****************************************************************************/
/* File: graphPack.c                                                         */
/* This file should be compiled in a project along with the following:       */
/*                                                                           */
/* - XCMDshell.c                                                             */
/* - the MacTraps library                                                    */
/*                                                                           */
/* The original code is from 1988.                                           */
/*                                                                           */
/* PRP: March 2025: updated this file for use with THINK C 5, changed        */
/* function definitions from K&R style to ANSI C style, improved             */
/* comments, and made some minor code improvements. Got rid of a number      */
/* of NewHandle and NewPtr calls for Str31 strings, as these did not         */
/* seem necessary. Ideally I would use statically-allocated file-scope       */
/* (so-called "global") Str31 variables for temporary strings, instead       */
/* of allocating them on the stack, but when an XCMD is executing, the       */
/* global variable base address in A5 is configured for HyperCard's          */
/* globals, not the XCMD's, and using globals will corrupt HyperCard's       */
/* state. Supposedly there are ways to work around this by saving,           */
/* altering, and restoring A5. I've been able to get this to work in         */
/* other types of MacOS code resources such as INIT files, but not           */
/* XCMDs.                                                                    */
*****************************************************************************/
#include <Memory.h>     /* handle routines */
#include <OSEvents.h>   /* declares FlushEvents() */
#include <Resources.h> /* declares ReleaseResource() */
#include <HyperXCmd.h> /* declares HyperCard callbacks */
#include <ToolUtils.h> /* declares GetString() */
#include "graphPack.h" /* declares DispatchCommand() */
```

```
/*
   myStringID is the ID number of the STR resource (in the HyperCard
   stack hosting this XCMD), which contains the name of the global
   HyperTalk variable (which should be defined in your stack script),
   which will hold a pointer to the window created by the XCMD. This
   allows the same window to persist between calls to the XCMD.
   This sounds overly complicated, but once it is set up, it is simple
   to use from your HyperCard stack.
*/
#define G_NAME_ID 100

/*
   Prototypes for the routines defined in this file.
*/
static short   paramToShort(XCmdPtr paramPtr, char *cstr);
static long    paramToLong(XCmdPtr paramPtr, char *cstr);

void           SelectCommand(XCmdPtr paramPtr);

static void    makeWindow   (XCmdPtr paramPtr);
static void    makeLine     (XCmdPtr paramPtr);

static void    putWindowPtr (XCmdPtr paramPtr, GrafPtr theWindow,
                             long strID);
static GrafPtr getWindowPtr (XCmdPtr paramPtr, long strID);

static void    trashWindow  (XCmdPtr paramPtr);
static void    cleanWindow  (XCmdPtr paramPtr);

/**************************************************************************/
/* A helper function to turn an incoming parameter into a short. This   */
/* uses glue functions to make two callbacks into HyperCard. The        */
/* parameters come in as zero-terminated C-style strings. Convert the   */
/* parameter to a Pascal-style string with a length byte, then turn it  */
/* into a short.                                                        */
/**************************************************************************/
static short paramToShort(XCmdPtr paramPtr, char *cstr)
{
    Str31 param_buf;

    ZeroToPas(paramPtr, cstr, (StringPtr)param_buf);
    return (short)StrToNum(paramPtr, param_buf);
}

/**************************************************************************/
/* A similar helper function that returns a long.                       */
```

```
/************************************************************************/
static long paramToLong(XCmdPtr paramPtr, char *cstr)
{
    Str31 param_buf;

    ZeroToPas(paramPtr, cstr, (StringPtr)param_buf);
    return (long)StrToNum(paramPtr, param_buf);
}

/************************************************************************/
/* This is the dispatcher which determines which one of the routines    */
/* in the graphPack XCMD to run.                                        */
/************************************************************************/
void SelectCommand(XCmdPtr paramPtr)
{
    short   which_command;
    GrafPtr saved_port;

    /* Save HyperCard's GrafPort */
    GetPort(&saved_port);

    PurgeMem((Size)5000);

    which_command = paramToShort(paramPtr, (char*)*paramPtr->params[0]);

    switch (which_command)
    {
        case 1:
            if (paramPtr->paramCount == 2)
            {
                makeWindow (paramPtr);
            }
            else
            {
                SysBeep(40);
            }
            break;

        case 2:
            if (paramPtr->paramCount == 5)
            {
                makeLine (paramPtr);
            }
            else
            {
                SysBeep(40);
```

```
            }
            break;

        case 3:
            if (paramPtr->paramCount == 2)
            {
                cleanWindow (paramPtr);
            }
            else
            {
                SysBeep(40);
            }
            break;

        case 9:
            if (paramPtr->paramCount == 1)
            {
                trashWindow (paramPtr);
            }
            else
            {
                SysBeep(40);
            }
            break;
    }
    FlushEvents (everyEvent, 0);

    /* Restore HyperCard's GrafPort */
    SetPort(saved_port);
}

/*****************************************************************************/
/* This function will initialize a dBoxProc window at the location of   */
/* the PICT read, and draw the PICT in it. To start with a blank window */
/* use an empty PICT                                                     */
/*****************************************************************************/
static void makeWindow (XCmdPtr paramPtr)
{
    long      which_pict;
    Rect      bounding_rect;
    WindowPtr theWindow;
    PicHandle myWindowBox;

    which_pict = paramToLong(paramPtr, (char*)*paramPtr->params[1]);

    myWindowBox = GetPicture(which_pict);
```

```
    bounding_rect = (*myWindowBox)->picFrame;

    theWindow = NewWindow(0L, &bounding_rect, "\P", true,
                          dBoxProc, (WindowPtr)-1L, false, 0L);
    SetPort(theWindow);
    DrawPicture(myWindowBox, &theWindow->portRect);
    ReleaseResource((Handle)myWindowBox);

    putWindowPtr(paramPtr, theWindow, G_NAME_ID);
}

/*************************************************************************/
/* This function reads two coordinate pairs in order to draw a line      */
/* segment in a window owned by the XCMD.                                */
/*************************************************************************/
void makeLine(XCmdPtr paramPtr)
{
    GrafPtr xcmdWindow;
    short   horiz, vert, newhoriz, newvert;

    horiz    = paramToShort(paramPtr, (char*)*paramPtr->params[1]);
    vert     = paramToShort(paramPtr, (char*)*paramPtr->params[2]);
    newhoriz = paramToShort(paramPtr, (char*)*paramPtr->params[3]);
    newvert  = paramToShort(paramPtr, (char*)*paramPtr->params[4]);

    xcmdWindow = getWindowPtr(paramPtr, G_NAME_ID);
    SetPort(xcmdWindow);
    ForeColor (magentaColor);
    PenSize(2,2);
    MoveTo (horiz, vert);
    LineTo (newhoriz, newvert);
}

/*************************************************************************/
/* This function will redraw the window owned by the XCMD. This will     */
/* effectively erase anything drawn in the window since it was created.  */
/* The first parameter is the ID of the PICT resource to be opened.      */
/* This can be the PICT resource that was used with createWindow, or a   */
/* different one.                                                        */
/*************************************************************************/
void cleanWindow(XCmdPtr paramPtr)
{
    GrafPtr   xcmdWindow;
    long      which_pict;
    PicHandle myPict;
```

```
    which_pict = paramToLong(paramPtr, (char*)*paramPtr->params[1]);

    myPict = GetPicture(which_pict);

    /* Retrieve the window pointer from the HyperTalk global */
    xcmdWindow = getWindowPtr(paramPtr, G_NAME_ID);

    /* Draw the PICT */
    SetPort(xcmdWindow);
    DrawPicture(myPict, &xcmdWindow->portRect);
    ReleaseResource((Handle)myPict);
}

/*****************************************************************************/
/* This function will get the current window pointer from HyperCard,    */
/* set the global variable that holds it to NULL, and dispose of the    */
/* window.                                                               */
/*****************************************************************************/
void trashWindow(XCmdPtr paramPtr)
{
    GrafPtr xcmdWindow = getWindowPtr(paramPtr, G_NAME_ID);

    DisposeWindow(xcmdWindow);

    /* Store NULL into the HyperTalk global */
    putWindowPtr(paramPtr, (GrafPtr)0, G_NAME_ID);
}

/*****************************************************************************/
/* This routine will call back into Hypercard to store a pointer to the */
/* window, in string form. The strID is used to look up  the string     */
/* resource, which should be in the HyperCard stack, containing the     */
/* name of the HyperTalk global variable to store the pointer in.       */
/*****************************************************************************/
void putWindowPtr(XCmdPtr paramPtr, GrafPtr theWindow, long strID)
{
    Str31        param_buf;
    StringHandle global_name;
    Handle       new_value;

    /* Put the GrafPtr, in Pascal string form, into the given buffer. */
    LongToStr(paramPtr, (long)theWindow, param_buf);

    /* Get the global variable name as a handle to a string resource. */
    global_name = GetString(strID);
```

```
    /* Get the new value as a handle to a C string. */
    new_value = PasToZero(paramPtr, (StringPtr)param_buf);

    /* Set the global variable to the new value. */
    SetGlobal(paramPtr, (StringPtr)*global_name, (Handle)new_value);

    ReleaseResource((Handle)global_name);
    DisposHandle(new_value);
}

/**************************************************************************/
/* This function will return a pointer to the window owned by the        */
/* graphPack XCMD graphPack. It accepts the ID number of the string       */
/* resource which contains the name of the HyperTalk variable holding    */
/* the pointer.                                                           */
/**************************************************************************/
GrafPtr getWindowPtr(XCmdPtr paramPtr, long strID)
{
    GrafPtr      xCmdWindow;
    StringHandle global_name;
    Handle       value;

    /* Get the global variable name as a handle to a string resource. */
    global_name = GetString(strID);

    /* Get the value of the HyperCard global as a handle to a C string */
    value = GetGlobal(paramPtr, (StringPtr)*global_name);

    xCmdWindow = (GrafPtr)paramToLong(paramPtr, (char *)*value);

    ReleaseResource((Handle)global_name);
    DisposHandle(value);

    return xCmdWindow;
}
```

Source code file 3 of 3: XCMD.shell.c

```
/**************************************************************************/
/* File: XCMDshell.c                                                      */
/* This file should be compiled in a project along with the following     */
/* files:                                                                 */
/*                                                                        */
/* - graphPack.c                                                          */
/* - HyperXLib (for XCMD interface callbacks)                             */
/* - the MacTraps library                                                 */
```

```
/* - XCmdshell.c                                                          */
/*                                                                        */
/* The original code is from 1988.                                        */
/**************************************************************************/
#include <HyperXCmd.h>
#include "graphPack.h"

/*
    Prototype for the XCMD main function, defined to use Pascal calling
    conventions as required by HyperCard's XCMD calling mechanism.
*/
pascal void main(XCmdPtr paramPtr);

pascal void main(XCmdPtr paramPtr)
{
    SelectCommand(paramPtr);
}
```

How to Get the Source Code

I have made the code available in two ways. If you want to work with the source code, HyperCard stack, and THINK C 5 project on a Macintosh or Macintosh emulator, you will need to download it in a format that preserves the original MacOS file metadata and resource forks. I've made a StuffIt archive available: talking_back_graphPack.sit. You'll need to put StuffIt Expander on your real or emulated Macintosh.

If you want to just browse the source code and don't need the stack, THINK C projects, and MPW Makefile, you should be able to use this zip file on just about any computer: talking_back_graphPack.zip.

Keep the vision of old Macintosh programming alive!
