main screen. Otherwise we go to the next card (dialog, pop-up screen) which prompts for information that will be needed for the histogram. Again, if the user chooses the "cancel" button we return to the main card. Otherwise a card showing the sample report pops up and the user can stare at it as long as desired. Clicking the mouse in the "done" button takes you back to the main screen.

The purpose of this stack is to provide the user with a general idea of how the application will actually look and feel. The feel of an application is important. If it is chunky and internally inconsistent the user will grow to dislike the application. A smooth and consistent feel is required and the user knows best what his or her needs are. Also, keep in mind that you need not describe in detail each aspect of the interface that you are simulating. Though you should try to simulate all the major points of the interface, minor ones can be left alone. For example, most of the radio buttons in my stack actually do anything. Another example involves the "Class History" button. In the final version the data screen that pops up from that button varies in size depending on how much schooling the student has had. In the stack, though, the size is fixed. The stack is a supplement to all the paper that you have probably provided, that details just what the various menu commands do, what data types will be permitted in what fields, what files are required, and so forth.

As has been mentioned before, the stack is quite simple. Most of the scripts are merely go card x scripts. The exceptions to that rule include the openStack, closeStack, hiliteButton, dimMenu, and hiliteMenu scripts. The openStack script puts up all the necessary menus. The closeStack handler cleans up the menu bar and clears out the various global containers. The dimMenu and hiliteMenu functions are a bit more complex. Thier purpose is to grey out (de-activate) and blacken (active) the requested menus respectively. They are called from various other handlers like the "Print Stats Report" menu option, and the "AP Placement" and "Class History" buttons.

In the real application you may not always want a menu to be active. That is a fairly major point in any interface and ought to be represented in the model as such. The hiliteButton function activates a specified radio button and dims out its neighbors. Such actions can be a vital part of the user interface and therefore are included. What follow are the openStack, closeStack, dimMenu, hiliteMenu, hiliteButton, doMenu, idle, the "done" button, and the "AP Placement" button handlers. They ought to give you a good idea of the mechanics of the stack. The various cards were created using HyperCard's various painting tools. However, the important idea to take away here is that HyperCard can be used as an aid to software design and development.

Writing XCMDs in Lightspeed C:

About the Aboutbox

by Paul Potts

This simple project creates a new "aboutbox" command for HyperCard which will display a picture, wait for a mouse click or keypress, and return to HyperCard. To construct this project you will need Lightspeed C version 3.0, HyperCard version 1.2, and ResEdit. You will also need an object-oriented or bit-mapped drawing program to create a PICT resource.

To access a resource of type PICT, the toolbox function GetPicture can be used. We can get the handle like this:

PicHandle the_pic;

the_pict = GetPicture(id_num);

where id_num is an integer. Now that we have a handle on the PICT, we can access the parts of the structure. In C, referencing the elements of a structure pointed to by a handle can be done in three ways. The first uses a double dereference to access the elements of the structure directly. The parentheses are necessary in order to dereference the handle rather than the field of the structure:

the_rect = (the_pict).frameRect;**

The second method uses the "->" operator, which does one dereference and also accesses a particular field in a structure. This is the equivalent to "Pointer ^. field" in Pascal, where the carat indicates the dereference and the period the field offset. Thus,

the_rect = (*the_pict)->frameRect;

is the equivalent of our first example. It is also possible to dereference the handle, store the result in a pointer, and access the field through the pointer:

```
Ptr deref;
deref = (*the_pict);
...
the_rect = deref->frameRect;
```

Note that this method is either extremely *unsafe* or *unfriendly*, depending on how it is done. Why? First of all, if the the handle is not locked there is a chance that memory may be rearranged behind your back, and that **deref** may no longer point to the PICT by the time you get around to accessing it later in your program. The alternative is to lock the handle using HLock. This is very unfriendly, because if another application needs memory space it will be unable to compact the heap properly due to your locked handle. I include this example to show how *not* to dereference a handle.

Since C is a weakly typed language, I like to program with the "Check Pointer Types" option turned on. This ensures that C will not do any implicit type conversion for you without your knowledge: you must do typecasting on your own. Unfortunately, this often makes code harder to read. Let us take a specific example from my **Convert** procedure to discuss typecasting:

pstr=(StringPtr)CtoPstr((char *)*c_han);

The function CtoPstr is a Lightspeed™ C built-in function (I give the prototype) which converts a zero-terminated string to a Pascal type string. Note that it works on a pointer to the string rather than the handle, and this explains the dereference of the handle *c_han. CtoPstr accepts a generic pointer char* , so we must cast the pointer to this type. Since pstr is defined as a (Pascal type) StringPtr for the later call to the toolbox routine StringToNum, we must force the function to return type StringPtr: hence, the cast of the entire function call.

In order to know which PICT to display with this XCMD, I let HyperCard pass the XCMD an argument.

Understanding how this works requires an explanation of a special data structure maintained by HyperCard called the **XCmdBlock.** The XCmdBlock is a non-relocatable block referenced by a simple pointer: HyperCard passes this pointer to your XCMD when it is called. Here are the first four elements of the XCMDBlock structure:

```
short      paramCount;
Handle     params[16];
Handle     returnValue;
Boolean    passFlag;
```

When calling an XCMD from inside Hypercard™, you simply give the name of the XCMD and follow it with a series of up to 16 parameters. Hypercard™ stores these parameters as zero-terminated strings (C - type strings) and gives your XCMD handles to them, contained in the **params** array. **paramCount** holds the number of parameters sent. If we call our XCMD "aboutbox," the XCMD call

aboutbox 35

would make the start of the XCMDBlock look like this:

paramCount = 1;
params [0]=Handle->Pointer->35

The next element of the XCMDBlock is also simple: **returnValue** is used for an XFCN (External Function). The only difference between an XCMD and and XFCN is that an XFCN is called as a function from HyperCard™, and should put a return argument into a zero-terminated string and a handle to that argument in **returnValue.**

If your XCMD is called from within Hypercard, it can choose to ignore the call and simply return. The Boolean variable passFlag can then be set to True by your XCMD to indicate that it did not handle the command, and that the message should continue up the inheritance chain (see the HyperCard Help stack for more information on messages and the inheritance chain.) The default value of **passFlag** is False, so we will not be concerned with it.

I have included the definition of the XCmdBlock in the file "working.xcmd.h." This is an extremely abbreviated version of Apple's header file. The complete XCMD include files as they are written for MPW C do not function properly in Lightspeed™ C, but this header file provides the definitions necessary for this introductory project.

Now we are ready to construct our XCMD. Find a HyperCard stack (your Home stack will do nicely). Select part of a drawing and copy it into the Scrapbook, then paste it into your HyperCard™ stack using ResEdit. You will need to know the resource ID number that ResEdit assigns your PICT when you paste it into your stack. Use ReSEdit's Get Info command to find out this information.

Now create a new project in Lightspeed™ C and add the MacTraps library to it. Use the Set Project Type menu option as shown in figure 1. You can use any ID number as long as it doesn't conflict with another XCMD resource already in your stack.

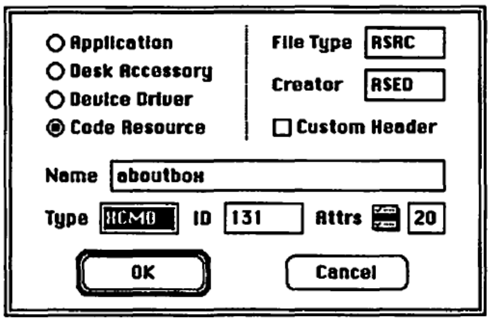

○ Application
○ Desk Accessory
○ Device Driver
● Code Resource
File Type: RSRC
Creator: RSED
☐ Custom Header
Name: aboutbox
Type: XCMD ID: 131 Attrs: 20

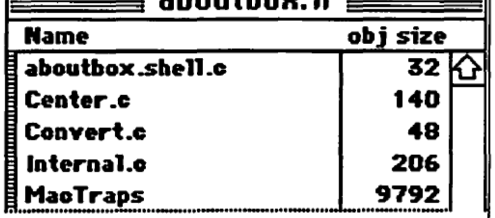
Figure 1: Project type dialogue for aboutbox XCMD

I have set the creator of the resulting file to RSED. This way you can launch ResEdit by opening the code resource file. Now that you have created the project, type in each of the files *as a separate document* and save it in the same folder with the project document. This ensures that the compiler can find all the included files quickly. Then use the "Add" command to add each file to your project. (**Working.xcmd.h** should be in the folder but not added to the project.) Figure 2 shows the files that should be included in your project.

Finally, choose "Build Code Resource" from the Project menu and copy the resource from the file LSC created into your Hypercard™ stack. You should now have a new command in your stack. Entering the command into the message box or running it in a script will draw the about box and wait for a mouse click or keypress. The command is:

aboutbox <PICT number>

➥ 54


aboutbox.π

Name	obj size
aboutbox.shell.c	32
Center.c	140
Convert.c	48
Internal.c	206
MacTraps	9792

Figure 2: Files to include in your project

13 ➥ GameSIG

The regular monthy GameSIG meeting on November 3rd featured a surprise reappearance of Ron Wartow, our Chairman Emeritus and favorite ninja. Ron suddenly materialized wielding a nasty-looking hard disk and ready to demo **Legerdemain,** his new "hypercard fantasy/role playing all-text all-graphics adventure database spreadsheet telecommunications and desk accessory" program. Despite technical difficulties (only partly remedied by Jeff Stetekluh, David Porter, and David Romerstein), Ron gave us a good look at the game and signed up play-testers. More details on **Legerdemain** will follow (since Ron is a little paranoid about revealing too much too soon!).

Ron and current chairman Charles Don Hall also shared news on recent and forthcoming games, including **Neuromancer, Might & Magic II** and **Wasteland II,** all due to be released shortly for the Apple II series, along with **Dungeonmaster** for the Apple IIGS (a "must buy"). **Times of Lore** and several of the new Infocom games are already out. Charles distributed the following programs for review:

Quarterstaff: The Tomb of Setmoth (Infocom, Mac + or better): Infocom's first fantasy/role-playing game;

Deja Vu: The Nightmare Comes True (Mindscape, Apple IIGS): conversion of the famous detective "MacVenture," this time with *color* graphics;

Uninvited (Mindscape, Apple IIGS): icon-based "haunted house" graphic adventure game in the same popular "MacVenture" series;

Shadowgate (Mindscape, Apple IIGS): third in the GS conversions of the "MacVentures," set in an enchanted castle where you battle an evil warlock.

The meeting ended with the monthly **Robot Tanks** tournament, in which Davy Hakim's "Crystal Raider" defeating all other contenders (David Porter, Jamie Kowalski, Jeff Stetekluh, and Richard Clark); Jeff's program is now up to Version .6, and he is still looking for beta-testers.

Sorry, folks! We forgot to compile our annual "Suggestions for Santa." If you're desperate for advice the November issue of **Computer Gaming World** has an extensive Christmas Buying Guide.

Aboutbox XCMD

```
/************************************************************/
/*      File: Working.xcmd.h
        Information ©Apple Computer, Inc. 1987
        Abbreviated version of XCMD header info */
typedef struct XCmdBlock
{
        short       paramCount;       /* We are only concerned */
        Handle      params[16];       /* With the first half of the */
        Handle      returnValue;      /* XCMDBlock in this project */
        Boolean     passFlag;

        char        *entryPoint;      /* to call back to HyperCard */
        short  request;
        short       result;
        long        inArgs[8];
        long        outArgs[4];
} XCmdBlock, *XCmdBlockPtr;

typedef struct Str31
{       char guts[32];
} Str31, *Str31Ptr, **Str31Handle;
/************************************************************/
/************************************************************/
/* File: aboutbox.shell.c                                  */
/* Simple XCMD shell by Paul Potts                         */
/* Needs MacHeaders turned on.                             */

#include "working.xcmd.h"       /* needed to define XCmdBlock, etc */

void Internal (XCmdBlockPtr paramPtr);          /* prototype for Internal*/
pascal void main(XCmdBlockPtr paramPtr);        /* prototype for self */

pascal void main(paramPtr)
        XCmdBlockPtr paramPtr;
{
        Internal(paramPtr);
}
/************************************************************/
/************************************************************/
/* File: Internal.c                                                  */
/* Function to display a specified "PICT" resource as an "about      */
/* box" type of dialogue.  The input argument, pointed to by the     */
/* handle in inArgs[0], is the ID of a PICT resource, which should   */
/* be contained in the calling stack.  Needs MacHeaders on.          */
#include "working.xcmd.h"

Rect Center (Rect the_rect);
long Convert(Str31Handle c_han);
void Internal (XCmdBlockPtr paramPtr);          /* This function */

void Internal(paramPtr)
        XCmdBlockPtr paramPtr;
{
        long                    which_pict;
        EventRecord             theEvent;
        Rect                    bounding_rect;
        WindowPtr               theWindow;
        PicHandle               AboutBox;

        FlushEvents(everyEvent, 0);
        which_pict = Convert ((Str31Handle)paramPtr->params[0]);
        AboutBox = GetPicture(which_pict);
        bounding_rect = Center ((*AboutBox)->picFrame);

        theWindow = NewWindow (0L, &bounding_rect, "\P", TRUE,
                                        dBoxProc, -1L, FALSE, 0L);
        SetPort(theWindow);
        DrawPicture (AboutBox, &theWindow->portRect);
        while (1)
                {
                        GetNextEvent (everyEvent, &theEvent);
                        if (theEvent.what==mouseDown) break;
                        if (theEvent.what==keyDown) break;
                }
        DisposeWindow(theWindow);
        ReleaseResource(AboutBox);
        KillPicture(AboutBox);
        DisposHandle(AboutBox);
        FlushEvents(everyEvent, 0);
}
/************************************************************/
/************************************************************/
/* File: Center.c
/* Needs MacHeaders turned on.  This function accepts a rectangle   */
/* represented in Global Coordinates and centers it on the screen,  */
/* using the Window Manager to determine the size of the screen.    */

Rect Center (Rect the_rect);            /* self-prototype */

Rect    Center (the_rect)
                Rect            the_rect;
{
        GrafPtr         wp;
        short           width, height, new_top, new_left;

        GetWMgrPort (&wp);                      /* Gets the whole screen */
        width= wp->portRect.right;
        height = wp->portRect.bottom;

        new_top = ((height / 2) - (the_rect.bottom - the_rect.top) / 2);
        new_left = ((width / 2) - (the_rect.right - the_rect.left) / 2);
        OffsetRect(&the_rect, new_left - the_rect.left, new_top - the_rect.top
);
        return the_rect;
}
/************************************************************/
/************************************************************/
/* File: Convert.c
/* Needs MacHeaders turned on.  A prototype is provided.             */
/* This function accepts a handle to a c (zero-terminated) string and */
/* returns a long int containing the number represented by the string */
#include "working.xcmd.h"       /* needed for Str31 types */

long Convert(Str31Handle c_han);        /* this function */

long    Convert(c_han)
                Str31Handle         c_han;
{
        long                thenum;     /* the longint to return */
        StringPtr           pstr;       /* pointer to a pascal string */
        pstr = (StringPtr)CtoPstr((char *)*c_han);      /* convert to Str255
*/
        StringToNum(pstr, &thenum);                     /* convert to a long
*/
        return thenum;
}
/************************************************************/
```