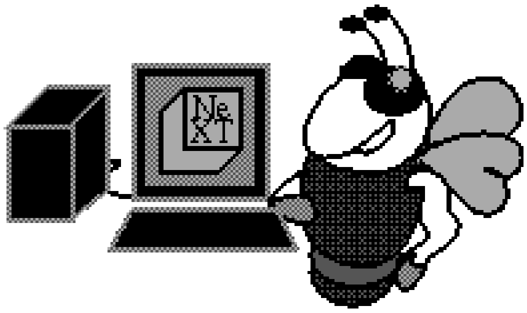
IBM officially announces that AI/X will support NextStep -- see p. 42



BuzzNUG Buzzings

February 1990 - Issue 3

Contents

The BuzzNUG General Council
Erica J. Liebman, Pres. erica@kong.gatech.edu
David Rosenbaum, V.P. daver@pyr.gatech.edu
David Samuel King, Tres/Sec dsking@pyr.gatech.edu
Phyllis A. Huster, Editorial Consultant, ccuseph@hydra.gatech.edu
Keith Edwards, keith@kong.gatech.edu, NeXT Campus Consultant
Tamora V. Sealey, Accounts tammy@cadnext2.gatech.edu

Contributers : Erica J. Liebman, Jiro Nakamura, Andrew C. Stone, Mike Gourlay, Ian Smith, Alan Chung, Jonathan Schwartz, Dick Silbar, Carl Sutter, David "Deebers". Kay, Edward Jung, John "Baker" Corey,James E. Burns, Phyllis A. Huster

```
Question :   Why did the NeXT Designer Cross the Road?
Answer :     It was elegant.
```

Welcome

Erica J. Liebman

Welcome readers, new and old. This third issue of the Buzzings is the largest yet. An offhand trend analysis indicates that if issue size increases keep pace, a single laser-printer toner cartridge will be insufficient to print an issue by the end of the current year. Realistically, I anticipate that the issue size will tend to level off in the thirty-to-fifty page range over the next few months.

There have been changes. We've beefed up the Market View section, trying to include as many current and potential third party developers as possible. Time didn't allow product confirmation w/ delivery dates, etc for this issue so you'll have to wait a month before we can include the initial results of a massive phoning onslaught. I've included the User Group section again -- hope it helps. Two new contacts have been added to the list respectively in the California and Texas areas. If you have a User Group starting up, please let us know contact names, phone numbers, addresses, meeting times and special events in plenty of time for advance printing. You may notice that the Object Buzz-line is no longer included. Any tips or hints about building custom objects will be covered in individual articles or the hint-corner.

The ID Cards have been designed and are awaiting the development of our subscription database. If you haven't sent in contact information (name, address, phone, e-mail), please do so at your soonest convenience. The ID Cards are adorable and are sure to soon become a fantastic status symbol among NeXTies. Send information to erica@kong.gatech.edu or the editorial address below. Please always note NeXT-specific mail in the subject line. Thank you. (Send SASE for USMail).

On another note, John Corey ("Baker", to friends) of NeXT was nice enough to slip us some official information about the AI/X-NextStep connection with IBM. The news included should be up to date and relevant. Baker, who pretty much led the effort for NeXT , promises that next month he'll write about the implications that this coalition will bring about. A *big* thank you is owed to him for pushing to get this information to us at the very last moment.

Finally, I decided that since this is a message from the editor, I'd like to editorialize. If I may be permitted to ascend the soapbox for once, at least once officially, I'd like to talk about the NeXT Dock. The dock is a lovely creature. It gives form and structure where certain other machines (some beginning with the prefix "mac") have not. It also remembers the home addresses of icons. The dock is clean. The dock is elegant. This is not enough.

Let me pontificate. I have at this time, at least three modes of NeXT-interaction : programming-, user- and demonstration-mode. In programming mode, I want access to (in no particular order) : Interface Builder, Scene, Icon, Shell, Terminal, TextArt, Draw, Edit, WriteNow, Digital Librarian, Mail, Preferences, Mixer, SoundPlayer and Printer. You may count these. Note, that even moving the black hole to another side of the Screen does not work. The dock is still limited to twelve berths. Mostly I compromise. I leave out the sound stuff and either Mail or Printer. I also have to sacrifice "Eyecon".

Then there's my normal user mode. That's when I crave NX_VOID, NetHack, my usual Shell, Term, Kermit, SoundPlayer (for Wagner in the background) etc. When I'm not programming (which happens on occasion), I want to use *applications.* I can't keep changing my dock around, so I wearily launch Mach shells and run apps from there.

And don't forget when Mom visits. Or when a friend sees the cube for the first time. ("*Awesome!* What can it do?) Then it's demo time. It's time for ScorePlayer & Billiards & BreakApp. I have to tell a non-computer-person "Ok now, click on the word Next Developer". After all, none of these end up being on the dock.

So what can be done?

Here's my suggestion. First, allow dock space above the NeXT cube as well as below it. This will provide twenty-four dock spaces. Next, allow the user to have a "carousel" of dock choices. By control-alt-clicking on the NeXT cube, let me circle through the dock modes. This solves my two major problems : too many icons for a single task and too many tasks for a single dock.

I don't want NeXT to compromise the elegance of their workspace management system, but I do want it to stretch the functionality within these limits.

Editorial Stuff

Articles for Buzzings are accepted in various forms, NeXT mail enclosures and Internet .wn.tar.Z forms are preferred but ascii text via the net, IBM and Mac Disks via USMail and (yikes) written text via the same USMail are happily accepted. We can guarantee no return of materials without SASEs, sorry. Our focus is how-to articles, especially with sample code. All articles are subject to editorial review.

We also welcome copies of new (and old) software for review from third party vendors. Again, we can not guarantee material return without SASE or guarantee publication dates, if at all, although we try to be prompt.

"Feedback from the Trenches" is open for comments/letters of limited length from all readers. Please write and tell us what you liked and what you disliked.

Mailed subscriptions should start with the April Issue, with any luck. Please write for information.

Editorial matters to :
BuzzNUG c/o EJ Liebman
1150 Collier Rd/NW L-12
Atlanta, GA 30318.
1-404-352-5551.

There is always an answering machine, but please respect relatively normal hours. Long distance phone calls may not be returned by the impoverished student at the other end. Please send any deliveries of items that will not fit within a tiny mailbox care of the Leasing Office. To contact me directly for subscription information, corrections, requests, or just to say hi, write via internet : erica@kong.gatech.edu.

Feedback from the Trenches

• I was glad to see notes about some commercial software development going on for the NeXT. I think this is an important thing to continue; many potential NeXT users are waiting to see if there will be a good base of commercial software available. Are there any juicy rumors about possible portings from the major software houses? (I've heard that Gates says Microsoft will never support the NeXT). P.S. Erica: I loved the first two issues. Keep up the good work.
Scott Coulter; scott@kong.gatech.edu

• As a sideline, I actually managed to get the read the latest issue of BuzzNUG this last weekend, but I didn't make it back into work until just today. For a newsletter that is only in its second issue, you (the BuzzNUG staff and contributors) have managed to include an interesting range of articles. The only suggestion that I could think of as I was browsing through it was that the articles need to be better delineated. One article ends and the next begins right after. This in itself is fine, but the beginning of the next article needs to be made more prominent such that it would most likely be the first thing spotted when flipping to a page. In addition, you might also consider putting a cute (but not too cute) little icon at the end of an article to denotes its conclusion (a little black box, er, cube, would be adequate).
Once I was done, I tried to think how I could best describe the newsletter and what I came up with was that it had somewhat of a MacTutor-esque style to it. You had an article about machine specifics replete with source code, in addition to another (somewhat insightful) article on interface design. I applaud your efforts. One thing that you might have considered, but I think would be a bad idea would be to include a rumor column of some sort. Everyone wants to hear the latest in what's soon to become the latest, but I think that sort of thing is best left on Usenet, IMHO :)
Frank Guerra, guerra%cae.llnl.gov@lll-lcc.llnl.gov

• I have a suggestion for your newsletter. How about a complete step-by-step guide to getting news and sendmail running on a NeXT?
Jeff Scott, atariman@bsu-cs.bsu.edu

•There is supposed to be a scanner and software. Sorry, I don't remember the vendor's name -- I don't have the 3rd party list, and people at my local Businessland can never find their copy... If you can get a hold of it, I'd love to hear how it works. I'm particularly interested in OCR software, especially if it's trainable (to read Russian).
Thanks for your efforts!
Jacob Gore, Jacob@Gore.Com, boulder!gore!jacob

• Thanks a bunch! I distributed Buzzings to our software department as well as a few other people. It got very favorable reviews. Keep up the good work Erica!
Morris Meyer, mmeyer@next.com

•Hello - I was just flipping through BuzzNUG 2 and wanted to tell you that you folks are doing a really great job with it. It is impressive. If you need any help, I'd be willing. [*Yes, please! -- EJL*]   BuzzNUG 2 was really great. I enjoyed seeing my two programs, reviewed by Gerrit..

Oh yeah, there was a bug on the last page in a letter by Andrew Stone. His first bit about the typo in the docs was right, but his example was wrong. It should be

 [window setMiniwindowIcon: "myCoolIcon.tiff"]

But I managed to figure out what he was saying and because of it, Cassandra now has miniwindow icons. Thanks! Keep up the good work!

 Jiro Nakamura, jiro@heights.cit.cornell.edu

• Erica, I have just seen issues 1 and 2 of BuzzNUG (ftp from purdue). Although I've still not seen a NeXT in the UK, or had any replies of email/snailmail from enquiries to NeXT (despite asking about the status of UK developers) it looks interesting enough to keep up-to-date with. If possible, could you put something to the effect 'How do we get one of these NeXT boxes in the UK, as quickly and as cheaply as possible (developer status)'. Thanks.

Andrew D. Nimmo, VLSI and Computer Graphics Research Group, EAPS II, University of Sussex, Falmer,BRIGHTON, East Sussex, BN1 9QT, UK, andrewn%uk.ac.sussex.syma@nsfnet-relay.ac.uk. Ph:+44 273 606755 x 2617

[NB : *We've found quite a lot of interest in Common-Market distribution. We'll try to address this in the next issue or so. Phyllis Huster reports that in Austria last summer, she met a software development consultant who claimed he could sell thirty NeXTs before breakfast -- if he had the access.*]

•I liked them [*the first two issues*] very much. I particularly like the code examples, several of which have given me some ideas and insight. I'd [*also*] like to see a good review of the new Terminal/Shell program people are chatting about in the group. Communicae is it? I wonder if listing of docs available from Next might not be nice.

I know I sometimes feel behind in what is available and this would help.

Doug Brenner, Weeg Computing Center, The University of Iowa, Iowa City, IA dbrenner@umaxc.weeg.uiowa.edu | dbrennpg@uiamvs.bitnet | 319-335-5444.at&t

• Erica - Perhaps in "Buzz' Hints Corner," you could describe how to take a non functioning NeXT and make it a lamp. Important points might be how to get power from the monitor cable, how to avoid violating FCC regs, and creative ideas for what to put into the OD slot. Sounds great! Remember, the article deadline is soon....

David Kay, dbk@esl.com

Finis

The Distribution of Low-Cost Software: What should we do?

Jiro Nakamura

Ever since its release two years ago, the NeXT personal workstation with its optical drive (OD) has been both praised and criticized by market pundits. Supporters of the OD call it the wave of the future, detractors say that it will effectively kill the low-cost software market for the NeXT. I believe both sides have valid arguments.

The OD's main advantages are simple: it can store tremendous amounts of data on a relatively quickly accessible medium and very cost effectively for its size. The NeXT's floptical can hold approximately 230 megabytes (formatted) and it is sold for $50 on the academic market and $100 retail, which is approximately 20 to 40 cents per megabyte.

Unfortunately, it is only cost-effective if you use all 230 megabytes of space. You still have to pay for all 230 megabytes even if the program you have bought on optical only occupies 1 megabyte. And because the base medium costs $50~$100, that means that any software will necessarily have to cost more than that. This doesn't affect large software packages, however it seriously damages the low-end of the NeXT software market, packages that go from $10 to $100. There has been some rumors of inexpensive mini-OD's using the same size format as CD-singles, but these have remained rumors.

One reaction to the high cost of the media has been the "Refundable-OD" used by some companies. They sell their software on OD, but allow the user to send back the OD for a refund, typically $50~$70. Another related method is to allow the user to send in his or her own OD to copy the software onto.

This method benefits both the publisher and the buyer and may become the main way for commercial operations to distribute low-cost software on the NeXT. There are some slight problems with it, such as the psychological barrier of buying a $90 package even though you know you will be eventually refunded $70, not to mention the hassle of shipping the OD back to the publisher.

Some organizations, especially lone-wolf programmers, do not have the facilities or the finances to run such an operation. They need a cheaper and easier way to distribute their software. One center of attention has been how to use the networks to distribute software. Both bulletin-boards (BBS's) and anonymous ftp sites seem like ideal transport mediums. The inherent problem of such distribution methods are that they so easily abusable for pirating. Some way of distributing software while at the same time be properly reimbursed for its use has to be developed.

The concept of ShareWare is not new, but as things stand it may the only way for small software developers to get their products out. For those not familiar with ShareWare, ShareWare products are usually distributed in their entirety in some easily accessible place like a BBS or ftp site with a notice somewhat like:

> "This product is SHAREWARE. You are free to copy and distribute it as much as you want, just as long as you distribute it in its entirety including this ShareWare notice. If you use this product for more than a reasonable 'testing period,' you are obliged to send the author $XYZ.'

The idea is to get the product in as many hands as possible and to rely on people's honesty. Most authors also provide support to the users that send in their ShareWare payments. The problem with ShareWare is its inefficiency. Although this author has no hard figures, saying that less than 10% of users who use ShareWare send in their ShareWare payments would not be totally unfair. Sadly, human nature does not want to pay for things that it thinks it can receive for free.

One thing that needs to be clarified by both the programmers and the users is exactly **what** is being sold. When we buy software, do we buy a product or do we buy a service contract? If software is a pure commodity, then ShareWare will naturally fail since it solely relies on honesty. But if software also entails service, then there would be no problem with ShareWare.

Because its inefficiency and uncertainty over what it means, many programmers shy away from the pure ShareWare concept and adopt other variations on it. Many of the variations do not have popular names so I have taken the liberty of naming them with what I think is the most appropriate or most widely used terminology.

One of these variations of ShareWare is LockedWare (also known as ObnoxoWare). The most widely known example of this is the FrameMaker program that comes on all NeXT system disks. LockedWare has also spawned many different variations of itself, mostly because the concept is simple and the code needed to implement it trivial. Also, LockedWare truly sells a product, not a promise.

Programs distributed under LockedWare usually have one important feature disabled. For example, FrameMaker has the ‘save’ function disabled and will not let you save your work between sessions. The idea is to give the user a taste of the program but not give him or her enough functionality to really use it. If the user really likes it, they can send their money to the company and the company will release the password needed to unlock the program’s full capabilities. Usually this password is linked to something machine specific like its Ethernet address so that it can be used only on one machine.

A hybrid version of LockedWare is LimitedWare. This is what is used by the Sybase database server on the NeXT. It is fully functional, except that it will not accept more than five clients at a time. You must buy the ‘real’ Sybase package if you want more than five clients on a single server.

Another variation of LockedWare is TimeBombWare. Programs under this usually operate normally, but if the user has not registered the password by a set number of days of uses, it self-destructs or destroys its datafiles. This is by far the most obnoxious form of ObnoxoWare’s variations and many people including this author have strong feelings against it.

Another one of LockedWare’s cousins is CrippledWare. This is most often found in ‘demo’ programs of commercial software. This is similar to LockedWare, but all the important code has been yanked, not just simply disabled. If the user wants the software, she or he must buy the whole package from the distributor. It is used more often by larger companies and software packages than by smaller ones.

LockedWare is extremely simple to implement, the code needed to get the network address is only a couple of lines long and all you need is a sufficiently complicated hash routine to work it from there. LockedWare and most of its variations however, share the same problem in that the unlocking password is necessarily linked to only one machine and its network address. This becomes a problem in shared environments or instances where the user will be using the software both at home and at the office, etc. It seems that there is no easy solution to this problem. LockedWare seems the way to go right now, unless the human conscious take a turn for the better and users become much more honest.

Another course of action is to believe what the Free Software Foundation (FSF) says and consider software to be public domain material with everyone having the freedom to copy and to share it. I wholly agree with the FSF's belief that more software sharing will benefit everyone.

The difficulty with Free Software lies in the fact that programmers need to live too. If we all had jobs and could afford to program for fun and for mutual benefit, then there would be no problems. However, many programmers program in order to support themselves and their families. One FSF guru's advice to me was to seek support from a commercial company in developing Free Software.

Sun supports free software, a lot of new GNU software is done in-house by Sun's programmers. GNU apparently supports some development. NeXT also has its own in-house Free Software effort. They are the ones who gave us the GNU Objective-C compiler, Objective-C debugger (GDB), and front-end to the GNU chess program. I can only hope that large corporations will start supporting smaller individual projects. It is beneficial to the companies because the more software their platforms can run, the more popular their machines will be.

Software really is for sharing, it is a commodity that should reach the greatest number of people for the greatest benefit to all. However, the people who write software should be properly reimbursed for their efforts too. My gut feeling is that there should be greater corporate sponsorship for Free Software to small individual efforts. The market needs the diversity that small programmers brings to it and it needs more low-cost software. And so I call on programmers and NeXT alike to form larger channels of communication in an effort to benefit everyone.

Jiro Nakamura
Independent Developer
jiro@vax1.cit.cornell.edu

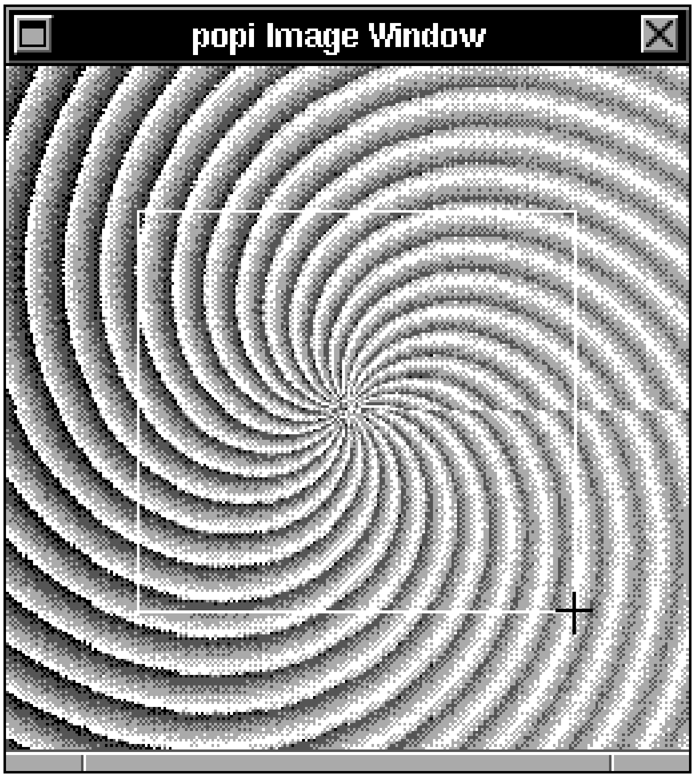
Finis

Copying Arbitrary PostScript in a Custom View

Andrew C. Stone

Abstract

The code to copy an arbitrary rectangle specified by the user is presented. This allows users to simply click and drag out a rectangle to copy the enclosed PostScript to the PasteBoard.



Introduction

Without too much work, you can add methods to copy just part of your custom view to the pasteBoard. This code comes from a very user friendly version of popi, the digital darkroom, that Joe Freeman and I are preparing for release into the public domain. We have added some nice features: the entire language is specified by buttons [no typing necessary], resizable windows, preferences to save set up, you can save images that you generate to an "image manager" for reloading and mixing later, and of course, the ability to copy any part of the image, with the following code. To use the following source code, be sure to add "cross.tiff" to your IB project. It’s the cross cursor which is available in /NextDevelopers/Examples/Draw.

In this example, we also have a method for copying everything in the view: -copyPS:sender, so you can have a menu item to get the entire view. We include an -acceptsFirstMouse method so that the first mousedown begins the process of repeatedly drawing the enclosing rectangle.

Note we keep switching the gray of the instance drawing [the temporary drawing of the rubberbanding rectangle to specify the source rectangle while in the mouse down loop] from black to white, since we don't know what color the drawing in the view is. This could be set to a more appropriate gray level, depending on the contents of the view. Since with popi, there is no telling what color the image is, we are content to keep flipping from white to black, which guarantees that we will see it. Enjoy!

Your bud in Buzz,

Andrew Stone

HEADER DECLARATIONS:

```
- (BOOL) acceptsFirstMouse;
- copyPS:sender;
- gutsCopyPS:(const NXRect *)r;
- mouseDown:(NXEvent *)event;
```

IMPORT FILES:

```
#import <appkit/Cursor.h>
#import <appkit/Pasteboard.h>
#import <appkit/Window.h>
#import <dpsclient/wraps.h>
#import <math.h>
```

CLASS METHODS & 1 STATIC FUNCTION:

```
- (BOOL) acceptsFirstMouse
{
   return (YES);    /// instant reaction
}
- copyPS:sender
{
  return [self gutsCopyPS:&bounds];
}
```

```
- gutsCopyPS:(const NXRect *)r
{
   NXStream *s;
    NXStream *stream;
    NXTypedStream *ts;
    int length,maxlen;
    int i = 0;
    char *data;
    const char *types[2];
    id pb = [NXApp pasteboard];

    [NXWait push];
    types[i++] = NXPostScriptPboard;   /// add other types here
    [pb declareTypes:types num:i owner:self];

    stream = NXOpenMemory(NULL, 0, NX_WRITEONLY);

    [self copyPSCodeInside:r to:stream];
    NXFlush(stream);     // Bruce  Wed Nov  1 09:00:36 MST 1989

    NXGetMemoryBuffer(stream, &data, &length, &maxlen);
    [pb writeType:NXPostScriptPboard data:data
         length:length];
    NXCloseMemory(stream,NX_FREEBUFFER);

    [NXWait pop];
    return self;
}
```

```
// Returns the rectangle which has p1 and p2 as its corners.

static void getRegion(NXRect *region, const NXPoint *p1, const
NXPoint *p2)
{
    region->size.width = p1->x - p2->x;
    region->size.height = p1->y - p2->y;
    if (region->size.width < 0.0) {
      region->origin.x = p2->x + region->size.width;
      region->size.width = abs(region->size.width);
    } else region->origin.x = p2->x;
    if (region->size.height < 0.0) {
      region->origin.y = p2->y + region->size.height;
      region->size.height = abs(region->size.height);
    } else region->origin.y = p2->y;
}

#define DRAG_MASK (NX_MOUSEUPMASK|NX_MOUSEDRAGGEDMASK)
- mouseDown:(NXEvent *)event
{
    NXPoint p, last, start;
    NXRect region, oldRegion,r;
    static const NXPoint hotChamama = {7.,7.};
    int oldMask;
    static id crossCursor = nil;
    float rectGray = NX_WHITE;
    BOOL doubleclick  = (event->data.mouse.click == 2) ? YES : NO;

     // quick copy
    if (doubleclick) return [self copyPS:self];

     oldMask = [window
          addToEventMask:NX_MOUSEDRAGGEDMASK|NX_MOUSEUPMASK];

    if (!crossCursor) {
        crossCursor = [Cursor newFromMachO:"cross.tiff"];
        [crossCursor setHotSpot:&hotChamama];
    }
```

```
[crossCursor push];
p = start = event->location;
[self convertPoint:&start fromView:nil];
last = start;
[self lockFocus];
PSsetlinewidth(0.0);
PSsetinstance(YES);
event = [NXApp getNextEvent:DRAG_MASK];
while (event->type != NX_MOUSEUP) {
  p = event->location;
  [self convertPoint:&p fromView:nil];
  PSnewinstance();
  if (p.x != last.x || p.y != last.y) {
     getRegion(&region, &p, &start);
     NXInsetRect(&oldRegion, -1.0, -1.0);
     PSsetinstance(YES);
     PSsetgray(rectGray=1.-rectGray);    /// alternate b&w
     PSrectstroke(region.origin.x, region.origin.y,
                  region.size.width, region.size.height);
     PSsetinstance(NO);
     oldRegion = region;
     last = p;
     NXPing();
  }
  p = event->location;
  event = [NXApp getNextEvent:DRAG_MASK];
}
 getRegion(&r,&start,&last);    /// final rect
 PSnewinstance();        ///flushes offscreen buffer
 [self gutsCopyPS:&r];   /// send the rect to be copied
 [self unlockFocus];      /// the rest is clean up
 [window flushWindow];
 [window setEventMask:oldMask];
 [crossCursor pop];
 return self;
}
```

Finis

Build a Stereo Cable for your NeXT

Mike Gourlay, Ian Smith

To build a cable connecting the NeXT to a stereo device, find two male 1/8" stereo plugs and three pieces of wire. Cut these to the final desired cable length. Unscrew the base, or *shield*, of the plugs. There will be three terminals -- places to solder wire. Two are short, one long. The long one probably has a U-shaped end. This is for crimping wires to relieve strain.

All you need to do is solder each wire to a similar post inside the plug. If there are three lengthed terminal posts, and you have three wires, one wire goes from long to long, one from medium to medium, the last from short to short. It is very simple.

To get a good connection, either buy shielded stereo cable to make this plug (it will have three wires -- the ground is braided around the two signal wires to keep out radio wave interference) or simply buy a cable pre-made to these specs. I've never seen such, but then, I've not looked very hard.

```
              wires |        terminal |        plug  |
                    V         posts   V              V
  _______                                     _______
/       )-----c~~~~~~~~~~~~~~~~~~~~~~~~c-----(       \
)---------c~~~~~~~~~~~~~~~~~c--------(        ====>
\_______)---c~~~~~~~~~~~~~~~~~~~~~~~~~~~~c---(_______/
```

I hope this picture helps. It's not difficult. By the way, you don't have to solder if you really don't want to. You can find plugs which have screws rather than solder locations. You can make the wire without having to solder, but you'll probably lose the surety of the better connections made through soldering.

Finis

Pull-Down Lists: Discovering the Appkit's Hidden Treasures

by Alan Chung and Jonathan Schwartz

Introduction

We're all fans of the familiar pull-down menu. As brought to the masses by another large California-based computer company, the pull-down menu features prominently in a variety of graphical user interfaces. In fact, just the other day we were trying to work around a user interface problem on our EE CAD tool when we found ourselves saying, "What we need is a pull-down list." Figuring NeXT would have predicted such an occasion, we set the Digital Librarian rifling through the tech docs to find references to "pull-down list." No luck, "0 found."

So we tried “pull-down”—still no luck. In fact, the only reference we could find to pull-down lists ended up in the user interface guideline section—nothing in the Appkit references. It was a bit hard for us to believe that NeXT would have given us pop-up lists without pull-downs, but after spending a considerable time looking through the 0.9 Reference Manual, we concluded this was the case. Faced with adversity, however, development must go on.

So we started to write our own pull-down list. Unfortunately, we were a few days into it before Alan made the *big* discovery: pull-down lists have been in the Appkit all along—just undocumented! We suspect this oversight could be attributable to a certain patent held on the pull-down menu by another large California-based computer company, but that’s a debate for the lawyers. In the meantime, we thought we’d pass on the solution to the absent pull-down list problem, and hope to spare other hapless developers from our sleepless nights.

The Pull-Down List, Documented

The first step to creating your own pull-down list is to create a pop-up list object:

```
pullDownList = [PopUpList new];
```

After you’ve created the object, go ahead and insert items into the list, leaving out the list’s title.

```
[pullDownList addItem:"This"];
[pullDownList addItem:"is a"];
[pullDownList addItem:"test"];
```

And now we perform the magic. The default value of the boolean **changeButtonTitle** is **YES**, which means the list will be regular pop-up. By setting **changeButtonTitle** to **NO**, however, we get our pull-down list.

```
[pullDownList changeButtonTitle:NO];
```

The final step is to attach this pull-down list to a control object either by **NXAttachPopUpList()** or by **NXCreatePopUpListButton()**. These procedures even give the button the correct pull-down list icon. It’s that simple.

And remember, **PopUpList** is a subclass of **Menu**, so it’s not only possible to have separate actions for each item in a pop-up/pull-down list, but to have keyboard equivalents as well. Just replace the above **addItem:** message with **addItem:action:keyEquivalent:** and the Appkit handles the rest.

```
[pullDownList addItem:"Deposit"
          action:@selector(deposit:)
          keyEquivalent:68];
```

As a final note, before you go ahead and use pull-down lists in your application, we'd recommend familiarizing yourselves with the user interface guidelines to better understand their role in the NeXT universe. The pull-down should be considered a last resort user interface option, to be used only if all other methods, including regular menus, are deemed unsatisfactory.

The authors of this article, Alan Chung and Jonathan Schwartz, are partners in Lighthouse Design, creators of electrical engineering CAD tools for the NextStep environment. They can be reached by electronic mail at lighthouse@lighthouse.com or ...!uunet!lighthouse!lighthouse, by physical mail at 6516 Western Avenue, Chevy Chase, MD 20815-3212, or by telephone at 301-907-4621.

Finis

Working Through the NeXT Developer Camp Laboratories, Or, How I Spent My Christmas Vacation

Dick Silbar, Los Alamos National Laboratory

In December at Los Alamos we are forced to take vacation between Christmas and New Year's Day. This is normally not a serious problem for me because, if there is snow, my wife will take me skiing every day. This year, however, there was not enough snow to ski at all, here in the Great Southwest.

On the other hand, that allowed me to catch up on my homework. In early December some NeXT People came to Los Alamos to teach the first Developer's Camp off the grounds of NeXT. I participated and had a great deal of fun in the process. The unfortunate part was that, at the time, I really only had a chance to *touch* on the Labs that go with the course. Thus, with a snowless Christmas break looming ahead, I arranged to detach "my" NeXT from the Laboratory Net and take it home for Christmas.

I really had in mind to get started on an application that I've planned for some time. However, being a beginner (and you will see in just a minute just how much of one I am), it occurred to me that I would be better off if I first finished working through the Camp Labs in detail. The story that follows is somewhat reworked from a more stream-of-conscious version that went off to our Camp Instructor, Bruce Blumberg, for purposes of feedback on the content of the course and the associated Labs. In the process of writing these experiences down, however, it finally dawned on me that what I had learned here might be of interest to *other* people who are also starting out to program the NeXT.

Why and How Should *You* Do These Labs?

Why should you do these labs? To learn how to use your NeXT, of course. It is possible to do this even in the absence of being able to go to Developer's Camp. The Labs themselves are available for downloading by anonymous FTP from, say, Gerritt Huizenga's NeXT archives at "j.cc.purdue.edu". There are four of them, one for each day of the Camp: **CalculatorLab**, **CompositeLab**, **TextLab**, and one of undetermined form (since it is a choose-it-yourself project). Basically, these Labs are partially-baked applications that are broken in some places and incomplete in others. The Exercises are to fix the broken places and complete the coding.

Before you begin on the Labs themselves, however, I would suggest first working through the simple examples of how to use the Interface Builder (henceforth, "IB") given in Chapter 8 of the (on-line) System Reference Manual. Follow this by printing out and *reading* Chapters 6 and 7, which fill in on the philosophy of NeXT Program Structure and Dynamics. After, or during the time that you then start in on the Labs themselves, you may also want a printout of the Class Specifications given in Chapter 22.

The Author Really *Is* a Beginner!

That is to say, I don't qualify as a trained-in-school Computer Scientist, as I assume most of the readers of Buzzings are. I'm a practicing nuclear physicist, and have been for some time. Recently, however, I have become interested in somewhat more modern programming techniques than are exemplified by Fortran programs for solving singular integral equations for scattering amplitudes.

I came to the NeXT machine last April with essentially no experience in UNIX or in C programming, much less in Objective C. On the other hand, for somewhat peculiar reasons, I already had fairly extensive recent experience programming in Common Lisp and Scheme and I did know a fair bit about object-oriented programming in general.

By the time I got the to Development Camp course, I had used the NeXT machine in my office for about seven months, and thus I knew basically how to get around in the file system and get things done. I had also worked through the "Adobe Blue Book", the PostScript Tutorial and Cookbook (and had written *one* PS program, which is mentioned below). Moreover, I had read the first two-thirds of the book on Objective C by Brad Cox, but I had never really tried writing any code in Objective C before the Camp.

Heading into Camp I had two things on my agenda. The first was, "How do I make an Emacs Tool?" Something with a lot of mouseability like the Tool which comes with SunView. (I also began using a Sun 4, for separate purposes, at about the same time the NeXT arrived.) The second intended project was a more "traditional" situation for a NeXT Application: to make a "**LensLab**" for designing optical systems, either of light or charged-particle beams.

I was somewhat disappointed in not learning, in the Camp, just how to get a NeXT application to talk to an existing non-NeXT application, such as the GNU Emacs that comes bundled with the system. Evidently the answer has something to do with pipes or Unix sockets or Mach ports, and none of this material was covered in the Camp. On the other hand, this sort of material is -- I am told -- part and parcel of the regular baggage carried by ordinary C programmers. I guess that means I still have to do some homework before I can tackle my EmacsTool project!.

General Remarks on the Interface Builder

The Interface Builder itself is a remarkable Black Box, but one which deserves some comment. Besides the general remarks given here, there are particular criticisms I will make in the Lab discussions, in the context of the troubles I was having at the time. Blumberg tells me that these kinds of comments are all "good", but that most of them had already been made to the people at NeXT in charge of IB improvements. Perhaps some of the things I am nagging about here will be cured in Version 2.0. One good reason for mentioning these sorts of things here is that it might serve as a public reminder of Things That Ought To Be Fixed in Version 2.0.

I don't like the choice of words "parse" and "unparse", and I don't even know what the two words mean. (I suspect they were chosen by someone who was mostly concerned with building compilers). More to the point, however, I am frankly still not sure what all happens when these things are done in the IB. There seem to be some dangers here even beyond the obvious one of over-writing all your previous programming efforts. For example, will "parse" sometimes disconnect things? I can't prove it, but I have the impression that it might.

The documentation on "parse" and "unparse" is, to coin a phrase, sparse. There are some words and warning on p. 8-31 and four not very informative paragraphs on p. 8-81 of the System Reference Manual. In particular, I also have the impression there have been times when I *needed* to "parse" one of my classes -- as when I added outlets or actions -- before things would work (allow me to make connections). There is nary a word to that effect in the docs.

It would be helpful if the unparser put in the "`#import <appkit/appkit.h>`" line automatically. I realize that might well include more than one actually uses, but it is very easy to forget this line. At the beginning of a project, the programmer doesn't often know all of the appkit things that he is going to use. I suspect it only costs in compilation time, but if each programmer makes about two mistakes by not having the proper ".h" files imported, the recompilations will eat up all that economy.

It seems to me that if I make a change somewhere in the IB files for a project, the little "X" in the northeast corners of *all* the IB windows should change into the broken "X" icon, which indicates that the IB files have not been saved. Apparently the only such "X" which changes is that on the "Files Window". However, many times that window (or the NE corner thereof) is covered over by something else. I was stung many times this Christmas by not saving the IB project before doing a make.

That last point requires some clarification, perhaps. It became my habit to do the successive makes and loads from a standard Terminal window. One of my reasons for that was that, if I tried using the Make item in the IB main menu, I got hung up with an interactive input request I had stuck in my .login file. (From the UNIX prompt there is no way for the Makefile to know whether the IB files had changed.) However, when I remove the extraneous prompt from .login and choose the IB menu Make item, the IB does ask if it should update its files first. [*Me too. I always use the shell for Make--EJL*]

On another topic, it seems to be *very* inconvenient to change the name of a project or a class. (In contrast, it is no trouble at all to change the name of an outlet or action.) All unparsed files, the IB.proj file, and the Makefile must be thrown away or laboriously edited. Not fun at all.

As things now stand, there is a lot of iteration of the **{fix-code, make, execute}** cycle , and compilation of even my relatively small codes is pretty slow. (Blumberg has assured me that improved compilation speed is a very high priority for Version 2.0.) It would be nice to be able to see how a method behaves *in* the IB, i.e., to be able to interpret pieces of code interactively. And then make the fixes on the fly. This is clearly possible, since the LISP machine people have enjoyed such a development environment for some time. In fact, I've heard rumor of such interactive tools for C++. Maybe there even is something like this for Objective C.

Particular Remarks on the Labs Themselves

The following paragraphs may only be understandable if you've already looked at the Labs and tried some of their exercises.

There were no particular problems, for me, in finishing the **CalculatorLab** (Lab 1), but as a C-beginner I have a lot of trouble remembering to put the ";" at the ends of lines. (I almost never seem to forget the "]", of course, but that's a LISPishness learned some time ago.)

The **CompositeLab** (Lab 2) was somewhat more trouble. First, in my fumbling around, I somehow ended up defining Subclass1 (of Object) and Subclass2 (of Subclass1) before I realized that the custom CompositeView class was really a subclass of View. In the meantime, I evidently parsed and unparsed and instantiated some of these things at various odd times (I don't remember it all). In any case, when I decided it was time to *delete* these extraneous classes, I could not do so: "Subclass2 is being used". (If it weren't, the appropriate procedure is to highlight it in the Classes Browser and then hit the <delete> key.) There needs to be a way so that one can find out, in the IB, *where* a class is being used, to be able to delete it. For that matter, I could find nothing in the System Reference Manual on deleting a class, but that is surely something a general IB user will want to do sometimes.

I cured this particular problem of not being able to delete one class by scratching all my previous work on this Lab and starting over. Clearly that is not something you want to do if you are deep into a *big* application.

My second adventure in Lab 2 was in learning how to set the sliders in the proper initial positions at load-time. The easy solution (which I presume is what was intended, since there is no reference to this in the Solution) is to set the initial values for each slider in the Attributes Window of the Inspector. However, in class Bruce discussed a way to do so programmatically, by setting up outlets from the CompositeView instance to each slider:

a) put "id sourceAlphaSlider" and "–setSourceAlphaSlider" into the .h file.
b) put "-setSourceAlphaSliderPosition:sender" into .h.
c) put "[sourceAlphaSlider setSourceAlphaSliderPosition:self];" into +newframe.
d) define methods as follows:
e) make sure everything is parsed and saved.

```
-setSourceAlphaSliderPosition:sender
{
   [sourceAlphaSlider setFloatValue:sourceAlpha];
   return self;
}

-setSourceAlphaSlider:aSlider                          {
   sourceAlphaSlider = aSlider;
   return self;
}
```

Alas, this never worked for me (at Christmas time). I now think that, perhaps, I forgot to actually define and connect the outlets here! So, someday soon I may go back and try this again. I can see where this programmatic technique could be very useful in other contexts.

My next adventure had to do with learning how to use "pswrap"for drawing a PostScript picture. Unfortunately, pswrap is *very* poorly documented; it badly needs a more complete example about how to use it *in* the IB and *with* ObjC. All I could find was a skimpy "man" entry and a small section "Using pswrap" in Ch. 4 of the System Reference Manual (pp. 49-50). (There is more information, I am told, on "pswrap" in Vol. 3 of the Tech Docs, but those are not presently on-line and searchable with Librarian.)

What I was trying to do was to replace the FLOWER option with a piece of PS code that I had developed earlier, a Zia Sun symbol with crossed squash racquets (the logo of the New Mexico Squash Racquets Association). Eventually (after a lot of hacking) I found that the simplest way to make it work (that I could find) was:

a) to put the PS code in a separate file, "drawNMSRA.psw" (note extension), with the "defineps drawNMSRA()" and "endps" around my code.

b) have a "void drawNMSRA();" at the top of the -drawSource body.

c) define an internal method in .m (and note it in .h):

```
-drawNMSRA
{
   void drawNMSRA();
   drawNMSRA();
   return self;
}
```

d) call the method in drawSource with:

```
case NMSRA:
   [self drawNMSRA];
   break;
```

e) be sure the Project includes "drawNMSRA.psw" in the .psw files.

This works. But there may be a better way.

The **TextLab**, Lab 3, is a lot harder than the earlier two, particularly for those who don't know much about pointers and declarations in C (like me). I had to cheat a bit, by looking at the solution file, to get Exercise 1, which is to add scrolling capability to the Text Window, to come out right. (During the class my lab partner and I *were* able to get it right without cheating, but I had forgotten how in the intervening two weeks.)

Going on, I then got very much hung up on Exercise 2, which is really a very simple thing, once you know where things are. My problem was *not being able to find* the instantiation for the application's main menu. Double-clicking on the menu icon in the Files Window does *not* bring the menu up onto the screen!

I eventually found the main menu *under* the IB menu. Where it evidently is always put, whenever a new application is loaded into the IB. I consider this a bad feature. The IB would be much more understandable to a beginner if the menu for the application under construction were visible *from the start*, e.g., just under the IB menu. (Perhaps greyed?). At the very least, double-clicking on the MainMenu icon in the Files Window ought to bring it up to the surface, somewhere, even if only temporarily.

For Exercise 4a I had to cheat quite a lot to understand what to do. In fact, I really *don't* understand the declaration

```
const char *const tlType[2] = {"tl",NULL};
```

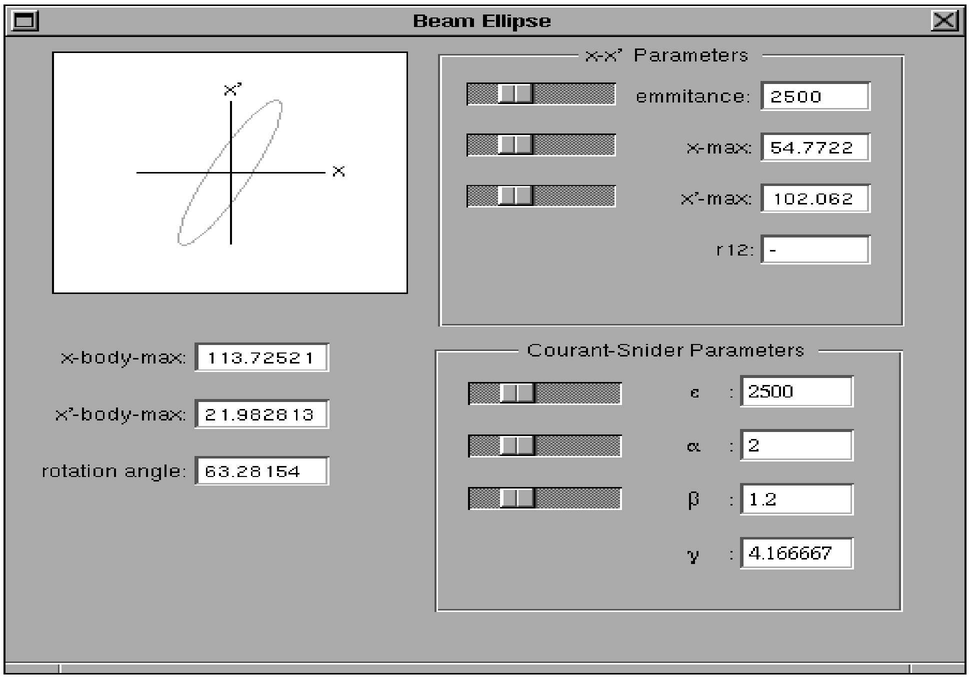
at all. The books on C that I had at home during the Christmas break are not very highbrow, but neither of them says anything about "const" or "*const". That, I now know, is because the above declaration is something "new", namely ANSI-C. OK, I am exhibiting my ignorance of the base language here, but I suspect that getting to the above declaration from what's in the Spec Sheets on "runModalForDirectory:file:" is a tough problem even for moderately good C programmers.

I *might* have gotten to the above declaration by myself if the statement of the problem had given a hint to look at the `openRequest` method. (I confess to not having read over the code very carefully before trying the exercises.) Even so, I still wouldn't have understood the declaration.

When I finally got **TextLab** to work, I was a little surprised *not* to see the "X" become the "broken X" after something is written to the text-window. I am still curious how one does that. Also, the File Name displayed in the textfield of the Save Panel is the *entire* pathname, not just the name as it appears in the directory. It would be nice to know how to truncate this to something a bit more readable.

Beyond the Cookbook Labs

For my Lab 4 -- a project of one's own from scratch -- I decided to build an interface to set up and display changes to a beam phase-space ellipse. (A phase-space ellipse is a first-order optics description of a non-point-source beam.) On New Year's morning I succeeded in getting it (sort of) right:



Apart from my problems with the spelling of "emittance", there were other problems along the way (that are of general interest).

Most scientists and engineers need to use an occasional Greek symbol. I wanted to have coordinates *x* and θ = *dx/dz* (instead of *x*') for my phase space. There is the Symbols Font available to me, but *within* a Matrix of Form Cells everything must be either *all* Roman or *all* Greek. Damned inconvenient.

Then I had the question of how one sets the texts in a Matrix of TextFields. I was expecting a method like "setFloatValue:at:" In this regard, I must confess to being a bit confused; I couldn't see at first, from the documentation, that "setFloatValue:" would work on a TextField. Even though I knew that we did just that in "displayValue" for the field "viewer" in the CalculatorLab. I now know that "setFloatValue:" is, through TextFieldCell, inherited from Cell. Thus, another question is, how can one learn of *all* the methods available in a class? The write-ups in Ch. 22 only give those methods defined in that subclass -- one has to also look at the write-ups of all the ancestors?

The situation that gave rise to the Matrix of TextFields questions was eventually finessed by going instead to a Matrix of Forms. Since then, however, I have found that what I wanted to do could have been done in steps using something like

```
[[myTextFieldMatrix cellAt:3 :0] setFloatValue:thetamax]  .
```

Another problem I had arose, once again, from my lack of C experience. One has to be sure to import *every*thing one needs into a class definition. I was getting an innocuous warning "method not found" when messaging my EllipseView instance inside a BeamEllipse class method. But since that flashed by and the make otherwise compiled, I didn't pay all that much attention. On loading and executing, however, the instance variables in EllipseView were not actually being set. Thus I was then getting all sorts of strange errors, all of which looked like PostScript problems that made no sense at all (in view of the PS code I had written). These all miraculously disappeared when I finally realized I had to add '#import "EllipseView.h"' to BeamEllipse.m.

I also had mysterious problems in drawing my (tilted) ellipses in my CustomView, which came out filled and looking like PacMan, even though there was no fill and I completed my arc from 0 to 360 degrees. At first I thought this was a bugginess of the PS C-functions, but on trying it also with regular PS code in the Yap App, it eventually became clear that it was me who messed up, not them. I was drawing my ellipses with *very* thick linewidths.

Another PostScript Surprise was to find that my labeling of axes (with PSshow) at first had the characters upside down (rotated about the *x*-axis). I couldn't see where, in the documentation, this is discussed as the normal DPS behavior. However, it is curable by using "1 -1 scale" (or its PSscale(...) equivalent) to get Post Script to print out strings upright.

And a final surprise: on loading of my application, the first drawing (during initialization) in the CustomView comes in greyed, not with full black lines. In particular, the NXFrameRect() at the beginning of my drawSelf method didn't show up black the first time around, unless I specifically require it so with a PSsetgray(0).

And Conclusions

Well, the major conclusion here is that I did have some fun over the Christmas break and in fact got a good start on learning more about how this beautiful black machine on my table actually works. It might have been better, however, if there had been a *little* snow this year.

Finis

silbar@whistler.lanl.gov (505)-667-5253

Creating Simple, Elegant Icons - A Brief Introduction

Erica J. Liebman

Abstract

Simple, elegant, professional icons can be created using the "icon" drawing program included on the NeXT Distribution disk. Stone Design's TextArt adds flashy titles to these icons. Additionally ray-tracing icons is considered. Examples are included from Quality/Liebsoft's public domain disk documentation program for Lighthouse Designs..

Introduction

In late January, I needed to create icons for a *ReadMe* program that Quality Corp (the family company) prepared for Lighthouse Designs. This called for a large variety of distinctive icons in a relatively short period of time. I turned to the *Icon* program from the standard NeXT Distribution and, to my delight, also found the true purpose for which *Text Art* was designed. Now, Andy Stone may disagree with me as to *Text Art*'s "true purpose", but he won't be put off by sales figures when people see what a great little programming tool it is. *Text Art* was made for creating icons.

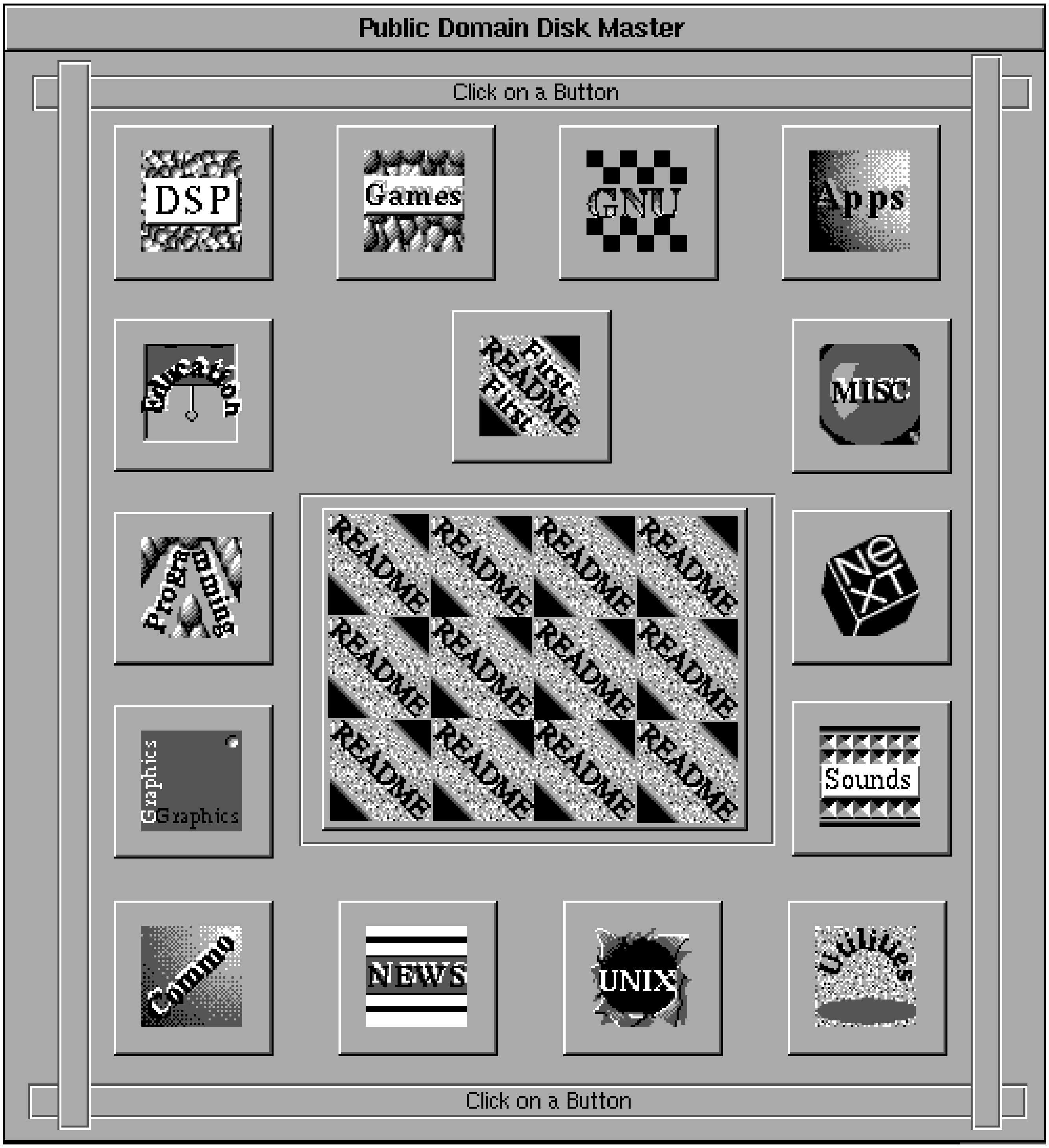
Three Steps to Icon Creation

It took only three "manufacturing steps" to produce my icons for *ReadMe.* Gaining familiarity with *Icon* and *Text Art* does require some time, but is well worth the investment. After the first two or three icons, I found I really started to achieve mind-bending creation speed. To begin make sure you have both programs running and a fresh "new" worksheet to play with in *Icon.*

Before starting the first icon, try to become familiar with how the "detail" window works. This small window is exactly the size of a standard NeXT icon. All your artwork must fit within this window. Moving the detail tool on the worksheet changes the contents of the detail window. Follow these steps to create the icons.

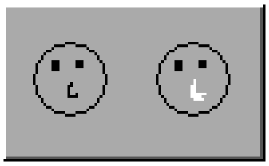
1. Create a Nice Background. Use the rectangle or paintbrush tool to create a background larger than the finished icon. The Shades palette has a variety of regular patterns. You may also paste in some pre-existing artwork from the Shapes menu. Bored with this trivia? Adventurous and creative? Create your own "brush" and use the Effects palette for picking up and pasting down. If you do this, make sure to set the composite option in the Tools palette to Sover or a similar compositing function. Can't remember what that compositing function is? There's a handy reference to compositing in the Shapes palette. Open a fresh window and drag down the shape from the lower right hand corner.

2. Create and Place Your Title. Click on the *TextArt* mini-window to bring the application to the forefront. Use 12 or 14 point text, and shape as desired. Copy the graphic and move back to *Icon.* Open the detail window, and move the detail icon to an interesting part of your background. Click on the detail window and paste in your artful text. Move to where it looks nice and use the Effects palette to paste down the title. Be sure you're in Sover mode on the Tools pallet or else you'll end up with a lot of white-space you didn't intend.



Icons Panel created for ReadMe

3. Add Borders, Touch-up and Save Make any finishing touchups. You may want to add a border to demark a textured background. Just beware that you are using *alpha* values in your creation of icons. For example, if making corrections by "painting in" with white, you'll have white spots on your finished icon. Click on the detail window and Select-All. Save the selection by choosing "Save-As" from the Window menu. Be sure the selections are set to Tiff, Save-Selection and With Alpha (versus EPS, Save-Document and No Alpha) before changing the name and clicking OK.



Using "White-Out" doesn't work with Alpha-Values

Adding Icons to Interface Builder

Icons can be added to interface builder by dragging the TIFF Icon from the directory browser onto a project window. The appropriate suitcase will automatically open. To add an icon to a button, just drag it from the icons window onto the button. When the button gets a border, let go. The button will automatically resize.

The exception to this method is the Application icon. In the Project Inspector, (for "Attributes" not "Files"), you have to add the application icon separately by choosing the set button and selecting the tiff file. If the file isn't in your current directory, you will be offered the option to make a copy.

What Makes a *Good* Icon?

Icons differ from pictures in that they have implied functionality. Good icons should reflect information. The instructions above will create only workable, even professional icons, but it's important to understand the techniques of good icon design. Icons come in three basic types . *Picture* icons directly reflect their purpose, such as a pencil for drawing. These are easily recognized, retained and designed. These often are direct pictures of tools they emulate. *Symbol* icons are metaphors. These too are easily taught and retained, but harder to design. An example is a picture of a train labeled "+" dragging two cars with numbers behind it, indicating that this icon is used for adding two quantities. *Sign* icons are arbitrary. An example of a *Sign* Icon is the radiation symbol. This symbol was designed and has no relation to the reality of radiation. The advantage of *Sign* icons are their long lifetimes once they enter into popular use, but they have no direct associations and are harder to learn.

To Trace or not to Trace?

Probably the best way to design picture icons, outside of digitizing a snapshot, is to use Ray-Tracing or similar drawing methods to create a realistic icon with shading, shadows, etc. Ray-tracing works by following a single light-ray from an "eye-point" and determining the ray's source. When a traced-ray hits an object, it will reflect, refract, or simply indicate the degree & absorption of "photons" coming from various light-sources hitting this surface. Ray-traced icons, face it, are *cool*. Ray tracing can be used by those of us without any artistic talent whatsoever. Until a cheap package comes along though, this method of icon creation is beyond most of our reach. C'mon NeXT, tell us how you *really* created that floating screwdriver for Interface Builder's icon.

Conclusions

With just two programs, you can create some spiffy looking icons. *Icon* is available in the NeXT distribution package under /NextDeveloper/Demos. *Text Art* is available from Stone Designs. Together, they reduce icon building to a few simple steps.

Finis

Using the Interface Builder to include pre-drawn bitmaps in a program:

Carl Sutter

There are several ways to include bitmaps in an application through the Interface Builder. Each method has its quirks, but the process ends up being quite simple once you know what to look for.

The first step is to make a bitmap (perhaps with the Icon application). Save it as a TIFF file like Monkey.tiff. There are two ways to load it:

1) Drag the TIFF file into the icon suitcase. If the bitmap is used on a button, it will be included as a named bitmap in your final application. You can then load the bitmap with the findBitmapFor: method of the Bitmap class.

```
Bitmap *bmpMonkey;
bmpMonkey = [Bitmap findBitmapFor:"Monkey"];
```

Note that `findBitmapFor:` will also work with the system bitmaps listed in the Bitmap class documentation. You can put these system bitmaps on a button by typing the name in the button inspector icon field. (Note that a bug in IB 1.0 forces you to first drop any icon on the button before typing the system bitmap name). In your program, you can access the Application Icon with the name "app". Any document icons can be found under those file's extensions set in the IB like "txt" or "wn". Note that the bitmap must be used on a button for the findBitmapFor method to work! If it is not used, it will not be loaded from the .nib file and subsequently not added as a named bitmap.

2) Add the TIFF file to the Project Inspector .tiff Files section. You can then load the bitmap with the newFromMachO: method of the Bitmap class. This method has the advantage that the bitmap isn't used on a button.

```
Bitmap *bmpMonkey;
bmpMonkey = [Bitmap newFromMachO:"Monkey.tiff"];
```

Using Bitmaps for Fast Drawing

The basic technique on the NeXT machine is to create an offscreen bitmap, draw into it, and then composite (bit-blt) it on the screen when you need to see it. The NeXT machine can composite extremely fast, and you can have transparent parts in a bitmap. You can also use all the powerful compositing modes when drawing. (see the docs for details)

First, declare a pointer to a Bitmap instance in the interface file:

```
Bitmap  *bmpMonkey;
```

Don't forget to `#import <appkit/Bitmap.h>`

The rest of the code goes into the implementation file. Next, make a new instance of the Bitmap object: (maybe in the new, or newFrame method)

```
{
 NXSize  nxsMonkey;
// size of desired bitmap

 nxsMonkey.width = 50.0;
 nxsMonkey.Height = 100.0;
 bmpDemo = [Bitmap newSize:&nxsMonkey
         type:NX_UNIQUEBITMAP];
 }
```

To draw in the bitmap:

```
{
 [bmpMonkey lockFocus];
    PSsetgray( NX_BLACK );
         /* whatever PS drawing code you want    */
    PSmoveto( 10.0, 10.0 );
          /* this just draws a short line segment */
    PSlineto( 30.0, 30.0 );
    PSstroke();
 [bmpMonkey unlockFocus];
}
```

The creation and drawing could be combined by loading an existing bitmap with the newFromMachO or by a similar method. To draw the bitmap on the screen: (maybe in the drawSelf method, so the focus is already set, otherwise lock the focus on the receiving view first)

```
NXPoint  nxpLocation;

nxpLocation.x = 50.0
nxpLocation.y = 100.0
[bmpMonkey composite:NX_COPY toPoint:&nxpLocation];
```

Carl F. Sutter
Academic Software Development Consultant
University of Southern California
University Computing Services
(213) 743-3826   sutter@ozone.usc.edu (NeXT mail)

Finis

An Open Hypermedia System for the NeXT

David Kay, ESL

ESL's new non-commercial hypermedia system provides "integration by reference" between NeXT applications.   Hyperworld, allows application builders to add hypermedia functionality to products while conforming to NeXT interface guidelines. Hyperworld is intended to make creating and traversing inter- or intra-application links as accessible to the user as cutting, copying, and pasting. Like cut, copy, and paste, the user need not know the mechanics of the process or the differences among data types; hypermedia functionality is there when needed and otherwise unobtrusive.

What It Is

Hyperworld furnishes various hypermedia functionality. It provides:

- Transparent link management with Sybase
- A powerful application programmers interface
- An open architecture for all-media integration
- An efficient annotation capability
- A complete linking facility, with user-definable link types
- The HyperBrowser, a sophisticated tool for link navigation.

Additional tools already included are HyperWrite, a Hyperworld text editor, and DataDisplay, a Hyperworld signal viewing application. A Hyperworld image editor, map editor, and graphical signal processing language are currently under development.

About Hypermedia

Hypermedia allows data in applications to be linked with other data in the system. This linking accomplishes integration by reference; applications can be launched transparently by other applications by following links. For example, a picture of a person might be linked to biographical text about him, and these might be linked to by a mention of his name in another document. NeXT Applications integrated through hypermedia allow users to create chains of reasoning, complex presentations, and institutional memories in a way that no single tool alone could support.

How You Add It To Your Application

The Hyperworld class library should allow the application developer to add hypermedia functionality to a new or existing application with minimal effort. (Our experience indicates that integration takes an experienced NeXT developer about one week per application). Each Hyperworld application must be able to refer to blocks in its data; these blocks are the starting or ending points of Hyperworld links. For example, a word processor could use a range of characters as a block; a spreadsheet could use a range of cells, and a graphics display could use minimum and maximum x and y values. Once the application can represent its chosen block style, Hyperworld takes over. The Hyperworld BlockManager class handles block representation, links, link types, and the Hyperworld class implements the Sybase interface.

What This All Means

Unlike previous hypermedia solutions, Hyperworld is not a single closed application, but part of the application infrastructure. Any NeXT application can support linking and link navigation with minimal development. Just like cut, copy, and paste, hypermedia functions can become a user-expected feature of NeXT applications. And because Hyperworld takes advantage of the Sybase data server that comes with every NeXT computer sold, link data is quickly, efficiently, and securely stored.

How To Get More Information

Please note, Hyperworld is *not* a commercial product. For more information about Hyperworld technology, contact ESL’s Workstation Products Manager at (408) 752-2525.

Finis

An Objective-C Run-time Class Browser

Edward Jung, DTG

Abstract

NeXT[1] has chosen Objective-C® as their system-wide programming language. Objective-C is an object-oriented language standard developed by Stepstone Corporation (formerly Productivity Products International), and implemented by NeXT as extensions to the Free Software Foundation GNU C compiler "gcc".

Introduction

The NeXT Objective-C compiler is one of many object-oriented language systems available today. Objective-C differs from most other hybrid object-oriented languages, however, in its flexibility at run-time; this flexibility is exploited by tools such as Interface Builder.

The run-time flexibility is available because the NeXT compiler produces "meta-data" for every object and class – data stored about the object system that is accessible at run-time. I will present the meta-data stored for objects, and how the run-time system uses this data. In the course of the presentation, I will demonstrate how to access the run-time system by developing an application that can browse all the Objective-C classes that are linked into a run-time image. Note that the documentation shipped with the NeXT system is unclear and in some places incorrect.

The reader is expected to be somewhat familiar with the fundamentals of object oriented languages in general, and Objective-C. I will be using the Smalltalk term "class" in place of the term "factory" used in Stepstone documentation.

NeXT Objective-C is not Objective-C

Objective-C is a message-typed, hybrid, compiled, object oriented language, more closely related in philosophy to Smalltalk than to C++ or Object Pascal but lacking garbage collection. Its object model is a single inheritance, class based, late-bound, message passing, message (non-strict) typed, compiled language. Importantly, this model is expressed through a run-time support system rather than being wired into the language, and therefore is amenable to use in other languages, such as Fortran or Common Lisp's foreign language interface. The NeXT Application Kit and Interface Builder® components of NeXTStep® are strongly dependent upon the use of the run-time system of the Objective-C object model.

---

1 Objective-C and Software IC are registered trademarks, and ICpak is a trademark of Stepstone Corporation. NeXT and Interface Builder are registered trademarks of NeXT Incorporated.

The NeXT Objective-C implementation differs from the Stepstone standard in many subtle ways. Firstly, the NeXT Objective-C compiler is integrated into the normal (ANSI) C compiler rather than being a separate front-end compiler (sometimes called a translator) emitting C source code. This results in faster compilation, better shell-level semantics, and more friendly source-level debugging. As a consequence, some debugging features of the Stepstone system are no longer necessary and are deleted.

Secondly, NeXT has introduced the concept of class categories, which may promote modularity and flexibility of class interface and implementation, potentially at the expense of class maintenance in programming-in-the-large (by introducing project context sensitivity to class declarations).

Thirdly, NeXT's run-time system uses delayed or lazy initialization of classes; this decreases start up time, but has the unfortunate side-effect of altering the semantics of class posing (a powerful albeit seldom-used feature). NeXT's current Objective-C compiler does not support stack-allocated objects, nor the ability to statically bind classes (an option usually used to reduce message passing overhead when the classes have been frozen). NeXT maintains internal data structures in the run-time image that facilitate run-time object behavior modification; these structures are different than those of Stepstone.

One resulting side-effect is that class objects are no longer considered rvalues (more on this later). NeXT uses a vastly different archiving mechanism than Stepstone, and thus cannot use much of the Stepstone ICpak "Software IC" libraries without modification. In general, however, NeXT's compiler is *mostly* source-level compatible with that of Stepstone, and is certainly identical in philosophy.

An Aside: a Brief Comparison to C++

Inevitably comparisons between Objective-C and C++ 2.0 (the emergent object-oriented "standard") will be made. The languages are both hybrids of C and an object model, but they are quite different in syntax and semantics. For purposes of this article, C++ will refer to the AT&T CFront 2 standard released mid-1989. Since this topic is sufficiently complex to warrant an article of its own, I will make some simplifications in this comparison.

C++ has features such as distinction between private, protected and public members, which is both important and useful for software engineering purposes. The explicit declaration of virtual methods may result in improved performance. Multiple inheritance may prove to be useful, although the semantics may prove to be unwieldy in the maintenance of large evolutionary projects. C++ allows stack-allocated and reference objects, however careful understanding of the allocation of these objects is important in order to avoid severe performance penalties.

The validity of message passing in C++, like Object Pascal, is controlled by subtyping. It has been shown that subtyping is not sufficient for encapsulated behavior, and subtyping interferes with tools such as Interface Builder which are purely message-based. The subtyping model makes the inheritance matrix of classes extremely important; it becomes the most important issue in class design. In summary, C++ is a large, powerful language that can be difficult to master.

Objective-C is purely message-based, which means that the validity of message passing is controlled only by the message, and has little to do with the inheritance matrix (subtyping). This allows instances to delegate behavior, or respond to any message regardless of where it is declared in the inheritance matrix. This model permits the passing run-time computed selectors as messages, and has other features lending exceptional run-time flexibility. Although good for prototyping, incremental class definition, and tools such as Interface Builder, this model can be detrimental to correct implementation of projects and management of complexity. The compiler, however, is able to perform subtype-based static checking (and emit appropriate warnings), which may be defeated using casts or ignored entirely. Many errors, however, still cannot be caught until run-time. This seems to be a comfortable compromise, although its scalability remains unproven.

The bottom-line is that C++ will almost always out-perform Objective-C in speed, at the expense of run-time flexibility. There are, however, many "features" of Objective-C that can be used to maximize performance at the expense of readability and generality. Objective-C is also a very open object system. Both suffer from being hybrid languages.

A Brief Review of Syntax

Objective-C adds few syntactic constructs to ANSI-standard C. First is the message passing syntax:

```
return_value = [receiver_object selector];
```

The bracketed expression passes the message selector “selector” to the object “receiver_object”. The return value of the method invoked by this message becomes the value of the expression. This is closely patterned after the Smalltalk syntax. Message selectors can be unary selectors that do not pass arguments, or keyword selectors that do pass arguments. Keyword selectors have colons embedded into their names; every colon represents an argument. Thus the selector “setIntValue:” takes one argument, while “setValue:andUpdate:” takes two. These arguments are passed embedded into the keyword selector as follows:

```
return_value = [an_object setValue:10 andUpdate:YES];
```

A special case is worthy of note: the receiver “nil” is the universal receiver. Any messages sent to “nil” always return “nil” without error. This can be used to reduce the amount of checking in code for valid receivers, or for terminating cascaded messages, so in the following example:

```
[[anObject findLastObject] doThis];
```

the “doThis” message will only have effect if the method invoked by the “findLastObject” message returns a non-nil value.

The next syntax extension is in the declaration and implementation of classes. By convention, a class declaration appears in “ClassName.h”, while the corresponding implementation appears in “ClassName.m”.

A class is declared using the following syntax:

```
@interface ClassName:SuperclassName
{
// instance variables
}
// class and instance method declarations
@end
```

Method declarations are of the forms:

```
+ (return_type) selector;
+ (return_type) selector:(arg_type)arg...;   // class method
- (return_type) selector;
- (return_type) selector:(arg_type)arg...;   // instance method
```

Unlike ordinary C where the implied type is “int”, an integer, Objective-C declarations have the implied type of “id”, which is an object reference[2]. As a convention, all method return some value. If the method cannot return an otherwise useful value, it should return a reference to the receiver.

Implementations use another syntax extension:

```
@implementation ClassName:SuperclassName
// method implementations
@end
```

The appearance of the superclass name is optional. Method implementations use forms similar to method declarations, for example:

```
+ classMethodDeclaration
{
// C code
}
```

2 Technically the type id is a pointer to a structure describing an object – the exact definition appears in /usr/include/objc/objc*.h.

Within a method, three special symbols may appear: “self”, which is a reference to the receiver, “super”, which is a reference to the receiver’s superclass, and less frequently (almost never, actually), “_cmd”, which is the method message selector.

Categories are a NeXT extension to Objective-C. Given a class that has already been declared using the aforementioned syntax, a category or extension to the class may be declared at some other point in the source:

```
@interface ClassName(CategoryName)
// category class and/or instance method declarations
@end
```

Note that there is no provision for declaring additional instance variables. Likewise, implementations use a similar syntax:

```
@implementation ClassName(CategoryName)
// category class and/or instance method implementations
@end
```

This allows the methods of a class to be distributed across different source files for purposes of modularity or incremental compilation. No methods may be duplicated across ordinary or category declarations or implementations, however there are no rules governing where the implementations may appear. A method declared in the interface category X in class Y can be implemented in the ordinary implementation of class Y; the implementation category X may not exist at all (although this probably would not be wise). Categories therefore allow adding methods to an existing class, but do not allow redefinition of an existing method.

There are no conventions in place governing the naming of files in which categories appear, although I would guess that categories would most often be used to create functional packages. Notice that there is a bug in the implementation of categories that prohibits any class name and any category name from being the same in any application.

In the following sections, I will introduce the internal structures that the compiler and run-time system utilize to implement Objective-C.

Message Passing

The chief duty of the compiler is to translate the Objective-C message passing and class definition/implementation syntax into a form that the run-time system expects. So we will first delve into the run-time system:

The run-time object system is actually fairly simple. A message dispatch expression is divided into two categories: those messages sent to “super”, and all others. A message sent to “super” translates into a call to objc_msgSendSuper, while a message sent to any other receiver translates into a call to objc_msgSend. All message sends therefore dispatch through those two functions. The essential difference is that messages sent to “super” begin searching for a method implementation in the superclass of the calling object, whereas all other messages begin their search for a method implementation in the class of the receiver.

Development Process

Here at DTG, we have a development system that we affectionately call *ROMPER* that handles much of the "scut work" involved in the software life-cycle, library maintenance, and class reuse. I generally start with a software requirements specification (SRS).

In brief, the SRS involves:

1. general descriptions,
2. functional and operational interface requirements,
3. a specification of functional requirements that are transformed into a project work suite,
4. a specification of operational criteria that are transformed into a formal or informal test suite, and
5. a user interface prototype expressed as a set of Interface Builder .nib files.

Although a report can be generated from an SRS, it tends to be quite long. For this application, the SRS is quite simple: the project is single-tiered, there are no unexpected functional nor operational interface requirements, the work suite is standard and therefore substantially encapsulated in the design specification, all test suites are informal, and the prototype is enclosed.

After the SRS, the software design specification (SDS) is created. The SDS ranges from an overview of the design issues, to functional decomposition to low-level pseudo-code. Again, this application was very simple, so a very high-level design was immediately coded (I do not believe that tools used to enforce an engineering methodology should be so cumbersome as to interfere with "hacking" an application together; the tools, however, should sufficiently empower the implementer such that resultant applications employ sufficient formality to permit maintenance and reuse).

The actual implementation has an associated software information record (SIR), which is used with repositories and other tools for maintenance and auditing purposes. Sufficiently mature modules are deposited into a globally-accessible hierarchy of software components. Note that all repositories contain active modules rather than passive code. Modules are preserved in and out of context, and have references to clients, servers and humans for code, debugging, testing, maintenance and support. Repositories may be accessed at specification time or run-time, and are generally late-bound to their contents.

I generally customize my environment to divide the specification, design and implementation phases into three components: foundation, interface, and glue. Each has its own section in the SRS and SDS.

Foundation Modules

Since this project is concerned with certain data structures that may change, a primary order of development was to create the foundation classes representing these structures. These classes would serve to insulate clients from the details of the data structures, and would also serve as foundation classes for other applications. This is an example of a functional class design influenced somewhat by implementation details.

The fundamental structures of interest are:

1. a collection of all classes in the current image;
2. class information;
3. instance variable information;
4. class variable information;
5. instance method information; and
6. class method information.

A natural question that arose concerns the manner of ordering the collection of all classes. The run-time system does not care about any global ordering: the only requirement is that each class know its superclass. It would seem useful, however, to present a hierarchical ordering to the collection, since such a collection is fundamental to the programming model. Moreover, a multiple inheritance model is more general. Looking at the library of available classes, I found a class **Graph** that maintains directed graphs, which is ideal since both single and multiple inheritance can be embedded in directed graphs. I therefore create a subclass of this class, **ClassGraph**. When an instance of ClassGraph is created, it immediately initializes itself as a directed graph comprising the entire run-time class hierarchy.

The application will assume that instance and class variables are substantially the same, except for their property of belonging to an instance or class. I therefore create a **IVarRef** class (instance variable reference), a property of which is whether it is an instance variable or a class variable[3]. Likewise I can get away with creating only a single method reference class, **MethodRef**, which has a property that encodes whether the object is an instance or class method. Note that this is probably a poor assumption; if in the future method and class information become substantially different, I will wish that I had kept them in separate classes. With that caveat suitably ignored...

Interface Classes

I steered toward a window that displayed a class hierarchy in a manner much like the NeXT file browser uses in the Open and Save panels. Class names would appear in the browser wells. A class with subclasses would have the small arrow icon denoting a non-leaf node. Clicking on a class would select it; if it were non-leaf, the subclasses would appear. Double-clicking on a class would “open” the class.

Opening the class would cause a browser to appear. The browser would have a toggle button that would switch its view between that of variables and methods. Another toggle button would switch between class and instance inspection. The net effect is similar to that of the Smalltalk-80 class browser.

I expect that one utility for a run-time browser would be to gain information concerning undocumented classes. Thus an feature that produces an Objective-C header file from the run-time information would be useful. A window containing a scrolling text

3 Actually the property is whether the instance variable belongs to a class or a metaclass (a class variable is actually an instance variable of a metaclass). A metaclass is the class of a class.

area is therefore created; the header file is output to the text area from where it may be copied to the pasteboard.

In order to support this interface specification, I needed a browser class. As it happens, NeXT has a class, **Browser**, that is used by the Workspace Manager file browser, as well as the system Open and Save panels. It is undocumented, but I used an early version of this run-time browser project to generate a header file for Browser. I then used the header file to subclass Browser twice: **HashTableBrowser** can browse instances of HashTable; **NodeBrowser** can browse the instances of Node in an instance of Graph (or in this case, ClassGraph) in a hierarchical manner.

All the other interface pieces were available from the standard AppKit “parts bin”; I employed Interface Builder for interface layout.

Glue Classes

In order to tie the interface and foundation classes together, I created a class and a non-class module. First I made the **RTBController** class to control the direct interaction between user-interface pieces and foundation class manipulations.

Then I made the **TypeParser** module. Note that the run-time system stores typing information in ASCII filer strings, and method selectors as single strings with embedded colons. This module contains utility routines that convert the ASCII filer strings and split the selector strings into a string suitable for inclusion in an Objective-C class interface declaration. TypeParser was put in a separate module (instead of being integrated with the Ref classes) because I felt it likely that people would want to change the format of the output at this level, and that as handling of the ASCII filer strings matured, a single point of reference would be easier to maintain.

Putting It Together

Integrating the modules was a “snap”. It’s really a simple application; it took about a day to put it together. Unfortunately, it took four days to remove all the dependencies of the source code upon our special development system (for example, our code maintenance system does not store nor present code as linear files -- the system is also beta and had not been used to export code to the world outside of ROMPER).

You should have no problem using the NeXT-standard compilers and libraries with the enclosed source code. If you do, please submit a bug report to the address given at the end of this article.

Some caveats:

1. the Browser subclasses are based upon a correct but undocumented specification. It is within the full rights of NeXT to change this specification without notice. This may break these interface classes.
2. the TypeParser module has many short-comings. It does not handle all types. It is not re-entrant. It should be rewritten.
3. the internals are neither space nor time efficient.

I sincerely hope that the application is useful both as an example and as an exploratory utility.

Closing Comments

As a development tool, a static class browser/editor (or *inspector*) would be more useful than the run-time browser presented herein. Rather than operating on a running image, however, such a browser would have to either parse through class source files (probably in conjunction with the pre-processor), or read the Mach-O format object (.o) files produced by the compiler. The former has the disadvantage of requiring a syntax-directed parser with its attendant maintenance headaches (i.e. keep up with the evolving Objective-C definition from NeXT). The latter has the disadvantage of only parsing class information after successful compilation, and not being as generically applicable to editing the information.

Note that due to the dynamic nature of Objective-C and the potential distribution of class information among many source files (due to NeXT’s categories), a static browser may not accurately reflect the run-time image. Categories, moreover, may complicate the ordinarily hierarchical view of class information, possibly requiring a separate functional module view.

The author is perfecting a static inspector and other related tools, and would welcome any suggestions from the readership, especially in the area of user interface and feature set.

About the Author

Edward Jung has nearly completed his third (and final) year as Director of Research at the Deep Thought Group, Limited Partnership, a Pacific-Northwest research and development organization working on the application of physiological neural network theory to analog and digital parallel distributed computation (among other things). Previously he “did time” as an academic researcher in biophysics, where he published and did other equally amazing academic things.

Edward is also co-moderator of the NeXT computer conference on the BYTE Information Exchange (BIX). He enjoys stimulating electronic correspondence and may be reached using regular or NeXT-format mail at:

Internet	**ed@dtg.com**
UUCP	**...uunet!dtgcube!ed**
BIX	**ejung**

Finis

AI/X and NextStep

John "Baker" Corey, NeXT Systems Engineer

As the NeXT Program Manager for the NeXT/IBM joint project, I am rather pleased that IBM Corp. has made public their plans for NextStep. I have asked Erica to include the press kit information which NeXT sent out in this issue of Buzzings. I think it will make clear what the announcement was all about. In the March issue, I will provide an inside view of the AI/X NextStep port (which means that all the announcements will be complete so the details will be better).

On a related note, I will be writing a short piece which describes the R&D project undertaken by NeXT and IBM to port the NextStep software to the various IBM platforms.

One last comment given the confusion on the net. IBM will ship NextStep with Objective-C. They did not use C++ as was reported from Uniform.

For other questions, etc., please contact Allison Thomas Associates [NeXT's PR firm]. Ask for Allison Thomas or Emily Brower. Their phone number is 818-981-1520.

IBM Release:

Immediate Release, February 5, 1990

IBM TO OFFER NEXTSTEP ON AIX WORKSTATIONS

NEW YORK, February 5, 1990 . . . IBM and NeXT, Inc. today announced that IBM plans to offer NextStep on AIX. IBM's NextStep offering will provide AIX users with a major new application environment for enhanced business and professional productivity.

NextStep is an application software development and user interface environment, created by NeXT and licensed to IBM in 1988. IBM will support the same applications programming interfaces (APIs) as NextStep, providing compatibility and consistency so that developers can offer applications on both machines, resulting in a larger market for their efforts.

NextStep will join OSF/Motif as graphical user interface offerings planned for the IBM PS/2 and RISC computers running AIX, IBM's open-standard UNIX operating system based on AT&T System V and BSD 4.3. Specific product offerings and availability will be made at a future date.

"The innovative NextStep application environment will offer outstanding ease-of-use and development productivity," said Nick Donofrio, president of IBM's Advanced Workstation Division. "We're especially excited about the benefits of the NextStep Interface Builder and Application Kit, which bring significant value to our customers."

The UNIX operating system offers sophisticated features such as powerful networking and multitasking, but it may been considered, by some users, to be too complicated for those who are not UNIX experts. NextStep, which hides the complexity of the UNIX operating system under an object-oriented environment, will allow users to take advantage of the benefits of UNIX.

"We believe IBM's support of NextStep will have profound implications over time," said Steven P. Jobs. "UNIX is destined to be a crucial operating system this decade. NextStep tames UNIX so business users can tap its power. NextStep offerings from both IBM and NeXT will be a dynamic combination."

Quotes from NeXT developers about developing with NextStep:

Adamation, Stephen Adams, president, (415) 452-5252

"We enthusiastically support IBM's decision to license NextStep® for their AIX platforms. The strategic alliance between IBM and NeXT will be a boon to both software developers and end users alike. Developers will gain the advantage of two markets for the price of one and end users will benefit from the ability to choose software solutions that are not tied to one platform. Products such as Who's Calling?™, a fully automated client management system built for the NeXT Computer, will be marketed on two different hardware platforms that share the same user environment."

Adobe,John Warnock, chairman of the board and chief executive officer, contact Pat Marriot (415) 961-4400

"IBM's endorsement of NextStep for its AIX platforms has an enormous impact on the UNIX market. NextStep, which incorporates the Display PostScript® system as the underlying imaging model, provides a powerful and innovative development environment. Software vendors now have not only state-of-the-art tools, but and expanding base of platforms and customers for their applications. We intend to port our application programs to the AIX NextStep environment to take advantage of this expanding market."

Ashton-Tate,William Lyons, vice president and general manager, applications group (408)927-5300 or (408) 927-5538

"We are very excited about the user capabilities of NextStep and we are confident the putting NextStep on IBM's AIX platforms will broaden the market and increase customer acceptance for NextStep applications."

Conextions, Inc.,Fred Hammond, vice president, sales and marketing,(603) 888-5525

"Support for IBM's version of NextStep is strategically important to Conextions, Inc. Products that have been or will be developed for the NeXT Computer will also be offered on IBM's AIX platforms. Our IBM terminal emulator (to be released later this month) will be Conextions' first NextStep product available for both NeXT and IBM/AIX computers."

Frame Technology,Steve Klann, vice president, sales and marketing,(408) 433-3311

"Frame Technology has been a strong supporter of NeXT and NextStep because early on we realized the benefits to programmers and end users provided by the NextStep environment. The joint IBM/NeXT announcement sends a clear message to companies who have been taking a wait and see attitude: NextStep is a standard for the 90's. We're excited by this announcement because IBM computers have such a broad market and product appeal which opens up tremendous opportunities for developers like Frame."

Informix, Jeffrey Bork, vice president of marketing, (415) 926-6300

"The NextStep development environment on IBM AIX platforms provides an exceptional foundation for Informix's Wingz™ graphic spreadsheet. With its new DataLink capabilities, Wingz becomes an easy-to-use graphical interface to Informix SQL databases. This technology can integrate AIX platforms into a corporate-wide graphical information management system."

Lotus Development Corporation, Ed Belove, vice president of corporate research and development, contact Susan Earabino (617-225-1281

"The decision by IBM to offer NextStep on their AIX systems for business users further enhances the market potential for NeXT technologies."

Media Logic Inc., G. Hank Weghorst, contact Rob Batchelder (213) 453-7744

"We believe NextStep will make desktop UNIX a reality in the personal workstation market. NextSTep makes UNIX a superior environment for general business users as well as technical professionals. For software developers like ourselves, NextStep has enabled us to create an innovative application like TopDraw™ and bring it to market sooner than would be possible in any other environment. We are pleased that IBM has shared NeXT's vision and look forward to supporting their future products. Discussions are currently underway with IBM to begin doing so."

WordPerfect Coporation, Alan Ashton, president, contact Tom Mallory (801) 222-2350

"We are pleased with IBM's decision to adopt NextStep for their AIX platforms. Our plans are to port our NextStep software to IBM's AIX platforms under NextStep."

Legal Stuff:

Background: The Difficulty with Easy-To-Use User Interfaces

NeXT, Inc. created NextStep to solve two major problems. Applications without good user interfaces are difficult to use, and applications with good user interfaces are difficult to develop.

Before NextStep, many users admired the powerful attributes of UNIX® systems--multitasking, high-speed networking, robust operation--but were overwhelmed by the rigors of learning the arcane commands necessary to use the machines. In fact, even many personal computers were daunting for this same reason.

As personal computers with graphical user interfaces appeared, life for users improved. Unfortunately, however, application developers paid the price because these new user interfaces were very difficult to program. Easy-to-use programs required software developers to spend as much as 90 percent of their time programming the user interface, which typically represents only 10 percent of the total program. In practice, this imbalance prevented developers from allocating the time necessary to create graphical interfaces.

The NextStep Solution

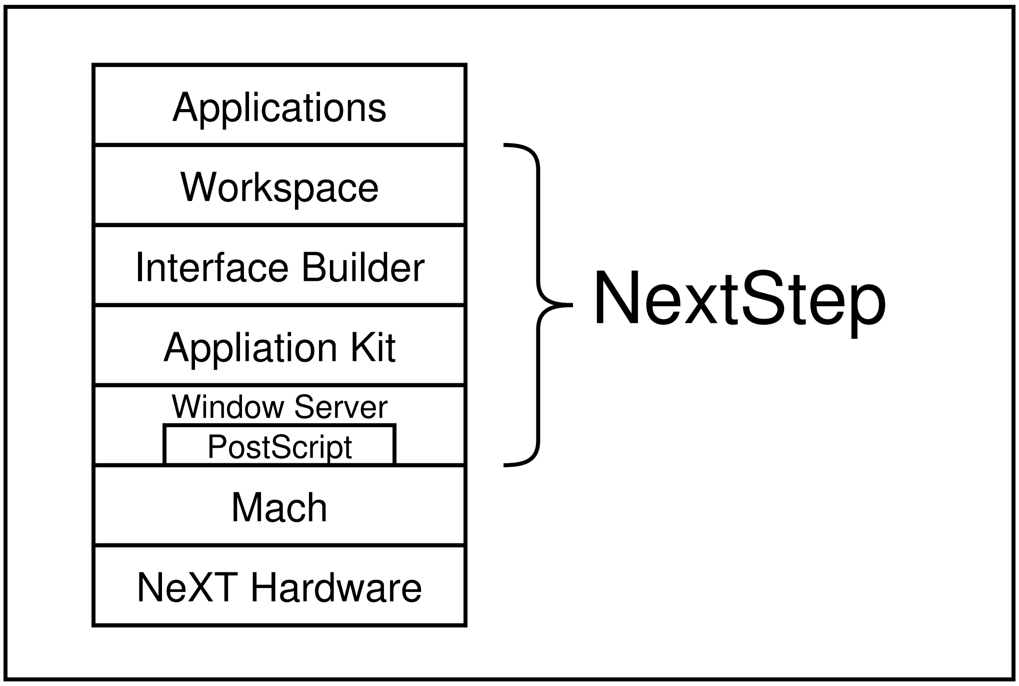
NextStep simplifies the creation of complex, multitasking applications that have graphical user interfaces. It provides software developers with a set of interactive ‘‘objects’’--e.g. buttons, scroll bars, windows--which they can arrange on the screen with the mouse, without any programming, to create their application’s user interface. They can also use the mouse to graphically connect these user-interface objects to the objects in the guts of their application.

This process removes a major obstacle in the development of an application. Freed from the burden of time-consuming and difficult user-interface programming, developers can concentrate on the features and capabilities specific to their applications.

Users benefit, too. In NextStep, once the user has mastered one application, other applications are easy to learn. Because all applications are built from the same basic set of objects, the user is guaranteed a consistent, high-quality user interface. Also, the shortened development time allows developers to produce a broader range of more sophisticated applications, thus expanding users’ options.

The Power of NextStep

In addition to the basic set of objects provided in NextStep, software developers can create their own user-interface objects and use objects created by others. For example, NeXT has already introduced special object ‘‘kits’’ for incorporating sound or music into applications. It is expected that entire object families will be developed to serve specific vertical applications, ranging from finance to construction to computer-aided design. NextStep also includes mechanisms for applications to communicate and cooperate with each other more easily. Combined with the UNIX operating system’s multitasking capability, this inter-application communication heightens the collective power of the user’s application set.



NextStep's Four Components

The Window Server

The Window Server routes signals from the keyboard and mouse to applications, processes drawing commands from applications and manages overlapping windows.

At the heart of the Window Server lies the Display PostScript system. All drawing of text and images in NextStep is done through the PostScript® language, whether that drawing is to appear on the computer screen or on a printer. Because of this unified imaging model, users can see their work on the screen exactly as it will appear when printed. Additionally, application developers do not need to add any special code to their program to support printing.

In NextStep's multitasking environment--in which the user can literally do more than one task on the computer at the same time--the Window Server performs two other vital functions: sending the user's keyboard and mouse actions to the appropriate application and providing the multiple windows on the screen within which applications' PostScript drawing is displayed.

The Workspace Manager

The workspace is what appears on the computer screen when the user starts working. It is controlled by the Workspace Manager, which displays document and application icons, launches applications, allows the user to copy files, and manages the computer's file system.

It includes a ‘‘docking’’ feature, whereby users can place up to 12 application icons in a convenient location, at the right hand side of the computer screen, for easy access.

The Application Kit

The Application Kit contains the set of predefined ‘‘objects”--e.g., a button, a menu, a scrolling bar--that developers combine to create their user interfaces. Developing a user interface is greatly simplified because developers work with a few self-contained, fully functional objects instead of a library containing hundreds of subroutines. This object-oriented approach gives programmers a more powerful set of building blocks than with traditional approaches, allowing them to think at a higher level about what they want their applications to do.

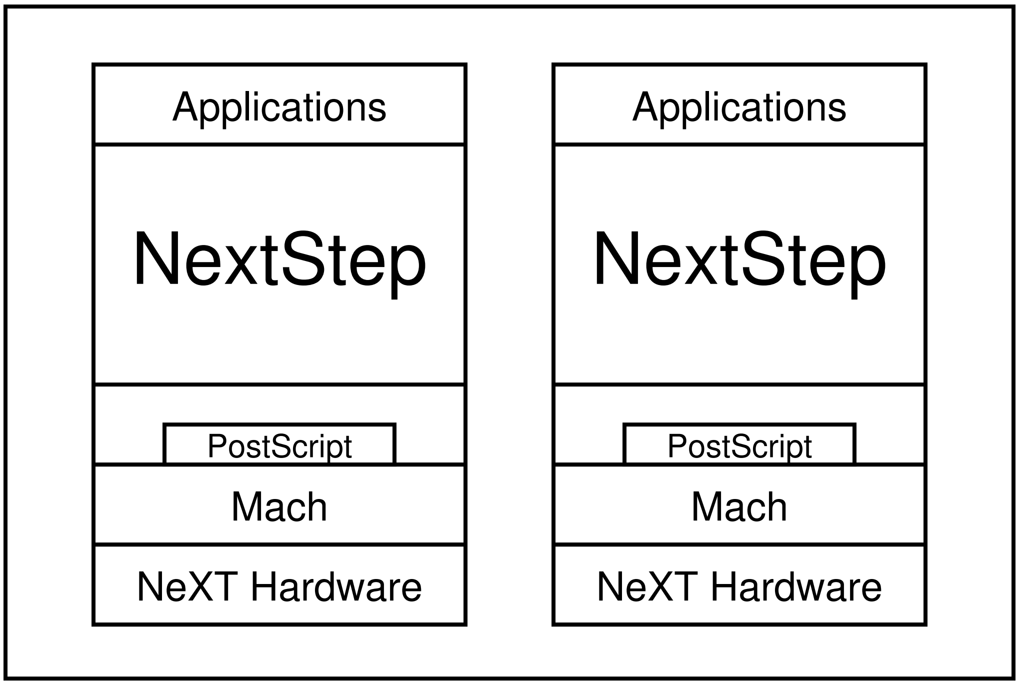
In addition to user-interface objects, the Application Kit contains objects that support such vital functions as copying and pasting between applications, making requests of other applications and communicating with the Window Server. These objects help make NextStep independent of the hardware on which it runs, while isolating the software developer from some of the more arcane aspects of the UNIX operating system.

Interface Builder

Interface Builder is a tool for the rapid construction of a graphical user interface for any type of application. It takes application programming to a new level by drastically cutting the time needed to create user interfaces. In addition, because its graphical mode of operation requires minimal programming expertise, Interface Builder makes application development accessible to people who may be experienced at using computers, but not at programming them. For example, people can use Interface Builder to create customized applications for their own offices, departments or companies.

Interface Builder provides a palette of software objects, such as those supplied in the Application Kit. Using the mouse, users can simply drag objects from these palettes and graphically create user interfaces for applications. Users can also edit the objects’ attributes, using the mouse to determine how objects will look and function.

Users can also graphically connect these objects to each other and to objects they creates themselves, thus specifying how objects will communicate and function together. These connections form the basic structure of the application and determine what happens within the program as the user manipulates the user interface.



NextStep and IBM

When NeXT introduced its NeXT™ Computer in October 1988, the company also announced that IBM had licensed NextStep. IBM has now announced that it is offering NextStep on AIX™ to provide its customers with a major new application environment for business and professional productivity.

AIX is IBM's UNIX operating system. NextStep will join OSF/Motif™ as IBM's graphical user interface offerings for its PS/2® and RISC computers running AIX.

IBM supports the same application programming interfaces (APIs) as NextStep, which provides compatibility and consistency to help users learn new NextStep applications quickly. Also, NextStep application software written for either the NeXT or the IBM PS/2 platform can be ported to the other platform with simple recompilation, which should create a much larger installed base of NextStep users. As a result, developers can justify devoting greater energies to creating a wider range of software programs under NextStep.

The partnership between NeXT and IBM over NextStep is expected to benefit both parties. They hope that the combination of IBM's broad market strength and NeXT's innovation will help NextStep become a world standard.

[*It may have been vendor biased propaganda -- but, hey, at least it was COOL vendor biased propaganda! Thanks Baker.--EJL*]

Finis

Notes on using Terminal windows

James E. Burns (burns@gatech.edu)

If you use a lot of different Terminal windows, you may find the following somewhat useful (although there is little here that you can't find in pages 18-6 and 18-7 of the System Reference Manual).

Terminal has a lot of attributes which can be set to your preferences. The default values are saved using dwrite: `dwrite Terminal <attribute> <value>` (You can check your current defaults with 'dread -o Terminal'.) The most useful defaults to set are given below (see the on-line documentation for the others):

```
dwrite Terminal NXFixedPitchFont <font family>
dwrite Terminal NXFixedPitchFontSize <point size>
dwrite Terminal Lines <number of lines>
dwrite Terminal Columns <number of columns>
```

The only values currently available for <font family> are Ohlfs (default) and Courier.

You can create a new terminal window from a shell or terminal window by typing `Terminal &` (the '&' lets Terminal run asynchronously so you can continue using your current shell). Any of the default Terminal attributes can be reset for an individual call. For example:

```
Terminal -Lines 24 -Column 60 -NXFixedPitchFontSize 24 &
```

The standard placement for a new Terminal window is 24 points down and 24 points to the right of the 'last moved' Terminal window. A Terminal window is 'moved' when it is created or moved about the screen (but not resized!). If you want to put a Terminal window in a specific location, you can do so with the WinLocX and WinLocY attributes. For example, the following Terminal will appear in the leftmost upper corner of the screen:

```
Terminal -WinLocX 0 -WinLocY 0 &
```

What I am most frequently using multiple window for is to rlog to other machines. The convenient way to do this is use the 'Shell' attribute, e.g.,

```
Terminal -Lines 24 -NXFixedPitchFontSize 18 -Shell "rlogin foobar" &
```

will set up a 24 line window with 18 point font which logs onto machine 'foobar'. If you have set up .rhosts properly on foobar, you won't even have to give a password to start using it. When you exit from rlog, the window quits.

OK, this is lot's of fun, but it also requires a lot of typing. What I have done is created a set of executable scripts (I keep this in a directory on my path called scripts.local) so that I can easily start up whatever remote sessions I need. By appropriate use of WinLocX & WinLocY, I can make sure my remotes to each machine always appear where I expect them. An example file, lfoobar, looks like this:

```
#!/bin/csh
#
#Shell script to open an rlog window to foobar
#
Terminal -Columns 80 -Lines 33 -WinLocX 1 -WinLocY 80 -Shell "rlogin foobar" &
```

Now when I type 'lfoobar' a window is opened & a session started on foobar. I also have a set of similar scripts to create terminals of different size fonts. For example,

```
#!/bin/csh
#
# Create a full size 10 point terminal window
#
Terminal -NXFixedPitchFontSize 10 -Lines 71 -Columns 80 &
```

I chose 71 lines since this is essentially fills the screen at 10 points. Corresponding choices for other font sizes:

FontSize	**Lines**
12	58
14	54
16	48
18	42
20	39

Finis

Market View

I've received literature from NeXT or Third Party Developers on the following products. No warranty, express or implied, is given. The quotes are *mostly* the developers' and may not reflect reality.

Communicae - *Active Systems* 1-617-576-2000 "high performance communications package. VT240 emulation"

Wingz *Informix Software, Inc.* 1-913-599-7100 "graphic spreadsheet featuring advanced charting, desktop presentation capabilities, and HyperScript"

Scan 300/GS *Abaton* 1-415-683-2226 "300 dpi flatbed scanner with TIFF compatility"

DM-N Digital Microphone *Ariel* Corporation 1-201-249-2900 "software-selectable sample rates from 88.2 kHz to 5.5 kHz per channel"

DaynaFILE *Dayna Communications, Inc.* 1-801-531-0600 "external, SCSI floppy disk drive to write to standard UNIX-formatted diskettes, as well as MS-DOS formats"

Smart Art *Emerald City Software, Inc.* 1-800-223-0417 "50 text and graphics effects and easily customized in any NeXT word processor, desktop presentation, or page layout program"

FrameMaker 2.0 *Frame Technology Corporation* 1-408-433-3311 "powerful, cost-effective workstation publishing software"

Artisan *Media Logic Incorporated* 1-213-453-7744 "high-resolution paint and image processing system"

TopDraw *Media Logic Incorporated* 1-213-453-7744 "complete and advanced page-based graphics software"

TextArt *Stone Design Corporation* 1-505-345-4800 "array of tools that allow immediate creation of outstanding PostScript images"

Encapsulated PostScript ClickArt *T/Maker Company* 1-415-962-0195 "combines ClickArt EPS portfolios into a collection of high-quality Encapsulated PostScript (EPS) artwork"

Public Domain Disk #1 - *Lighthouse Design* 1-800-FOOBAR9 "Public Domain Software & More" (I'm in on this one. Buy it. Please!)

Scematic Entry - *Lighthouse Design* 1-800-FOOBAR9 "CAD Tool for designing electrical circuit schematics"

Media Station - Imagine Inc 1-313-434-1970 "archival, retrieval and processing of multi-media information"

Fortran 77 -- Absoft 1-313-853-0050 "Objective Fortran-77"

DisplayTalk - Emerald City Software - 1-800-223-0417 "Complete development environment for Display PostScript programming" (I gave them a call in January. It looks like we may have a review of this for the April Issue.)

Video Monitor and Projector Interfaces *Extron Electronics* 1-800-633-9876 "offers three video monitor and projector interfaces"

Digital Ears *Metaresearch, Inc.* 1-503-238-5728 ""allows entering and recording compact disc-quality sounds"

Digtal Eye *Metaresearch, Incorporated* 1-503-238-5728 "allows entering and recording NTSC video images"

NVT High Density Video Drive *New Vision Technologies, Inc.* 1-415-285-8744 ""video playback device for interactive multi-media applications"

JETSTREAM Tape Backup System *Personal Computer Peripherals Corporation* 1-813-884-3092 "high performance tape backup system"

A/D64x Analog/Digital Interface *Singular Solutions* 1-818-792-9567 "a low-cost platform for sound recording, experimentation, and analysis"

Who's Calling *Adamation, Inc.* 1-415-452-5252 "lets sales & business professionals keep track of phone calls and other client information"

GEMS (Generalized Equilibrium Modeling System) *Data Transforms, Inc.* 1-303-832-1501 "a flexible way to model economic systems"

InDia (Influence Diagram Processor) *Data Transforms, Inc.* 1-303-832-1501 "graphical application for representing complex decision-making"

Knowledge Retrieval System (KRS) *KnowledgeSet, Corporation* 1-415-968-9888 "rapidly searches and retrieves information from large databases of text and graphics"

OMEN III *Microstat Development Corporation* 1-604-228-1612 "stock quotation and financial system"

TACTICIAN Plus *SouthWind Software, Inc.* 1-316-636-5100 "multi-user spreadsheet that supports high-level functions and adds built-in presentation graphics"

Adobe Illustrator *Adobe Systems Incorporated* 1-415-961-4400 "graphic design and illustration program for generating high-quality artwork"

Adobe Type Library *Adobe Systems Incorporated* 1-415-961-4400 "offers more than 500 different typefaces"

Flash Graphics *Flash Graphics* 1-415-331-7700 "extensive charting, illustration, and text functions in a graphics package for screen, slide and paper presentations"

InterFax 24/96N *Abaton* 1-415-683-2226 "combines a 9600 bps Group 3 fax modem with a 2400 bps MNP 5, Hayes-compatible data modem"

GatorBox *Cayman Systems, Inc.* 1-617-494-1999 "LocalTalk to Ethernet gateway that translates the Network File System (NFS) protocol into Apple Filing Protocol (AFP)"

MacLinkPlus/PC *DataViz Inc.* 1-203-268-0030 "kit for transferring and translating files between NeXT and Macintosh environments"

Ethernet PhoneNET, Sound and Interpersonal Communications *Farallon Computing, Inc.* 1-415-849-2331 "used to build LANs over standard telephone cables"

Etherport NL Kinetics 1-415-947-0998 "allows the NeXT computer to connect directly to standard twisted-pair Ethernet networks"

INFORMIX-TURBO *Informix Software, Inc.* 1-415-926-6300 "database engine for on-line transaction processing (OLTP)"

INGRES Relational Database Management System *Relational Technology, Inc.* 1-800-4-INGRES "SQL database engine provides on-line transaction processing (OLTP) in single- or multi- CPU and distributed environments"

DAN - The Data Analyzer *Triakis Inc* 1-505-672-3180 "data analysis package for reducing data and generating presentation-quality plots"

Math++ - *Triakis Inc* 1-505-672-3180 "C-language math library. Approx 100 math functions"

Dreams - *Innovated Data Design* 1-415-680-6818 "Frm the makers of MacDraft, drawing and drafting tools"

Cross Assember/Simulator Programs - *Motorola* 1-512-891-2030 "for the 56000 and 96000"

Fortran, C and Pascal Compilers - *OASYS* 1-617-890-7889

User Groups

Here are some pointers to Users Groups that may be in your area.

Maryland/Northern Virginia/DC
•Washington Apple Pi, Hugh O'Neill (301)-328-9510
•Ed Klein, the NeXT Consultant of UMD says he's started a user group : NUTS. Contact him at eklein@umd5.umd.edu.

Georgia
•BuzzNUG is sponsoring local demos and talks. Contact Erica Liebman at erica@kong.gatech.edu.

Massachusetts
•The Boston Computer Society (BCS) has a NeXT special interest group. Contact Dan Lavin at 1-617-969-6555, or Jan McPeek at 1-617-926-4027

California
• Robert D. Nielson 1-408-995-5775 and Jeff Wishnie 1-415-324-9567/1-415-780-2753 (voice mail) have started **BANG!** out of San Jose, loosely associated with Stanford. Jeff is apparently the Stanford NeXT Consultant. These guys are clearly in the best location possible for a NeXT user group and I was drooling at their speaker list. Definitely contact them if you live within fifty miles of San Jose. Or are willing to drive further. Or have your own 'copter. Robert has promised me Vietnamese Food if I make it to the Valley some day. No similar promises of food are made to any potential members. Sorry.
• Paul Lowe (714)787-3883 at the University of California at Riverside is interested in seeing what others are doing with their NeXT Cubes to distribute to the other NeXT Users (six so far) on campus. write to : plowe@ucrac1.ucr.edu

Texas *(yee-hah!)*
• The report goes that there "is some massive new NeXT user group down there". Says Jerry Goode : "Hi there! Just wanted to get back to you with some info about the NeXT user's group down here in Dallas, Texas...

First, we've had a grand total of 1 meeting with about 35 people in attendance. Metaresearch came down to speak and demo Digital Eye & Digital Ears. Pretty excellent stuff if you haven't seen it lately! Second, here is the name of the fellow who is heading up the group down here: Dirk Hardy/Hofbauer Information Systems/5080 Spectrum Drive/Suite 912W (Lock Box 21)/Dallas, Texas 75248/Phone: 214-385-2991

Hofbauer is a NeXT registered developer doing courseware authoring tools, and Dirk is working in conjunction with Dr. Ali from North Texas State U. to get the group going. I encourage you to get in touch with him - he's got a lot of creative ideas! Hope to have an email address for him soon.....

Finally, we plan to meet the third Thursday of every month at 7 pm, somewhere! The logistics are still a bit up in the air as we try to figure out how much this thing will grow. If the response after our first meeting is any indication, we could settle in at around 50 people.

By the way, I have had Buzzings forwarded to me and it's GREAT. Dirk has copies of both issues so far and was really impressed. You may want to talk with him about organizing a Texas contingent contribution as we try to get things rolling here. "

Scenes from the NeXT Issue

First off, we've got few articles that were missed this time. Midterm time prevented my finishing my Array-Processing with the DSP article. Morris Meyer is still signing off on his Godzilla Distributed Processing article and we had to get a designated hitter for the best of Public Domain article. Lighthouse, has been working hard, but still promises their Object Persistence article -- real soon now.

We've got an unspecified article from Dave Stutz out of NeXT, Bryce Jasmer of Oregon State will have his turn at describing his archive site at cs.orst.edu. Craig Shock will submit some article and William Shipley is looking at either an X.11 or Listener/Speaker writeup.

We've got another article promised on extending Interface Builder by adding an enhanced slider object from the authors (Judy Halchin & Chuck Fleming) of a new package called "Spring", which we'll be reviewing for March. We'll also be reviewing the Communae Demo from Active Ingredients.

Doug Brenner has offered to write about Custom Icon Association with Digital Librarian.

For April, we've got an interview lined up with Bruce Webster, the author of the NeXT Book. Please start sending questions to me at erica@kong.gatech.edu that you'd like included in the interview. We'll be reviewing the 1.0 version of the book. Emerald City has tentatively agreed to send a review copy (if I can get a disk out to them) of DisplayTalk for the April Issue. I talked to them in late January and probably should drop a pointed hint again soon. They sounded like nice people, so things should go well.

Finally, I'll try to summarize the results of an ongoing survey : "Which little things annoy you most about your NeXT?" and "What application software is missing for the NeXT?"

Buzz’s Hint Corner

• Bored with your login window? Change it! Copy the default NeXT login window, /usr/lib/NextStep/nextlogin.tiff to your directory, edit it it with the Icon program and reset the default by

```
dwrite loginwindow ImageFile your-file-path
```

Try this out by logging out, typing "exit" instead of your login and press return twice.

• Want to read your Next-Specific mail without running "Mail"? David Kay and Jacob Gore came to my rescue with this well-timed advice & code.

1. Save the mail to a local file, say foo (leave, mail using "x" to preserve all your mail) & delete up to (but not including) the word begin in ed or emacs or vi. Yes, NeXT mail does uuencode. Decode the file. **uudecode foo** A file named .tar.###.something will be created. (### refers to some arbitrary number)
2. Rename the .tar file to, perhaps, foo.tar.Z. **mv .tar* foo.tar.Z** Change the protections to readable/writable by doing a **chmod 644 foo.tar.Z.** Uncompress the file. **uncompress foo.tar.Z**.
3. The tar file will probably contain a file called index.rtf, containing the heart of the mail, not including the NeXT Enclosures. Use David’s spiffy rich-text-file program, that follows to "clean" it and then read it. **stripText index.rtf** then **more index.rtf.clean**.

Caveat : You can’t run the rich-text-reader from a remote login. You may get the "gist" of the message by doing **more index.rtf**, but it’s messy.

```
/*
cc -ObjC rich.c -o rich -lNeXT_s -lsys_s
       dbk 15/18 december 1989

        This program reads in rich text and writes out
        plan text.  It uses a Text object to do the work.
        BUG:  It doesn’t handle non-RTF input well.  Plain
        text is deformed; badly-formed RTF can hang it.

 */
```

```
#import <stdlib.h>
#import <sys/file.h>
#import <appkit/appkit.h>

void main(int argc, char *argv[])
{
        int             outFd;
        id              someText;
        char            outFile[1024];
        short           ix;
        NXStream        *inStream;
        NXStream        *outStream;
        if (argc < 2) {
                printf("Usage: stripText f1 [f2,...fn]\n");
                exit(-1);
                }

        NXApp = [Application new];      // Application new does some
                                        // initialization of the Text
                                        // class for us.
        someText = [Text new];

        for (ix = 1; ix < argc; ix++) {
                sprintf(outFile,"%s.clean",argv[ix]);                   //
      output = input.clean
                inStream = NXMapFile(argv[ix],NX_READONLY);             //
      open input NeXT stream
                outFd = open(outFile,O_WRONLY|O_CREAT|O_TRUNC,0666);    //
      open output file...
                outStream = NXOpenFile(outFd,NX_WRITEONLY);             //
      ...and associated stream
                [someText readRichText: inStream];                      //
      read text in as rich...
                [someText writeText: outStream];                        //
      ...and out as plain
                NXCloseMemory(inStream,NX_FREEBUFFER);                  //
      close input...
                NXClose(outStream);                                     //
      ...and output
                close(outFd);
                }

        exit(0);
}
```