ENCLOSED:

The Second Issue of *Object-Based Computing* -- A resource for the NeXT Programmer

ARTICLES:

Object-Based Computing (Part 2) *by Dan McCreary*

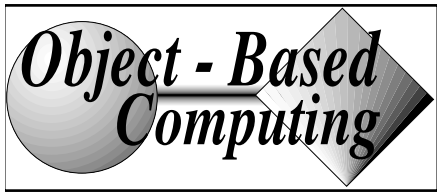
Postscript Window Manipulation *byTed Shelton*

Object Oriented Programming: The Probable Person Period *by Dan McCreary*

nextIssue
Object-Based Computing (pt. 2)
NeXT/OOP: A Case Study
usernamed: A Network Daemon
Flexible Object Design

Object-Based Computing is published monthly by Information Technology Solutions, Inc.

Information Technology Solutions, Inc. - -- 400 West Erie, Suite 402 - -- Chicago, IL 60610 - -- (800) 394-4487 - -- gwkiv@its.com



400 WEST ERIE, SUITE 402
CHICAGO, ILLINOIS 60610

In future issues an important part of this publication will be a descriptive list of commercial and "shareware" objects available to the NeXT Developer community.

Your participation in creating this marketplace of ideas is essential: we'd like to make it easier ---

TEMPLATE FOR CATALOG SUBMISSIONS

In order to make your lives a little easier we will be using the same template as NeXT for submissions to our objectCatalog. The information can be mailed, faxed or emailed to gwkiv@its.com

Object Category

Product Name

Product description here (up to 100 words)

Company Name or University

Street Address

City, State, Zip code, Country

Phone:

E-Mail address:

Pricing Information (including educational discount, if applicable):

Source code available:

Support available:

On-line Help (if a UI object):

Documentation Availability:

Localization Support:

Availability:

MAIL ADDRESS:

Geordie Korper
Object-Based Computing
400 West Erie, Suite 402
Chicago, IL 60610

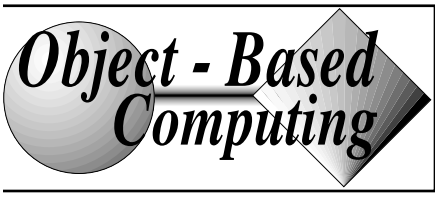
FAX: 312.664.8409

EMAIL: gwkiv@its.com

Please use one of these category names for consistency and clarity:
UI Object
Information Display Object
Analytical Tool (includes objects to be used with Mathematica)
Networking & Telecommunications Object
Device Driver & Communications Object
Multimedia Object
Document & Publishing Object
Database Object
Financial Object
Tool & Utilities -- General
Business Graphic
Digitized Sound
Graphic
3D & Rendering Tool
Voice & Speech Object
Telephony & ISDN Object
Text & Language Tool
Scripting & Macro Object
Indexing & Retrieval Object
Workflow Object

Deadline for object submissions:

July 24, 1992



mands that could use a better user interface?

3.) Now that you have finished the program, try it again. This time yourself. How long do you think an experienced object based programmer would need to create a new object? How many lines of code did you have to enter to make this program work? How many lines of code would you need to do this same program in other graphics or iconic programming systems?

4.) Instead of just the same action taking place, what if you wanted to have two buttons connected up to the same object. How would you change the above procedure?

NEXT MONTH:

Encapsulation and Inheritance

objectCatalog

Companies currently marketing programming tools and objects for the NeXT Developer:

Impact Software
210 Lake Street
Ithaca, NY 14850

(607) 277-8623

email: impact@impact.shamen.com

Objective Technologies
Suite 1502
7 Dey Street
New York, NY10007

(212) 227-6767

email: lg@object.com

RDR, Inc.
Suite 350
10600 Arrowhead Dr.
Fairfax VA 22030

(703) 591-8713

email: info@rdr.com

Threaded Technologies
339 Wiltsee Avenue
Loveland OH 45140

(513) 677-2106

email: !uunet!tti!dennisg

We would be happy to list your company -- please let us know how to list your company and what objects are available from your company. Filling out the object submission form on the next page will also help us to provide a complete list of resources for the object based NeXT developer.

```
debug_obj/hello_main.o -
lNeXT_s -lsys_s
```

We can then enter the following in the shell:

```
cd Programming/Hello
hello.debug
```

The “hello.debug” command causes our program to be executed. Each time you press the hello button you should see the “hello, world” message printed on your shell output. When you are done select the “quit” from the hello main menu.

Congratulations! If you made it this far you have completed the most difficult section of this entire book: creating your first object. You have created a structure which is encapsulated, you have used Inheritance, your have sent a message from a user interface object (in this case a button object), and you have, perhaps for the first time, used event based programming and connection based programming tools. It is time to pat yourself on the back and tell everyone around you that you have entered the world of Object Based Computing!

But what was it that we really did? What did that class browser have to do with re-using objects? How do I create new objects that are sub-classes of existing objects and connect them all together? How do I get this new object to send messages to other objects? What if you wanted to have the output of “printf” go to a screen text object rather then the standard output of the shell? These questions and many more will be explained in the next chapter. Before we answer them we need to know a bit more about events and the way they are handled in the Application Kit.

Exercises

1.) Create another button and another instance of the MyHelloClass. Connect them together and then save. After you do another make and rerun the program, what happens to the output?

2.) Change the printf statement to the line

```
system(“date”)
```

You will need to another import file called <stdlib.h> instead of <stdio.h>. This will cause the UNIX date command to be run whenever the button is pressed. Can you think of any useful UNIX com-

```
/* Generated by Interface Builder */

#import "MyHelloClass.h"

#import <stdio.h>          // add this for type checking

@implementation MyHelloClass

- helloAction:sender

{

printf("hello, world \n");     // add this line

return self;

}

@end
```

Note that the one additional line which includes the file <stdio.h> should be included to get rid of compiler warnings but is not necessary for the program to work. To compile we go to the main Interface Builder menu, select "File" and "Make" and we will see the following messages being sent to a UNIX shell:

```
pushd /dan/Programming/Hello; make debug; popd

mspdemo> pushd /dan/Programming/Hello; make debug; popd

~/Programming/Hello ~

make hello.debug "OFILE_DIR = debug_obj" "CFLAGS = -g -DDEBUG -Wimplicit"

mkdirs debug_obj

cc -g -DDEBUG -Wimplicit -c MyHelloClass.m -o debug_obj/MyHelloClass.o

MyHelloClass.m: In method `helloAction:'

MyHelloClass.m:10: warning: implicit declaration of function `printf'

cc -g -DDEBUG -Wimplicit -c hello_main.m -o debug_obj/hello_main.o

cc -g -DDEBUG -Wimplicit -segcreate __ICON __header hello.iconheader -segcreate __ICON app /usr/lib/nib/default_app_icon.tiff -segcreate __NIB hello.nib hello.nib -o hello.debug debug_obj/MyHelloClass.o
```

This actually requires two smaller steps. One involves "Saving" your project in a new directory and the other involves "unparsing" the files related to our new custom object. The first step can be done by going to the main menu and selecting "Files" followed by "Save". This will bring up a save panel from which you can create the directory and the name of the file you want to save your work in. One suggestion would be to enter the path:

```
Programming/Hello/
    hello.nib
```

This will create the directories (folders) for Programming and Hello if they do not already exist and save your work in the file "hello.nib". After you have created a directory folder for your work you will also need a file to keep track of all your objects and the steps necessary to compile and link them. This is called a "project" file. It contains information similar to a UNIX Makefile. The user does not need to know what is in the files other that it contains the "recipe" for building the program. This information includes things like instructions for compiling, linking and installing the program. To create the project file from the main menu select File and then "Project...". It will bring up an inspector for the project file and tell you there is currently no project file. To create one just enter Return or select the OK button.

Our last step is to create the template file into which we will enter our line of program source code. To do this we need to return to our class editor. Make sure that "MyHelloClass" is the chosen class (MyHelloClass must be below the ".h" icon) and select the Operations pop-up list and then use the "Unparse" selection. It will then ask you if you want to add these files to the project manager. You should select the default "Yes". You have now created all the files you need for the last step. You should be able to view all of the files you created from the browser by selecting the icon view after you are in the Programming/Hello folder. From the Workspace Browser, the icon view of these files should be similar to the Workspace Browser on top of this page.

Note that the files that have the "~" (tilde) characters after them are the backup files created by Interface Builder. You can revert back to these if you make a mistake on the current version.

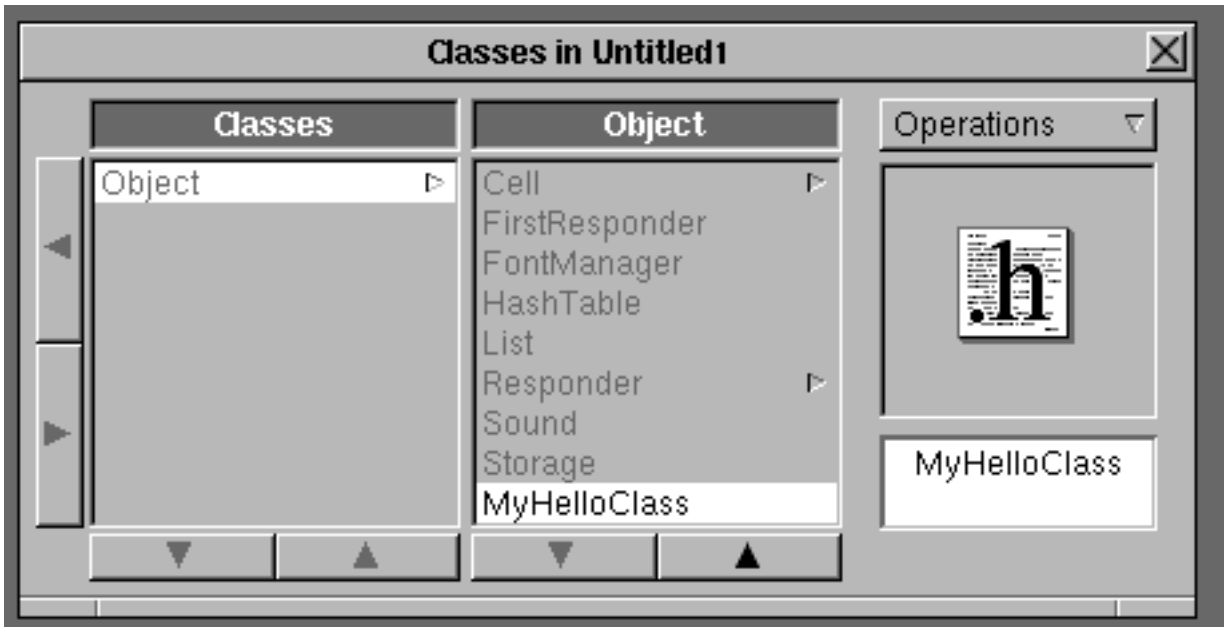
7) Add print command statement and compile.

Our last step will be to add a two lines of code to the file MyHelloClass.m and compile it. This will actually display the message in the shell window. The line that will do the work is the following:

```
printf("hello, world
    \n");
```

To edit the file you can double click on either the line in the project inspector or the icon in the Browser.

The entire file should look like this after you have added this line. The two lines you add are in **bold**. Comments (which you don't have to add) are all the text after the double slashes (//).



object which has a sphere as its icon (the generic object icon) and below is the text "MyHelloClass…".

4) Create an action message for MyHelloClass.

We will now create a link between our button and our object. This link is a "message" that we will add to our class. Before we do this we need the Inspector window. To get this to appear on the screen we go to the main menu and select Tools and Inspector. Select the MyHelloClassInstance object by clicking it. The inspector should have a pop-up menu at the top of it that is, by default, set to Attributes. Select this pop-up menu and make the Class selection. In that window you will see two columns: one for Outlets, and one for Actions. We want the actions selection since pressing our hello button is technically an "action" caused by the user clicking the mouse over the button. To do this, click on the button below the two columns so that the word "actions" is in bold. Now type in the word "helloAction" followed by a <Return>. When you are done the Inspector panel should have a single entry in the action column and the text "helloAction:" (Note: the colon is automatically added for you at the end).

5) Make the connection from the button to the instance of the object.

This is done by pointing to the button and Control-Dragging a line from the button to the icon in the lower left corner of the screen labeled "MyHelloClass…".

As you release the mouse over the object the inspector will again appear. But this time the pop-up menu will be changed to be the connection panel. As each instantiation of an object can have several actions, we must let IB know which action we intend to occur when the button is pressed. After the inspector panel comes up you must click on the word "helloAction" in the column on the right labeled "Actions of the Destination". After you have selected it you must then click on the "Connect" button or enter a carriage return. A small round knob will appear next to the message name indicating that it has been connected.

6) Create Objective C files.

We must write code to display the message "hello, world" in a shell window.

objects. It will be different in several ways from the simple "C" program that is created to do similar things. This program will contain an "Object" that prints "hello, world" whenever a button is pressed. After this object is created, we will be able to re-use this object over and over again without recompiling the object. This object will respond to mouse clicks and will aid our understanding of event based programming in a later chapter. This object can also be part of an application object that can later be used to respond to messages from other programs.

Our first task is to start a new project within Interface Builder. If you are already running Interface Builder with another project you should go to the main menu and chose File… and then Close File. If you are just starting, select File and New Application as in chapter 2. Our interface will be very simple. It will only have one button.

This will be connected to one "custom object" we will call "MyObject". When this button is pressed it will run the line of program that will print a message to the shell.

We will create the object using the following steps. You are not expected to understand what is going on in each of these steps. We will take a detailed look at each of them in the next chapter.

1) Add a button to the main menu.

To do this, drag the object marked "button" from the palette menu and drag it into the main window. You can then double click on the text of the button and type in the word "hello<CR>". You can then resize the main window to fit around the button so your man window will look like the following:

2) Create a sub-class of the "Object" class called MyHelloClass.

This can be done by double clicking on the brief-case icon in the lower left window labeled "Classes". This looks like the icon below:

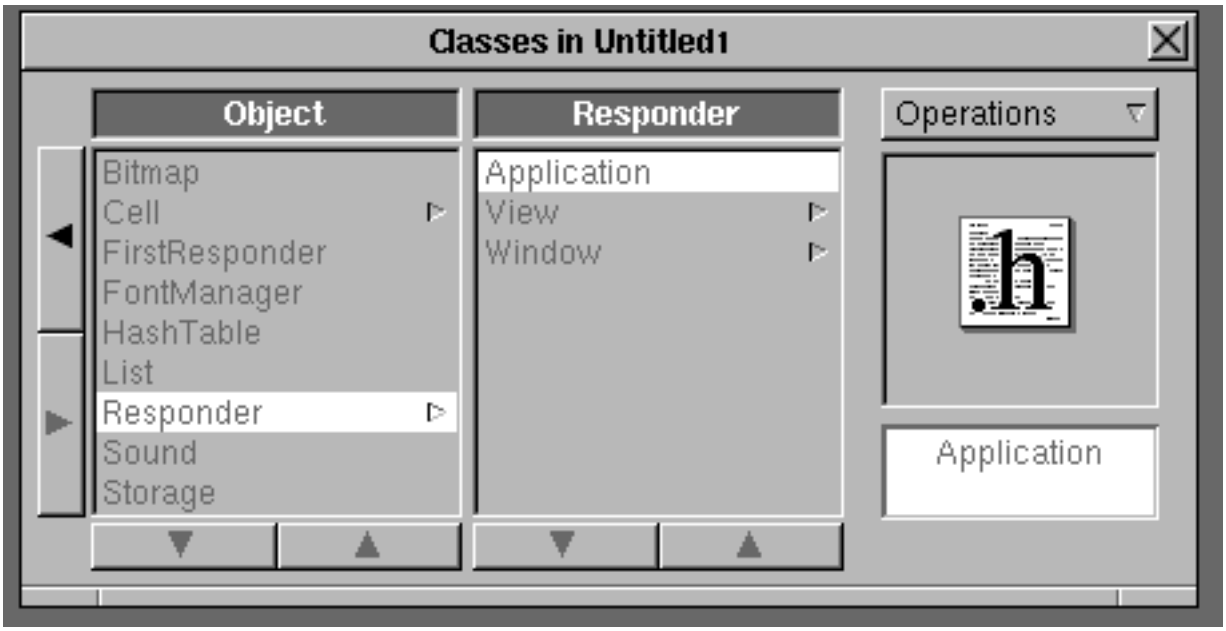
After you double click this icon a "Class Browser" will appear that looks like the figure below.

We will first click the black arrow on the upper left edge of this browser to go to the left-most column so that the word "Object" is the only word showing in the left column. We then want to select "Object" and verify that the object was selected by noting that the word "object" also appears below the ".h" icon in the right section of the window. We want to use the pop-up list above the ".h" icon labeled "Operations" and select the "Subclass" button. The text below the icon should now be "Subclass1". You should double click over that text and change it to be "MyHelloClass". The class editor should now show our new object (see figure on the next page).

Note that we used an uppercase letter to begin the class name. This is an important convention which we will explain in the next chapter.

3) Create an instance of the MyHelloClass.

This is done by clicking on the Operations pop-up menu of the class browser and clicking the "Instantiate" selection. You will now notice that the window in the lower left corner of the screen has an



(the slider) and CONTROL-DRAGGING a connection to the "target object" (the text).

After you release the mouse over the target object an Inspector panel will appear in the lower right corner of the display. This tells you what types of messages the destination object can receive. In this case, select the "takeFloatValue:" message. After you have selected the appropriate item under the label "actions of the destination" you must then press the "Connect" button at the lower right hand corner of the Inspector panel. A small round knob will appear next to the message name to indicate that the connection has been made. Now try testing the interface again using the Test Interface selection of the File menu. When you move the slider the number in the text object should change. Note that the default limits of a slider are zero and one. Select quit when done.

You can change the default values of objects using the inspector panel. By selecting the object that you want to modify and then using the pop-up menu on the inspector and going to the attributes selection you can see and modify the minimum, default and maximum settings on a slider or other object.

Exercises

1.) Try selecting other palettes. Add a menu selection and a panel. Have the menu selection send a "orderFront": message to the window.

2.) Try changing the text on the button object by double-clicking over the text.

3.) Try changing the font of the button object using the "Font" menu. Can you get Greek characters to appear using the symbol font? See the NeXT User Manual for alternate keyboard layouts.

4.) Try using all the different layout tools. Note that the "Same Size" will make the second object selected with the SHIFT-click the size of the first object selected.

5.) What other palettes would you like to use? What types of inspectors would they have? What might an object which is a front end to a modem look like? How would you integrate a front end to a SQL database server into IB?

6.) Many people have commented that Interface Builder is as easy to use as a database program with a graphics user interface such as Hypercard(TM) or Supercard(TM). Can you create any arbitrary sets of panels with Interface Builder and have them send messages to any other panels? What are the restrictions? What can you do with databases that you can not do with Interface Builder? What can you do with Interface Builder that you can not do with database programs? What are the basic differences between the output of database programs and software development tools like Interface Builder?

Hello World: Creating a New Class of Objects

Some day, far in the future, we will create the first being that can speak intelligently. Even odds the first words we hear will be: "hello, world".

- Anon

Now that you are familiar with using tool-kits to create user interfaces, lets take a look at what we must do to create one of our own objects. Our first program will be very simple and yet very important. It will have one button that when pressed will print out the words "hello, world" in a "UNIX shell" window. It will demonstrate the foundations of the actions required to build our own

program and for other Interface Builder tools.

Ignore the menu and controller panel for a moment and go back to our palettes and main window. Add a button to our main window by first pointing to the palettes panel and dragging the object labeled "Button" to the upper left corner of the main application window. You will notice that immediately after you let go of the mouse, six "knobs" appear around the object. These knobs are very similar to the knobs used in programs like MacDraw. IB allows you to grab the knobs with your mouse and drag them. This will allow you to re-size the button in any dimension to meet your needs. Try dragging the lower right had corner down and to the right till the button is about two inches wide and an inch high.

Suppose we had sound in our program and we want to create a graphic equalizer to control the bass, mid-range and treble of our sound. The user interface for this might be a row of knobs that slide up and down. We call these objects "sliders". Our left palette (called the View palette) has both vertical and horizontal sliders to choose from. To create a row of vertical sliders we will drag a vertical slider into the main window and place it in the lower left hand corner of the screen. Rather than resizing the slider by dragging the right middle knob, hold the Alternate key down at the same time you drag the knob. You will notice that as the "rubber-band" line reaches twice the width of the slider another slider will appear next to the original. As you ALT-DRAG further you can get an entire row of sliders. Create a row that fills about half the screen.

Now supose we wanted to create a program that enters names and phone numbers. Lets group each name and number together in a unit so the user will know they are related and so that they can be moved together.

First drag a "Box" object to the upper right corner of the screen. Resize it to fill about one fourth of the main window. Then drag a "form" object, which is a white text box with the label "Field:" to the left of it. While the knobs are still showing on the form, go to the main menu and select "Edit" and then "Copy" (or type COMMAND C). Then select "Paste" (or type COMMAND V). You should see two forms on top of each other. Drag the top one to the lower right of the Box object. Then hold down the SHIFT key while you click the initial Field. From the main menu select the Layout, Align, and Make Column options. Your forms should now be directly under each another.

Our last exercise before we test our interface will be to select the edge of the box we created and from the main menu select the "Layout" and "Size to Fit". You should now see you user interface look something like the following with the exception that your interface may have different spacing between the objects (see below).

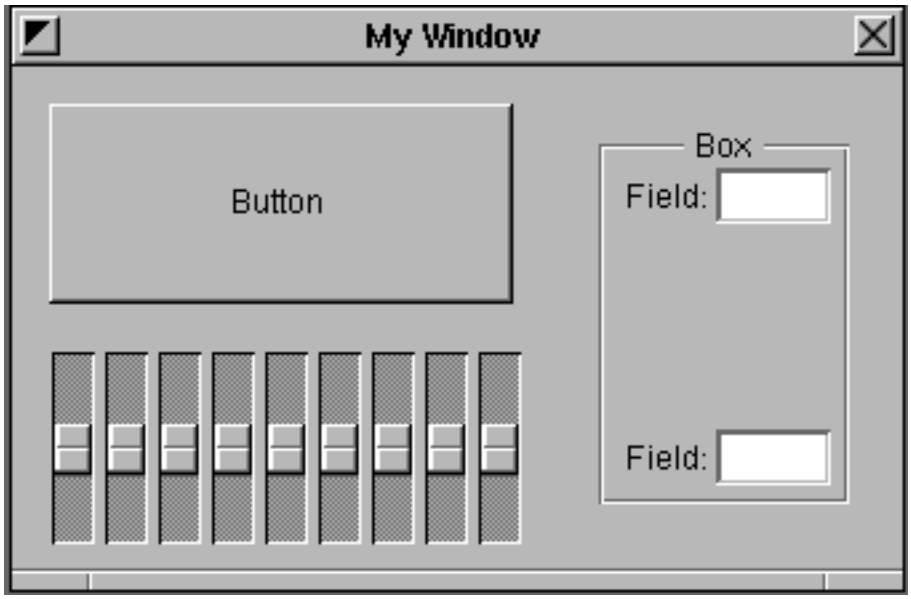
We have shown that the Interface Builder is a tool that allows you to create user interface objects that are meticulously laid out as fast as you can draw the objects on the screen

But this is only the beginning. The objects that you have placed in the main window are more than just graphic objects. They are software objects. Each one has a specific function. One such function would be transferring user-generated events (such as pressing the mouse over a button) to a receiving object which would cause a section of a program to be executed. To show this, lets use the "Test Interface" mode of the Interface Builder. This selection is available from the "File" command of the main menu.

Once you are in the test mode you will find that all of the windows involved in creating the interface have disappeared and the only windows left are the ones that would exist if you were actually running the application. In this case the main menu and the main window.

Try pressing the button. Notice that it will highlight to indicate that it is being pressed. Try to move the sliders in the graphic equalizer. Now try typing text into one of the forms. You should be able to use the COMMAND-C and COMMAND-V keys to copy and paste text from one form into the other. When you are done you can select the quit menu and you will be returned to the build mode of Interface Builder.

Our next step is to make connections between objects. To see the effects of this connection, drag a horizontal slider and a text box into the middle of your main window. Now make a connection from the slider object to the text object. This is done by pointing to the "source object"



interface (and in some cases, an entire application) graphically rather than by writing Objective-C code. With Interface Builder, you manipulate graphic representations of Application Kit objects just as if you were using a graphics editor to create a drawing.

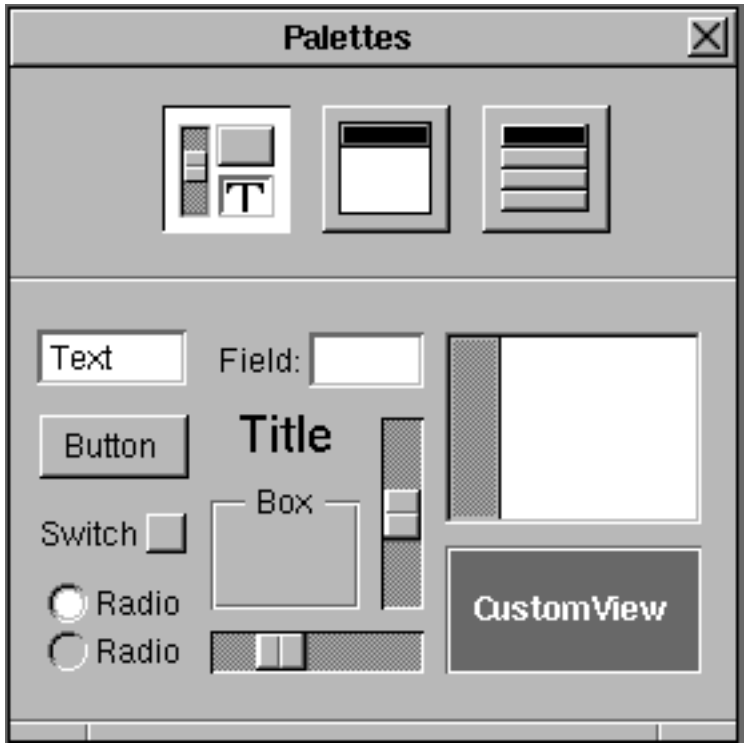
NeXT System Reference Manual - chapter 8, page 1

Let's first get a feeling of what it's like to build a user interface using tool-kits. Thismonth's article will give you a short guided tour of the Interface Builder. It assumes you are familiar with a mouse and have had at least a half hour exposure to an icon based computer such as the NeXT or a Macintosh. Keep in mind that although we will be discussing user interface objects, the idea of using tool-kits and creating connection based programs are very general and can be applied to many other areas.

The NeXT on-line documentation contains over a hundred page description of and tutorial for the Interface Builder. (See /NextLibrary/Documentation/ NeXT/SysRefMan/ 08_IntfBuilder.wn) This chapter is meant to complement that information. One of the best ways to learn about the Interface Builder is to start with this chapter and then later go through the on-line documentation for additional details as needed. We will also give you a more general background on encapsulation and inheritance in later chapters, so don't worry if what you are doing when you sub-class an object is not crystal clear.

If you have not done so already, drag the icon form /NextApps/InterfaceBuilder to you application dock. It should look like the following:

Once it is on the dock, double click the mouse button over the icon. The icon will "highlight" itself for less then ten seconds and then the following panel will appear:



This highlighting stage between when you double click the icon and when the main menu changes is called the "launching" stage of a program. During that time the program is being read off the disk and loaded into memory. If you attempt to do other things during a program's launch stage the system will usually react sluggishly.

The panel above is perhaps the most important single panel you will see in computing in the 1990's. It is much like the painters palette. It contains the elements that you will use in creating your user interface. We will discuss later at length how it will affect the way we create programs. In this version of Interface Builder, the top row contains three buttons. Initially only the first one is highlighted. This is the "views" panel. It contains all user interface objects that occupy space on the screen.

Try clicking on the object marked "button" or some other objects and dragging them to different parts of the screen. Notice that the selected button will be highlighted but it will always zoom back to the palette when the mouse button is released. The reason is that because we have not created a new application, there is no place for the objects to anchor themselves. The new application will be the "canvas" on which we will paint our objects. To create a new application, we must go to the main Interface Builder menu, with the letters "IB" on the menu title bar, and choose the "File" menu followed by the "New Application" selection.

After a new application is created, three new panels appear on the screen. The largest is the window titled "My Window" at the top center section of the screen. This is the "canvas" on which we will "paint" our user interface objects. We often refer to this as our "main window" because it is usually the first window to be displayed when a program launches. There are two other panels that also appear. One is a small menu to the left of the screen directly under the main "IB" menu. This is the menu associated with the program we are creating. The last new panel is the one in the lower left corner of the screen. This is usually titled "/PATHNAME/Untitled" where PATHNAME is the path to your home directory. This last panel controls which of the panels on the screen are active. It is the "controller" for the rest of the panels in

Object - Based Computing

Published Monthly

A programmer's guide to object based computing on the NeXT Computer.

July, 1992
Issue #2

VOLUME 1, NUMBER 2, JULY 1992

editorsDesk
Our second issue!

We received a lot of mail regarding our first issue -- praise, complements, and a few complaints as well. The complaints are actually the most highly cherished (though we love the complements) since it is through your criticisms that we can become more useful and thus BETTER...

One of the topics on which we received mail had little today with the publication and more to do with our mailer... Those of you who received our last issue via email received a short **.eps** file as part of the mail message. This file caused the rotating words "Object-Based Computing" to appear on the screen.

A few people wrote nice notes asking how we had performed this trick and complementing us. Others felt that this was a bad thing to do to an unsuspecting reader. One even asked us not to do this sort of thing since he did not want his customers to know that it was possible!

So after all of these comments we have decided not to ever send out this sort of executable postscript again but also to explain how it was done -- so in this issue there is an article (with the PostScript code) explaining how one takes over the screen...

Please write and tell us what kinds of articles you'd like to see. We are also looking for article submissions so please consider sharing some of your ecperiences with the NeXT community.

yours,

Ted Shelton, Editor
Object-Based Computing
ems@its.com

thisIssue
A little about the articles in this issue.

This first issue of Object-Based Computing owes a great deal to Dan McCreary and his Minneapolis based NeXT development group. A conversation with Dan four months ago first caused me to think of creating this newsletter. And now, our first issue contains three articles from members of his new company, Integrity Solutions, Inc.

The first of a series of six articles on object-oriented computing appears in this issue. Written by Dan McCreary of Integrity Solutions, Inc. this series represents the first part of a new book on Object-Based Computing which focuses on the NeXT computer. It is a pleasure to include this serialized version of an outstanding book in our publication.

By Lionel P. Aboulkheir, also a member of the Integrity Solutions staff, we have an article on the "Linguistics of Object-Oriented Programming."

Finally, a second article by Dan McCreary exploring some of the reasons behind the industry switch to object oriented development environments titled "Object-Oriented Programming: The Probable Person Period."

I hope that you enjoy reading this first issue of OBC as much as we have enjoyed starting this new enterprise.

nextIssue

Object-Based Computing (pt. 2)
NeXT/OOP: A Case Study
usernamed: A Network Daemon
Flexible Object Design

Object-Based Computing (pt. 2)
Dan McCreary - Integrity Solutions

(This is the second of six articles on Object-Oriented programming)

Interface Builder speeds the creation of applications by letting you define an

index

features

400 WEST ERIE, SUITE 402
CHICAGO, IL 60610