

# ***Building Applications***

# Building Applications

**Two important application builder programs:**

***Interface Builder (IB)*** - for building the screen interface for the application.

***Project Builder (PB)*** - for managing the files needed to build the application.

This section provides an overview of these two programs. It is not intended to be a tutorial on how to use them.

# Interface Builder

*Interface Builder (IB) lets the user:*

Use ***Application Kit*** objects to design user interfaces.

***Inspect***/change initial values of objects.

Make ***connections*** between objects.

***Test*** the interface.

Design ***custom objects*** (*subclasses of AppKit objects*).

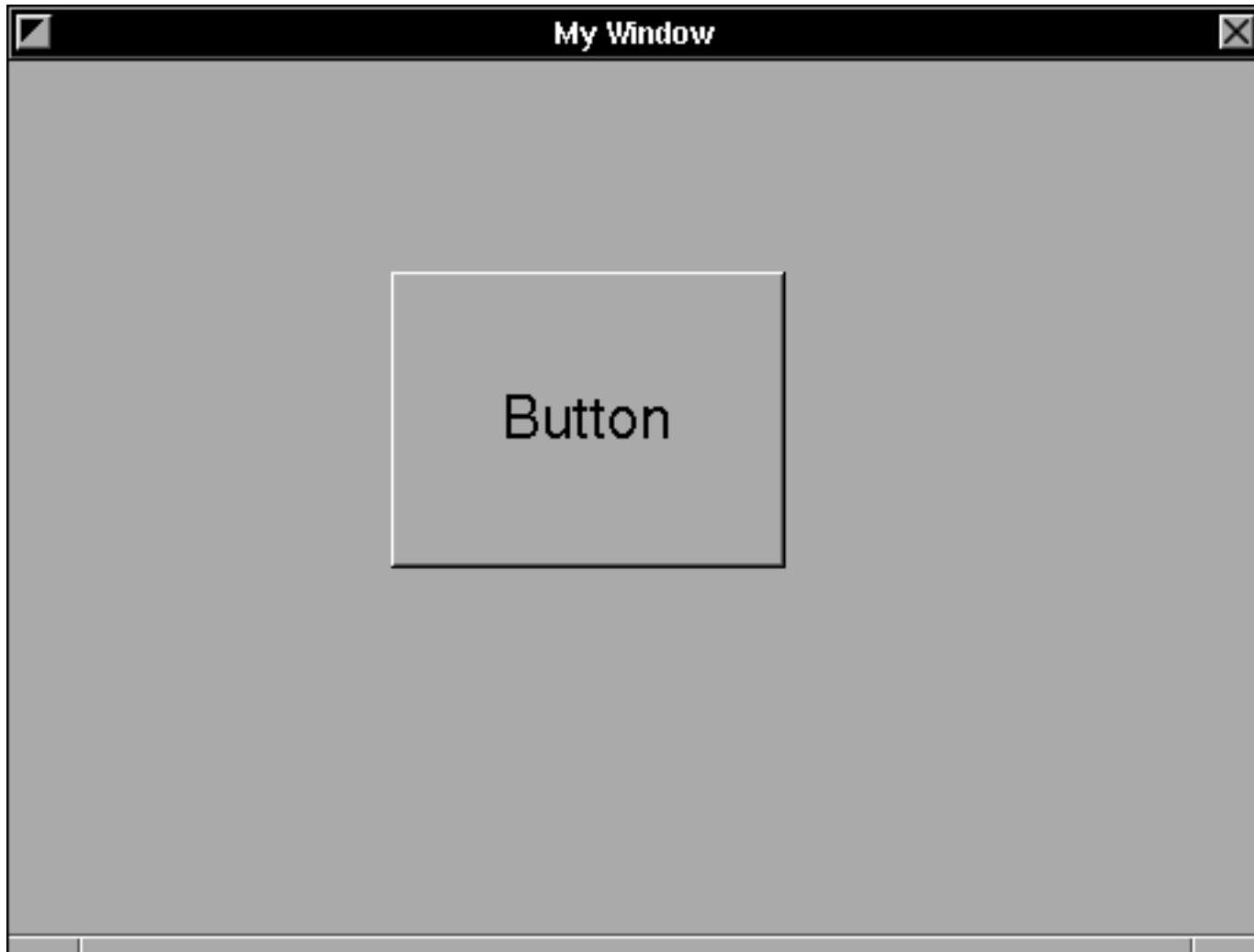
# ***The Application Kit***

A library of ***user-interface objects*** that you can select from to be placed into your application's ***window***.

***AppKit*** objects include such items as buttons, sliders, panels, switches, views, etc.

In general, your application will include a number of AppKit objects and one or more subclasses of ***Object*** (containing the logic of your application) and ***View*** (drawing code unique to your application).

## ***Building Applications***



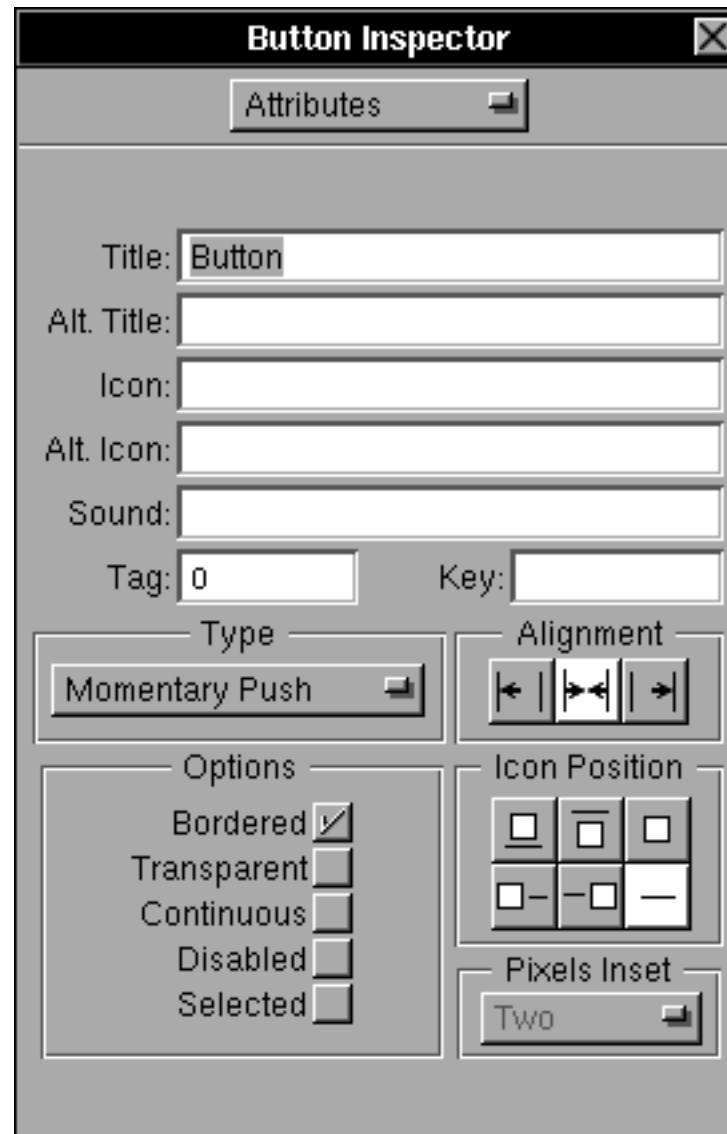
## ***Changing instance variable values***

Once an object is added to your application, IB allows you to change the values of many instance variables directly.

For example, changing the size of a button on the screen changes the values of some of the Button object's instance variables.

Instance variables for objects not easily changed graphically can be changed using the ***Inspector Window***.

## ***Building Applications***



The image shows a 'Button Inspector' dialog box with a title bar containing a close button. The dialog is organized into several sections. At the top is a tab labeled 'Attributes'. Below this are five text input fields: 'Title' (containing 'Button'), 'Alt. Title', 'Icon', 'Alt. Icon', and 'Sound'. Below these are two more input fields: 'Tag' (containing '0') and 'Key'. The main body of the dialog is divided into four sections: 'Type' with a dropdown menu showing 'Momentary Push'; 'Alignment' with three icons representing different alignment options; 'Options' with five checked/unchecked checkboxes for 'Bordered', 'Transparent', 'Continuous', 'Disabled', and 'Selected'; and 'Icon Position' with a 2x3 grid of icons representing different icon positions. At the bottom right is a 'Pixels Inset' section with a dropdown menu showing 'Two'.

**Button Inspector**

Attributes

Title:

Alt. Title:

Icon:

Alt. Icon:

Sound:

Tag:  Key:

Type:

Alignment:

Options:

- Bordered ☒
- Transparent ☐
- Continuous ☐
- Disabled ☐
- Selected ☐

Icon Position:

<input type="button" value="Top Left"/>	<input type="button" value="Top Center"/>	<input type="button" value="Top Right"/>
<input type="button" value="Bottom Left"/>	<input type="button" value="Bottom Center"/>	<input type="button" value="Bottom Right"/>

Pixels Inset:

## ***Making connections***

Interface Builder lets you interconnect objects so they can send messages to one another.

Simply ***control-drag*** from the source object to the destination object, then specify the connection.

Connections are made through an object's ***outlets***.

An ***outlet*** is an instance variable of type ***id*** that is a pointer to the intended receiver of a message.

*When your application begins execution, outlet variables are automatically initialized to the ***ids*** of the objects you connect to in Interface Builder.*



## ***Two types of connections:***

1. ***Simple Outlet:*** *Outlet of a non-control object to some other object.*

*Example:* a Window object has an instance variable **delegate** which needs to be initialized to the **id** of the object that is going to be the delegate (so the window can send messages to it).

In IB the user can ***control-drag*** from the window to the chosen object, and then specify a "connection" for the **delegate** outlet.

## Building Applications

Window's *delegate* outlet is connected to an instance of MyObject:



## ***Building Applications***

2. ***Target-Action:*** Outlet of a control object to the action method of some other object.

Controls have a special outlet called **target** which can only connect to ***action*** methods. The paradigm is:

*Control sends action message to target:*

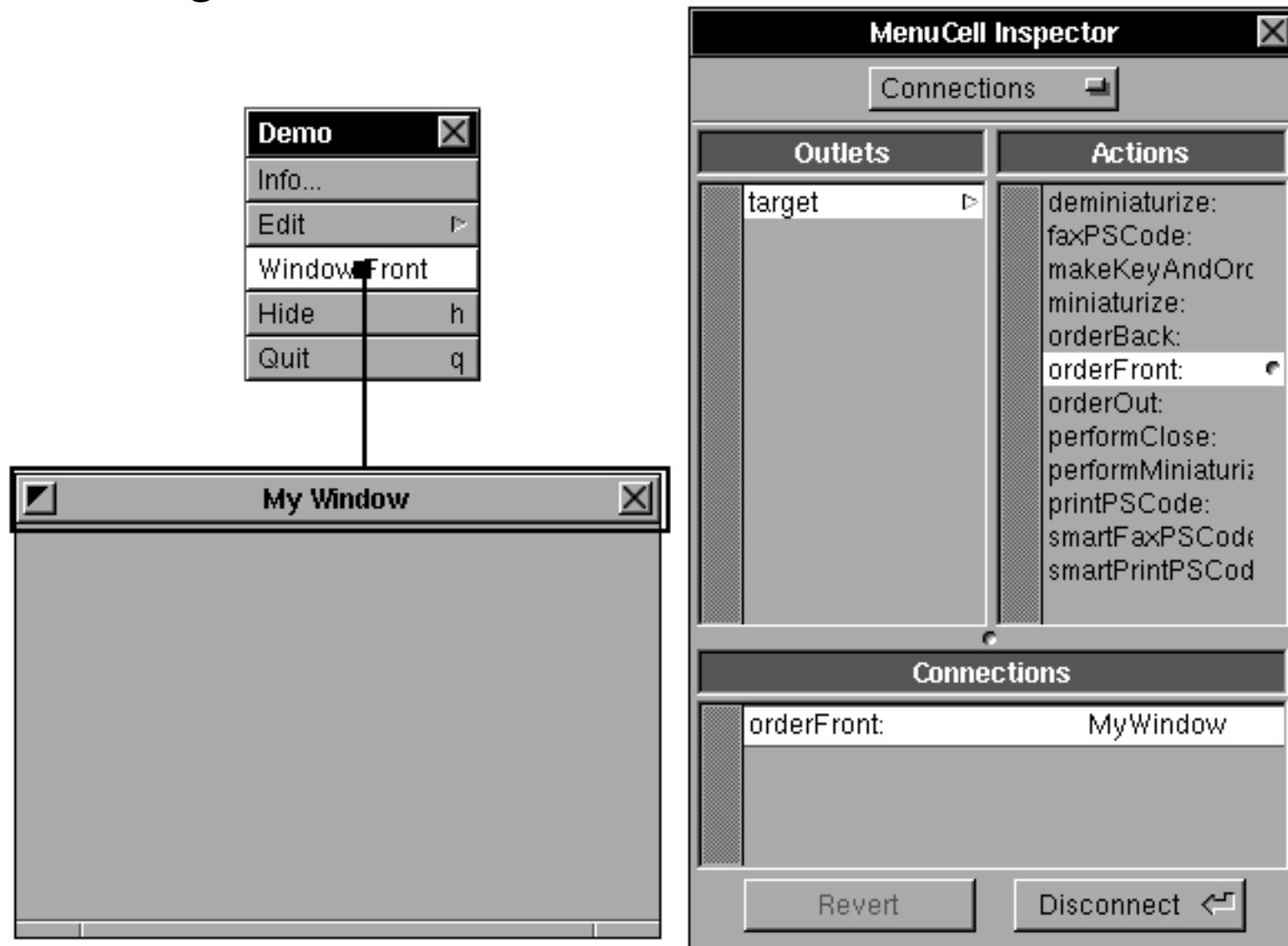
```
[target actionMethodName:self];
```

*Action method is defined in target object:*

```
-actionMethodName:sender {  
<code to take action>  
return self;  
}
```

## Building Applications

MenuCell's **target** outlet connects to Window's **orderFront:** action method.



## ***Testing the interface***

Interface Builder has a ***test interface*** mode which allows testing of the newly-connected interface objects.

Only the objects obtained from Interface Builder palettes are fully functioning in the test interface mode. New custom objects are not.

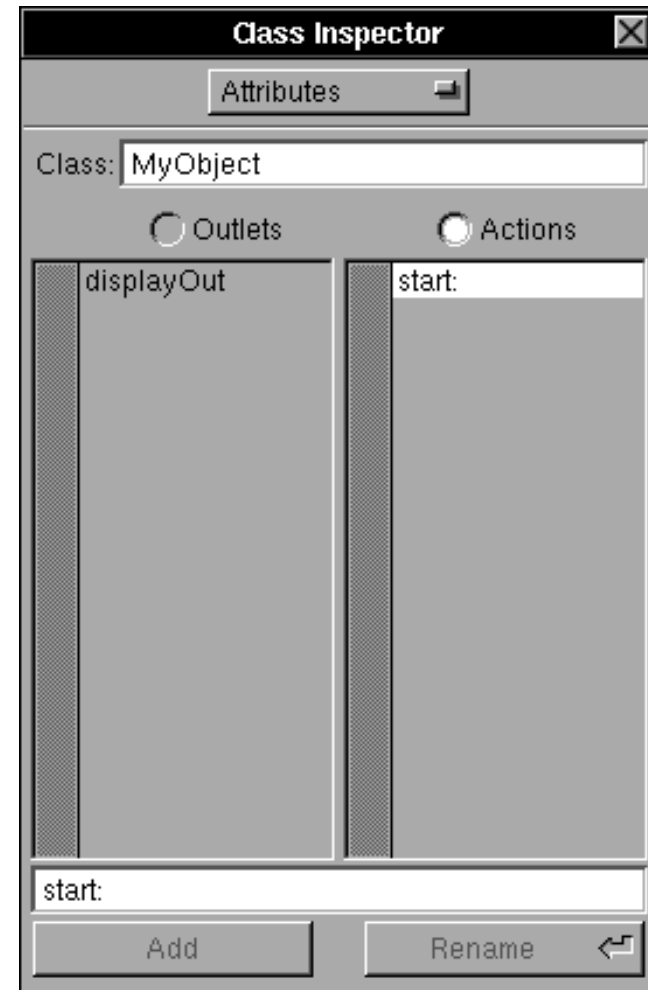
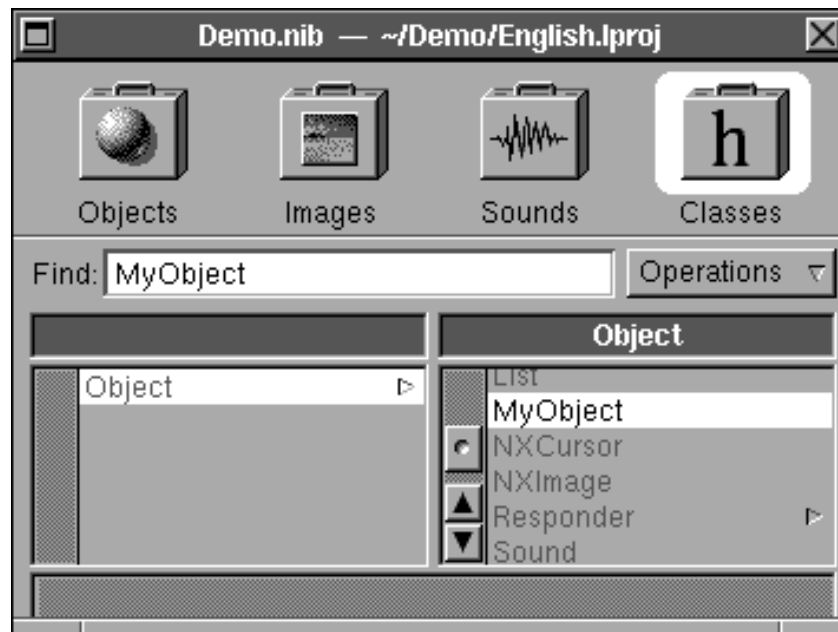
# *Designing custom objects*

Interface Builder allows you to specify new **custom objects** as **subclasses** of AppKit objects or of the root Object class.



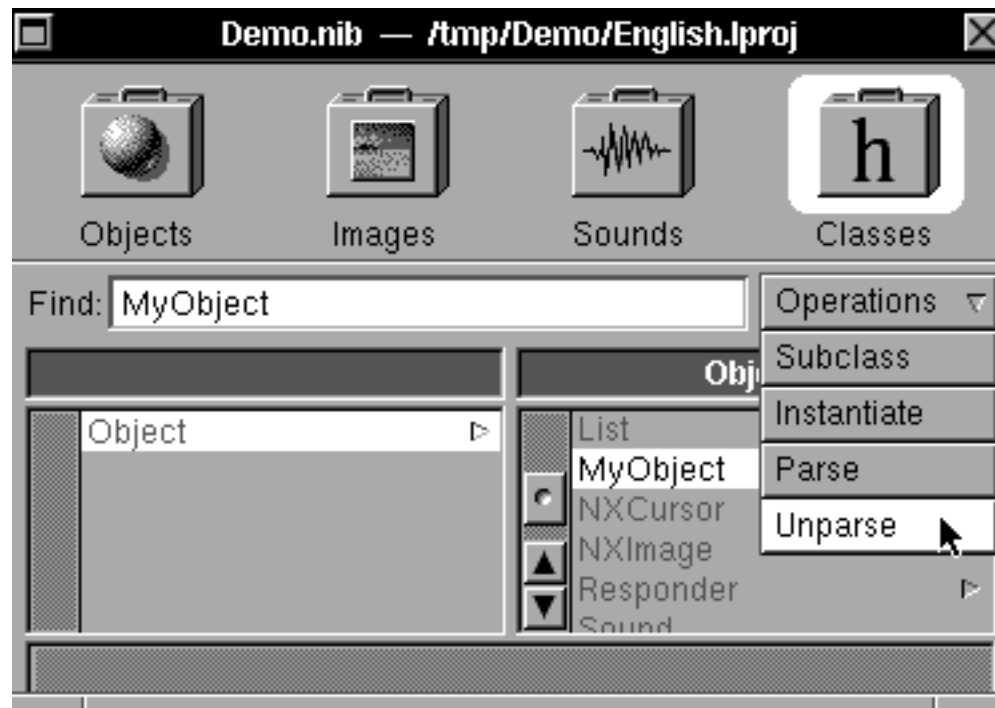
## Building Applications

Here we define a **displayOut** outlet and a **start:** action method for the new MyObject class.



## ***Building Applications***

IB will ***unparse*** the new class definitions to generate skeleton files of these new classes. The remainder of the class definition must be typed in by the user (using Edit, for example).





## *Building Applications*

The skeleton interface file (MyObject.h) created by the ***unparse*** operation in IB for the MyObject custom object above is:

```
#import <appkit/appkit.h>

@interface MyObject:Object
{
    id    displayOut;
}

- start:sender;

@end
```

## *Building Applications*

The skeleton implementation file (MyObject.m) created by IB for the same custom object is:

```
#import "MyObject.h"

@implementation MyObject

- start:sender
{
    return self;
}

@end
```

## The "nib" File

The interface you develop is saved in an Interface Builder file which has a **".nib"** extension.

The **nib** file contains all of the class information and all specifications for the AppKit objects in the interface.

The **nib** file contains information about how outlets should be initialized, about action messages and their targets, sound and icon data, and a reference to an owner object ("File's Owner").

# The "nib" File

(continued)

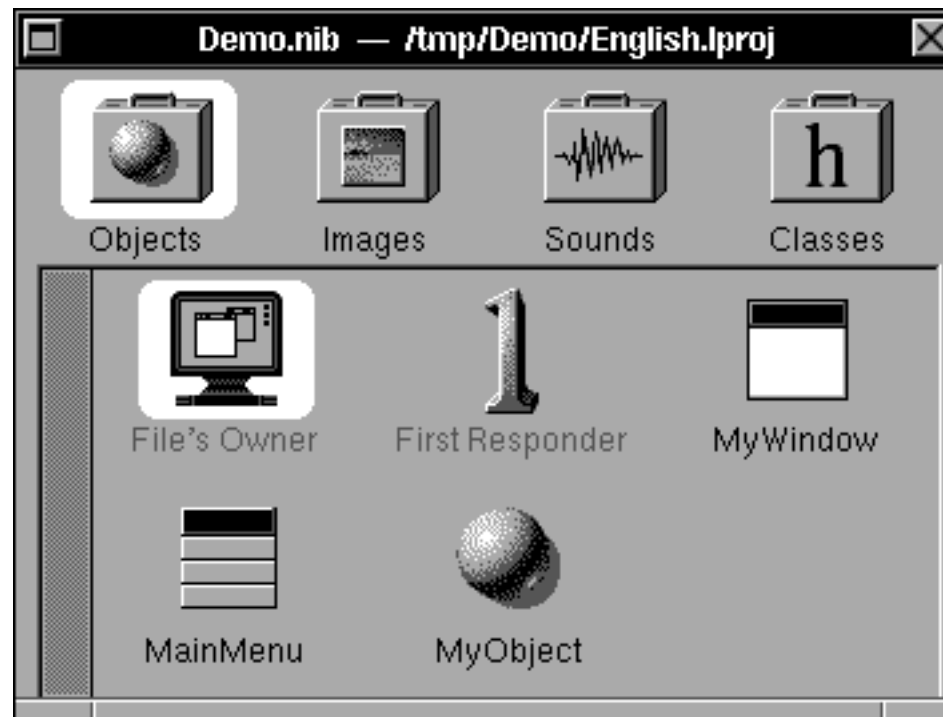
When your application is compiled and built (see Project Builder later), the **nib** file is included in the application's file package.

When your application is run, your ***Application Object*** creates objects and initializes outlets using information from the **nib** file.

An application can have more than one **nib** file, but only one can be the *main nib* file (which includes the main menu).

## ***Building Applications***

The objects stored in the **nib** file are those in the main window, plus those represented by icons in the File Window:



***File's Owner*** is the only object outside of the **nib** file that a connection can be made to. By default, the owner of the **main nib** file is the ***Application Object***.

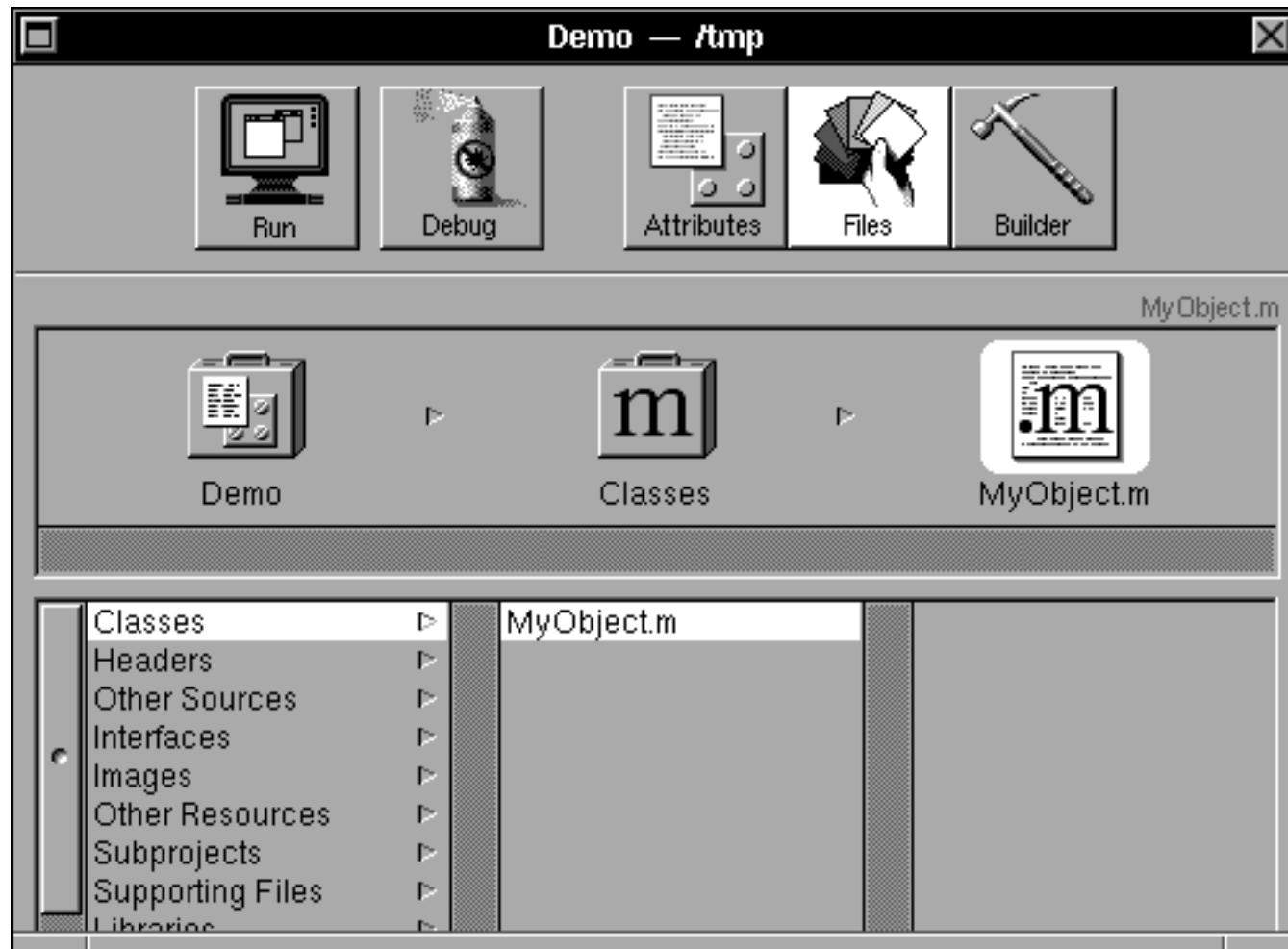
# Project Builder

*Project Builder (PB):*

Manages in one ***project directory*** all of the files and resources used to build the application. These files include the nib files built in Interface Builder.

Maintains a ***project file*** called ***PB.project*** to organize the files.

## *Building Applications*



## ***Building Applications***

### ***Project Builder:***

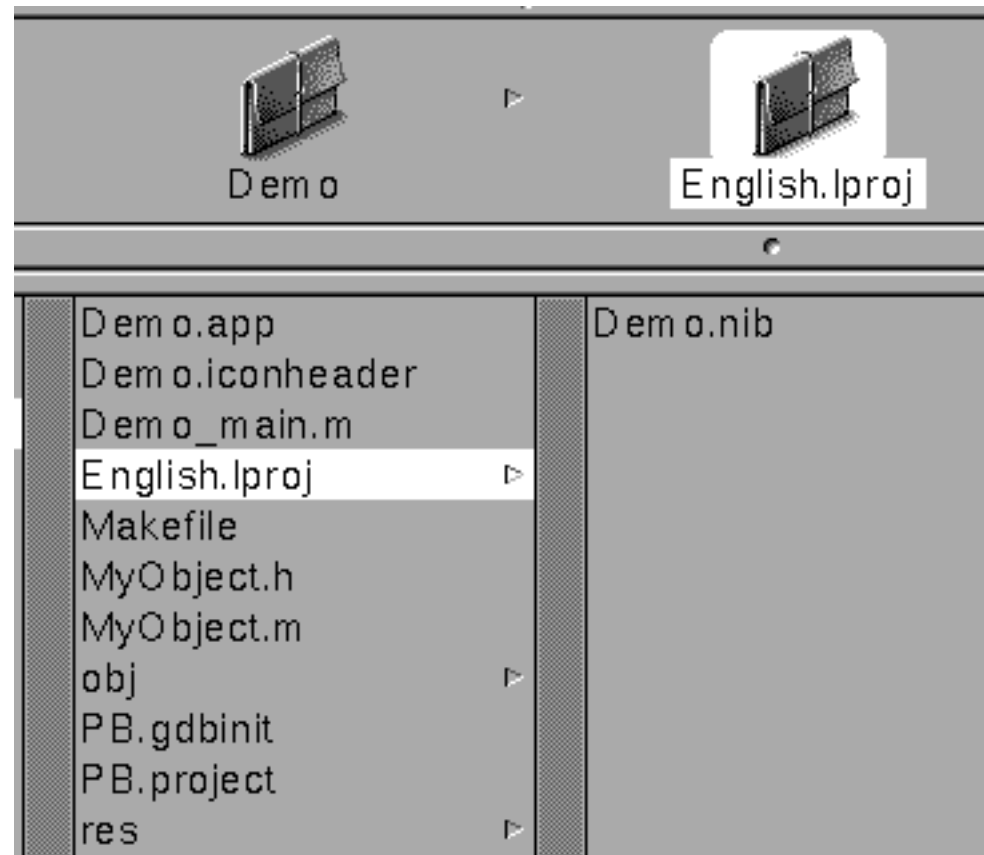
Creates and updates several other files, including:

- the ***Makefile*** for compiling and linking.
- the default ***main file*** containing the ***main()*** function. This function messages the Application object to load the nib file(s).
- the ***icon header file*** (***.iconheader*** extension) containing information about the icons associated with the application and its documents.



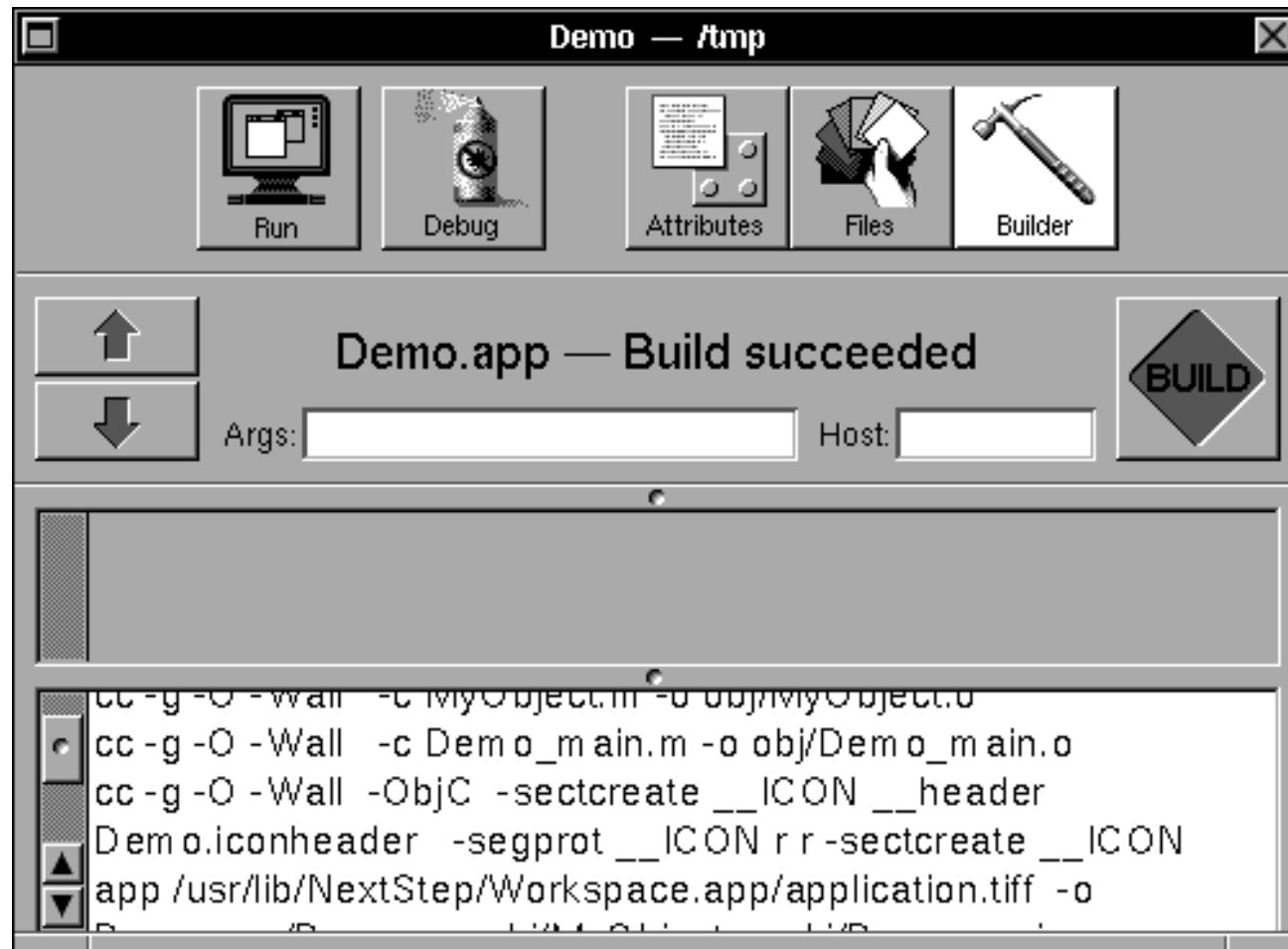
## ***Building Applications***

All of these files are kept in the project directory and its subdirectories:



## ***Building Applications***

*Project Builder: **Builds** and **runs** the application, optionally using the debugger.*



# Steps for Building Applications

*If no custom objects, one main nib:*

- 1) In PB, create a new project and main nib file.
- 2) Open the nib file in IB and draw the user interface.
- 2) Make connections in IB.
- 3) Save IB file.
- 5) In PB, build and run.

## ***Building Applications***

### ***If custom objects, one main nib:***

- 1) In PB, create a new project and main nib file.
- 2) Open the nib file in IB and draw the user interface.
- 3) Make connections in IB (between interface objects).
- 4) *Create custom object(s) in IB*
  - Subclass Object or View (or some other)*
  - Instantiate custom object*
  - Edit object using Inspector*
  - Add actions and outlets*
  - Make connections to custom object*

## ***Building Applications***

***If custom objects, one main nib (cont'd):***

*5) Unparse custom object*

*6) Add instance variables to .h file*

*7) Add code and #import's to .m file*

*8) Save Files*

*9) In PB, build and run.*

# ***References***

*NeXT documentation manuals.*

*NeXTSTEP Applications Programming: A Hands-On Approach*, by  
Simson Garfinkel and Michael K. Mahoney, Spring-Verlag, 1992.  
The best book anywhere on developing applications under  
NeXTSTEP.