# Creating a User Interface in Apple Dylan

**Preliminary**

Developer Press
Apple Computer, Inc.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

# Contents


Preface About This Book v

What to Read v
Conventions Used in This Book vi
  Special Fonts vi
  Types of Notes vi
For More Information vii

Chapter 1 Learning the Interface Builder 1

Interface Builder Overview 2
  Steps in creating a user interface 2
  Using a project window 3
  Using the Windows menu to open tools 5
  Creating a window 6
  Altering windows with the Window Inspector 7
  Using catalogs of elements 8
  Elements in the Views Catalog 9
  Elements in the Additions Catalog 11
  Altering elements with the Property Inspector 12
  Adding color 13
  Creating menus 14
  Setting preferences 17
  Using the Class Importer 19
  Adding your interface to the development environment 21
Interface Builder Tutorial 21

Chapter 2 Using the Interface Builder 43

Using the builder from the development environment 43
  Running the builder within the development environment 44
  Adding custom elements to the catalogs 45

Making interface elements available to your application 48
Loading a window 48
Loading menus 50

Appendix A Palette Buttons 51

Command Palette 51
Text Palette 54
Windows Palette 56

Index 65

# About This Book

This book, *Creating A User Interface in Apple Dylan*, gives you a good look at the Apple Dylan interface builder, which is a separate component of the Apple Dylan development environment. The interface builder can be used within the development environment or as a standalone application.

The Apple Dylan interface builder is probably different from any similar tool you have used in the past, but it uses the metaphor of a graphics program so many of the tools and features will be familiar. By following the information presented in this book, you will soon find yourself interacting with the interfaces you develop in a new and highly interactive way.

The term **Apple Dylan** refers to the development environment and associated tools, extensions and capabilities developed by Apple Computer for programming in the **Dylan** language, an **object-oriented dynamic language**.

For more information on programming in the Dylan language, see the books *Programming in Apple Dylan* and *Apple Dylan Extensions and Framework Reference*. For more details on using the development environment, see the book *Using the Apple Dylan Development Emvironment*. For information on how to install and configure Apple Dylan on your system, see the booklet *Apple Dylan Quickstart*.

## What to Read

This book has two chapters and an appendix. You can either read it sequentially or move around in it from one topic to another.

- Learning the Interface Builder—introductions to and explorations of the interface builder. This chapter should give you a chance to learn how to interact with the builder without a lot of explanation. This chapter primarily familiarizes you with the windows, catalogs, controls, and other features of the interface builder.
- Using the Interface Builder—in-depth tasks on how to use the builder from within the development environment. This chapter begins with a

description of how to open the builder from within the development environment, continues with details on how to add custom elements to the builder's catalogs, and also includes details on how to add your interface elements to your development environment project.

- Appendix A—descriptions of all the buttons on the Command Palette, the Text Palette, and the Windows Palette.

The two chapters include examples and step-by-step descriptions of common tasks.

# Conventions Used in This Book

This book uses various conventions to present certain types of information.

## Special Fonts

All code listings, reserved words, and the names of data structures, constants, fields, parameters, and functions are shown in a monospaced font (`this is monospaced`).

## Types of Notes

There are several types of notes used in this book.

**Note**
A note like this contains information that is interesting but possibly not essential to an understanding of the main text. Often, these Notes include additional information. See page 48 for an example.

▲ **WARNING**
A note like this contains information that is especially important. As this is an early release of Apple Dylan, not all interactions are as smooth as we want them to be. See page 2 for an example. ▲

# For More Information

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

| | |
|---|---|
| Telephone | 1-800-282-2732 (United States)<br>1-800-637-0029 (Canada)<br>716-871-6555 (International) |
| Fax | 716-871-6511 |
| AppleLink | APDA |
| America Online | APDAorder |
| CompuServe | 76666,2405 |
| Internet | APDA@applelink.apple.com |

# Learning the Interface Builder

Apple Dylan is a new programming language and development environment. The development environment has a separate component, the Apple Dylan interface builder, to help you create a user interface for your application in a quick and easy manner.

The interface builder has been designed to be used as a library that is linked into the application under development in the Apple Dylan development environment. This means that the builder is always easily accessible, simply by showing the builder from a menu-item and hiding it again. The application and builder can also be suspended to drop back into the Apple Dylan development environment. This makes it very easy to develop some portion of the user interface and attach functionality to it while the application is running. The design/implementation/test loop is much tighter than it would be if the builder were a separate application. The builder requires a color monitor.

A standalone version of the builder is also provided. You can double-click on this version in the Finder to run it without running the Apple Dylan Development Environment.

The interface builder is built entirely on top of the Dylan framework and is, therefore, really just a higher level abstraction of the framework. Every user-interface element that is manipulated in the builder represents an element from the framework, including views, adorners, determiners, and behaviors. One of the driving concepts behind the builder is to make the capabilities of the framework more accessible. For more information on the framework, see the books *Programming in Apple Dylan* and *Apple Dylan Extensions and Framework Reference*.

▲ **WARNING**
Any interface builder projects created with earlier versions of the interface builder will not open using this version of the builder without taking the following actions. If you have an old project you want to open with the new builder, you must first copy the resource counter for menus, "mcnt" id: 0, and the resource counter for windows, "wcnt" id: 0, from a new project into the old project with a resource editor, such as ResEdit. ▲

# Interface Builder Overview

Both the Apple Dylan development environment and the interface builder use projects to organize their contents. Each user-interface project corresponds to its development environment project. Projects in the development environment consist of such objects as modules, subprojects, source folders, and source records, while user-interface projects consist of the windows and menus in the user interface. In the development environment projects are displayed in its project browser, while in the interface builder a project is displayed in its project window.

## Steps in creating a user interface

The basic steps you need to perform to create a user interface are to open the interface builder, create a new project, create the windows and menus you want in it, customizing them as needed, and then save the project to a project file, which contains its resources. You then return to the development environment, add the interface project file to the development environment project it goes with, and do whatever coding needed to utilize the high-level elements in your application. In addition to these basic steps, you can also create your own custom types of user-interface elements by coding them using framework code in your development environment project, and then adding them to the catalogs in the interface builder.

## Using a project window

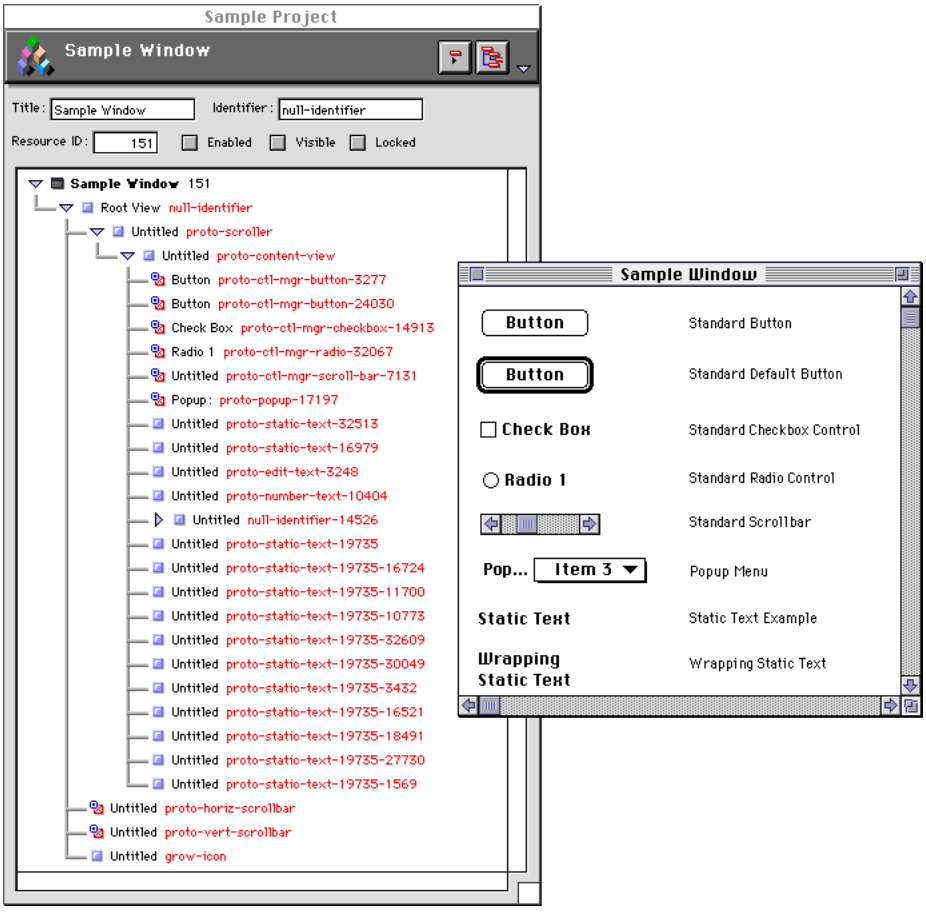
The contents of a project are displayed in the project window in the form of a hierarchical tree diagram. Individual high-level elements (such as windows and menus) are displayed at the root level of this tree. The root-level nodes for windows, and any node within them containing children, have disclosure triangles at their left edges that allow them to be expanded or collapsed. Menus cannot be expanded in the project window, but must be double-clicked to see their contents. Two buttons at the right of the project window's header collapse or fully expand the tree diagram.

When a window node is expanded you can view the element hierarchy (view hierarchy) for the window, which shows the embedding relationships of all the elements in the window. The project tree also displays some of an element's properties, allowing you to edit them directly from the project window and to see the values for their more common properties, such as their identifiers.

The project window in the following figure lists the entire hierarchical tree diagram of the elements created for a sample window. When you select an element in the project window, its visual representation in the sample window is highlighted. When you select an element in the sample window, its entry in the project window list is also highlighted. If you double-click on a window entry in the project window that is not currently open, the window is opened.

Italics in the project window indicate that the window or menu has not been saved. You do not need to save each menu or window individually; they are not lost when they are closed, only hidden. When you save the project, all the windows and menus you have created for it are saved.

The following figure shows the project window for the project named "Sample Window". The project window has only one high-level element listed for this project, a window named "Sample Window," which is the window to the right in the figure.

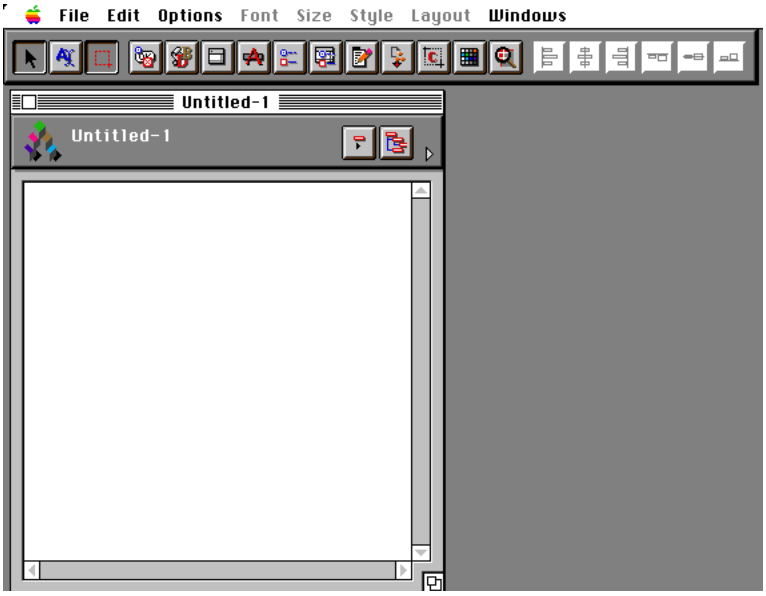



The data input panel for the project window can be expanded to reveal some important values for a project. The data input panel is expanded using the disclosure triangle on its far right. In the preceding figure the disclosure triangle has been clicked, so the data input panel for the project window is shown. It contains such fields as Title, Identifier, and Resource ID. Changes made to the project from the fields on the data input panel are displayed immediately throughout the project. For instance, if you change a project's name in the Title field, the name of the project window immediately changes.

The interface builder uses the metaphor of a graphics application, so you will see in it such things as selection tools, palettes, and text styles. The menubar

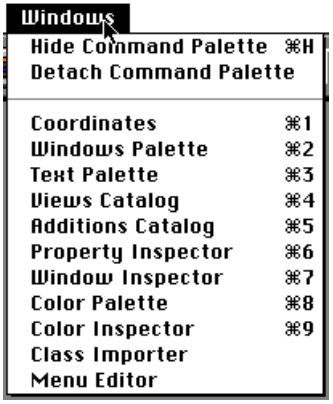
contains commands to perform these actions and open windows for these tools. The Command Palette, which is attached just below the menubar, contains buttons for many of the more commonly used commands and tools. You can detach the Command Palette from the menubar using the Detach Command Palette command on the Windows menu.

The following figure shows the builder's menubar, Command Palette, and an empty, new project window titled "Untitled-1", which is the default name for your first new project.



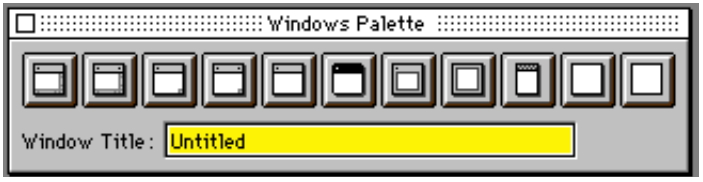

## Using the Windows menu to open tools

The interface builder's palettes, windows, and catalogs of elements are the primary tools you use to create your interface. These are available from the Windows menu, which is shown in the following figure. When you click on a menu item below the separator bar on the Windows menu, the corresponding window opens and a check mark appears by the menu item. If you click on a checked menu item, it brings the corresponding window to the front.




## Creating a window

From the Windows Palette, you can create the windows for your interface. The interface builder provides a set of eleven ready-made window styles you can choose from. To create a new window using the Windows Palette, enter in the Window Title field the name you want your window to have, if any, and click one of the buttons from the Windows Palette's set. The following figure shows the default Windows Palette.

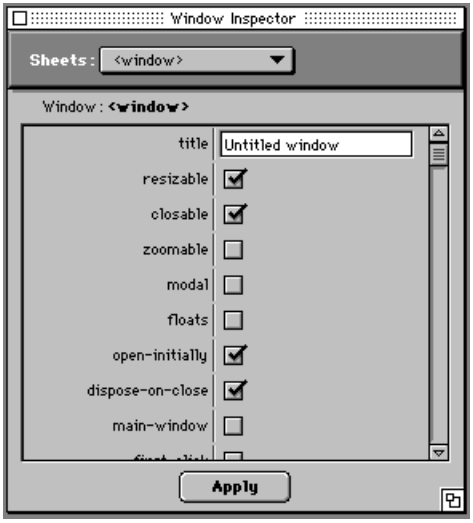



You can use the mouse to reposition or resize elements in a window and see the new locations as you are changing them in the Coordinates palette. The following figure shows the default Coordinates palette.

The following figure shows the Text Palette, which contains buttons for the commonly used text manipulation commands. You can use the Text Palette to style text in window elements. You can set whether the Tool Palette is automatically shown or is attached to the Command Palette when opened by using the Window Preferences page of the Preferences command.


## Altering windows with the Window Inspector

If you need to change a window once you have created it, you can alter many of its characteristics using the Window Inspector. To use the Window Inspector, make the window you want to alter active and choose Window Inspector from the Windows menu. The following figure shows the Window Inspector displaying the properties for a newly created window.



## Using catalogs of elements

The interface builder contains two catalogs of ready-made elements you use to construct your user interface: the Views Catalog and the Additions Catalog. The Views Catalog contains pages of view elements. The Additions Catalog contains adorners, behaviors, and determiners that you can drag onto view elements to alter their qualities, such as to add a drop shadow to a button. On the right side of both catalog's header is a button that takes you to the other catalog.

You drag and drop the ready-made view elements, such as buttons, from the Views Catalog into the windows you are designing. Once an element is in the window you are designing, you can alter it to meet your exact specifications. For instance, you could drag a drop shadow from the Additions Catalog onto a button you dragged into a window from the Views Catalog, causing a drop shadow to appear on that button. You could also move the view element around in the window, resize the element, apply color to it, make copies of it, and change its name. If you do this, you might find that you have created an element you'd like to use again, such as an OK button. To do this, simply drag the element from your window back onto a page in the Views Catalog.

You can also alter elements on the Views Catalog's pages without ever dragging them into a new window by holding down the Option key as you click on the element on the page. This opens a Property Inspector Window for the element, displaying various values you can alter. You can also add entirely new pages of customized elements to the Views Catalog.

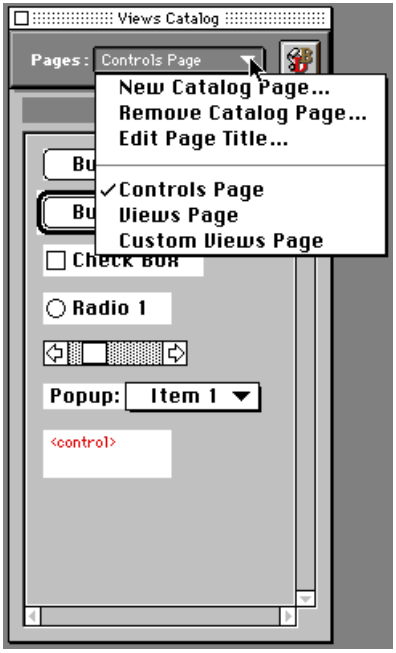
Changes you make to the catalogs, such as adding elements or pages, are persistent only if the catalogs are saved. If you add anything to or change the catalog pages in any way, be sure to use the Save Catalog command before quitting the builder. The best way to be sure you won't lose any new elements from the catalogs is to issue the Save Catalog command as soon as you run the builder for the first time. The name of the file you save the catalogs to appears in the title of each catalog. The default name for your catalog file is Interface Builder Catalog, but you can change that to whatever you want and select the location you want. You can alter the way the catalogs work using the Catalog Preferences page of the Preferences command.

## Elements in the Views Catalog

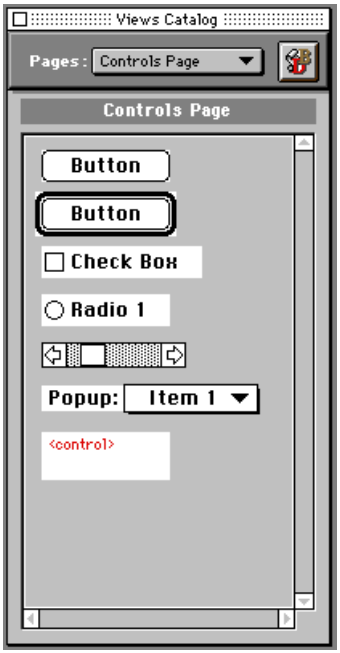
All the subclassed views from the framework's <view> class appear in the Views Catalog by default, including such elements as buttons, check boxes, and static text fields.

You can add and remove pages to the Views Catalog, as well as edit page titles, using the commands under the Pages popup. You can choose the page you want to see from the same popup. To the right of the Pages popup is a button that takes you to the Additions Catalog.

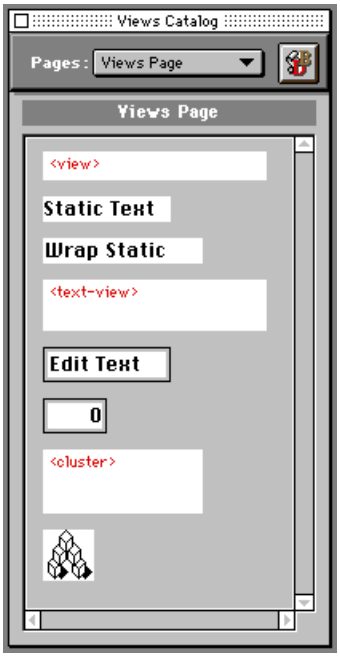
The following figure shows the list of pages and commands in the Views Catalog.




The following figure shows the Controls Page of the Views Catalog.




The following figure shows the Views Page of the Views Catalog.

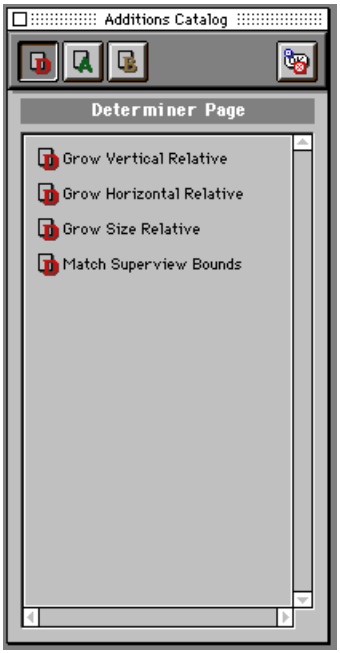



## Elements in the Additions Catalog

The Additions Catalog has three pages of elements you can drag onto view elements, the Determiner page, Adorner page, and Behavior page. The elements from the Additions Catalog alter the behavior, appearance, and determiners (constraints) applied to the view element they are dropped on, such as whether the element moves when the right edge of its window moves. You access the three pages using the three buttons in the header of the Additions Catalog. To the right of the three buttons is a button that takes you to the Views Catalog.

You cannot alter the elements in the Additions Catalog or add and delete pages from it. However, you can add elements to it.

The following figure shows the Determiner page of the Additions Catalog.



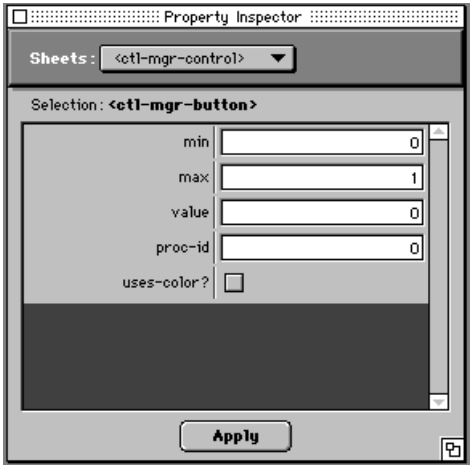

## Altering elements with the Property Inspector

One important way to change an interface element is to use the Property Inspector. You can alter elements in a window you are designing or on a page in the Views Catalog. The Property Inspector displays an editable property sheet for the selected element, based on the type of element it is. When a property sheet is opened, you can edit each slot in its class based on the slot's type. For example, an integer that is editable would have a number text field on the sheet in which a value is displayed and edited. A color displayed on the sheet shows the color and its values, and also allows access to the standard system color picker, so you could change the color by changing its RGB value or choosing another color from the color picker. The sheets also contain read-only values that are not editable.

The Property Inspector can support new types and you can also replace the standard editors with versions more to your liking. Support for this is provided through commonly available generic functions in the framework. These custom

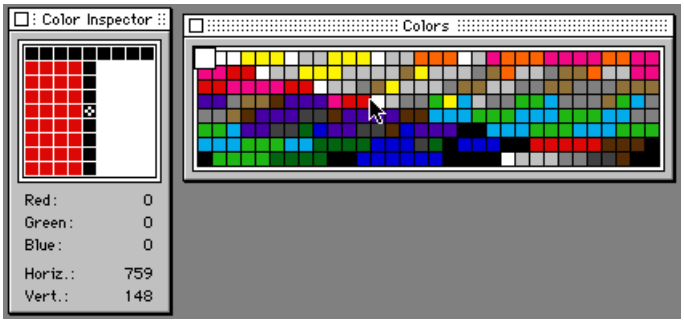
editors are only available when the builder is used as a library that is linked into the application being developed in the Apple Dylan development environment, not when the builder's standalone is run.

The following figure shows the Property Inspector displaying the properties for a button.




## Adding color

Color can be applied to the foreground and background of user-interface elements using the Colors Palette. The Color Inspector window enlarges the view of the pixels surrounding the cursor's location so you can see the color of each pixel. The following figure shows the Colors Palette and the Color Inspector. The cursor is on the black line between a red and white box on the Colors Palette and an enlargement of the pixels under the cursor are displayed in the window at the top of the Color Inspector. The lower half of the Color Inspector shows the RGB values for color under the cursor, as well as the location of the cursor.




The Colors Palette's shape can be altered using the General Preferences page of the Preferences command.

## Creating menus

The Menu Editor allows you to create the menus and submenus for your application. You use the Menu Editor to create each menu you want on your menubar, and then you write code in the development environment to identify the order they are displayed on your application's menubar. When you choose Menu Editor from the Windows menu, the New Menu window opens.

Menus appears in the project window as high-level elements without disclosure triangles on the left. Therefore, you cannot expand them to see their contents from the project window, you instead double-click the element to open the menu.

The following figure shows a window named New Menu, which is the default name for a new menu. You can change the name of the menu by clicking in the field in the window's header named New Menu. Above that field, the New Menu window has four buttons that contain the commands for constructing your menu. The menu items for your menu appear within the lower pane. You can set your menu's resource ID in the New Menu window's header.

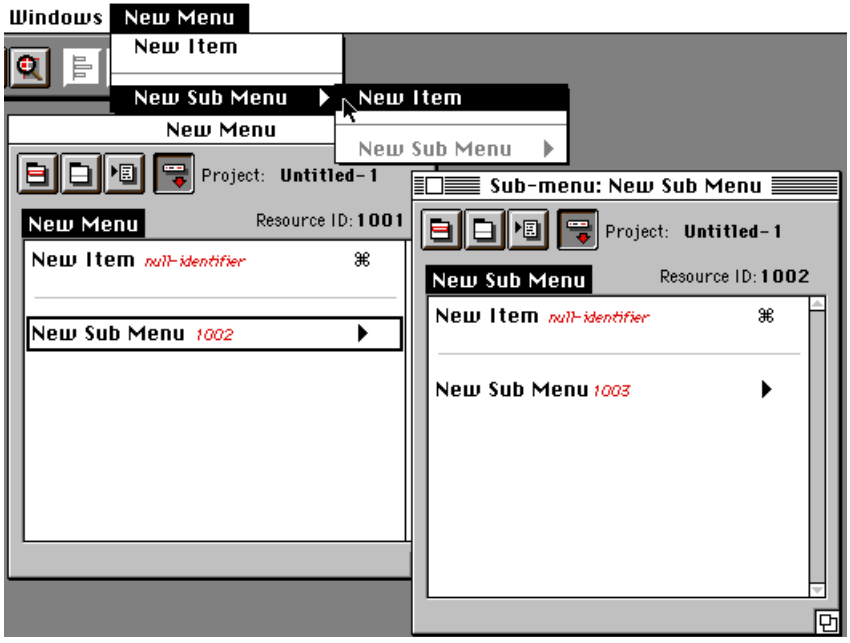
To create a menu item, you select the first button in the header. The second button creates a separator line. If you want the menu to have a submenu, you select the third button, which creates the line "New Sub Menu" and sets its default resource ID inline. The fourth button displays the new menu at the end of the interface builder's menubar, where you can open it to see how your

menu looks. The text in the menu items is editable and stylable inline in the New Menu window. To delete a menu item, select it and choose the Delete key.

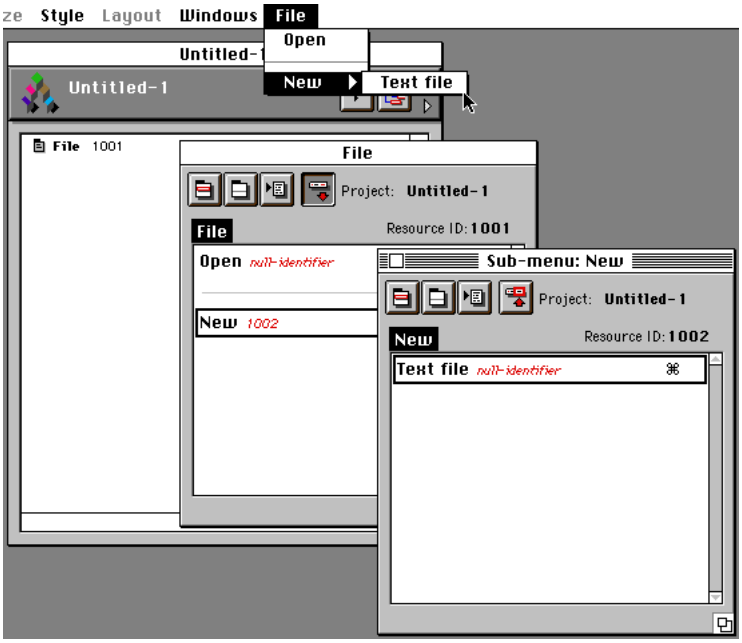
The following figure shows the result of choosing all four buttons in the window's header from left to right.


You create a submenu by double-clicking in the New Sub Menu line. This opens the window "Sub-menu: New Sub Menu," which has the same buttons and functionality as the New Menu window. You can reset the resource ID number of a submenu within the header of the Sub Menu window or inline with the New Sub Menu item in the New Menu window. In either case, the change is immediately reflected in both places, as well as in the project window.

The following figure shows the new menu and its new submenu. The contents of the new submenu are displayed in the Sub-menu: New Sub Menu window. The menu and its submenu appear on the interface builder's menubar.




The following figure shows the result of changing the name of the new menu to File. The New Menu window's name automatically changes to File, the name of the menu listed in the project window changes to File, and the name of the menu on the menubar also changes. The menu items on the new File menu have been renamed Open and New. The menu item New is a submenu, whose contents are displayed in the "Sub-menu: New" window. The submenu New has one menu item, "Text file". Notice that all your changes to names in the menu and submenu windows are automatically reflected on the menubar.

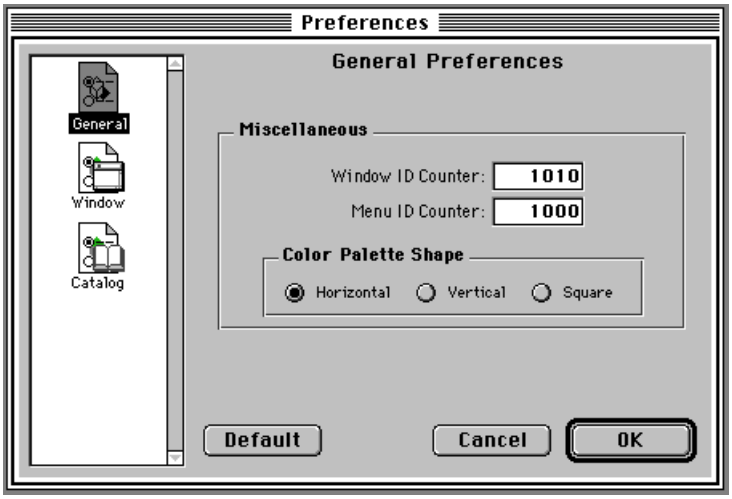



## Setting preferences

Be sure to check the three pages of preferences using the Preferences command to see what defaults you might want to alter. There are three pages of preferences you can choose from, General Preferences, Window Preferences, and Catalog Preferences.

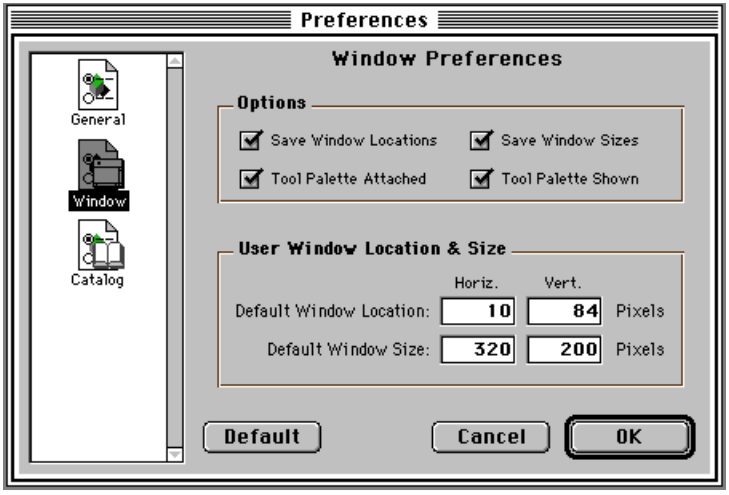
The General Preferences page allows you to set the resource IDs for windows and menus and the shape you want the Colors Palette to have.

The following figure shows the default General Preferences page.



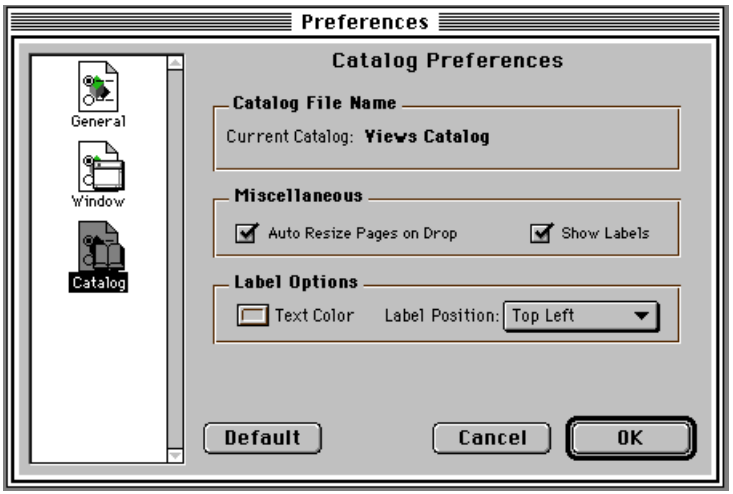

The Window Preferences page allows you to set certain characteristics for the windows you create, such as where they will be located when the user opens them. You can also set whether the Tool Palette will be attached or shown.

The following figure shows the default Window Preferences page.



The Catalog Preferences page allows you to set certain characteristics for how the catalogs behave, including the position and color of labels on elements in the catalogs.

The following figure shows the default Catalog Preferences page.




## Using the Class Importer

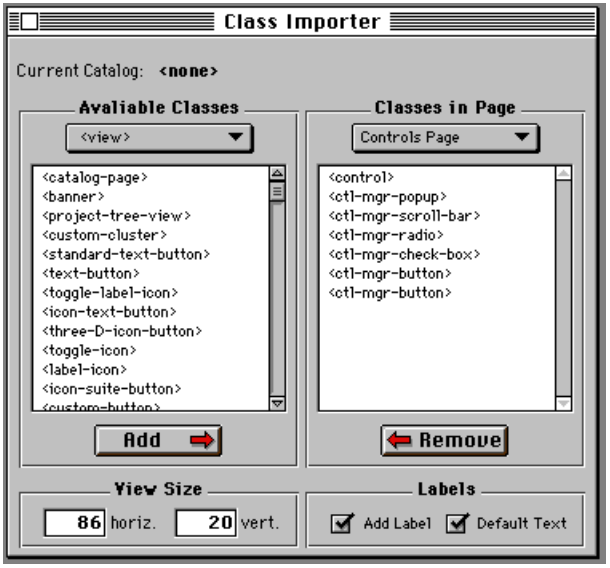
Besides altering windows and classes from within the interface builder, you can also create your own custom types of elements by writing framework code in the development environment. The Class Importer window in the builder allows you to add these elements to a catalog page in the interface builder. You would then be able to reuse the custom elements just as you would any of the built-in elements.

You create your own types of custom elements by subclassing the framework classes <view>, <adorner>, <behavior>, or <view-determiner>, or one of their subclasses. See the books *Programming in Apple Dylan* and *Apple Dylan Extensions and Framework Reference* for more information on how to create your own elements.

The following figure shows the default Class Importer. The subclasses of the <view> class are currently listed in the pane on the left. Each subclass represents a user interface element in the Views Catalog. The pane on the right

lists all the subclasses representing user interface elements on the Controls Page of the Views Catalog.

To use the Class Importer, you must start the builder from the development environment with an application running that uses the framework. Then, return to the builder, using the commands Load UI Builder and Show Interface Builder if necessary, and open the Class Importer. Within the Class Importer, first choose from the Available Classes popup the class you subclassed in the framework. Then choose the subclass you created from the list below the popup (you might have to scroll the pane on the left to find it). Next, from the Classes In Page popup on the right, select the catalog page you want the element to be added to. Last, choose the Add button to add the selected subclass in the left-hand pane to the page listed on the right. You can delete an element in the right-hand pane by selecting it and choosing the Remove button. You can also set the view size for a view element and how the label on the element will look using the View Size and Labels boxes at the bottom of the Class Importer window.




After you add your custom element to a catalog, be sure to use the Save Catalog command, if you haven't already, or your element will be lost from the catalog the next time you quit the builder.

## Adding your interface to the development environment

User-interface projects are saved to project files, which contain the framework resources that are generated from the high-level elements in the user interface. The project file can be added to its corresponding Apple Dylan project, allowing the user-interface elements to be utilized from within your application. A project file from the interface builder which is added to a development environment project is treated as any other resource file and has the resource file icon placed to the left of the file name. As with other resource files that are added to a Dylan project, the development environment automatically opens any project file that has been added in this manner.

# Interface Builder Tutorial

The easiest way to learn about the interface builder is to run its standalone application from the Finder. The standalone version of the builder creates a project file that you can add to your development environment project. This is a valid container for your user interface elements. However, to see them within your Apple Dylan application, you must connect them to your development environment project and then run your application from there. For more information on that, see the chapter "Using the Interface Builder" on page 43 and the book *Using the Apple Dylan Development Environment*.

The following tutorial briefly introduces the common features and fundamental concepts of the interface builder. This section consists of steps to help you create a new project, then build a window and menu for that project.

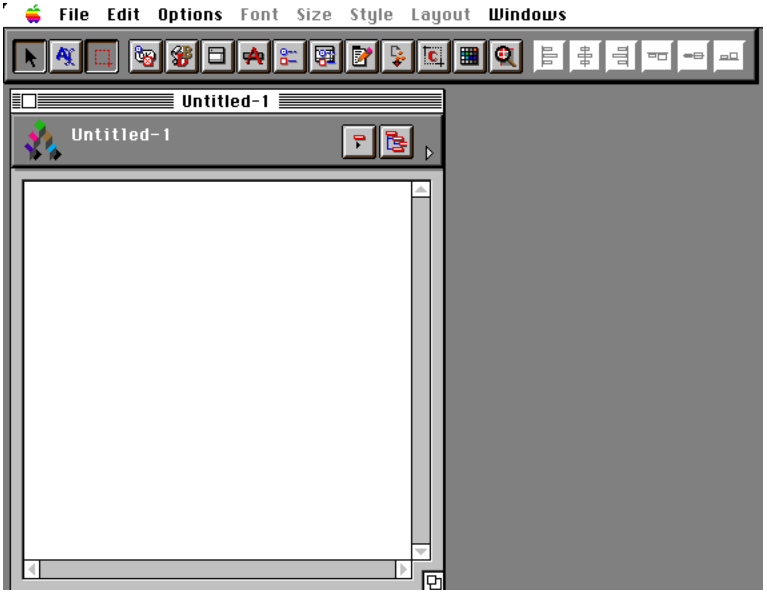
1. **Launch the Apple Dylan interface builder application by double-clicking on its icon in the Finder.**
   You can drag and drop a project file onto the builder to launch it. You can also double-click a project file to launch the builder and open the project.

2. **Select the New Project command from the File menu.**
   A new project window opens and the Command Palette appears below the menubar.

   The Command Palette is used to access the basic editing tools, the alignment commands, and the main windows of the builder. Some of the items are dimmed in the palette because there is currently no selection. The alignment

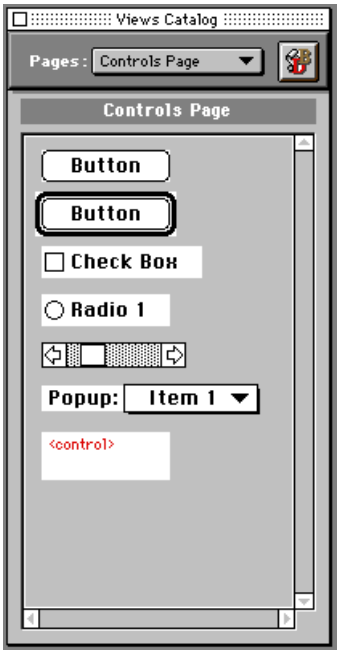
buttons, on the right of the Command Palette, are only enabled when there is more than one selection.

The builder's menubar, the Command Palette, and a new project window, named "Untitled-1" by default, are shown in the following figure. Currently there is nothing in the project window because we have not yet created a high-level element in the new project.




3. **Open the Views Catalog and choose the Save Catalog command from the File menu.**
   You only have to issue the Save Catalog command once. It is best to save the catalogs to a file as soon as you run the builder, because any changes you make to the catalogs are lost if you haven't saved the catalogs when you quit the builder. You can change the default name and location of the catalog, if you want, when the Save Catalog dialog box opens. The saved catalog file name is used in the title of both catalogs.

   The Views Catalog contains all of the view elements that are used to build user interfaces. The Views Catalog is shown in the following figure.




The Views Catalog contains a number of separate pages which are used to hold different categories of view elements. These pages are accessed by using the Pages popup in the catalog's header. There are two pages, the Controls Page and the Views Page, which contain all the views provided by the framework. A third page, the Custom Views Page, is blank so you can add your own elements to it.

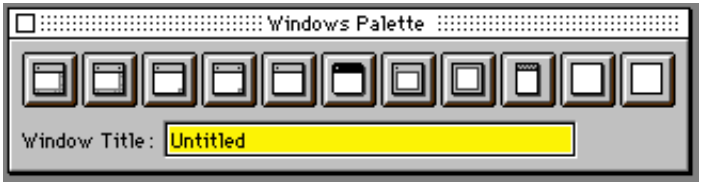
The catalogs are dynamic entities that new elements can be added to. In the Views Catalog, the elements within it can be edited. The elements can have properties edited, as well as their location and size changed. To edit an element, hold down the Control key and select the element.

For the Views Catalog, you can create new catalog pages, add elements to pages, and then save them.

If you create a customized element in a window you are creating, such as an OK button, that you think you want to use repeatedly, you can drag it onto a Views Catalog page wherever there is available space. The page grows dynamically to accommodate the new element, if you have checked Auto Resize Pages on Drag on the Catalog Preferences page.

4. **Open the Windows Palette from the Windows menu.**
   You can press the third button in the center group of buttons on the Command Palette. The Windows Palette contains a set of predefined windows that can be created by clicking one of its buttons.

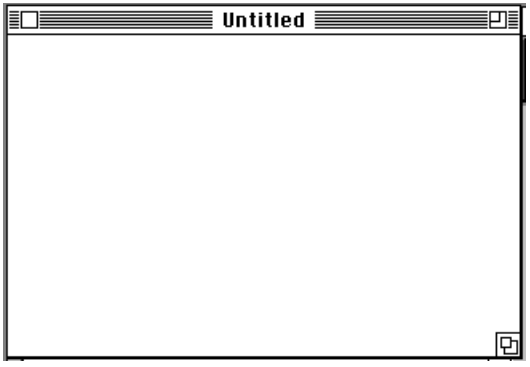
   The following figure shows the default Windows Palette.




   The modal windows created from this palette cannot have their size and location manipulated using the mouse. They can, however, be moved by pressing down the Control and Option keys as you drag the window. These windows sometimes also behave a little differently in the presence of the builder's floating windows.

   Additionally, if a floating window is built, it is important when the window is reconstituted from the resource to make sure that the floats? property is correctly set to true.

5. **Click the fourth button from the left on the Windows Palette.**
   This opens a new resizeable document window with no scroll bars, as shown in the following figure.




   When building a user interface, you are working with real windows, as opposed to sketching the contents of the window within another window and then viewing the window as a separate step.

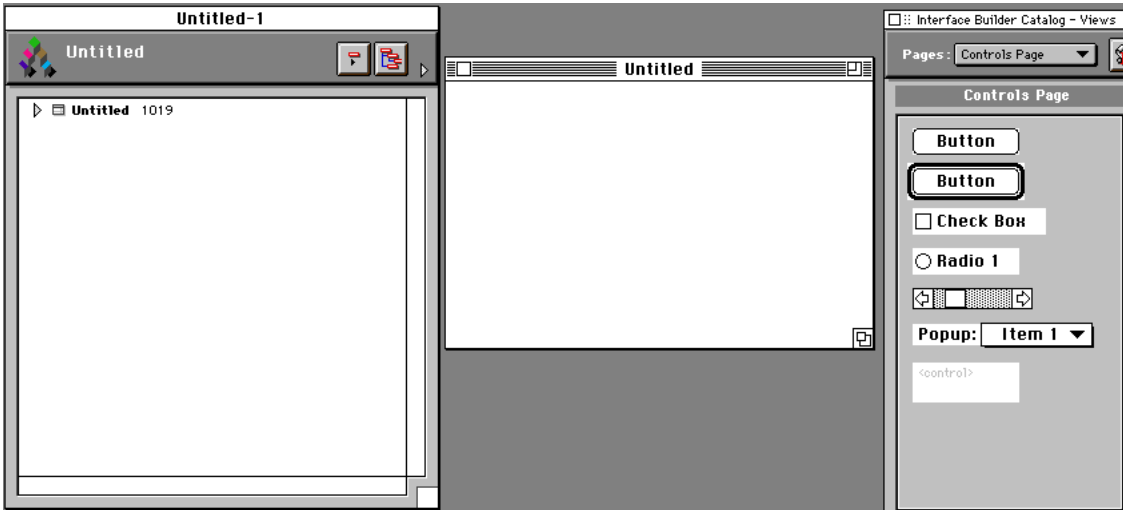
The Windows Palette provides a title input field, Window Title, so you can designate a name for the window. For purposes of our tutorial, we will not enter a title at this time. The position and size of windows created from the Windows Palette can be controlled from the Preferences dialog which provides fields for setting these default values.

At this point the Windows Palette can be closed as we now have a window to work with. If you create a window and then want to delete it without saving it, select the window's node in the project window and choose the Delete Window command from the Options menu. This deletion is not undoable.

6. **Drag the newly created window off the project window and off the Views Catalog so you can see all three.**
   The act of creating the new window causes a new node to be added at the root level of the project window. The node is named "Untitled" with a number next to the title. The name of the node is the new window's title and the number is the resource ID for the new window. This resource ID is used when the window is saved (when the project is saved) to create a unique identifier for its resource. It is also used to load the window when the end user uses it in your standalone application. The builder automatically generates a resource ID for the window starting at the current value specified on the General Preferences page, so use the Preferences command to customize the resource IDs, if you want.

   The following figure shows the project window on the left, the new window in the middle, and the Views Catalog on the right.

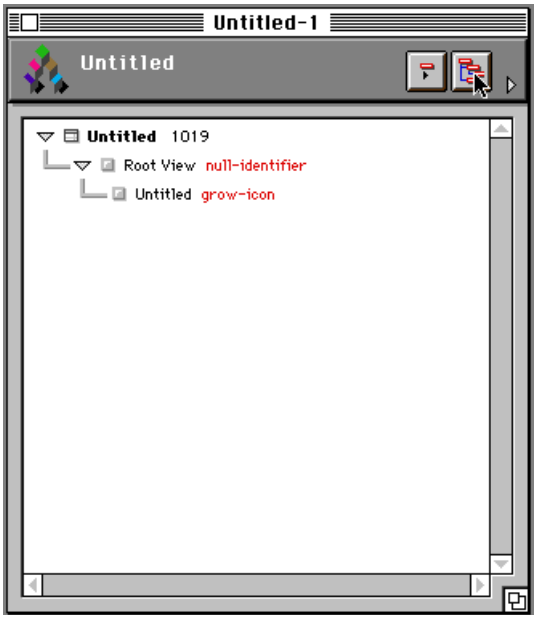
   In the project window, the node for the new window has a disclosure triangle to its left. The disclosure triangle is used to expand and collapse this branch of the project tree. Only windows, not menus, have disclosure triangles; menu nodes cannot be expanded in the project window. The node can also be expanded or collapsed by clicking one of the two buttons on the right of the project window's header. The two buttons on the project window's header expand or collapse all window nodes, not only a selected node. Next on the node's line in the project window is an identifying icon. The icon in this figure is a small window, indicating that this is a window node. Next is the actual name of the node, followed by the resource ID for this window.




7. **Click the window node and press the Expand All button in the project window's header, which is the button to the far right.**
   This results in all window nodes expanding to reveal all their elements.

   The following figure shows the project window expanded. It shows the Root View and a grow icon, which both have the generic view icon.

   The root view is needed because the Apple Dylan framework is different from most frameworks; the window class is not a sub-class of a view. This means that there needs to be a generic view within the window that is used to embed views contained in the window. This is the root view's role. It is automatically created by the framework and is setup to always match the size of the window, such that it occupies the entire content area of the window.




8. **Click the disclosure triangle in the bottom, right-hand corner of the window's header.**
   This opens the project window's data input panel, which is used to change the data displayed by a node in the tree, and thereby, change the data for the selected element the node corresponds to. The data input panel provides fields for the title, identifier, and resource ID, and checkboxes for the Enabled, Visible, and Locked flags of an element, which are some of the most commonly used properties of elements. For complete access to an element's properties, use the Property Inspector as described later.

**Note**

Depending on the type of element selected, some items on the date input panel might be disabled.

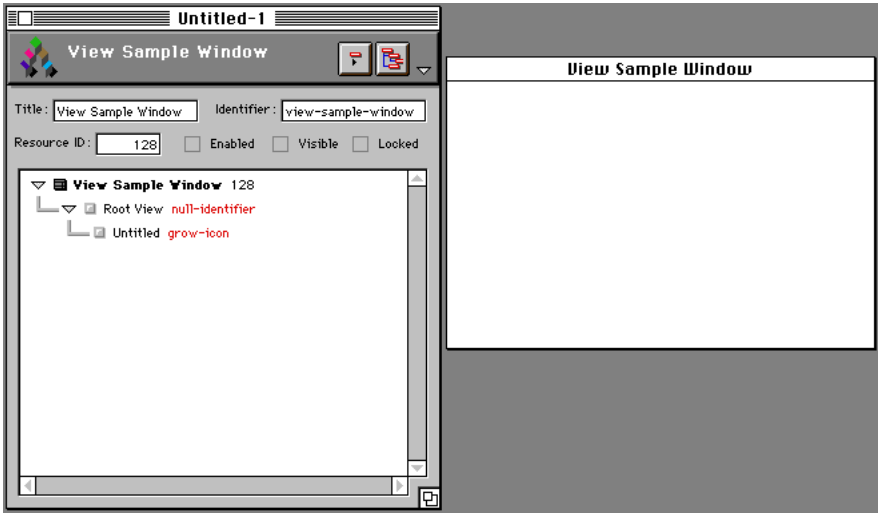
All nodes in the project tree can have titles and in some cases these titles are the names of the actual elements, such as a button or window. However, in many cases they simply serve as descriptors, since some elements do not have visible titles.

9. **Click the window node in the project tree.**
   This causes the data for the corresponding new window to be displayed in the data input panel.

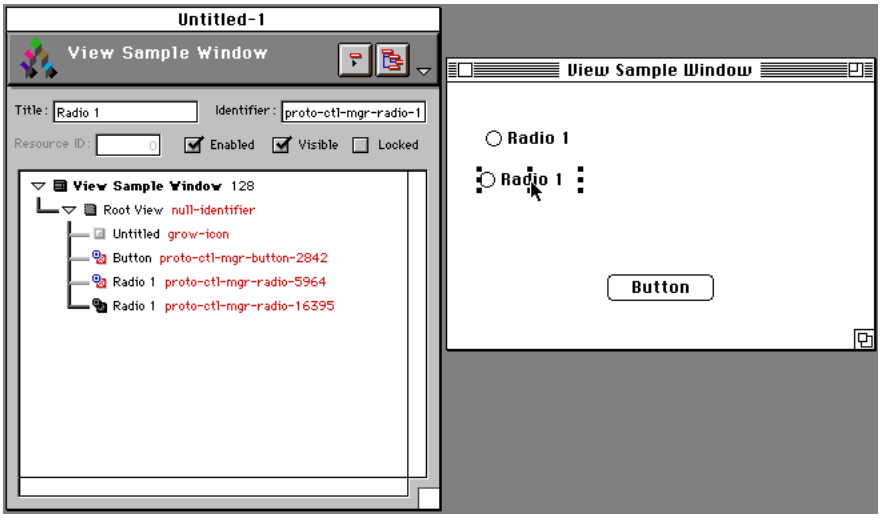
   Enter a name for the new window by double-clicking in the Title field of the data entry panel and typing in the name. Enter the title “View Sample Window”.

   Press the tab key to advance the cursor to the Identifier field and enter the identifier “view-sample-window”. Finally, advance to the Resource ID field and enter the resource ID “128”. Press the Enter key to have this data applied to both the node and the actual window.

   This results in the window node changing to show the newly entered name and the resource ID. The window titles in both the project window and in the new window also change to show the new title.




10. **Drag elements, such as a button and two radio buttons, from the Views Catalog into the window View Sample Window.**
    Notice that, as elements are dragged from the catalog and dropped into the window, a corresponding new node is created and added to the project tree.



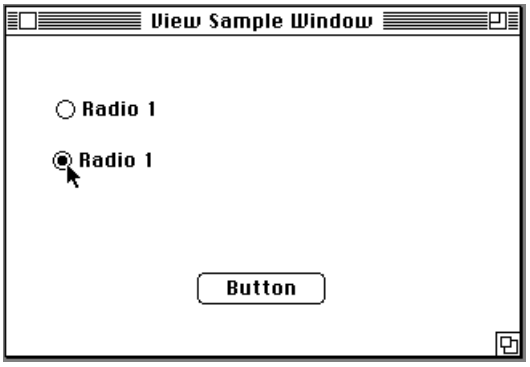

When an element is selected in a window its corresponding node is also selected in the project window, if the window node is expanded.

A selected node in the project window has its icon darkened and a selection path is shown back from the node to its parent window. This provides a clear indication of a node's embedding hierarchy.

Selecting a node in the project window results in the corresponding element being selected in the new window.

The duplication between the window and the project tree is particularly useful for elements that you might wish to hide in the layout until some operation results in them becoming visible. When working in a layout the minute an element in made invisible, there will be no way of making it visible as it is not enabled and there is nothing to click on. In the project window all elements are visible whether they are visible or not in the new window. The same applies to the enabling and disabling of elements. This can all be controlled from either the checkboxes in the data input panel or from the Property Inspector.

11. **Select the Edit Front Window command from the Options menu.**
So far you have been in edit mode so a click on an element results in it becoming selected so you can alter it. When you click Edit Front Window so that it becomes unchecked, it puts the window View Sample Window into run mode. The run mode allows you to experiment with a new window as it would act when in use in an application.

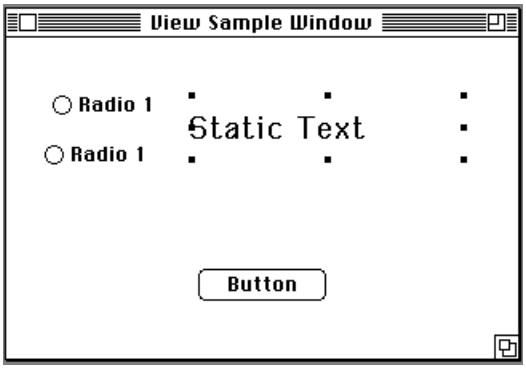



Floating windows and modal dialogs do not behave modally nor float while being edited. However, they do take on the correct behavior when opened by the end user.

12. **Switch back to edit mode by selecting the Edit Front Window command again to unselect it.**

13. **Drag out a static text element from the Views Page of the Views Catalog.** Make sure that it is selected as we are now going to change some of its properties using some of the other features of the builder. Resize the element by dragging on one of the handles on the edges or side of the element, the goal is to make the element about twice the size of the text shown in the element.

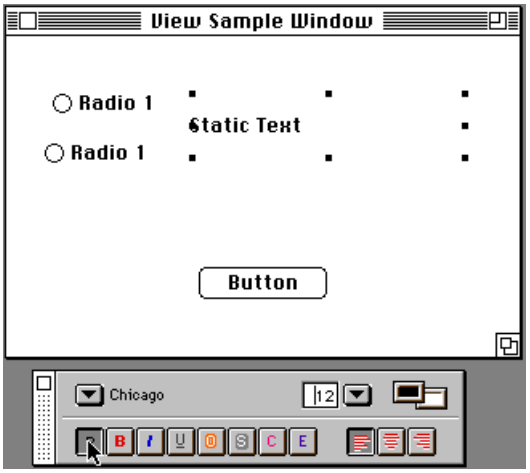
    Now, continue to change the static text element, which you can do with the Text, Size, and Style menus whenever the window being edited is the frontmost window. From the Font menu change its font from Chicago to Geneva, from the Size menu change its size from 12 point to 18 point, and from the Style menu change its style to bold. All of these operations can also be performed using the Text Palette.




Text styling is applied to all elements in the current selection. All framework views can have a text style associated with them, but many of them do not make use of the text styles.

Experiment with the Text Palette by opening it from the Windows menu and using it to change the text style properties of the static text element back to the original values, Chicago, 12 point, Plain. The font popup is used to select a font. The size control allows you to select a size by typing it directly into the field (press the Enter key for it to take effect) or to select the size from the popup. The set of style buttons on the Text Palette that are additive (with the exception of the first one which is the plain style), in that the characteristics of each is added to any others clicked. The group of buttons on the right of the Text Palette shows the current justification setting for the element. Finally, the foreground and background colors for the element are shown in the color wells at the right of the palette.

The following figure shows the static text field returned to its original font and the Text Palette.




You can duplicate elements in a window, such as a radio button or the static text element, by pressing the Option key while dragging one of the element's handles. You will see the number of elements you are creating as you drag the handle; the farther you drag, the more duplicates you create.

14. **Open the Colors Palette from the Windows menu.**
You could also select the Colors Palette from the Command Palette.

    The Colors Palette is used to set the foreground and background colors for an element. Elements use these properties in different ways. In the case of the static text field the foreground color is used to draw the text and the background color is used to fill the background of the entire element.

    Colors can be applied to an element either by clicking on the color in the Colors Palette or by dragging the color from the palette and dropping it on the element.

    Clicking on a color will result in it being applied to the foreground color for all of the currently selected elements. An Option-click results in the color being applied to the background color of all the elements in the current selection. Dragging and dropping of colors only affects the color of the element on which it is dropped, the element does not need to be selected. An Option-drag of a color results in the background color being changed.

    To set its background color to gray, select the static text element and Option-click on a gray in the Colors Palette.

    Notice that as colors are changed, the color wells on the Text Palette change as well. The colors for the current selection can be changed directly from the

color wells, either by double-clicking on them to display the standard system color picker, or by dragging and dropping a color onto either one of the wells. The color changes are applied to all of the currently selected elements.

Colors can also be dragged from a color well and dropped on an element which will result in the color for that element changing to match the color from the color well.

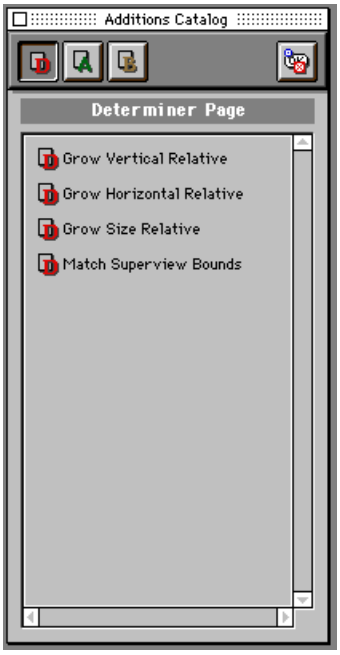
15. **Choose the Color Inspector from the Windows menu.**
    The Color Inspector can be used to examine the color values of the pixels under the cursor. In addition, it also shows the current global coordinates of the mouse.

**Note**
The Color Inspector does not allow colors within four pixels of the edge of the screen to be inspected.

16. **Open the Additions Catalog by clicking the button on the right of the Views Catalog header.**
    You can also open the Additions Catalog from the Windows menu or the Command Palette.

    The Additions Catalog contains items that can be added to view elements within a layout. These items are determiners, adorners, and behaviors. Three buttons are provided in the catalog's header for accessing the page for each of these items. Determiners are used to constrain elements in relation to their superview. Adorners are used to alter the appearance of elements. Behaviors are used to alter the behavior of elements dynamically.

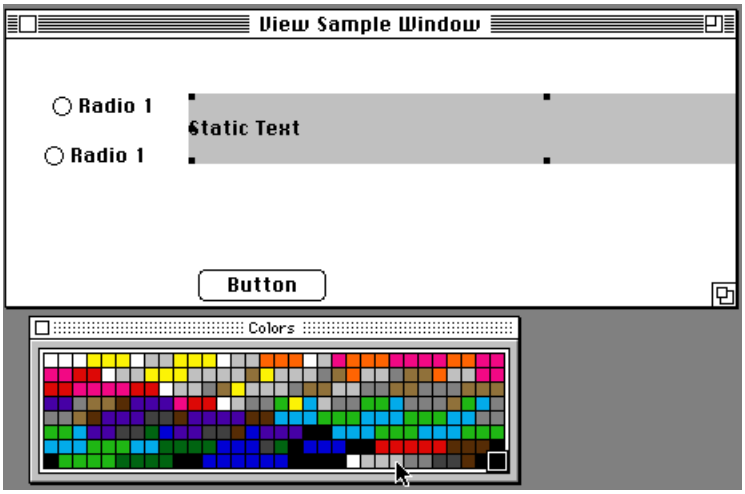
    The following figure shows the Determiner Page of the Additions catalog.




Drag the Additions Catalog off the window View Sample Window.

17. **Drag "Grow Horizontal Relative" from the Determiner page of the Additions Catalog onto the static text element.**
Once this determiner has been applied to the static text field, resize the window and the text field will grow horizontally. This provides an example of how determiners can be used to alter the behavior of elements when there superview undergoes a change in size.

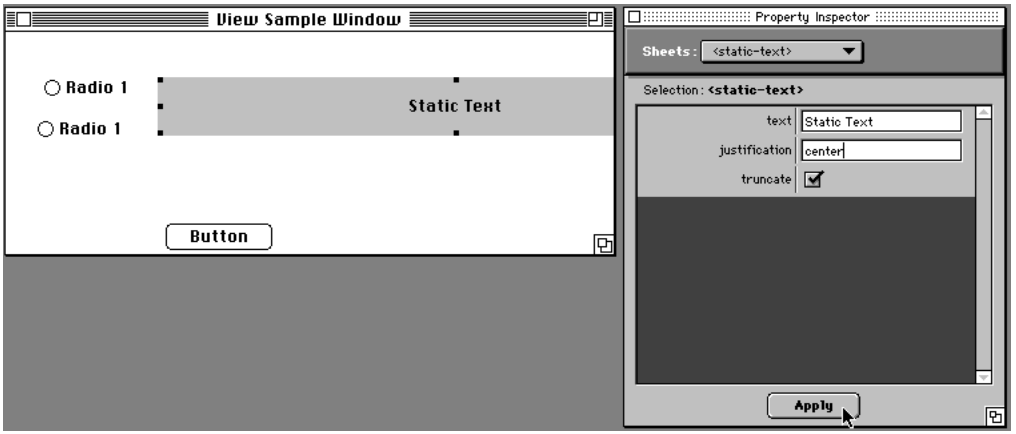
    The following figure shows the result of dropping Grow Horizontal Relative onto the static text field and dragging the lower-right corner of the window to widen it.




18. **Double-click the static text element to open the Property Inspector.**
    To open the Property Inspector you can also choose it from the Windows menu or choose it from the Command Palette.

    The Property Inspector is used to view and edit all the properties for a selected element. The inspector displays these properties on property sheets, each corresponding to the superclass of the selected element. Note that any deletions you make using the Property Inspector's Delete key are not undoable.

    The following figure shows the <static-text> sheet of the Property Inspector. You can edit the justification field to "center", as shown in the following figure. Choose Apply to make the edit take effect.

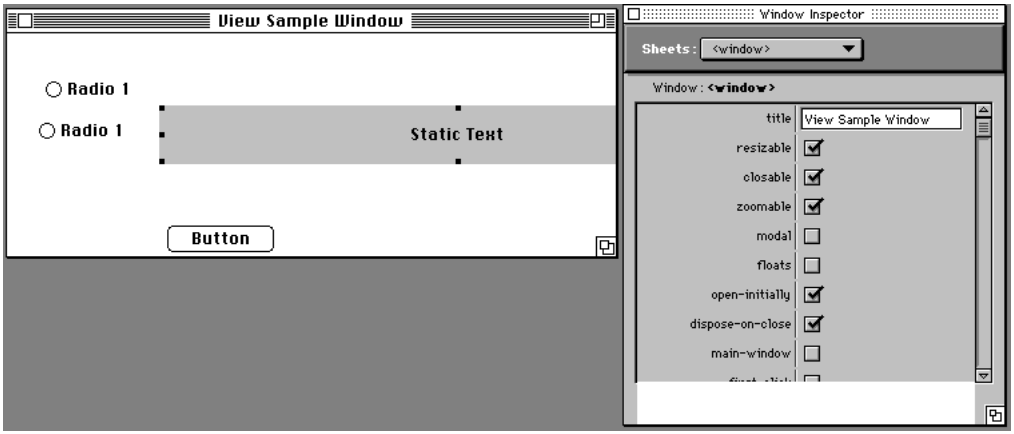



These property sheets are constructed dynamically with separate editors for each property type. This mechanism allows the builder to display properties for any element without the need for hard-coded property sheets. The information used to construct these property editors is provided by the meta information needed by the framework to support the streaming of views.

Open the Frame property sheet using the Sheets popup in the Property Inspector's header. Now enter a new value for one of the location properties and press the Apply button, which will result in the element moving in the layout.

Open the Window Inspector by selecting an element in the window View Sample Window and choosing Window Inspector from the Windows menu. The Window Inspector displays the properties for the window containing the selected element, View Sample Window in this case. The Window Inspector works much like the Property Inspector, but it displays a different set of property sheets.

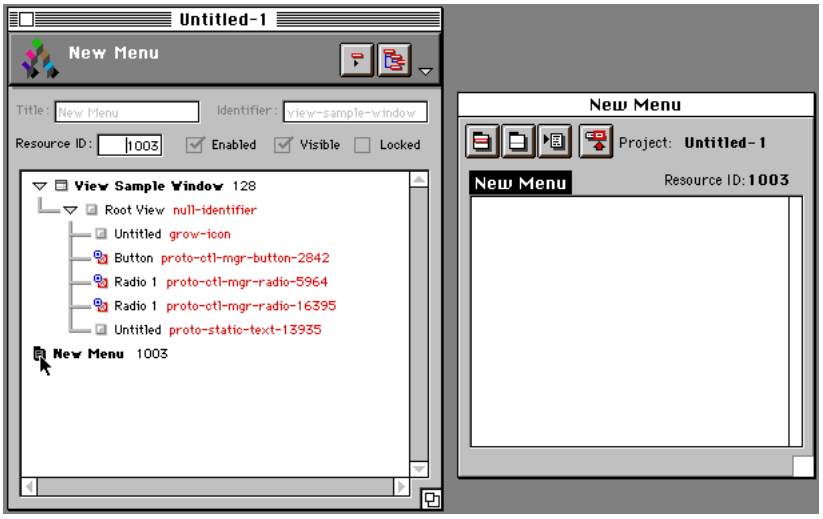
The following figure shows the result of changing the location of the static text field, selecting the static text field, then opening the Window Inspector.



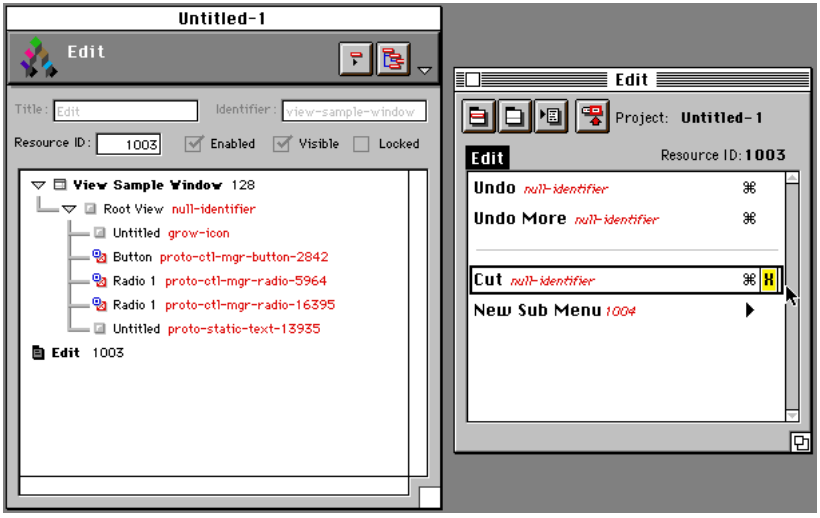

If either inspector window is open and all selections are turned off, the contents of the window are cleared and the popup and button are dimmed.

The width of both inspector windows is determined by the builder based on the type of property sheet being displayed.

19. **Close all the windows except the project window and choose Menu Editor from the Windows menu.**
A New Menu Editor window, called "New Menu", opens. Notice that a new node is created in the project window for the new menu and that there is no disclosure triangle for the new node.

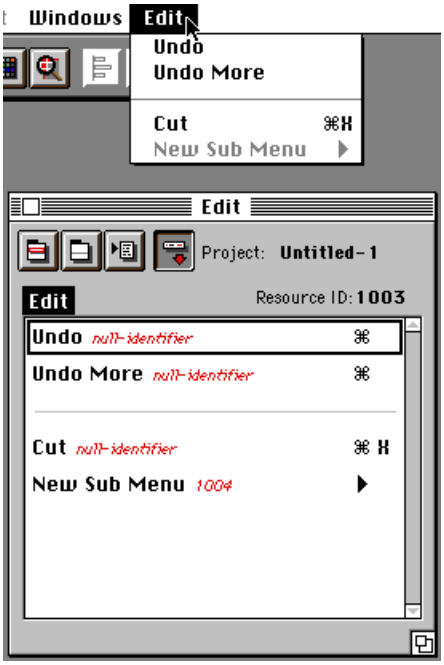



You can edit the name of the new menu by clicking in the New Menu text field, which is below the four buttons in the window's header. The new name appears in that field and also replaces the title of the New Menu window. It also replaces the title New Menu in the project window node that represents the menu and in the header of the project window, as shown in the following figure.




You can then use the four buttons in the Edit window to create menu items and separator lines, as well as preview the new menu on the builder's menubar.

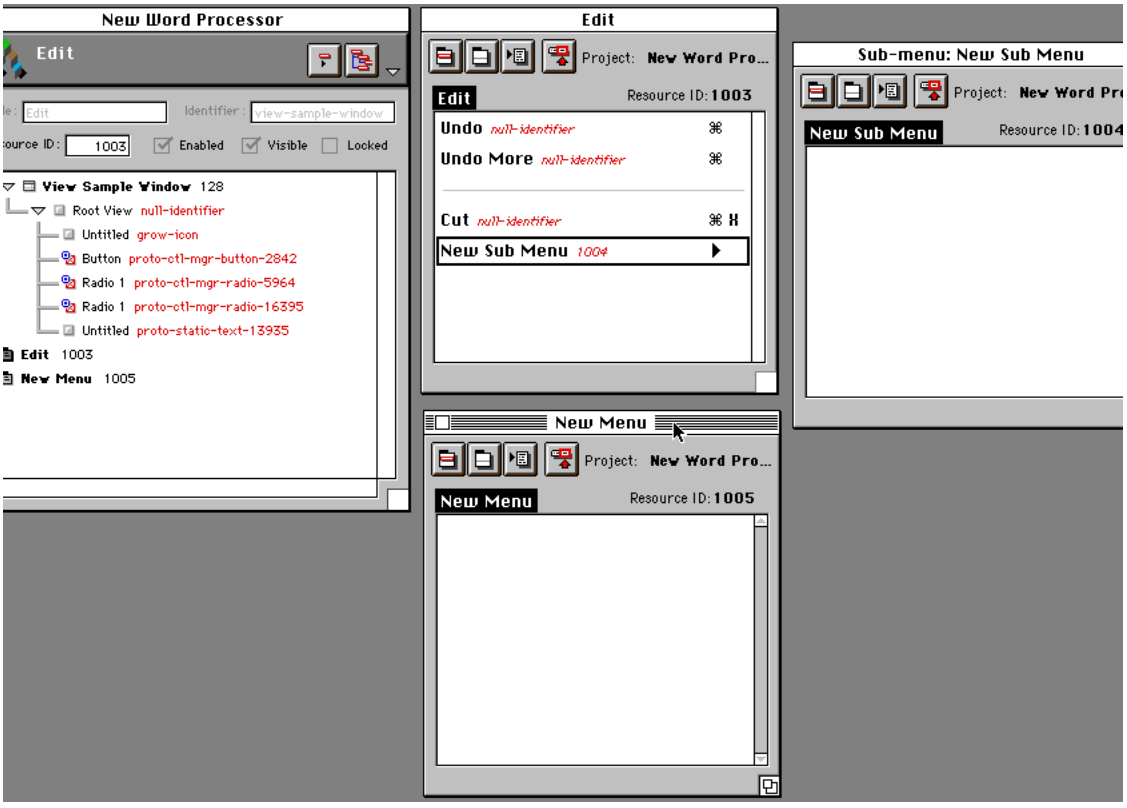
The following figure shows four new menu items on the new Edit menu, Undo, Undo More, Cut, and New Sub Menu. The Cut menu item has had a short-cut key added. To create the submenu, double-click in the New Sub Menu menu item. The Edit menu is shown at the end of the builder's menubar with all its menu items in place. You create this preview of your menu by clicking the fourth button and then opening the new menu on the menubar. If you had created a submenu, it would appear on the new Edit menu as well.




To create additional menus, select a window node in the project window and choose Menu Editor from the Windows menu. If you create a menu and then want to delete it without saving it, select the menu's node in the project window and press the Delete key. This deletion is not undoable.

The following figure shows the result of creating a submenu for the new Edit menu, and then creating a new, separate menu. The new, separate menu will be constructed in the New Menu window, which appears at the bottom of the figure. Notice that a new node has been created for it in the project window.

The resource IDs on the submenu appear both inline in the Edit menu and on the header of the Sub-menu: New Sub Menu window. If you edit the resource ID in either place, the other changes immediately, as does the node in the project window. The builder automatically assigns resource IDs to menus and submenus starting from the current menu counter value on the General Preferences page.




**20. Choose Save As from the File menu.**

If you want to save the windows and menus you created using this tutorial, you can use the Save or Save As commands. When a project is saved, the entire project is saved to disk, including the actual window and menu resources, as well as the size and location of the project window. Menu resources are saved as “dmnu”s.

You can always tell which windows or menus have not been saved by checking for italics in the project window. When a window or menu is edited, it is marked by the builder as unsaved and its node in the project window is written in italics, instead of regular font. The window or menu is cached when it is closed, so it is not lost and will be saved whenever needed, whether it’s open or not.

Notice that the saved project file in the Finder does not have the suffix ".rsrc" as resource files do. However, you could open the project file with a resource editor application, if you choose, and copy all the resources in the project file to it. If you do, the development environment will not recognize that resource file as an interface builder project file, but as a regular resource file.

Saving a project does not close the interface builder. You close the interface builder by choosing quit or by choosing Hide Interface Builder from the Apple menu.

See the chapter "Using the Interface Builder" on page 43 for more information on incorporating into your development environment project the windows and menus you have created in the interface builder.

CHAPTER 2

# Using the Interface Builder

The following sections describe certain additional features of the interface builder, including tasks on how to perform some optional tasks.

After you have run the standalone version of the interface builder and learned the basics of how it works, as described in the first chapter, there are other capabilities you need to know about. You need to run the interface builder from within the development environment project you want to design a user interface for. Before you do that, you will need to have written enough code for your project to run.

You might also want to write custom user interface elements in the development environment and then add them to the builder's catalog. That task is detailed in the section "Adding custom elements to the catalogs" on page 45.

## Using the builder from the development environment

When you want to use the interface builder from the development environment in order to do real development on your application, add the interface builder to your development environment project. You add the builder to a development environment project by running your application within the development environment and then using the Load UI Builder command. You can then run the builder by choosing the Show Interface Builder command from the Apple menu.

Remember, to use the interface builder from inside the development environment you must first add the framework to your project. You add the Apple Dylan framework to your project in the development environment as you would add any subproject, using the Add to Project command from the

File menu. See the book *Using the Apple Dylan Development Environment* for more information on adding files to a project.

You will probably want to add the Macintosh toolbox to your project as well. See the book *Using the Apple Dylan Development Environment* for more information on adding libraries to a project.

When your user interface is completed in the builder, you add to your development project the project file generated by the interface builder.

The following sections show you how to:

- run the builder from within the development environment
- add custom elements to the builder's catalogs
- make these elements available to your application

## Running the builder within the development environment

Once you have added the framework to your development environment project, have created a basic application and run it, you can run the interface builder from within the development environment.

1. **Add the framework to your project.**
2. **Run your application.**
3. **Choose Load UI Builder from the Project menu.**
4. **Go to the Apple menu and choose Show Interface Builder.**
   You should make sure your application runs before you run the interface builder. The application's startup function must be specified using the Set Project Type command before you can use the Run command.
5. **Open the existing builder project you want to work on or create a new one.**
6. **To return to your running application without quitting the interface builder, choose Hide Interface Builder from the Apple menu.**
   To quit the builder, choose the Quit command from the builder's File menu. This also quits the Application Nub and your application, returning you to the development environment.

## Adding custom elements to the catalogs

If you choose to, you can write your own user interface elements using the framework, and then add them to the builder's catalogs.

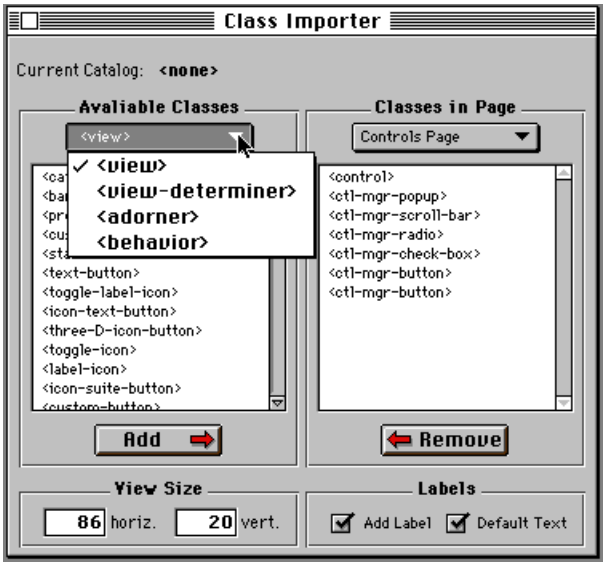
You create your own types of custom elements by subclassing any of the framework classes <view>, <adorner>, <behavior>, or <view-determiner>. See the books *Programming in Apple Dylan* and *Apple Dylan Extensions and Framework Reference* for more information on the framework.

Once you have created your custom elements using the development environment, you use the Class Importer in the builder to add these elements to a catalog page.

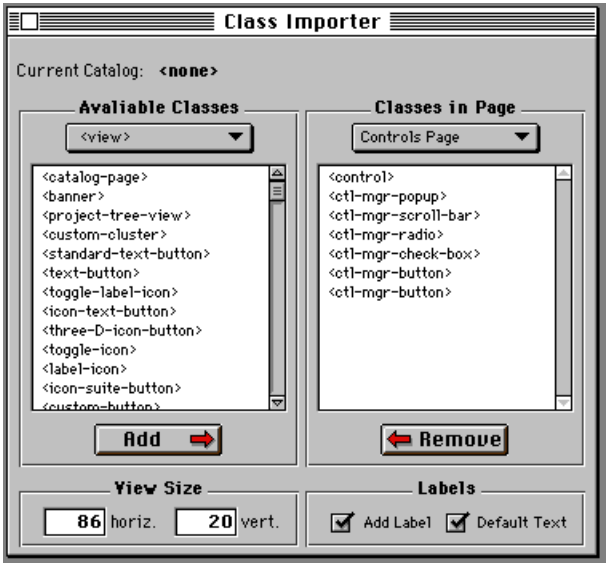
Be sure to use the Save Catalog command, if you haven't already, after adding an element to a catalog. You should also make sure Auto Resize Pages on Drag is checked on the Catalog Preferences page.

1. **Choose from the Available Classes popup the class you subclassed from when you created your element.**
   The classes listed in the Available Classes popup are the ultimate parents of the classes you subclassed, not necessarily their direct parents. For instance, you could subclass a subclass of <view>, such as <text-button>, but you would still choose <view> from the Available Classes popup.

   The following figure shows the Class Importer with the list of classes displayed that you might have subclassed from. When you have selected one, its subclasses are listed in the pane below the Available Classes popup.

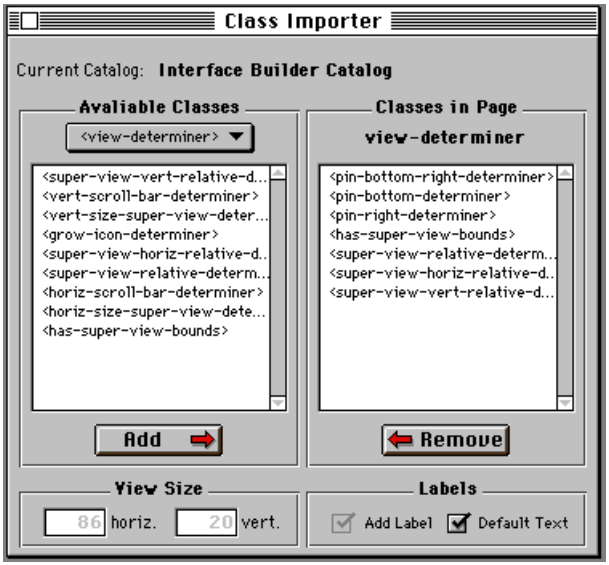



The following figure shows the list of classes for the <view> class. You might have to scroll to find your subclass.



2. **Choose the page you want the element to be on from the Classes in Page popup on the right.**
   If you choose one of the classes whose elements appear in the Additions Catalog, the page in the right-hand pane is chosen for you. This is because, for example, subclasses of <view-determiner> must go on the Determiner page of the Additions Catalog.

   As shown in the following figure, a subclass chosen in the left pane will go onto the view-determiner page, which is the Determiner page of the Additions Catalog.




3. **Select the subclass in the left-hand pane and click the Add button below it.**

4. **Set the view size of a view element, if you wish, using the View Size boxes.**
   You can alter the size of view elements as they will appear in the catalog using the View Size boxes.

5. **Set the way the element's label will appear in the catalog by checking the Labels boxes at the bottom of the window.**
   Labels can appear on any view element. The default for Labels is that both are checked. The default text used is the name of the element's class.

6. **If you want to remove an element from a catalog, select it from the right-hand list and choose Remove.**
   This deletes the element from the list in the right-hand pane and also from the catalog page.

**Note**
This is how all elements can be deleted from catalog pages.
This deletion is not undoable.

## Making interface elements available to your application

Once you have created the user interface and the classes that the user interface elements represent, you must write the framework code needed to load these elements from resources. You need to add the high-level elements (such as windows and menus) that you created into your existing development environment project.

The method in the following example shows how to access one of the window resources from the builder project file. Window resources have an OSType of "dwin", while views have a resource type of "dvew". This function should be written with some failure handling if the developer wishes to catch any errors; the framework will catch an error getting the resource or when the call to fail-nil() is made.

The Dylan framework provides a number of methods and macros for accessing resources from resource files that have been added to projects. The macros get-resource, get-string, get-indexed-string, etc., can be used to access resources from a file associated with a library. See the book *Apple Dylan Extensions and Framework Reference* for more details.

### Loading a window

The following steps show you how to create the code that loads a window from the builder's project file.

1. **Make active the Apple Dylan project to which the user interface elements will be added.**
   Be sure you have added the framework and Macintosh toolbox to this project.

2. **Choose Launch Application Nub, if you are not connected already.**
   You must load the builder into this project using the Load UI Builder command and then add the builder's project file to it as well.
3. **Create a source record and enter the following code.**
   Once the call to this method has been made, the window can be opened, or, if it is a modal dialog, the dialog can be posed modally.

```
define method new-template-window-from-file ( file :: <file>,
                    resource-id :: <integer> )
                    => new-window :: <window>;

    let resHandle =  get-resource-from-file
              ( file, ostype ( "dwin" ), resource-id );
    let stream =
        make ( <handle-object-stream>, handle: resHandle );
    let window = read-object ( stream );

    // • Return the new window if we were able to get it
    window;
end method
```

4. **Call the method wherever you need to load a user interface element.**
   For example, to load a window, you could use code similar to the following:

```
let window = #f;
// • Make sure that the resource fork is open
with-open-resource-fork ( file )
window := new-template-window-from-file ( file, window-res-id );
end;
```

**Note**
The `window-res-id` variable is an `<integer>` object.
The `file` variable is a `<file>` object.

5. **Compile the changes and save them.**
6. **Run your application to see if the user interface element works properly.**

## Loading menus

Once you have created the menus you want to have in your application using the interface builder, you can call the framework function `get-menu-from-resource` in the order you want the menus to appear on your menubar. You can use the GetResource call to the toolbox to get the menu resources, which are saved as "dmnu"s. You must load each menu in the order you want it to appear in your menubar without regard for the order of its resource ID number.

You can get and install your menus on your menubar using the following line of code for each menu.

```
install-menu(get-resource-menu(menu-id));
```

The variable "menu-id" is the resource ID you assigned to the menu in the interface builder.

# Palette Buttons

This appendix includes descriptions of all the buttons on the Command Palette and Text Palette, as well as an example of each of the kinds of windows you can create using the Windows Palette.

## Command Palette

The buttons on the Command Palette represent commonly used commands.


**Command Palette buttons**

| | | |
|---|---|---|
|  | 1— Selection Tool | Activates the tool for selecting a user-interface element in a window. The Selection Tool is the default tool and is active unless another tool is chosen. |
|  | 2— Text Tool | Activates the tool for editing the name of a user-interface element in a window. You choose the Text Tool command and then select the text to be edited. |
|  | 3— Marquee Selection | Activates a selection tool for drawing a rectangle around user-interface elements so they can be acted on as a unit, such as dragging them to another location or aligning them. |

*continued*

**Command Palette buttons (continued)**

| | |
|---|---|
|  4— Views Catalog | Opens the Views Catalog, which contains many user-interface elements, such as buttons, ready to be used in the windows you create. |
|  5— Additions Catalog | Opens the Additions Catalog, which contains items that are used to add characteristics, such as adorners, to interface elements from the Views Catalog. |
|  6— Windows Palette | Opens the Windows Palette, which you use to create new windows. |
|  7— Text Palette | Opens the Text Palette, which is used to modify the text style of the current selections. |
|  8— Property Inspector | Opens the Property Inspector, where you can edit common properties of interface elements. |
|  9— Window Inspector | Opens the Window Inspector, where you can edit common properties of windows. |
|  10— Menu Editor | Opens a Menu Editor, which allows you to create or modify your menus. If a menu is selected in the project, this button opens an editor for that menu, otherwise it creates a new menu. |
|  11— Class Importer | Opens the Class Importer, which is used to import user-defined classes into the catalogs. |
|  12— Coordinates palette | Opens the Coordinates palette, which you can use to see or set the location and size of an interface element as you are manipulating it. |

*continued*

**Command Palette buttons (continued)**

| | |
|---|---|
|  13— Colors Palette | Opens the Colors Palette, where you can change an element's foreground color by dragging a color from it onto the element. To change an element's background color, hold down the option key as you drag the color. Clicking on a color works the same way. |
|  14— Color Inspector | Opens the Color Inspector, which you can use to examine individual pixels near the cursor. |
|  15— Align Left Edges | Aligns the left edges of several selected user-interface elements in an active window. |
|  16— Align Centers Vertically | Aligns on a vertical line the centers of several selected user-interface elements in an active window. |
|  17— Align Right Edges | Aligns the right edges of several selected user-interface elements in an active window. |
|  18— Align Top Edges | Aligns the top edges of several selected user-interface elements in an active window. |
|  19— Align Centers Horizontally | Aligns on a horizontal line the centers of several selected user-interface elements in an active window. |
|  20— Align Bottom Edges | Aligns the bottom edges of several selected user-interface elements in an active window. |

# Text Palette

The buttons on the Text Palette represent commonly used text manipulation commands from the Font, Size, and Style menus, as well as the color wells.


**Text Palette buttons**

| | | |
|---|---|---|
|  | 1, top row— Font | Opens a popup list of fonts you can choose from. |
|  | 2, top row— Size | Opens a popup list of font sizes you can choose from or you can enter a font size in the field. |
|  | 3, top row— Color wells | Drop boxes where you can drop colors. The top box is for foreground colors and the bottom box is for background colors. The colors in these wells are also set if you click on a color in the Colors Palette. |
|  | 1, bottom row— Plain text | Changes the characters in selected text to the specified font and size with no other embellishments. Changes the style of the selected element to plain resets all previous styles. |
|  | 2, bottom row— Bold text | Makes the characters in selected text bold. Changes the style of the element's text to bold. |

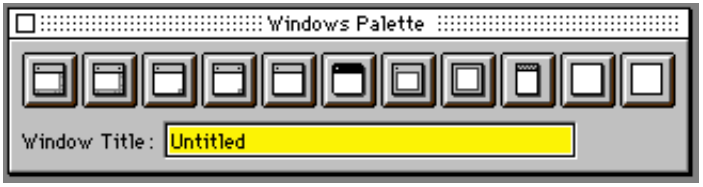
*continued*

**Text Palette buttons (continued)**

| | | |
|---|---|---|
|  | 3, bottom row— Italic text | Makes the characters in selected text italics. |
|  | 4, bottom row— Underlined text | Underlines the selected text. |
|  | 5, bottom row— Outlined characters | Outlines the characters in selected text. |
|  | 6, bottom row— Shadowed characters | Makes the characters in selected text appear shadowed. |
|  | 7, bottom row— Condensed text | Compresses the selected text so it fits into a smaller horizontal space. |
|  | 8, bottom row— Extended text | Extends the selected text so it fits into a larger horizontal space. |
|  | 9, bottom row— Left Justify | Justifies the selected text so it is as far to the left as it can be within its user-interface element. |
|  | 10, bottom row— Center Justify | Justifies the selected text so it is centered within its user-interface element. |
|  | 11, bottom row— Right Justify | Justifies the selected text so it is as far to the right as it can be within its user-interface element. |

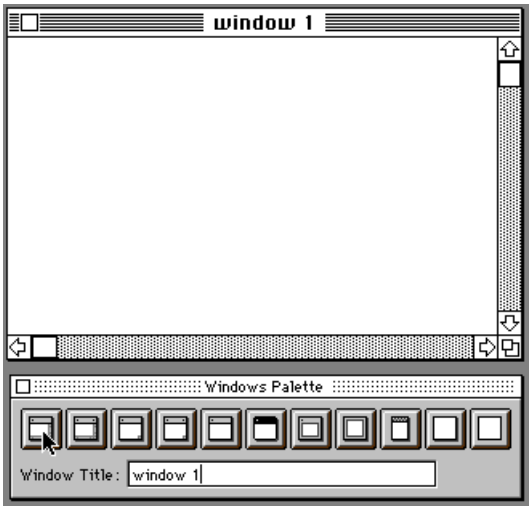
# Windows Palette

The buttons on the Windows Palette represent the pre-defined types of windows you can create using the interface builder. The following figure shows the default Windows Palette.

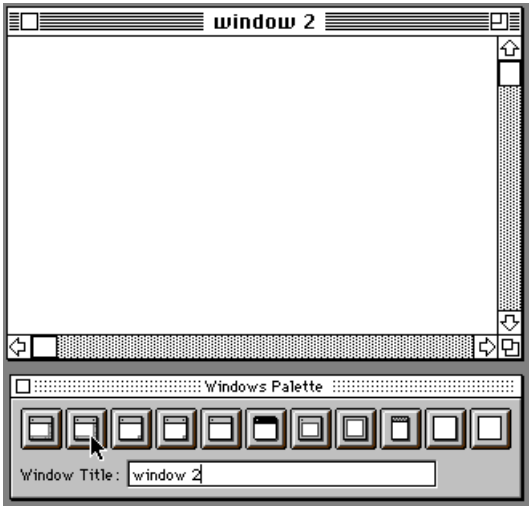



Each of the window buttons corresponds to a different type of window, some with scrollbars, some with close buttons, etc. When you choose the button representing a type of window, the window is created on the desktop and a representation of it appears in the project window. You choose the name for a window, if you want one, by entering it into the Window Title field before clicking on the window button.

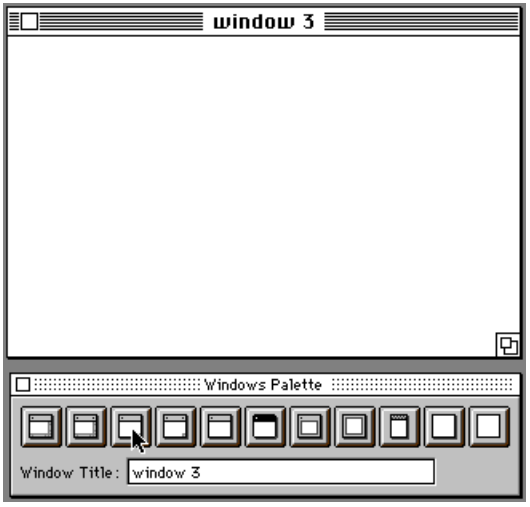
The following figure shows the window created by choosing the first button on the Windows Palette. This window is a standard document window with scrollbars, but has no zoom box. This is known as the “Scrolling Document window - documentProc” constant in the Macintosh toolbox. However, the scrolling capability is added by the Apple Dylan framework and interface builder.



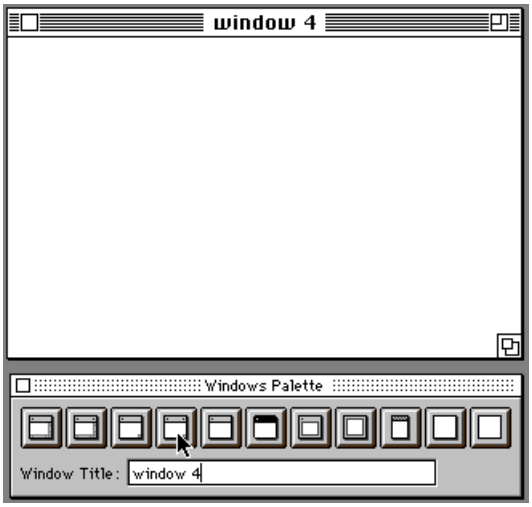

The following figure shows the window created by choosing the second button on the Windows Palette. This window is a standard document window including a zoom box and scrollbars. This is known as the “Scrolling Zooming Document window - zoomDocProc” constant in the Macintosh toolbox.



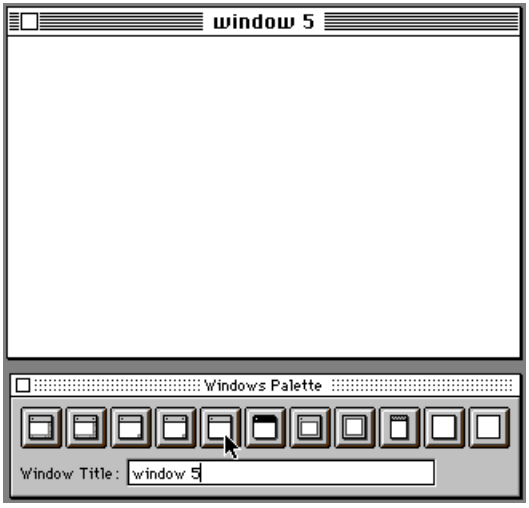
The following figure shows the window created by choosing the third button on the Windows Palette. This is a document window that has no zoom box, scroller or scrollbars, but is resizable. This is known as the “Standard Grow Document window - documentProc” constant in the Macintosh toolbox.



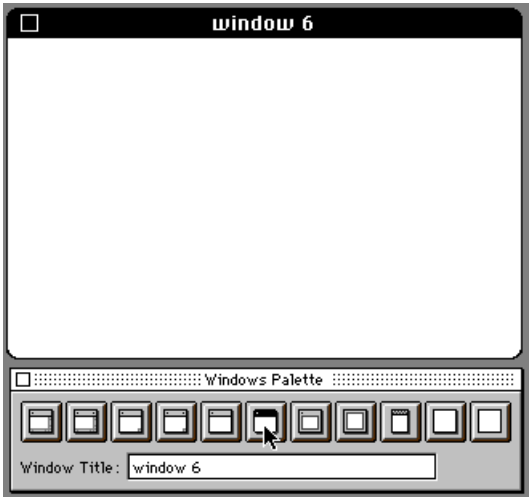

The following figure shows the window created by choosing the fourth button on the Windows Palette. This window is a document window that has no scroller or scrollbar, but has a zoom box and is resizable. This is known as the “Standard Zoom Grow Document window - zoomDocProc” constant in the Macintosh toolbox.



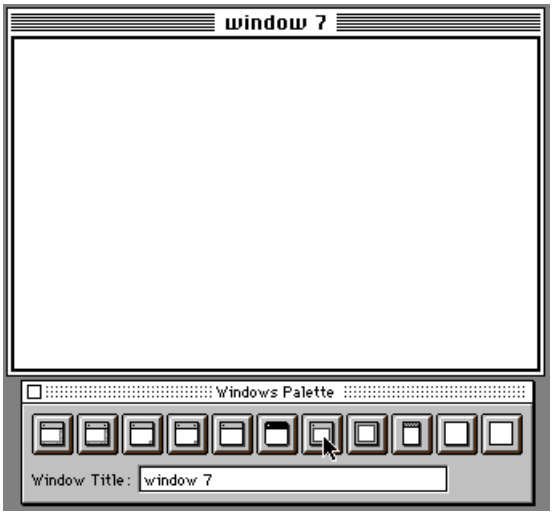

The following figure shows the window created by choosing the fifth button on the Windows Palette. This window is a document window but has not scroller or scrollbar and cannot be zoomed or resized. This is known as the “Standard No Grow Document window - noGrowDocProc” constant in the Macintosh toolbox.



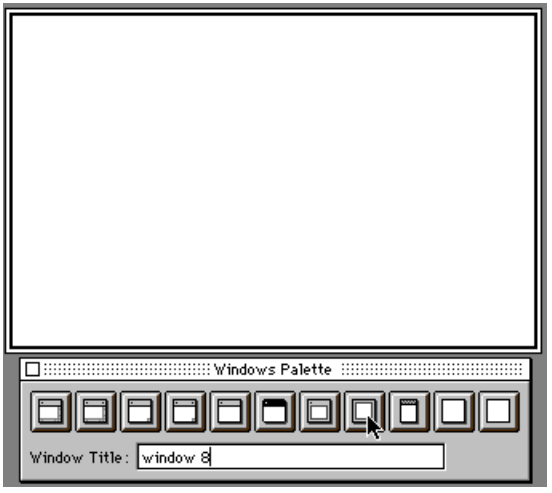
The following figure shows the window created by choosing the sixth button on the Windows Palette. This window is for a desk accessory. This is known as the “Rounded Corner window - rDocProc” constant in the Macintosh toolbox.



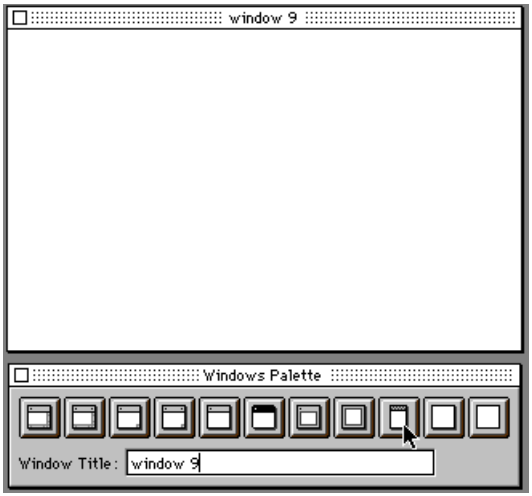

The following figure shows the window created by choosing the seventh button on the Windows Palette. This window is a movable modal dialog box. This is known as the “Movable Modal Dialog window - movableDBoxProc” constant in the Macintosh toolbox.




The following figure shows the window created by choosing the eighth button on the Windows Palette. This window is a standard dialog and alert window. This is known as the “Standard Dialog and Alert window - dBoxProc” constant in the Macintosh toolbox.

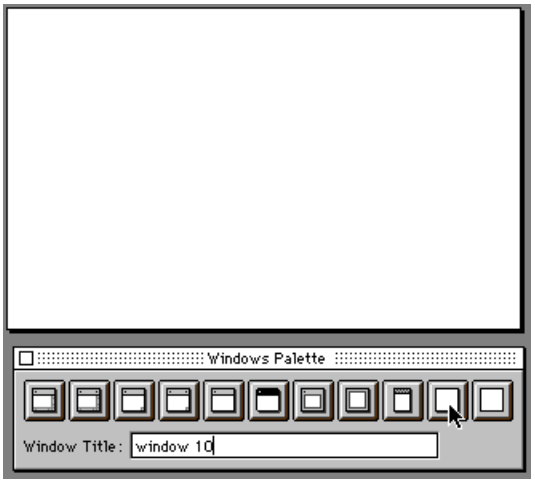


The following figure shows the window created by choosing the ninth button on the Windows Palette. This window is a movable utility window. This is known as the “Title Floating window” but it is not a Macintosh toolbox constant. It is provided by the Apple Dylan framework and interface builder.

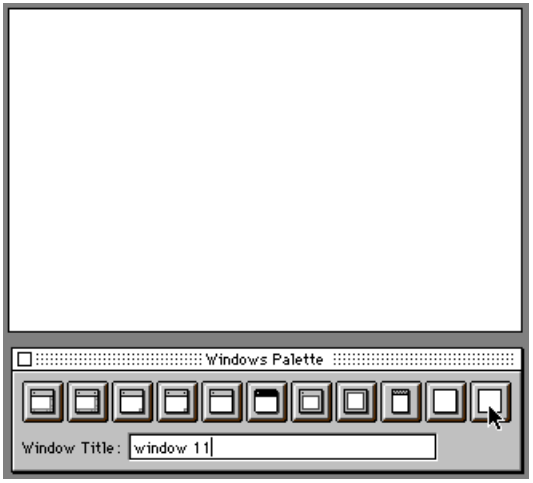



The window created by choosing the tenth button on the Windows Palette is a simple box dialog with a drop shadow. This is known as the “Shadowed Box Dialog window - altDBoxProc” constant in the Macintosh toolbox.

Palette Buttons




The window created by choosing the eleventh button on the Windows Palette is a simple box dialog without a drop shadow. This is known as the “Plain Box Dialog window - plainDBox” constant in the Macintosh toolbox.



APPENDIX A

# Palette Buttons

# Index


## A

Add button 20
Additions Catalog 8, 11, 47
Additions Catalog button 9, 33, 52
adorner class 19
Adorner page 11, 33
adorners 8, 33
Align Bottom Edges button 53
Align Centers Horizontally button 53
Align Centers Vertically button 53
Align Left Edges button 53
alignment 21
Align Right Edges button 53
Align Top Edges button 53
Apply button 35
Auto Resize Pages 23
Auto Resize Pages checkbox 45
Available Classes popup 20

## B

behavior class 19
Behavior page 11, 33
behaviors 8, 33
Bold text button 54
builder
  closing 42
  quitting 42
  running 21
builder menubar 5
builder overview 2
builder project 2

## C

catalog pages
  adding 8, 23
  removing 9
  saving 22, 23
Catalog Preferences page 8, 17, 19, 23, 45
catalogs
  using 8, 23
Center Justify button 55
Classes In Page popup 20
Class Importer 19, 20
Class Importer button 52
collapse 3, 25
Collapse All button 3
color
  adding 13
  background 32
  foreground 32
Color Inspector 13, 33
Color Inspector button 53
Colors Palette 13, 17, 32
Colors Palette button 53
color wells 32, 54
Command Palette 5, 21, 32, 33, 35
Command Palette buttons 51
Condensed text button 55
Controls Page 9, 20, 23
coordinates 33
Coordinates palette 6
Coordinates palette button 52
creating a project 2
cursor
  location 13
Custom Views Page 23

## D

data input panel 4, 27, 28, 29
Determiner page 11, 33, 47
determiners 8, 33
disclosure triangle 3, 4, 14, 25, 27
dmnu 41

## E

Edit Front Window command 29
editing 21
edit mode 29
elements
  adding 8, 28, 34
  changing 8, 12
  coloring 32
  creating custom 19
  editing 23
  saving 23
Enabled flag 27
expand 3, 25
Expand All button 3, 26
Extended text button 55

## F

floats? property 24
Font menu 54
Font popup 54
Frame property sheet 36
Framework 48
framework 1, 12, 19, 23, 26, 31, 36

## G

General Preferences page 14, 17
Grow Horizontal Relative 34

## H

Hide Interface Builder command 44
high-level elements 3, 14, 22

## I

Identifier field 27, 28
Interface Builder Catalog 8
Italic text button 55

## J

justification field 35

## L

Labels boxes 20, 47
Left Justify button 55
library 1, 13
Loading elements 48
Loading menus 50
Loading windows 48
Load UI Builder command 20, 43, 44
Locked flag 27

## M

Marquee Selection button 51
Menu Editor 14, 37
Menu Editor button 52
menu item 14
menuitem
  creating 39
menus 14
  creating 14, 37
  opening 14

previewing 39

## N

New Menu field 38
New Menu window 14, 37
New Project command 21
New Sub Menu 14

## O

Outlined characters button 55

## P

Pages popup 9, 23
pixels 13, 33
Plain text button 54
Preferences command 7, 8, 14, 17, 25
project
  creating 2, 5, 21
  saving 21, 42
project file 21, 42
project tree 3
project window 2, 3, 14, 25, 28, 37
  new 5
Property Inspector 12, 27, 29, 35
Property Inspector button 52
property sheets 35

## R

Remove button 20
resource editor 42
resource file 21, 42
resource ID 14, 17, 25, 27, 40
Resource ID field 28
RGB values 12, 13
Right Justify button 55
root view 26
run mode 29

## S

Save As command 41
Save Catalog command 8, 20, 22, 45
Selection Tool button 51
separator line
  creating 14, 39
Shadowed characters button 55
Show Interface Builder command 20, 43, 44
Size menu 30, 54
Size popup 54
standalone version 1, 21
Style menu 30, 54
sub-class 26
submenu 14, 15
  creating 14
Sub-menu window 15

## T

Text menu 30
Text Palette 7, 30
Text Palette button 52
Text Palette buttons 54
Text Tool button 51
Title field 27, 28
tree diagram 3, 25, 28

## U

Underlined text button 55

## V

view class 9, 19
view-determiner class 19
views 23
Views Catalog 8, 9, 19, 22, 25, 28
Views Catalog button 11, 52
View Size boxes 20, 47
Views Page 10, 23, 30
Visible flag 27

## W

window
  creating 6, 24
  floating 24
Window Inspector 7, 36
Window Inspector button 52
Window Preferences page 17, 18, 25
window properties 36
Window resources 48
Windows menu 5
Windows Palette 6, 24
Windows Palette button 24, 52
Windows Palette buttons 56
Windows Preferences page 7
Window Title field 6, 25

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter Pro printer. Final pages were created on a Docutek. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

PROJECT MANAGER
Trish Eastman

LEAD WRITER
Linda Kyrnitszke

WRITERS
Linda Kyrnitszke

ILLUSTRATOR
Sandee Karr

PRODUCTION EDITOR
Lorraine Findlay

SPECIAL THANKS TO
Robin Mair