

UIMAX

Beginner's Guide



**Integrated Computer
Solutions Incorporated**

Copyright © 2005-2007 Integrated Computer Solutions, Inc.

The *UIM/X Beginner's Guide*™ manual is copyrighted by Integrated Computer Solutions, Inc., with all rights reserved. No part of this book may be reproduced, transcribed, stored in a retrieval system, or transmitted in any form or by any means electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Integrated Computer Solutions, Inc.

Integrated Computer Solutions, Inc.

54 Middlesex Turnpike, Bedford, MA 01730

Tel: 617.621.0060

Fax: 617.621.9555

E-mail: info@ics.com

WWW: <http://www.ics.com>

UIM/X Trademarks

UIM/X, Builder Xcessory, BX, Builder Xcessory PRO, BX PRO, BX/Win Software Development Kit, BX/Win SDK, Database Xcessory, DX, DatabasePak, DBPak, EnhancementPak, EPak, ViewKit ObjectPak, VKit, and ICS Motif are trademarks of Integrated Computer Solutions, Inc.

Motif is a trademark of Open Software Foundation, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

X Window System is a trademark of the Massachusetts Institute of Technology.

All other trademarks are properties of their respective owners.

Contents

Preface	v
----------------------	----------

Chapter 1—Building Your First Project

The GUI You Will Build	2
The Steps in Building the To Do List	3
Step #1: Starting UIM/X Novice Mode	4
Step #2: Drawing Your Interface Window	5
Step #3: Moving and Resizing Objects	9
Step #4: Saving Your Work	12
Step #5: Adding Objects to Your Interface	14
Step #7: Revising the Menu Bar	40
Step #8: Adding Declarations and Callbacks	43
Step #9: Adding the “About” Dialog Box	55
Step #10: Testing Your Interface	59
Step #11: Generating Code	60
Where Do You Go From Here?	61

Chapter 2—Basics of UIM/X Novice Mode

What Is UIM/X Novice Mode?	64
Starting UIM/X Novice Mode	64
Parts of the Project Window	65
Using the Project Window	66
Using the Palette	72
Using the Pop-up Menus	75
Exiting from UIM/X	76

Chapter 3—Working with Objects

What Is an Object?	80
The Windows Category	81
The Primitives Category	86
Creating an Application Window	92
Adding Objects to an Application Window	93
Selecting Objects	93
Working with Objects	94
The Nine Regions of an Object	95
Moving Objects	96
Resizing Objects	97
Aligning Objects	98

Arranging Objects	100
Object Properties	101

Chapter 4—Viewing and Changing Properties

The Editors Provided	104
The Property Editor	105
The Color Viewer	113
The Color Editor	118
The Color Map	121
The Font Viewer	122
The Icon Viewer	125

Chapter 5—Building Menus

The Parts of a Menu	130
Tips on Creating Menus	131
Steps in Using the Menu Editor	133
Opening the Menu Editor	133
About the Menu Editor	134
Creating a Pull-down Menu	135
Creating an Option Menu	135
Using Mnemonics	136
Using Accelerators	137
Duplicating a Pane or Item	138
Deleting a Pane or Item	138
Rearranging Your Menu Bar	139
Rearranging Your Menu Items	140
Adding Callbacks to Menus	140
Testing Your Menu	141
Closing the Menu Editor	142

Chapter 6—Adding Connections

Steps in Using the Connection Editor	144
Opening the Connection Editor	144
About the Connection Editor	145
Loading a Source	146
Loading a Target	146
Defining Connections	147
Modifying Connections	148
Closing the Connection Editor	148

Chapter 7—Adding Constraints

Steps in Using the Constraint Editor	152
Opening the Constraint Editor	152
About the Constraint Editor.....	153
Defining Constraints	154
View Menu Options.....	158
Closing the Constraint Editor	159

Chapter 8—Browsing Object Hierarchies

About the Browser	162
Steps in Using the Browser	162
Opening the Browser	163
Loading an Interface into the Browser	163
Browsing the Interface	163
Viewing Instances of Objects.....	165
Changing View.....	166
Closing the Browser.....	167

Chapter 9—Managing Your Projects

What Is a Project?	170
File-Naming Conventions in UIM/X.....	170
Controlling Where Your Project Files Reside	171
Using the File Selection Dialog Box.....	173
Creating a Project	174
Saving Your Interfaces	174
Opening a Project or Interface File	175
Reusing Interface Files	177
Using the Program Layout Editor.....	178

Chapter 10—Generating Code

Generating Code for a Project	184
Generating Code for Selected Interfaces Only	184
Saving Time Regenerating Code.....	185
Troubleshooting Your Compiled Application.....	186

Chapter 11—An Introduction to Instances

How to Create an Interface Instance	190
How to Pop Up a Dialog Box	190

Chapter 12—Programming in UIM/X

The Ux Convenience Library	198
----------------------------------	-----

Setting and Retrieving Property Values	198
Creating and Displaying Interfaces	199
Using the Declaration Editor	200
Chapter 13—Beyond the Basics	
Building GUIs in UIM/X	204
Setting Properties	205
Programming in UIM/X.....	207
Editing Interface Code in the Declaration Editor	208
Generating Code.....	210
Generating Resource Files	211
Object-Oriented Programming	211
Programming with C++	212
Building Palettes	213
Non-Visual Objects	213
Appendix A—Effective GUI Design	215
The Principles in a Nutshell	216
Appendix A—Object Properties.....	221
Property Value Data Types.....	222
Special String Types	222
The Seven Core Properties for Each Object.....	224
Additional Properties Available for Certain Objects	225
Index.....	237

Preface

Overview

The *UIM/X User's Guide* describes the novice mode of UIM/X, the leading graphical user interface (GUI) builder for UNIX. The novice mode of UIM/X is for anyone who wants a simple introduction to building GUIs. Novice mode is a version of UIM/X with a limited Palette that is easier for new users to learn.

The main part of this guide is a step-by-step tutorial on how to build your first GUI. After you complete this tutorial, you will understand all the basics of UIM/X novice mode. Then you can use the following chapters to learn more details and some alternate ways to do various tasks. Desktop UIM/X is designed to work with the AIXwindows Desktop available with AIX version 4.1, which is based on Common Desktop Environment (CDE) technology. Help volumes, InfoExplorer information, and this *User's Guide* refer to the desktop as CDE.

Who Should Use this Guide

The *UIM/X Beginner's Guide* is intended for three types of users:

- Non-programmers interested in building GUIs
- C programmers just getting started with Motif
- C programmers already familiar with Motif

If you are not a C programmer, you should be able to understand this guide, and complete the tutorial given in Chapter 1, “Building Your First Project.” This chapter does not involve any C coding.

If you are a C programmer just getting started with Motif, make sure you read Chapter 3, “Widgets in Desktop UIM/X,” for an introduction to the objects supported by the novice mode of UIM/X.

If you are a C programmer already familiar with Motif, this guide should help you quickly get comfortable with UIM/X novice mode.

UIM/X novice mode is for anyone who wants a quick introduction to building GUIs with UIM/X, both programmers and non-programmers, including:

- Customers who want to create prototypes of their required applications to pass on to software developers.
- Builders of corporate standards who need custom interfaces and applications followed throughout their corporations.
- Business users or marketing personnel who want to develop simple applications or prototypes for their in-house needs.

Even if you are *not* a programmer, you should still be able to understand this guide, and complete the tutorial given in Chapter 1, “Building Your First Project.” You may find Chapter 12, “Programming in UIM/X,” more difficult to understand. Non-programmers can skip this chapter.

Before You Read this Guide

This guide makes the following assumptions:

- You are familiar with the basic functions of selecting from menus and dialog boxes; opening, moving, resizing and closing windows; and clicking icons.
- You understand the functions of the three mouse buttons, which this guide refers to as the Select button (left button), the Adjust button (middle), and the Menu button (right). See “Using the Mouse” on page xii.
- Either you have enough familiarity with programming to enter your own callback code; or you are using the novice mode of UIM/X to help design user interfaces for which a colleague can provide any code required.

The UIM/X Document Set and Related Books

This section lists the UIM/X document set, and provides a suggested list for further reading.

The following list is the complete UIM/X document set:

- *UIM/X Installation Guide*. Explains how to install and run UIM/X. Includes information on the files provided with UIM/X, backwards compatibility issues, and compiler considerations.
- *UIM/X Beginner’s Guide*. Introduces UIM/X by presenting Novice Mode, the simplified Palette that enables new users to be productive immediately. Includes information on a number of important features for creating, testing and running applications.
- *UIM/X Tutorial Guide*. A series of step-by-step tutorials, teaching tools and techniques that will greatly assist you in developing your own applications. Features tutorials in Novice Mode, Standard Mode, and on advanced topics.
- *UIM/X User’s Guide*. Explores the UIM/X features essential to GUI development. Includes discussions of how to use UIM/X’s editors to set properties, add behavior, etc.
- *UIM/X Motif Developer’s Guide*. An in-depth guide to the widgets, features and capabilities of UIM/X as they relate specifically to Motif development.

- *UIM/X Advanced Topics*. Describes how to customize UIM/X, including integrating new widget and component classes into the executable. Includes reference information of an advanced technical nature.
- *UIM/X Reference Manual*. A comprehensive list of properties, methods, and events, plus more, for Motif development. Designed for the experienced developer.

Suggested Reading

For more information on designing GUIs, see any of the following books:

- *OSF/Motif Style Guide release 1.2*
(Prentice Hall, 1993, ISBN 0-13-643123-2)
- *Visual Design with OSF/Motif*
(by Shiz Kobara, Addison-Wesley, 1991, ISBN 0-201-56320-7)
- *New Windows Interface: An Application Guide*
(Microsoft Corporation, 1994, ISBN 1-55615-679-0)
- *Human Interface Guidelines: The Apple Desktop Interface*
(Addison-Wesley, 1987, ISBN 0-201-17753-6)
- *Windowing Korn Shell for the OSF/Motif Graphical User Interface Programmer's Guide* (UNIX System Laboratories, Inc., 1992)

How this Guide Is Organized

This guide comprises thirteen chapters, two appendixes, and an index, as follows:

- *Chapter 1, “Building Your First Project”*, provides a tutorial on building your first user interface with the novice mode of UIM/X. If you do *not* plan to do the tutorial, you should read the following chapters more carefully.
- *Chapter 2, “The Basics of UIM/X Novice Mode”*, describes all the basics of using the novice mode of UIM/X, including the Project Window, the Palette, and the pop-up menus provided.
- *Chapter 3, “Working with Objects”*, describes each of the objects supported by the novice mode of UIM/X, how to work with these objects, and how to use the Palette.
- *Chapter 4, “Viewing and Changing Properties”*, describes how to use the Property Editor and the related viewers and editors provided: the Color Viewer, the Color Editor, the Color Map, the Font Viewer, and the Icon Viewer.
- *Chapter 5, “Building Menus”*, describes how to use the Menu Editor to create standard pull-down and option menus for your interface.
- *Chapter 6, “Adding Connections”*, describes how to establish a behavioral connection between a source and target object.
- *Chapter 7, “Adding Constraints”*, describes how to define form constraints graphically without needing to know the numerous Motif form constraint properties.
- *Chapter 8, “Browsing Object Hierarchies”*, describes using the Browser for viewing and working with object hierarchies.
- *Chapter 9, “Managing Your Projects”*, defines a project and its related files, how to save and manage files, the makefile and how to edit it with the Program Layout Editor.
- *Chapter 10, “Generating Code”*, describes how to generate the code for your project, and a few tips on troubleshooting your generated application. various options for generating code.
- *Chapter 11, “An Introduction to Instances”*, provides an introduction to creating instances of objects in UIM/X.
- *Chapter 12, “Programming in UIM/X”*, describes the basics of programming in UIM/X.

- *Chapter 13, “Beyond the Basics”*, describes the benefits of moving to UIM/X to build even more sophisticated interfaces than you can with the novice mode of UIM/X.
- *Appendix A, “Effective GUI Design”*, discusses the main principles of effective GUI design, and some common pitfalls to avoid.
- *Appendix B, “Object Properties”*, defines each object property available in the novice mode of UIM/X.

Some Terms You Should Know

Certain basic terms recur throughout this guide, and it helps to understand them from the outset.

An *object* is a building block you can use to build an interface with UIM/X. The novice mode of UIM/X supports two different types of objects:

- Motif widgets
- Compound objects

A *Motif widget* is an object whose appearance and behavior precisely follows the *OSF/Motif Style Guide*. The novice mode of UIM/X supports a number of popular Motif widgets, including Push Button, Label, Text Field, and more.

A *compound object* consists of several Motif widgets combined into one object for your convenience. The novice mode of UIM/X supports a number of compound objects, which save you the time you might otherwise spend creating them, including Application Window and Group Box.

An *interface* is a window or dialog box that you build up from objects with UIM/X. The novice mode of UIM/X supports four different types of interfaces: Application Window, Secondary Window, Message dialog box, and File Selection dialog box. Certain menu options refer to an interface, such as Save Interface; these act only on your selected interface.

A *project* contains all the interfaces (i.e., windows and dialog boxes) and their associated files for a certain GUI you are building with UIM/X. The program can automatically save and generate code for an entire project in one step. Certain menu options refer to a project, such as Save Project; these act on all the windows and dialog boxes in your project.

Conventions Used in this Guide

The following table describes the typographic conventions used in this guide.

Typeface or Symbol	Meaning	Example
AaBbCc12	The names of commands, files, and directories; or onscreen output; or user input.	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all the files. %You have mail.
<i>AaBbCc12</i>	A placeholder you replace with your actual value; or words to be emphasized; or book titles.	To delete a file, type <code>rm filename</code> . You <i>must</i> be <code>root</code> to do this. See Chapter 6 in the <i>User's Guide</i> .
File⇒Open	The Open option in the File menu.	Choose the File⇒Open command.
Alt+F4	Press both Alt and F4 at once.	Press Alt+F4 to exit.
Return	The key on your keyboard marked Enter, Return, or ↵.	Press Return.

Installation Directories

Product installation directories can depend on the platform or the user's preferences. To keep things simple, this guide uses general names for product installation directories. The following table lists the name and the corresponding product installation directory:

Name	Description
<i>uimx_directory</i>	The UIM/X installation directory.

Using the Mouse

Before starting the tutorial, take a moment to review the location and usage of your mouse buttons, as illustrated in Figure P-1 and the following table:

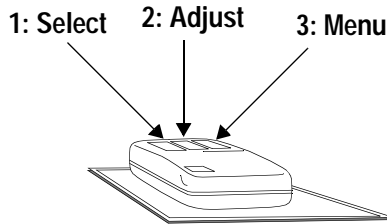


Figure P-1 The Mouse Buttons

Button:	Called:	Is used for:
1	Select	Selecting objects, menus, toggles, and options.
2	Adjust	Resizing and moving objects.
3	Menu	Displaying popup menus.

Throughout this book, you use the mouse buttons along with the mouse pointer to make selections, move the input pointer, or position the text insertion point. You can perform any of the following mouse operations.

Operation	Description
Point to	Move the mouse to make the pointer go as directed.
Press	Hold down a mouse button.
Release	Release a mouse button after pressing it.
Click	Quickly press and release a mouse button without moving the mouse.
Drag	Move the mouse while pressing a mouse button.
Double-click	Click a mouse button twice in rapid succession without moving the mouse pointer.
Triple-click	Click a mouse button three times in rapid succession without moving the mouse pointer.

In general, instructions for mouse operations include the name of the mouse button. The exceptions are Click, Double-click, and Drag. These common operations may be described without specifying a mouse button. For example:

- Click on the `applWindow` icon in the Interfaces Area of the Project Window.
- Drag the `pushButton` icon from the Palette.

In these cases, use the Select button to click and double-click, and the Adjust button to drag.

Setting Application Defaults

Application Defaults configure the way UIM/X looks and set the default preferences for many of its operations. You can set the Application Defaults for all UIM/X users or for a single user. For more details on setting your Application Defaults see the *UIM/X User's Guide*.

For optimum performance, set the following resources in your Application Defaults.

```
Mwm*autoKeyFocus: false
Mwm*clientAutoPlace: false
Mwm*focusAutoRaise: false
Mwm*focusFollowsPointer: true
Mwm*keyboardFocusPolicy: pointer
```

If you have a gray-scale monitor, you might try the following settings:

```
Mwm*activeBackground: #666666 (gray40)
Mwm*activeForeground: #e5e5e5 (gray90)
Mwm*background: #666666 (gray40)
Mwm*foreground: #e5e5e5 (gray90)
Uimx3_0*calculatedColors: false
Uimx3_0*background: #ededed (gray93)
Uimx3_0*BottomShadowColor: #000000 (black)
Uimx3_0*foreground: #000000 (black)
Uimx3_0*TopShadowColor: #ffffff (white)
Uimx3_0*XmText.background: #b3b3b3 (gray70)
Uimx3_0*XmTextField.background: #b3b3b3 (gray70)
```

Note: The resources above prefixed with `Mwm` are specific to the Motif Window Manager. If you are using a different window manager consult your Systems Administrator for the equivalent settings.

Building Your First Project

Overview

This chapter provides a complete tutorial on building your first project with the novice mode of UIM/X. You will learn how to quickly design, test, generate code, and run a sample interface. The features described in this tutorial are also covered in more detail in the following chapters.

You can do this tutorial to get started and then use the rest of this guide for more details and reference. Or, you can skim the later chapters to get a feel for the product and then do the tutorial to gain some initial hands-on use.

You don't have to complete the whole tutorial in one sitting. You can stop at any point, save your work, and continue later.

You do *not* need to be a programmer to understand and complete this tutorial. Non-programmers may find *Chapter 12, "Programming in UIM/X"*, more difficult to understand. They can safely skip Chapter 12, and still use the novice mode of UIM/X to design or prototype complete, useful interfaces.

Note: The project in this chapter was created using the Motif Window Manager (mwm) and its default resource values, *except* for `clientAutoPlace`, which was set to `false`. If you are using a different window manager, or have other than default values for window manager resources, you may see slightly different object appearance and behavior from that described in this chapter.

The GUI You Will Build

In this tutorial, you will create a To Do List, which lists your tasks to be done and their priorities. This interface provides push button control for modifying and deleting tasks from your list. The To Do window consists of three main areas, as shown in Figure 1-1:

- *Menu Bar*: Provides two pull-down menus: File for opening and saving files, and Help for seeing on-line help.
- *List Area*: Shows a list of tasks, and the priority for each one, along with PushButtons for editing and deleting the selected task.
- *Edit Area*: Provides PushButtons for adding a new task, replacing the current task, and clearing the Task field, and enables you to change the description and priority of any task.

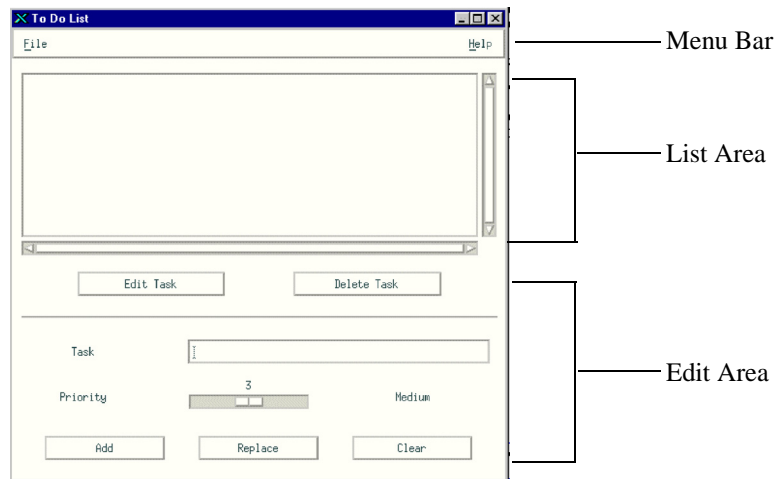


Figure 1-1 The GUI You Will Build

Note: A full, completed version of the project constructed in this tutorial can be found in `uimx_directory/contrib/ToDoList/ToDo.prj`.

Using the To Do List

You can enter or revise tasks in the Edit area and display them in the List Area. You can select any task to change or delete. You can reset the priority of any selected task using the HorizScale. You can use the pull-down File menu to save your current list of tasks in a text file.

The Steps in Building the To Do List

There are 11 steps in building this project with the novice mode of UIM/X:

Step #1: Starting UIM/X Novice Mode

Step #2: Drawing Your Interface Window

Step #3: Moving and Resizing Objects

Step #4: Saving Your Work

Step #5: Adding Objects to Your Interface

Step #6: Changing Object Properties

Step #7: Revising the Menu Bar

Step #8: Adding Declarations and Callbacks

Step #9: Adding the “About” Dialog Box

Step #10: Testing Your Interface

Step #11: Generating Code

Notice that some related background material accompanies each step. If you need more background on a particular step, see the related chapter reference.

Completing this tutorial will likely take you between one and two hours. Some of the simpler steps, such as starting the program and saving your work, take only a few seconds. Others, such as changing object properties or adding callbacks, take longer. However long you spend building an interface with the novice mode of UIM/X, you will save a lot of time compared to coding an interface completely by hand.

Step #1: Starting UIM/X Novice Mode

Now you are ready to start the novice mode of UIM/X. Before you start the program, create a new directory called `todo` to hold the files you create in this tutorial, and then change to that directory, as follows:

1. Start the X Window System.
2. Bring up a terminal window.
3. Make a directory to hold the files you create in this tutorial:

```
mkdir todo
```

4. Change to the directory you just created:

```
cd todo
```

5. Start UIM/X novice mode from that directory:

```
uimx -novice -language ansic &
```

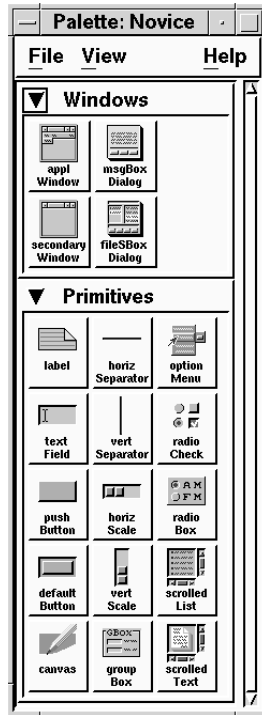
Note: The `-language` option instructs UIM/X to use ANSI C mode. While C++ mode accepts code written in C, for the purpose of this tutorial C mode is sufficient. By default UIM/X starts in C++ mode.

If your `PATH` variable does not provide the full path to the UIM/X executable, you will have to specify it when you run the program:

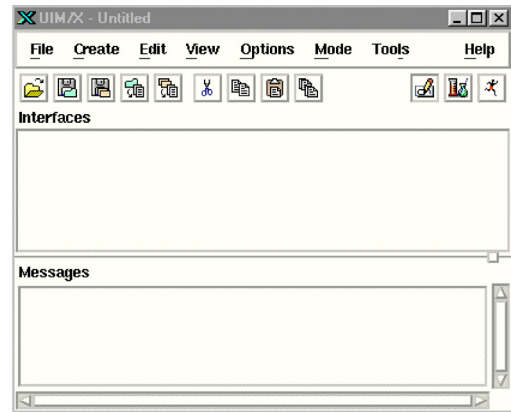
```
uimx_directory/bin/uimx -novice -language ansic &
```

6. After a brief pause, a copyright notice window appears on the screen. In a few seconds, the Project Window and Palette appear, as shown in Figure 1-2.
7. Minimize the terminal window and move it to one corner of your screen.

Note: To restart this tutorial at any time, begin again from Step 4 above.



Palette



Project Window

Figure 1-2 UIM/X Novice Mode Palette and Project Window

Step #2: Drawing Your Interface Window

An `ApplicationWindow` includes a menu bar. A `SecondaryWindow` has no menu bar and you can *not* add a menu bar to it. Otherwise these two windows look and work the same.

A window or dialog box can contain other objects, such as `TextFields` and `PushButtons`. A window or dialog box is said to be the “parent” of the objects it contains, while these objects are its “children”.

BUILDING YOUR FIRST PROJECT*Step #2: Drawing Your Interface Window*


The *Palette* is a toolbox of objects you can use to build interfaces. To create a window or any other object in the novice mode of UIM/X, you drag its icon from the Palette and drop it in the desired location on your screen.

For more details, see Chapter 3, “Working with Objects”.

1. Check that the Design icon in the Project Window icon bar is selected, as shown in Figure 1-3.



Figure 1-3 Design icon selected

2. Click on the applWindow icon in the Palette.
3. Notice that the mouse pointer changes to an upper-left corner shape: .
4. Press and hold down the Select button where you want the top left corner of the Application Window to be located on your screen.
5. While holding down the Select button, drag the mouse down and to the right to “draw” a rectangle, as shown in Figure 1-4.
6. Make your rectangle about 2 inches (5 cm) wide by 3 inches (7.5 cm) deep.

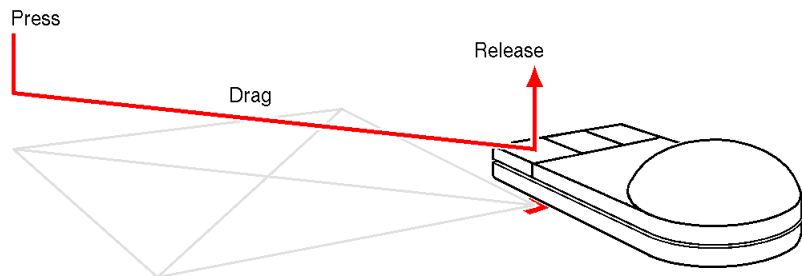


Figure 1-4 Creating an Application Window with drag and draw

7. Release the mouse button.
8. This process is called “drag and draw.”
9. Notice that the program creates an Application Window called `applWindow1`, as shown in Figure 1-5.

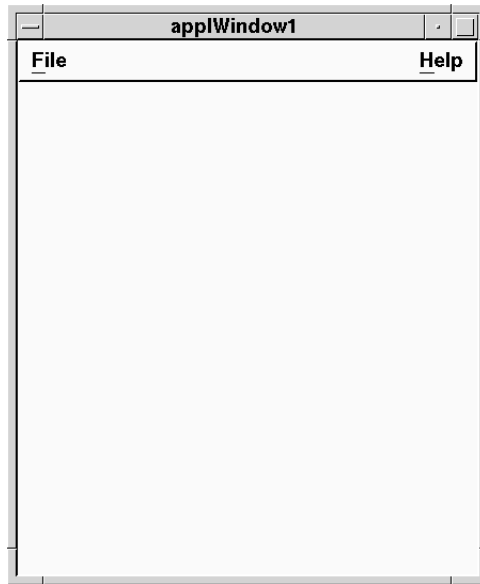


Figure 1-5 Your New Application Window

Note: If the size or location of the window is not what you wanted, don't be alarmed. The window manager may override your choice to display a window with its own defaults. You will adjust the window's size and location soon.

BUILDING YOUR FIRST PROJECT*Step #2: Drawing Your Interface Window*

Notice that the new window is represented by an icon in the Interfaces Area of the Project Window, as shown in Figure 1-6.

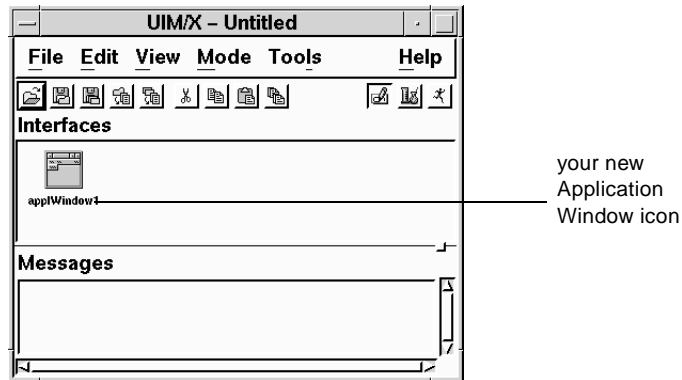


Figure 1-6 Application Window Icon

Also notice how the Application Window has eight handles around its perimeter, as shown in Figure 1-7.

These handles show an object is selected. After you create an object, it is automatically selected.

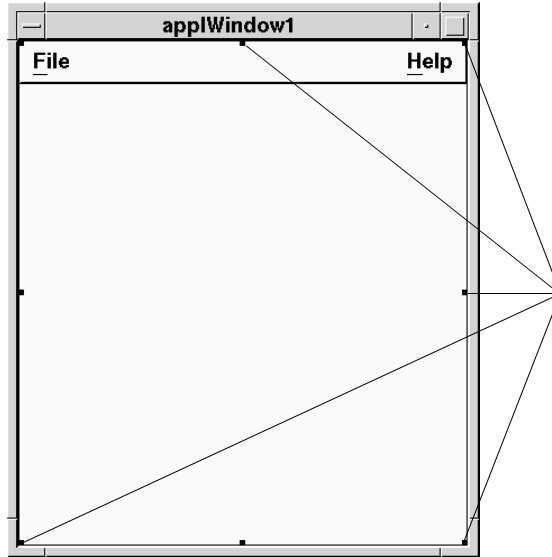


Figure 1-7 Handles Around a Selected Object

Step #3: Moving and Resizing Objects

Now you will practice moving and resizing your Application Window. You can move or resize any selected window, dialog box, or object the same way.

It's easy to move or resize the Application Window or any other object. When you move or resize an object, the position of the mouse pointer is important, since UIM/X divides each object into nine invisible regions, as shown in Figure 1-8.

1	2	3			
4	5	6			
7	8	9			

Figure 1-8 Nine Regions of an Object, and their Resize Pointers

To see the nine regions of an object, point to any corner of the object and press the Adjust button. The grid that appears is called the “resize grid”. Depending on which region of the object you point to when you press the Adjust button, you see a different resize pointer, as shown in Figure 1-8. Each resize pointer enables you to perform a different function, as listed in Table 1-1.

You can use the central region (5) of the resize grid, which displays the compass pointer, to move a selected object to a new location. You can use the other eight regions to stretch or shrink a selected object to a new size. For more details, see Chapter 3, “Working with Objects”.

Table 1-1 Functions of the Resize Pointers

Pointer Shape	Purpose
	To change the object’s height and width.
	To change the object’s height.
	To change the object’s width.

Moving the Application Window

1. Point to the center of `applWindow1`.
2. Press and hold down the Adjust button, and notice how the mouse pointer changes to a compass shape. This means you can now move the object.
(If you see the resize pointer, release the button and press again, closer to the center of the object.)
3. Drag `applWindow1` to a new location and then release the mouse button. The window moves, as shown in Figure 1-9.

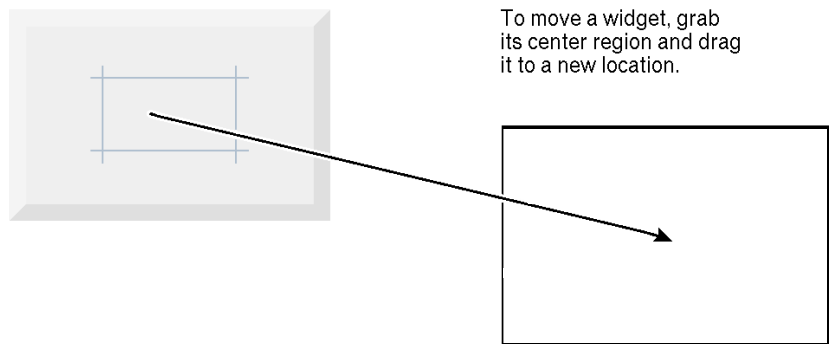


Figure 1-9 Moving an Object

4. Repeat steps 2 and 3 until you are comfortable moving objects around the screen.

Note: Do *not* move a window or dialog box with its title bar. This will *not* move the object permanently. Instead, the next time you open, show, or reload that window or dialog box, it returns to its previous location. Always use the compass pointer to move objects.

Resizing the Application Window

1. Point to one of `applWindow1`'s resize regions, such as the lower-right corner.
2. Press and hold down the Adjust button, and notice how the mouse pointer changes to a resize pointer, and the resize grid appears. This means you can now resize the object.
(If you see the compass pointer, release the button and press again, further away from the center of the object.)

BUILDING YOUR FIRST PROJECT

Step #4: Saving Your Work

3. Drag the resize pointer to stretch the outline of `applWindow1` to a larger size, then release the mouse button.

The window reappears larger, as shown in Figure 1-10.

4. Repeat steps 2 and 3, this time making `applWindow1` the size you want for your interface.

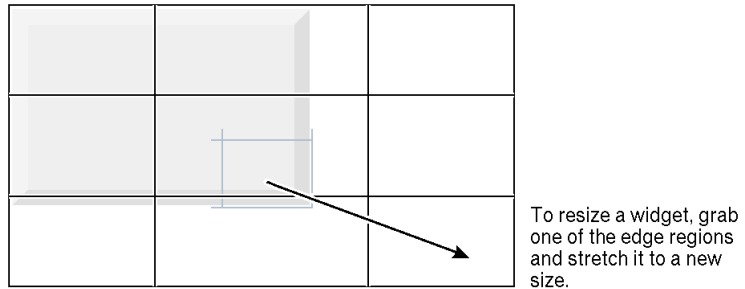



Figure 1-10 Resizing an Object

Note: Do *not* resize a window or dialog box with its resize handles. This will *not* resize the object permanently. Instead, the next time you open, show, or reload that window or dialog box, it will return to its previous size. This is so you can shrink down a window or dialog box on the fly to clear some space on your screen, without making a permanent change. Always use the resize pointers to resize an object permanently.

Step #4: Saving Your Work

As with any type of software, you should often save your work in UIM/X. The novice mode of UIM/X can save all your current windows and dialog boxes together in a *project*.

1. Choose the File⇒Save Project command from the Project Window or select the Save Project icon  from the icon bar.

The File Selection dialog box appears, with a default file name of `Untitled.prj` in your new `todo` directory, as shown in Figure 1-11.

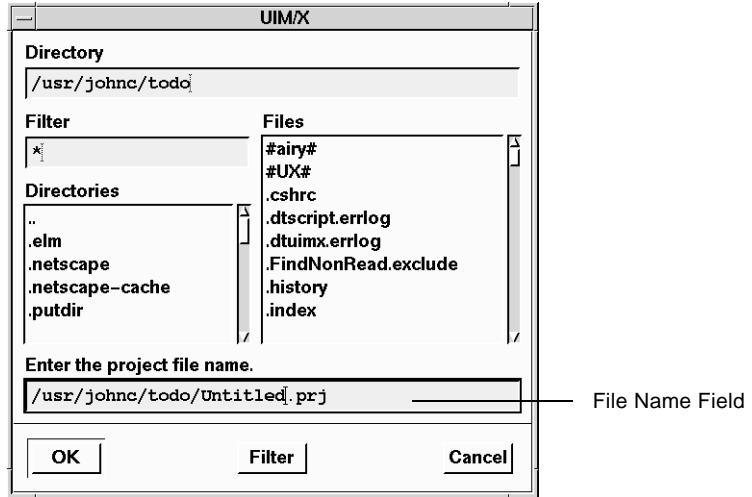



Figure 1-11 File Selection Dialog Box


2. Double-click in the File Name field and type:

ToDo.prj

3. Then click OK to save your work.

Your project is saved in a file named `ToDo.prj`. You can reload that file at any time, using the `File⇒Open` command in the Project Window or by selecting the Open icon  from the icon bar.

Notice that your project name is now shown in the title bar in the Project Window.

For the rest of this session, choosing `File⇒Save Project` or its corresponding icon  will save the latest version of all your current interfaces in the file `ToDo.prj`.

Step #5: Adding Objects to Your Interface

The Palette is a toolbox of objects you can use to build interfaces. By default, each object is shown in the Palette as an icon with a name. You can change this with the View menu to show an object with a name only, or an icon only.

The available objects are grouped into two categories in the Palette: Windows and Primitives. Windows include the four available windows and dialog boxes. All the other objects available in the novice mode of UIM/X are Primitives, including PushButtons, TextFields, Labels, and so on.

There are three ways you can add objects to an interface from the Palette:

- Click on an icon, position the mouse pointer on the interface, and click again, to create a default-sized object
- Drag and drop a default-sized object from the Palette into an interface
- Drag and draw an object in an interface, at the size and location you choose

For more details, see Chapter 3, “Working with Objects” and Chapter 7, “Adding Constraints”.

Now you will add the other objects you need to your Application Window. You need numerous objects, including:

- A ScrolledList to show your list of tasks
- PushButtons for entering, updating, and deleting tasks
- A HorizSeparator for visual design purposes
- A TextField for typing in and editing the selected task
- A HorizScale for setting the priority of each task
- Two Labels to show the purpose of the TextField and Scale
- Another label showing priority currently selected

First you will create the List Area at the top of your window, and then the Edit Area at the bottom.

Creating the List Area

You will create the List Area by adding a ScrolledList, two PushButtons, and a HorizSeparator to the top half of your window. You already used “drag and draw” to create your Application Window. Now you will use two more methods to create your next objects: “drag and drop” and “duplicate”.

Adding the Scrolled List

1. Point to the ScrolledList icon in the Primitives category of the Palette.
2. Press and hold down the Adjust button.

The pointer changes to a compass pointer, and an outline of the object appears. This means the object is ready for you to drag and drop.

3. Drag the outline of the object to the top of your Application Window.
4. Release the mouse button.

The new ScrolledList appears where you dropped its outline. This method is called “drag and drop”.

By default, a new object is given a default size when you drag and drop it. Don’t be alarmed if this default size is not exactly what you need in your interface. Next you will move and resize the ScrolledList.

Moving and Resizing the Scrolled List

The ScrolledList should almost completely fill the top of your Application Window, with the same border for its left, right, and top sides. If the ScrolledList does not fit as desired, move and resize it now.

1. Point to the centre of the ScrolledList, and hold down the Adjust button.
The pointer changes to a compass pointer. This means the object is ready for you to move.

2. Drag the mouse to move the ScrolledList, then release the mouse button.

3. Point to one corner of the ScrolledList, and hold down the Adjust button.
The pointer changes to a resize pointer, and the nine-region resize grid appears. This means the object is ready for you to resize.

4. Drag the mouse to resize the ScrolledList, then release the mouse button.

- Repeat steps 1 through 4 until your interface appears as shown in Figure 1-12.

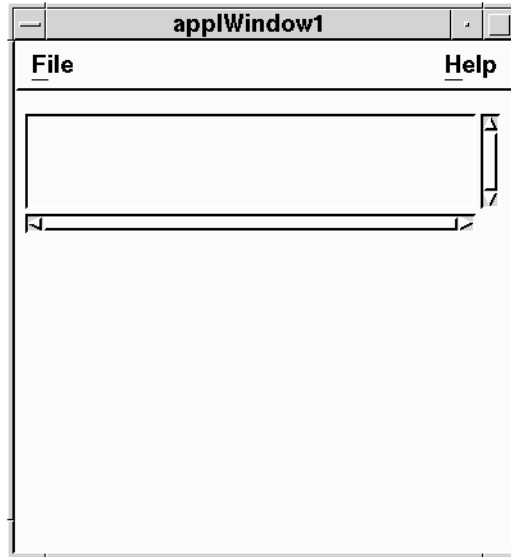


Figure 1-12 Your Interface with the ScrolledList Added

Adding the Push Buttons

Now you will create the two PushButtons below the ScrolledList. You will use the same drag and drop method to create the first PushButton. Then you will duplicate an exact copy of it for the second PushButton. Duplicating objects saves time and gives an uniform look to your interface.

- Point to the PushButton icon in the Palette.
- As before, press and hold down the Adjust button.
The familiar compass pointer and object outline appear.
- Drag the outline to your interface, on the left below the ScrolledList.
- Release the mouse button.

A PushButton appears where you specified in your interface. By default, it is called `pushButton1`. Don't be alarmed if `pushButton1` is too small to display its entire name properly. You will resize it soon.

Note: The font used for all labels in your interface depends on your default font, set in your `.Xdefaults` file. You will change this font later if necessary.

- With `pushButton1` selected, press the Menu button.

The Selected Objects pop-up menu appears, as shown in Figure 1-13. Notice the various options you have for your selected object.

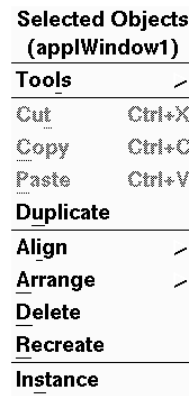


Figure 1-13 Selected Objects Pop-up Menu

- Choose the Duplicate option.

Another PushButton, called `pushButton2` appears, overlapping `pushButton1`. This is how to duplicate a selected object.

Note: You can also duplicate a selected object by choosing the Edit⇒Duplicate command from the Project Window or the Duplicate icon.



- Point to the center of `pushButton2`, press the Adjust button, and drag `pushButton2` to the right under the ScrolledList, as shown in Figure 1-14.

Your new PushButtons will most likely need to be moved, resized, or aligned. You will do that next.

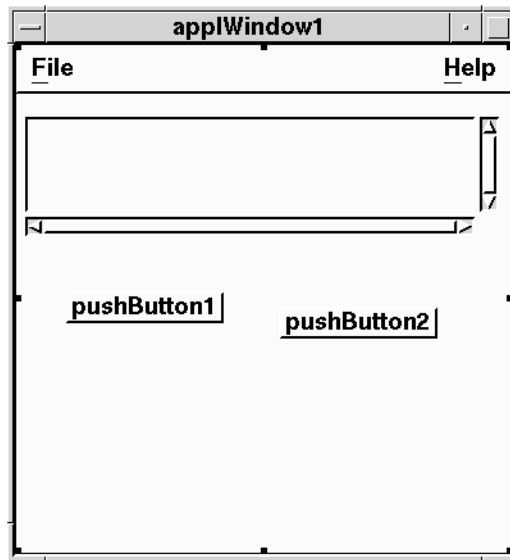


Figure 1-14 Your Interface with Two Push Buttons Added

Working with Multiple Objects

So far you have worked with one object at a time. Now you will select both your PushButtons and move, resize and align both of them at once. This saves time and helps you achieve uniform results. You can also duplicate or delete multiple selected objects in a single step.

Selecting Both Push Buttons at Once

1. Point above and to the left of `pushButton1` in your interface (*not* on the `ScrolledList`).
2. Press and hold down the Select button.
3. Drag the mouse pointer below and to the right of both PushButtons. The mouse pointer changes to an “O” shape, and a marquee (dashed box) follows the pointer, as shown in Figure 1-15.
4. Continue dragging the pointer to surround both PushButtons with the marquee.
5. Release the mouse button.

Notice that both PushButtons inside the marquee are now selected.

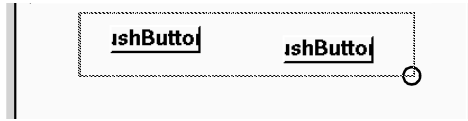


Figure 1-15 Selecting Both Push Buttons at Once

Note: You can also select multiple objects by holding down Ctrl and clicking on each object in turn.

Positioning Both Push Buttons at Once

Now that both PushButtons are selected, you can move, resize, and align both of them at once for a consistent look to your interface.

1. Point to the center of one PushButton, and when the compass pointer appears, drag the mouse to move the PushButton.

Notice that both PushButtons move at once.

2. Point to the edge of one PushButton, and when the resize pointer and outlines appear, drag the mouse to resize both PushButtons at once.
3. With both PushButtons still selected, press the Menu button, and choose Selected Objects⇒Align.

BUILDING YOUR FIRST PROJECT*Step #5: Adding Objects to Your Interface*

The Align submenu appears, as shown in Figure 1-16. Notice the various options for aligning objects.

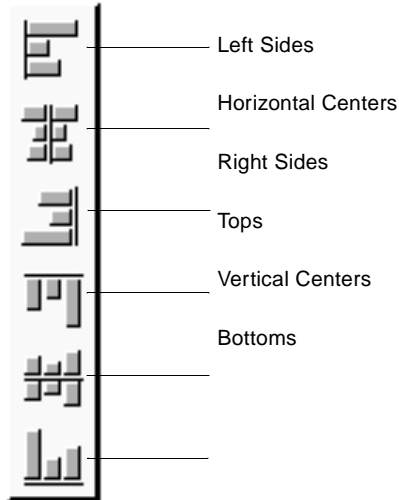


Figure 1-16 Align Submenu

4. Choose Align⇒  to align both PushButtons by their tops.

Your interface should appear as illustrated in Figure 1-17.

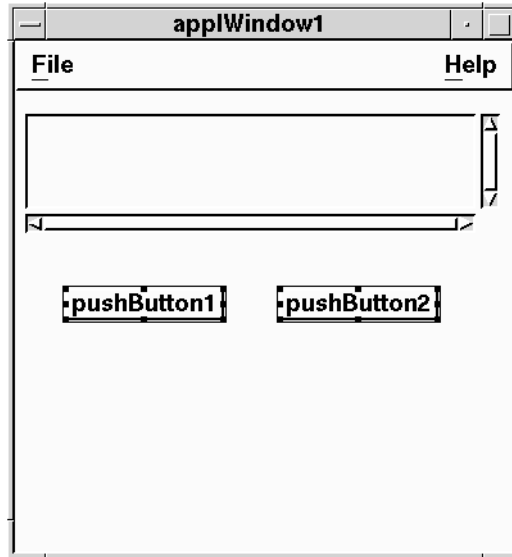


Figure 1-17 First Two Push Buttons Positioned

You can further refine the appearance of your interface by using the Arrange command to control the size of the space between objects, and the size of the space between the objects and the margin of their parent.

To arrange the placement of the PushButtons,

1. Select both PushButtons if they are not already selected.
2. Choose the Edit⇒Arrange command from the main window or press the Menu mouse button, and choose the Selected Objects⇒Arrange command.

The Arrange submenu appears, as shown in Figure 1-18.

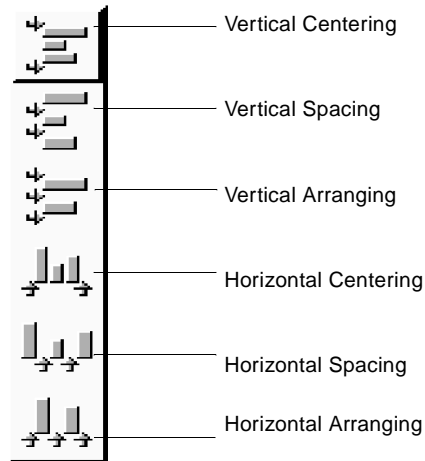


Figure 1-18 Arrange Menu Icons

3. From the submenu, choose the Horizontal Arranging arrangement icon.

The selected objects reappear, centered horizontally.

Adding the Separator

Now you will add a finishing touch: a `HorizSeparator` to help visually separate the List Area at the top from the Edit area at the bottom.

Separators are purely decorative; they can not have any behavior, but they are very useful for dividing up the various areas of an interface.

1. Point to the `HorizSeparator` icon in the Palette.
2. As you did earlier, hold down the Adjust button, and drag and drop the Separator in your interface below your two `PushButton`s.
3. Using the Adjust button and the compass and resize pointers, move and lengthen the Separator as required.

At this point, you have finished adding objects to the List Area.

Your interface should appear as shown in Figure 1-19.

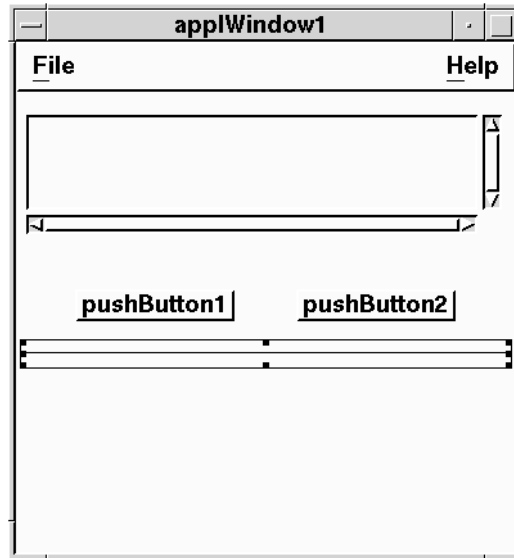



Figure 1-19 List Area with All Objects Added

4. Select File⇒Save Project in the Project Window or click on the Save Project icon  in the icon bar to save your work.
 This saves your project under the same file name you specified earlier.

Creating the Edit Area


Now you will create the Edit Area in the bottom half of your interface. The Edit Area includes three Labels, a TextField, a HorizScale, and three PushButtons. The Labels provide meaningful labels for the TextField and Scale. You type each new task into the TextField. You set the priority of each task with the Scale. You add, modify, or delete a task with the PushButtons. You will create these objects using a combination of the three methods you already learned: drag and drop, drag and draw, and duplicate.

Adding the Labels

1. Point to the Label icon in the Palette.
2. As you did earlier, hold down the Adjust button and drag and drop a Label to the top left of the area under the Separator.
 A new label appears, named `label1`.
3. With `label1` selected, choose Edit⇒Duplicate from the Project Window.

BUILDING YOUR FIRST PROJECT

Step #5: Adding Objects to Your Interface

This works the same as choosing Selected Objects⇒Duplicate or clicking on the Duplicate icon  in the icon bar.

A second label appears, named `label2`.

4. Move `label2` beneath `label1`.
5. Move and resize one or both labels as required.
6. Select both labels and align them by their left sides if required.

At this point, your interface should look as shown in Figure 1-20.

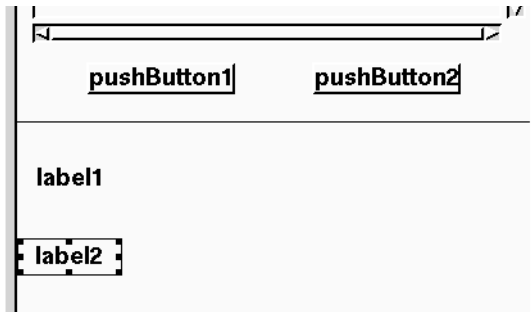



Figure 1-20 Your Interface with Labels Added

Adding the Text Field, Horizontal Scale, and its Label

Now you will drag and draw a TextField, the same way you created the Application Window. And you will drag and drop a HorizScale and another Label.

1. Click on the TextField icon in the Palette with the Select button. The pointer changes to the familiar upper left corner shape.
2. Point to the area to the right of `label1`, and drag down and to the right to create a TextField large enough to type in a task. A TextField appears as you specified.
3. Point to the HorizScale icon in the Palette.
4. Press and hold down the Adjust button, and drag and drop the Scale to the right of `label2`.
5. Select `label2` and choose the Duplicate icon  in the icon bar. A third label appears, named `label3`.
6. Move `label3` to the right of the Scale, as illustrated in Figure 1-21.
7. Move and resize the TextField, Scale, and Label until your interface appears as shown in Figure 1-21. Make sure to stretch the Scale taller as shown.

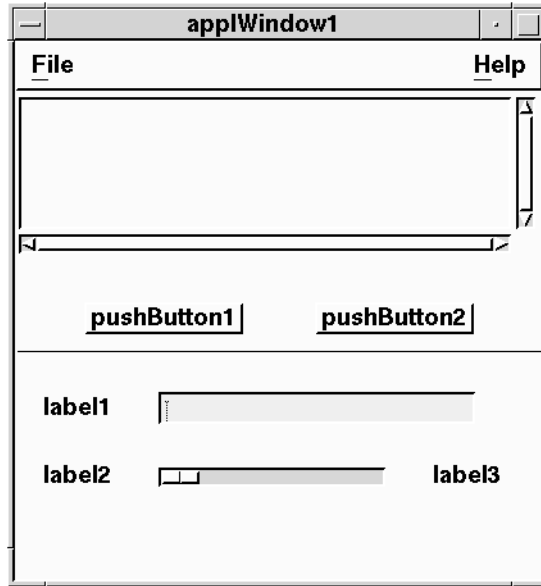



Figure 1-21 Your Interface with Text Field, Scale, and Label Added

Adding the Last Three Push Buttons

Now you will add the last three PushButtons to the bottom of the Edit Area. To save time, you will drag and drop the first PushButton, and then duplicate the other two.

1. Point to the PushButton icon in the Palette, hold down the Adjust button, and drag and drop a PushButton to the bottom left corner of your window. A new PushButton is created, called `pushButton3`.
2. Press the Menu button and choose Selected Objects⇒Duplicate or the Duplicate icon  to create a second PushButton, called `pushButton4`.
3. Repeat step 2 to create a third PushButton, called `pushButton5`. Move, resize, align, and arrange your three new PushButtons until your interface appears as shown in Figure 1-22. Don't be concerned whether the entire PushButton name shows in the last three PushButtons. You will change their names in the next step.
4. Select File⇒Save Project or click on the Save Project icon to save your work.

You have now finished creating all the objects in your Application Window.

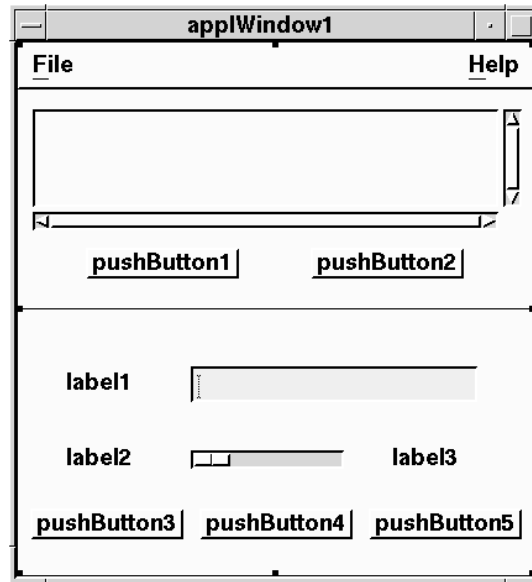


Figure 1-22 Your Interface with All Objects Added

Applying Constraints

You can define and apply constraints to the objects in your application window using the Constraint Editor. The Constraint Editor allows you to create interfaces which maintain proportion perfectly when resized.

As an introduction to the Constraint Editor, you will apply Dimension constraints to the ScrolledList object in your application window. The Dimension constraint allows you to constrain an object such that during subsequent move or resize operations, the position of the selected edge remains fixed at a proportionate distance from the left (y-axis) or top (x-axis) edge of the form.

1. With an object selected, press the Menu mouse button, and choose Selected Objects⇒Tools⇒Constraint Editor.

The Constraint Editor window appears with your interface loaded, as illustrated in Figure 1-23.

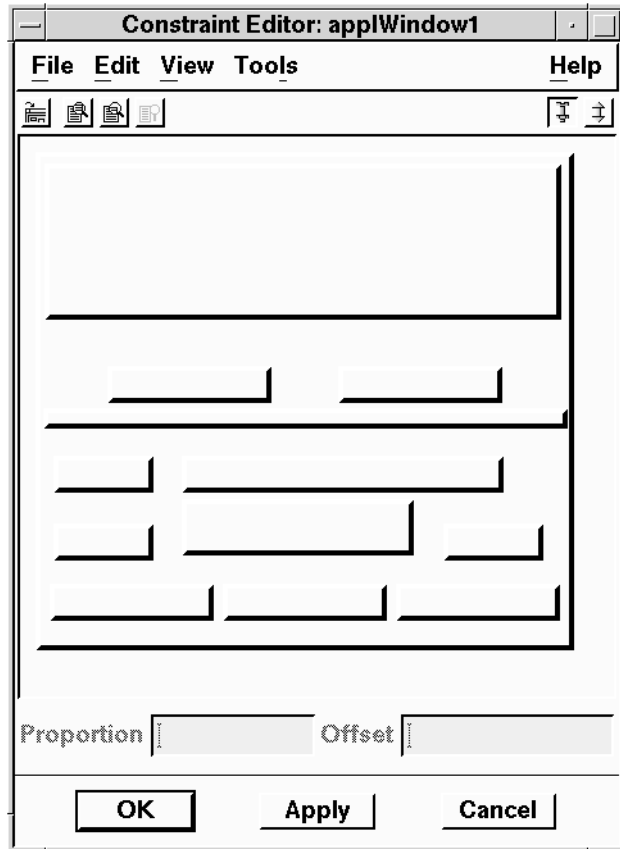



Figure 1-23 Constraint Editor Window with ToDo Interface

2. Select the Dimension icon  on the icon bar or Tools⇒Dimension from the menu bar.
3. Position the mouse pointer on the left edge of the ScrolledList object. When the pointer is over a valid destination edge, the edge of the ScrolledList will be shown highlighted.
4. Press the Select mouse button. The applied constraint is depicted graphically by an arrowhead linking the selected object to the parent.
5. Repeat steps 3 and 4, applying dimension constraints to the remaining three edges of the ScrolledList.

Your application window should appear as illustrated in Figure 1-24.

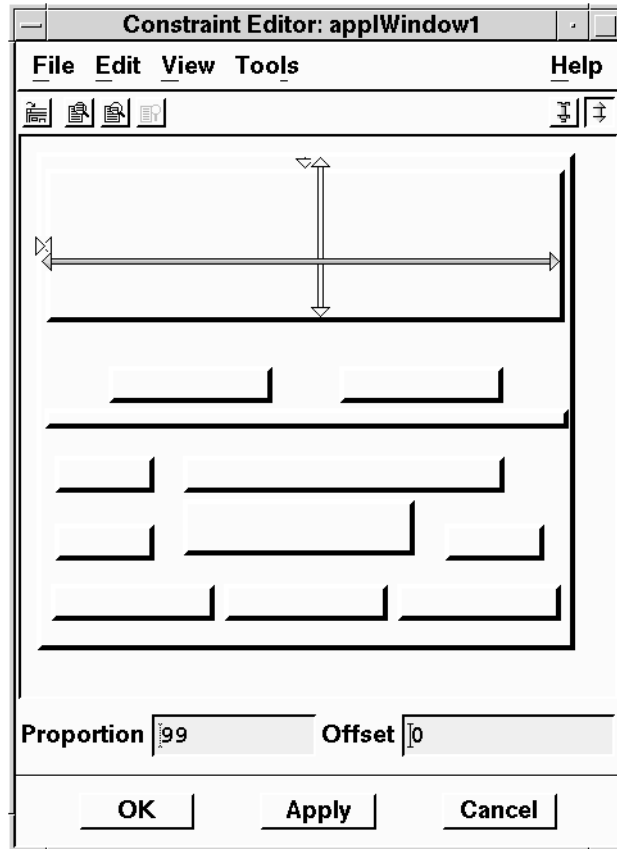




Figure 1-24 Scrolled List with Dimension Constraint

6. Close the Constraint Editor by selecting File⇒Close from the menu bar.

To demonstrate the effect of the dimension constraint, you will return to the application window and attempt to resize it.

1. From the Icon Bar, select the Test Mode icon .
2. Position the mouse pointer on the lower left corner of the interface.
3. Stretch the outline of the application window to a larger size, then release the mouse button.

The window reappears larger. Notice that the ScrolledList maintains a proportionate area of the application window as you stretch and shrink the form, while the objects without constraints do not.

4. Return to Design Mode by selecting its icon  from the menu bar.
5. Click on the application window to select it.
6. Select Edit⇒Recreate from the Project Window menu bar.

The interface reverts to its original size.

Note: To demonstrate the behavior of an interface with full constraints, load the project `uimx_directory/contrib/ToDoList/ToDo.prj`.

Applying constraints to your interfaces ensures that the end user is presented with a refined interface that behaves as expected when resizing. For a more detailed description of the Constraint Editor, refer to Chapter 7, “Adding Constraints”.

Step #6: Changing Object Properties

You have created all the objects you need for your To Do project. The next step is to refine your project by changing certain *properties* of these objects. A *property* is a value assigned to an object to determine its size, shape, color, font, behavior, or other characteristics.

All objects in the novice mode of UIM/X have several core properties, such as Height, Width, Background and Foreground colors, and X and Y pixels from the edge of the display (for a window) or the edge of the window (for an object). Each object has a further set of certain properties unique to itself. For more details on specific properties, see Appendix B, “Object Properties”.

You can view and change all the available properties for an object using the built-in Property Editor. For example, each new object you create is given a generic label, such as `pushButton1`. You can change these labels with the Property Editor to make them more meaningful.

From the Property Editor, you can also open most of the other specialized editors provided, including the Color Viewer, Color Editor, Color Map, Font Viewer, and Icon Viewer. These specialized editors help you quickly and easily compose the special strings required for setting a color, font, or icon name.

For more details, see Chapter 4, “Viewing and Changing Properties”, and Appendix B, “Object Properties”.

Now you will open the Property Editor and change several properties for certain objects. You will change the PushButtons and Labels to have more meaningful names, adjust the HorizScale to work better in your project, and

BUILDING YOUR FIRST PROJECT

Step #5: Adding Objects to Your Interface

change the name of `appWindow1`. You will also explore the Color Viewer and Font Viewer. Making all these changes will take a fraction of the time it would otherwise take to program them by hand.

Opening the Property Editor

1. Double-click on `pushButton1`.

The Property Editor appears, loaded with the properties for `pushButton1`, as shown in Figure 1-25.

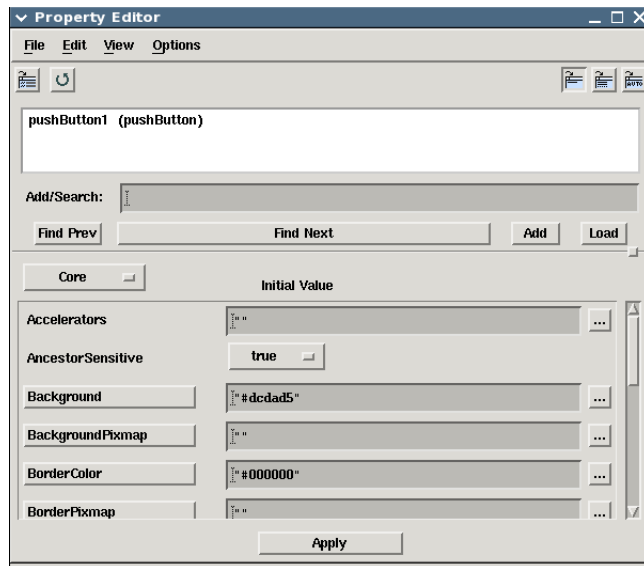


Figure 1-25 Property Editor Loaded with `pushButton1`

Changing the First Push Button Label

While you change the label for `pushButton1`, you will make a deliberate mistake to see how the program handles it.

1. With `pushButton1` loaded in the Property Editor, scroll through the properties until you reach `labelString`.

Notice that all properties are listed in alphabetical order.

Notice that the default `labelString` is "`pushButton1`", the same text that appears in your interface. You can change this label by changing this string.

2. Double-click in the `LabelString` text field to select all the text, and press the Backspace key to delete it.

Now you can enter an invalid value to see how the program handles it.

3. Type in `Edit Task` with no quotation marks.
4. Click Apply.

UIM/X checks all values before applying them to an object.

The error message shown in Figure 1-26 appears.

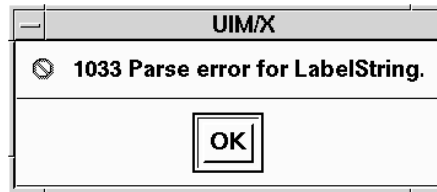


Figure 1-26 Error Message Dialog Box

5. Click OK.

The error message disappears. Notice that an “X” appears beside the property that caused the error, as shown in Figure 1-27.



Figure 1-27 “X” Marks an Invalid Property Value

6. To correct this error, click in the `labelString` text field, and type a quotation mark (") before and after the entry, as in `"Edit Task"`.
7. Click Apply.

Now `pushButton1` reappears with its new label, as shown in Figure 1-28.

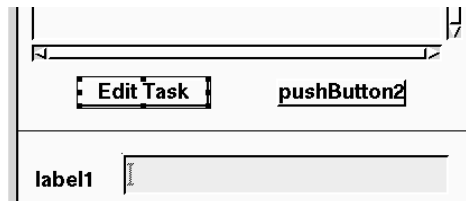


Figure 1-28 `pushButton1` with its New Label

Changing the Other Labels

Now repeat the same process for all the other `labelStrings` that need changing. In all, there are four more `PushButtons` and three `Labels` to change, as listed in Table 1-2.

BUILDING YOUR FIRST PROJECT*Step #5: Adding Objects to Your Interface*

1. Double-click on the next object to change.
2. When that object is loaded into the Property Editor, double-click in its `labelString` text field.
3. Type in the new `labelString` from Table 1-2. Remember to type in both quotation marks.
4. Click Apply in the Property Editor.
This applies your changes to the object.
5. Repeat steps 1 through 4 until all the labels are changed.

Table 1-2 `labelString` Values to Change

Object Name	labelString
<code>pushButton2</code>	"Delete Task"
<code>pushButton3</code>	"Add"
<code>pushButton4</code>	"Replace"
<code>pushButton5</code>	"Clear"
<code>label1</code>	"Task"
<code>label2</code>	"Priority"
<code>label3</code>	"Medium"

Since they now contain strings of different lengths, your two Labels may no longer appear to line up the same.

6. If required, move and resize `label1` and `label2`, using the Adjust button and the compass and resize pointers, until they line up with one another.

At this point, your interface should appear as shown in Figure 1-29.

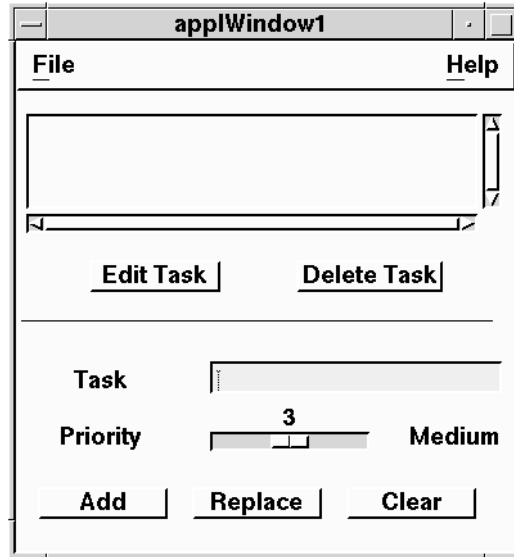


Figure 1-29 Your Interface with all Label Strings Changed

Changing the Scale Properties

The Scale is used to set the priority of each task in your To Do List. By default, the Scale's value ranges from 0 to 100, and is not shown. The priority for a task should be a much smaller range, perhaps between 1 and 5. A task with a priority of 0 doesn't make sense. You should display this value with the Scale to give proper feedback in your interface. You will make these changes now.

1. Double-click on the HorizScale to load it into the Property Editor.
2. Scroll through the Scale's properties, and notice how they are different from other objects. Notice that the `Maximum` value is 100 and the `Minimum` is 0.
3. Click on the option menu for `showValue`, and change it to `True`.
4. Click `Apply`, and notice that the current value of the Scale (0) is now shown.

Note: If you don't see the 0 above your Scale, turn back to Figure 1-21 and resize your Scale tall enough to contain the value.

5. Double-click in the `Maximum` text field and type in 5.

BUILDING YOUR FIRST PROJECT

Step #5: Adding Objects to Your Interface

6. Double-click in the `Minimum` text field and type in 1.
The `Value` is still set to 0, or less than your new `Minimum`.
If you click `Apply` at this point, you will get an error message.
7. Double-click in the `Value` text field and type in 3, midway between the `Minimum` and `Maximum` values.
8. Click `Apply`.
Notice that the slider moves to the midway point on the `Scale` and a 3 appears, as shown in Figure 1-30.

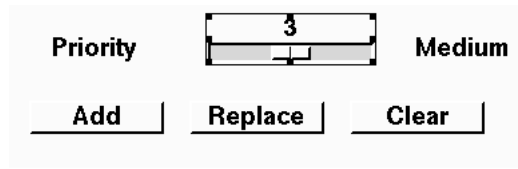



Figure 1-30 Horizontal Scale with its New Properties

Changing the Application Window Name

One final touch is to change the name of your Application Window.

1. Click on the Application Window to select it.
2. Press the `Menu` mouse button to bring up the `Selected Objects` popup menu and select `Tools⇒Property Editor`.
Your Application Window is loaded into the `Property Editor`.
You can load any other object the same way.
3. Scroll through the window's properties, and notice that both `IconName` and `Title` default to `applWindow1`.
4. Double-click in the text field for `Title`, and type in "To Do List".
5. Click `Apply`.
Notice how your window reappears with `To Do List` in its title bar.
At this point, your interface should appear as shown in Figure 1-31.
6. Select the `Save Project` icon  in the `Project Window` to save your work.

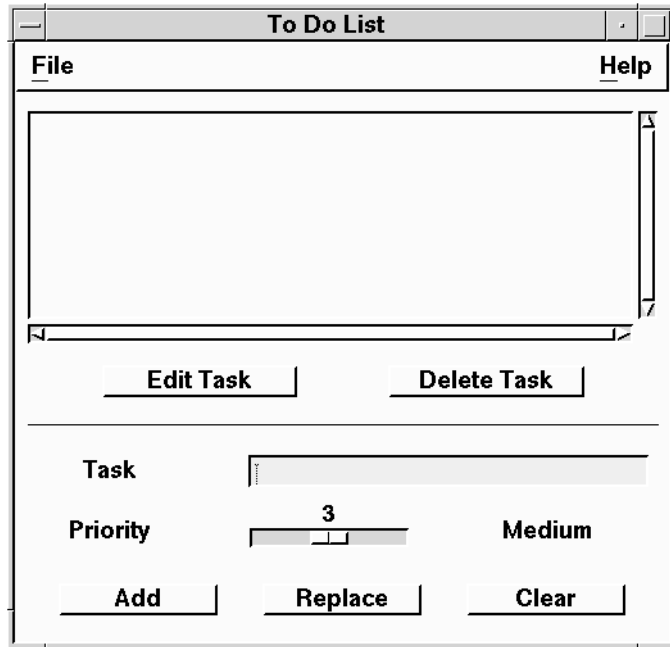


Figure 1-31 Your Interface to this Point

Changing Object Colors

The colors in your interface are automatically set to harmonize with one another. Each object shares the same background and foreground colors. All text is shown in the same color, black. So you have no pressing reason to change the color of any objects in your interface. But if you did want to change the color of an object, perhaps to highlight it, you do so with the Color Viewer and the Color Editor, which you open through the Property Editor.

Note: You can use five available properties to set object colors:

- Background controls the background behind the text in an object
 - Foreground controls the text displayed in an object
 - ListBackground controls the text area and scroll bars in a ScrolledList
 - MenuBackground controls the color for pull-down menus
 - TextBackground controls the text area and scroll bars in a Scrolled Text
-

1. Double-click on the TextField beside Task to load it into the Property Editor.

BUILDING YOUR FIRST PROJECT*Step #5: Adding Objects to Your Interface*

- Then click on the Background button.

The Color Viewer appears, as shown in Figure 1-32.

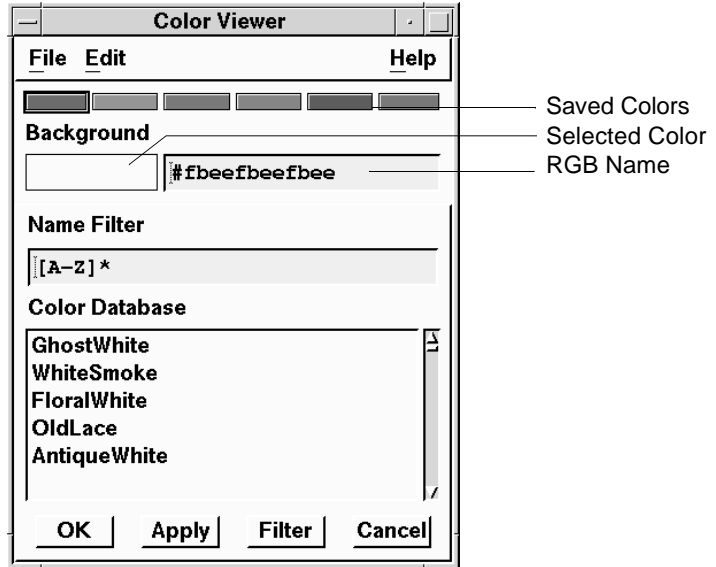


Figure 1-32 Color Viewer

- Move the Color Viewer to one side of your other windows.
If you need more space on your screen, choose File⇒Close in the Palette. You can re-open the Palette by choosing Tools⇒System Palette in the Project Window.
- Click on the saved colors across the top of the Color Viewer.
Notice that each time you click on a new color, the selected color and its RGB name change to display that color.
- Click on any of the color names from the Color Database.
Each time you click on a new name, the selected color and its RGB name change. You can use any color in the Color Database in your interfaces.
- Choose the Edit⇒Edit Color command from the Color Viewer.

The Color Editor appears, with the last color you clicked shown below the label Working and Original Color Samples, as shown in Figure 1-33. These two samples start out as the same color. Then, as you edit the Original Color, the Working Color changes to reflect your changes.

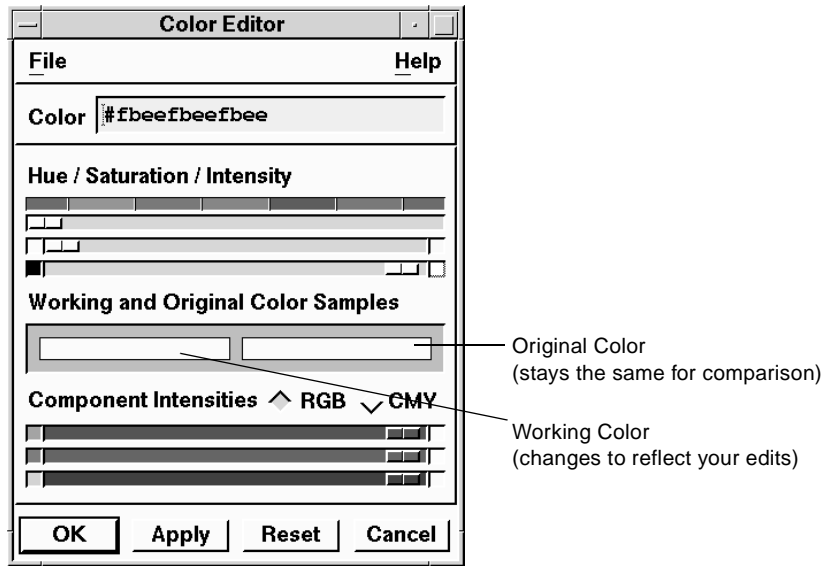


Figure 1-33 Color Editor

7. Move the Color Editor to one side of your other windows.
 If you need more space on your screen, choose File⇒Close in the Palette.
8. Use any or all of the six scales to mix a new color. The Working Color on the left changes as you move the sliders.
9. When you finish mixing your new color, click OK in the Color Editor.
 The Color Editor disappears, and the color you mixed becomes selected in the Color Viewer. Notice its new RGB string.
10. Click on the currently-selected color among the six saved colors at the top.
 Your new color replaces the selected color you clicked on.

Note: Your new color is saved for the duration of your current session. When you close the Color Viewer, your new color is not saved.

BUILDING YOUR FIRST PROJECT

Step #5: Adding Objects to Your Interface

11. Click Apply in the Color Viewer.
Notice the RGB string for your new color becomes the new value for the Background property. Your change is not yet applied to the object.
12. Click Apply in the Property Editor.
The background of the TextField changes to your new color.
The results may not harmonize very well with the rest of your interface. You can easily go back to the original color.
13. Choose Edit⇒Grab Color from the Color Viewer.
Notice that the pointer changes to the “grab” pointer.
14. Click on any object in your interface with the color you want to go back to.
Notice that the selected color changes to the color you clicked on.
15. Click OK in the Color Viewer.
The Color Viewer disappears.
16. Then click Apply in the Property Editor.
The background color of the TextField changes back to match the rest of your interface.

Changing Object Fonts

All the text in your interface is displayed in your default font, as set in your Application Defaults. For visual consistency, use as few fonts as possible in your interface. You may not want to change any fonts in your project. But if you did want to change a font, perhaps to emphasize a certain Label, you do so with the Font Viewer, which you open through the Property Editor.

Note: You can use five available properties to set fonts:

- `ButtonFontList` controls the font for all PushButtons in a dialog box
 - `FontList` controls the font for the text in any object in a window
 - `LabelFontList` controls the font used for all Labels in a dialog box
 - `MenuFontList` controls the font used for pull-down menus
 - `TextFontList` controls the font used for all TextFields in a dialog box
-

1. Double-click on the Edit Task PushButton to load it into the Property Editor.
2. Then click on the FontList button.
3. The Font Viewer appears, as shown in Figure 1-34.

4. Move the Font Viewer to one side of your other windows.

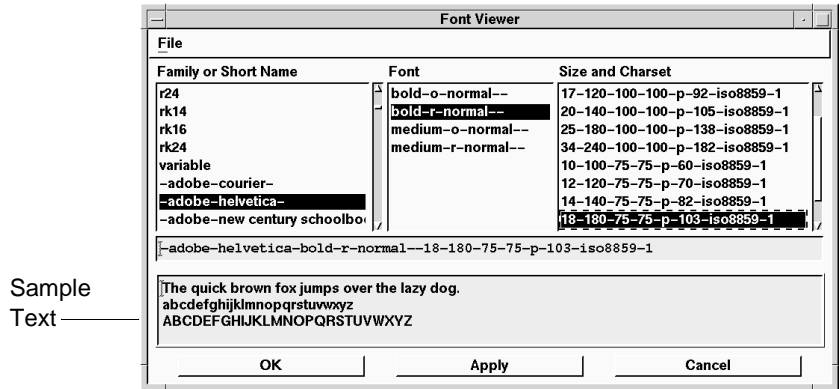


Figure 1-34 Font Viewer


5. Click on some names in the Family or Short Name list.
Notice that each time you click on a new font, the sample text changes.
6. Click on a combination of Family or Short Name, Font, and Size and Charset that looks bigger and bolder than the other fonts in your interface.
7. Click Apply in the Font Viewer.
8. Click Apply in the Property Editor.
The label for pushButton1 reappears in your selected font.

Note: To revert to the original font, reselect the original font in the Font Viewer and then go on to step 10.

9. Double-click on each PushButton in turn, and reset its font to the new font.
Remember to press Apply in both the Font Viewer and the Property Editor.

Note: As you load the objects in turn, you can ignore the messages about “loss of changes that you have made but have not applied”.

10. When you finish changing fonts, press OK in the Font Viewer.
The Font Viewer disappears.

11. Choose File⇒Save Project or select the Save Project icon  on the Project Window icon bar to save your work.
12. Close the Property Editor.

Step #7: Revising the Menu Bar

Like most GUIs, your To Do List interface includes a menu bar. As shown in Figure 1-35, a pull-down menu consists of the following parts:

- One *Cascade Button*, used to open the menu
- One *Pane*, used for the background of the menu
- Two or more *Items*, used for the menu options
- One *Mnemonic* per item (if desired)
- One *Accelerator* per item (if desired)

When you create a pull-down menu, it is automatically assigned a Cascade Button. You can give the Cascade Button any label, which is displayed in the menu bar. Pointing to the Cascade Button and pressing the Select button pulls down the associated menu. When a pull-down menu is open, it displays a background Pane. Each Pane is a RowColumn object that contains a list of Items or menu options.

A Mnemonic is a character you can press on the keyboard to open a pull-down menu or activate a menu option, such as E for Edit or C for Cut. A mnemonic must occur in the label for the pane or item, since it is underlined in the label.

An Accelerator is a keyboard shortcut you can use to activate a menu option, such as Ctrl+X for Cut or Ctrl+C for Copy. An accelerator need not occur in the label for the pane or item itself.

You can revise the existing menu bar in any Application Window using the built-in Menu Editor.

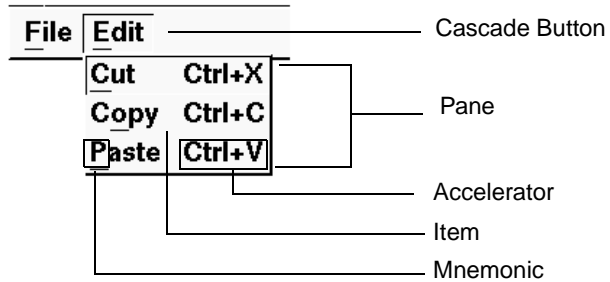


Figure 1-35 Parts of a Pull-down Menu

Now you will revise the existing menu bar in your To Do List window. You can add new panes and items, or change any of the available properties of the existing panes and items using the Menu Editor.

Adding Items to the File Menu

By default, the File menu has one item, File⇒Close. You will now add two menu items, File⇒Save and File⇒Exit, and rename File⇒Close to File⇒Open.

1. Double-click on the word File in the Menu Bar of the To Do List interface window.

The Menu Bar Editor appears, loaded with the menu bar for `applWindow1`, as shown in Figure 1-36.

Notice that `file_menurc1` is selected by default. This is the File menu.

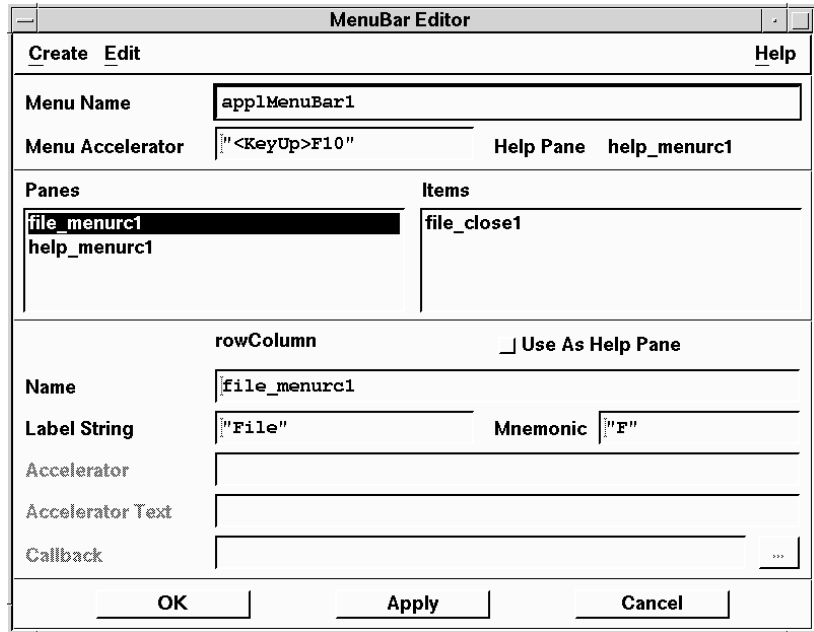


Figure 1-36 Menu Bar Editor

2. Click on `file_close1` in the Items area.
3. Choose the `Create⇒Item` command, and then choose `PushButton`.
A new item appears in the Items area, called `file_menusrc1_b2`.
4. Double-click in the `LabelString` text field and type in "Save".
5. Double-click in the `Mnemonic` text field and type in "S" for Save.
The `LabelString` and `Mnemonic` you enter will appear in the menu pane in your finished interface.
6. Double-click in the `Name` text field and type in `file_save`.
7. Click `Apply` in the Menu Bar Editor.
8. Repeat steps 3 through 7 to create another new menu item, originally called `file_menusrc1_b3`, with the properties listed in Table 1-3. Rename this item to `file_exit` as shown in Table 1-3. The `Accelerator` means you will be able to press `Alt+F4` to exit from your To Do List program.

9. Click on `file_close1` in the Items area, and revise its properties as shown in Table 1-3.
10. Click OK in the Menu Bar Editor.

Table 1-3 File Menu Items to Revise

	<code>file_close1</code>	<code>file_menurc1_b2</code>	<code>file_menurc1_b3</code>
Name	<code>file_open</code>	<code>file_save</code>	<code>file_exit</code>
LabelString	"Open"	"Save"	"Exit"
Mnemonic	"O"	"S"	"X"
Accelerator			"Alt<key>F4"
Accelerator Text			"Alt+F4"

Your menu items are now created, but they do *not* appear since you can not pull down the menus in Design mode. You can pull down the menus in Test mode, but selecting them will still have no effect, since you have not yet provided your menu items with any behavior.

Step #8: Adding Declarations and Callbacks

The “look” of your interface is now complete. The next step is to add its “feel”. You do this by typing in *callbacks* for various objects. A callback is a piece of code which make an object behave a certain way when a specific event occurs. You type in callbacks for objects with the Callback Editor, opened through the Property Editor. Eight callbacks are available in the Property Editor in the novice mode of UIM/X:

- `ActivateCallback` runs when you click on an object such as a `PushButton` or `TextField`
- `CancelCallback` runs when you click Cancel in a dialog box
- `CreateCallback` runs when an object is created; every object supports this callback so you can do any special processing before an object is realized
- `DragCallback` runs when you move a slide in a `Scale` object

BUILDING YOUR FIRST PROJECT

Step #8: Adding Declarations and Callbacks

- `HelpCallback` runs when you select the Help button in a File Selection Dialog or Message Dialog.
- `OKCallback` runs when you click OK in a dialog box
- `SingleSelectionCallback` runs when you select a line in a `ScrolledList`
- `ValueChangedCallback` runs when an object's current value changes.

You type in callbacks for menu items with a similar Callback Editor, opened through the Menu Bar Editor. One callback is available for every menu item; this callback runs when you select the associated menu item.

For more details, see Chapter 12, "Programming in UIM/X."

You can also use the Connection Editor to create callbacks quickly and easily. For more information on the Connection Editor refer to Chapter 6, "Adding Connections".

The Purpose of the PushButtons

The purpose of each `PushButton` should be clear from its name. In your To Do List project, there are five `PushButtons` as follows:

- `Edit Task` copies the selected task from the List Area to the Edit Area so you can modify it
- `Delete Task` deletes a selected task from the List Area
- `Add copies` a new task from the Edit Area to the List Area
- `Replace` replaces the selected task in the List Area with the new description and/or priority in the Edit Area
- `Clear` clears the current task from the Edit Area

In this step you will use the Declaration Editor to include application header files and declare a variable for the priority label, and the Callback Editor to add a callback for each of your `PushButtons` and each of your menu items. You add callbacks for objects like `PushButtons` with the Property Editor, while you add callbacks for menu items with the Menu Editor.

Adding Declarations

1. First you will select your Application Window and open the Declaration Editor by selecting Selected Objects⇒Tools⇒Declaration Editor.

The Declaration Editor opens, as shown in Figure 1-37.

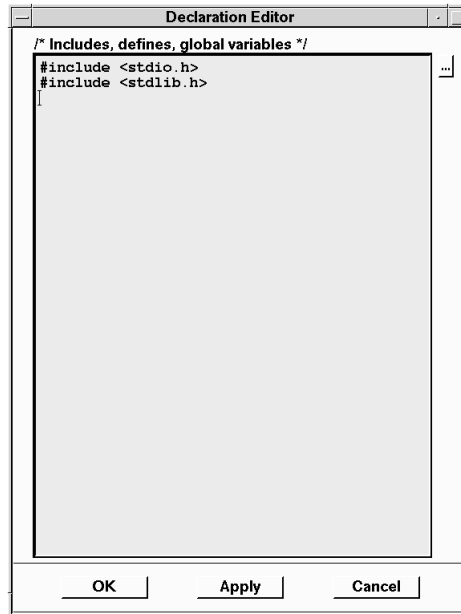


Figure 1-37 The Declaration Editor

2. Click in the Text Editor and type in the following immediately after the last statement:

```
char *priority_labels[5] = { "Very Low",  
                             "Low",  
                             "Medium",  
                             "High",  
                             "Very High" };
```

3. Click OK in the Text Editor to save your code.

The Declaration Editor closes.

You have now assigned corresponding labels to priority levels 1 through 5.

Adding Callbacks for the Push Buttons

To make a PushButton work, you provide it with an `ActivateCallback`.

1. Double-click on the `Edit Task PushButton` to load it into the Property Editor.
2. Click on the editor (...) button beside `ActivateCallback`.

The Callback Editor appears, with an empty text window ready for your `ActivateCallback`, as shown in Figure 1-38.

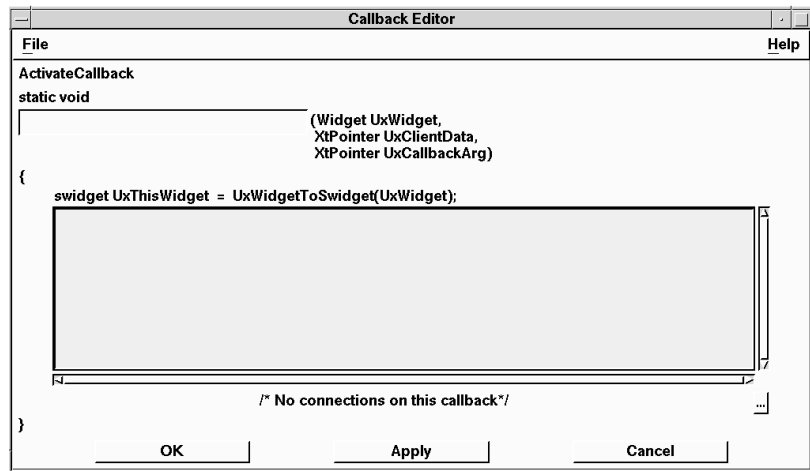


Figure 1-38 Callback Editor

- Click in the TextField, and type in the following code exactly as it appears:

```
int    *selpos, selcnt;
int    priority;
char   *taskLine;

if
(XmlListGetSelectedPos(UxGetWidget(scrolledList1),
                        &selpos,&selcnt)== True)
{
    taskLine =
    UxGetSelectedItems(scrolledList1);
    UxPutText(textField1,taskLine+6);

    priority = atoi(taskLine);
    UxPutValue(scaleH1, priority);
    UxPutLabelString(label3,
priority_labels[priority-1]);
}
```

This code does the following:

- Determines which task in the ScrolledList is selected
 - Extracts the selected task's priority and description
 - Copies the selected task's description into the Task TextField
 - Copies the selected task's priority into the Priority Scale
 - Assigns the appropriate text to the priority level label
- Click OK in the Callback Editor.

Note: When you click OK in the Callback Editor, your callback code is checked and stored with the associated object, but not run. You use the Test phase to test your callback code.

- Click Apply in the Property Editor.

If you made any errors typing the code, an error dialog box appears, an "X" appears beside the offending property in the Property Editor, and a message appears in the Messages Area of the Project Window, as you saw earlier.

BUILDING YOUR FIRST PROJECT*Step #8: Adding Declarations and Callbacks*

6. Clear up any error messages that appear, by checking your code against the sample code above.

Make sure to click Apply in the Property Editor.

7. Now double-click on the Delete Task PushButton to load it into the Property Editor.
8. Click on the editor (...) button beside ActivateCallback, and type in the following code:

```
Widget w;  
int *selpos, selcnt;  
  
w = UxGetWidget(scrolledList1);  
  
if  
(XmListGetSelectedPos(w, &selpos, &selcnt) == True)  
{  
    XmListDeletePos(w, *selpos);  
}
```

This code deletes the selected task, if any, from the ScrolledList.

9. Click OK in the Callback Editor.
10. Click Apply in the Property Editor.
11. Repeat steps 7 to 11 with the Add PushButton, typing in the following code:

```
int priority;  
XmString xms;  
char taskLine[256];  
char *task;  
  
priority = UxGetValue(scaleH1);  
task = UxGetText(textField1);  
  
sprintf(taskLine, "%3d : %s", priority, task);  
  
xms = XmStringCreateSimple(taskLine);  
XmListAddItem(UxGetWidget(scrolledList1), xms, 0)  
;  
XmStringFree(xms);
```


This code inserts the current priority from `scaleH1` and the description from `textField1` as a new task at the bottom of the `ScrolledList`.

12. Repeat steps 7 through 11 again with the `Replace PushButton`, typing in the following code:

```
int      *selpos,selcnt;
int      priority;
char     taskLine[256];
char     *task;
XmString xms;
Widget  w;

w = UxGetWidget(scrolledList1);
if
(XmListGetSelectedPos(w, &selpos, &selcnt) == True)
{
    priority = UxGetValue(scaleH1);
    task     = UxGetText(textField1);


    sprintf (taskLine, "%3d : %s", priority, task);
    xms      = XmStringCreateSimple (taskLine);
    XmListReplaceItemsPos(w, &xms, 1, *selpos);
    XmStringFree(xms);
}
```

This code replaces the currently-selected task in the `List Area` with the current task information from the `Edit Area`.

13. Repeat steps 7 through 11 one last time with the `Clear PushButton`, typing in the following code:

```
UxPutText (textField1, "");
UxPutValue (scaleH1, 1);
UxPutLabelString (label3, "Very Low");
```

This code clears the current information in the `TextField` and resets the `Scale` to 1 (the lowest priority).

14. Remember to click `OK` in the `Callback Editor` and then click `Apply` in the `Property Editor`.
15. Select `File`⇒`Save Project` or the `Save Project` icon  to save your work. You have now entered callbacks for all five `PushButton`s.

Adding Callbacks for Menu Items

Just as the purpose of each `PushButton` should be clear from its name, the purpose of each menu item should be readily apparent from its name. In your project, there are four menu items, as follows:

- `File⇒Open` is intended to open a saved file of tasks
- `File⇒Save` is intended to save your current list of tasks in an ASCII file
- `File⇒Exit` is intended to exit from the program
- `Help⇒About Application` is intended to display version information

To give a menu item its desired behavior, you enter a `Callback` using the `Text Editor`, which you open from the `Menu Bar Editor`. You will do that now.

1. Double-click on the `File` menu in the `Project Window`.

The `Menu Bar Editor` appears, loaded with the menu bar for the window.

2. Click on `file_exit` in the `Items` area.
3. Click on the `Editor (...)` button beside `Callback`.

The `Text Editor` appears. You can use it just like the `Callback Editor` in the `Property Editor`.

4. Click in the `TextField` and type the following code into the `Text Editor` to exit from the program:

```
exit(0);
```

This code handles the case when you explicitly exit from the program by choosing the `File⇒Exit` command.

You can also exit from the program by double-clicking the `Window` menu button. In this case, `applWindow1`'s `DeleteResponse` property handles this event.

5. Click `OK` in the `Text Editor`.
6. Click `Apply` in the `MenuBar Editor`.

7. Click on `file_open` in the Items area, and repeat steps 3 through 6, typing in the following code:

```
FILE *file;
char str[256];
Widget w;
XmString xms;

if ( ( file = fopen ("todo.out", "r" ) ) != NULL )
{
    w = UxGetWidget(scrolledList1);
    XmListDeleteAllItems(w);

    while ( fgets(str, sizeof(str), file ) != NULL )
    {
        /* remove trailing newline and add to the list */
        str[strlen( str ) - 1] = '\0';
        xms = XmStringCreateSimple(str);
        XmListAddItem(w, xms, 0);
        XmStringFree(xms);
    }
    fclose ( file );
}
```

This code loads the ScrolledList with the contents of the file `todo.out` from your current directory. The file `todo.out` is assumed to be a To Do List previously saved with your interface.

8. Click OK in the Text Editor.
9. Click Apply in the MenuBar Editor.

10. Click on `file_save`, and repeat steps 3 through 6, with the following code.

```
FILE *file;
char *taskList;
char *processList;

if ( ( file = fopen( "todo.out", "w" ) ) != NULL)
{
    taskList = UxGetItems(scrolledList1);
    if (taskList)
    {
        /* Replace commas by newlines and write out the
list */
        processList = taskList;
        while (*processList)
        {
            if (*processList == ',')
            {
                *processList = '\n';
            }
            processList++;
        }
        fprintf(file,"%s\n",taskList);
        fclose (file);
    }
}
```

This code saves your current To Do List as an ASCII file called `todo.out` in your current directory.

11. Click OK in the Menu Bar Editor.
12. Choose `File⇒Save Project` or select the Save Project icon to save your work.

You have now entered callbacks for three menu items. You can leave the `Help⇒Version` item without any callback for now.

Connecting the Priority Scale and Priority Level Label

Finally you will create a connection between the priority scale and priority level label such that the appropriate label string is displayed when the value on the scale changes. You will accomplish this using the Connection Editor.

1. Position the mouse pointer on the priority scale (`scaleH1`).
2. Press and hold the Shift key and the Select mouse button simultaneously and drag the mouse pointer from the priority scale to the priority level label (`label3`).

A line connecting the priority scale to the priority label is temporarily drawn on your interface and the Connection Editor is displayed, as illustrated in Figure 1-39. All available callbacks for `scaleH1` and methods for `label3` are listed in their respective scrolled lists.

By using this “shift-click” method, you will invoke the Connection Editor with source and target automatically loaded.

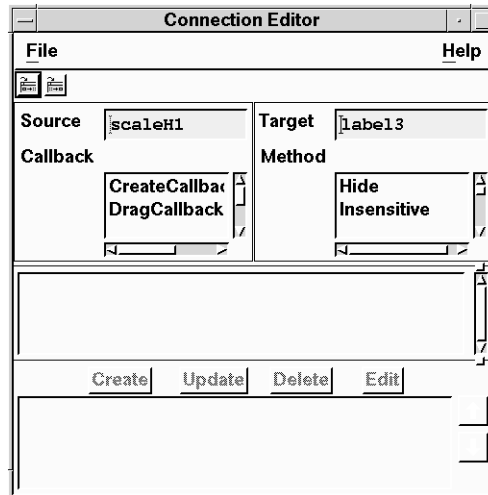


Figure 1-39 The Connection Editor with Source and Target Loaded

3. From the Callback list, select `ValueChangedCallback`.
4. From the Method list, select `SetLabelString`.

The `LabelString` argument appears in the Arguments area of the Connection Editor. The argument’s value is initially an empty string (“”).

BUILDING YOUR FIRST PROJECT*Step #8: Adding Declarations and Callbacks*

- Open the Text Editor beside the argument value, delete the quotation marks, and type in the following code:

```
priority_labels[UxGetValue(scaleH1) - 1]
```

This code assigns the appropriate string to the priority level label according to the value selected on the priority scale.

- Click Ok to apply your change and close the Text Editor.
- Click the Create button in the Connection Editor.

The connection appears in the List/Update area of the Connection Editor.

- From the Callback list, select DragCallback.
- Click the Create button once more.

A new connection using the same method and argument as was specified for the previous connection is created.

The Connection Editor should now appear as shown in Figure 1-40.

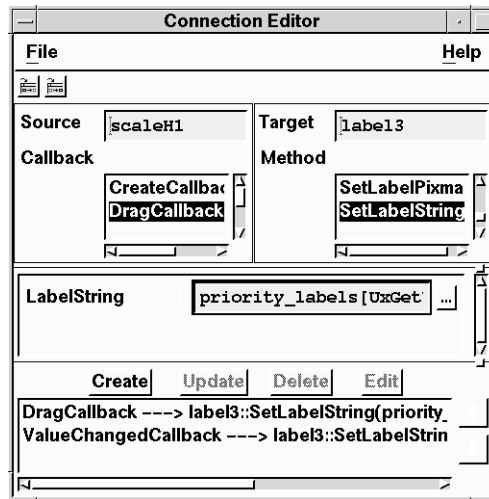


Figure 1-40 The Connection Editor with Callbacks Added

- Select File⇒Close from the Connection Editor’s menu bar.
The Connection Editor closes.

You have now defined the functionality for all but the Help menu. This you will do in the next step.


Step #9: Adding the “About” Dialog Box

In this step, you will assign a meaningful caption to the Help menu item and add the necessary functionality to pop up a dialog box. You will accomplish this by:

1. Creating a `MsgBoxDialog`,
2. Using the Property Editor to change the `MsgBoxDialog`'s `MessageString` property and set the `CreateCallback`,
3. Creating an instance of the `MsgBoxDialog`, and
4. Using the MenuBar Editor to change the Help menu item's `LabelString` and establish a behavioral connection between the Help menu item and the instance of the `MsgBoxDialog`.

By doing so, you will have defined the following functionality at run time: when `Help⇒About ToDoList` is selected from the menu bar, a `MsgBoxDialog` displaying “To Do List Version 3.0” will pop up.

First you will create a `MsgBoxDialog`.

1. Click on the `MsgBoxDialog` icon in the Windows category of the Palette. The mouse pointer changes to an upper-left corner shape .
2. Press and hold down the Select button where you want the top left corner of the dialog box to appear on your screen.
3. While holding down the Select button, drag the mouse down and to the right to draw a rectangle about 4 inches wide by 3 inches high.
4. Release the mouse button.

A `MsgBoxDialog` appears where you specified in your interface, as illustrated in Figure 1-41.

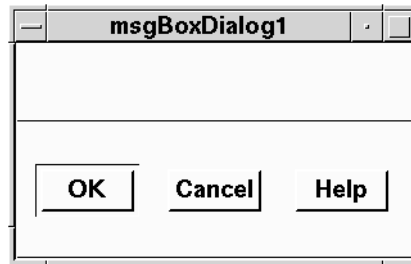


Figure 1-41 `MsgBoxDialog`

Next you will open the Property Editor to change properties of the `MsgBoxDialog` and set the callback behavior.

1. Double-click on the `MsgBoxDialog` to open the Property Editor.
The Property Editor opens.
2. Set its `MessageString` property to “To Do List Version 3.0”
3. Set its `DialogTitle` property to “About To Do List”
4. Open the Callback Editor for the `CreateCallback` property and type in the following code, exactly as it appears:

```
XtUnmanageChild (XmMessageBoxGetChild (UxWidget,  
XmDIALOG_HELP_BUTTON));  
XtUnmanageChild (XmMessageBoxGetChild (UxWidget,  
XmDIALOG_CANCEL_BUTTON));  
UxPutDefaultPosition( UxThisWidget, "true" );
```

This code removes the functionality of the Cancel and Help buttons and ensures that the dialog appears at the default position.

5. Click on Ok to close the Callback Editor.
6. Click on Apply in the Property Editor to apply your changes.
7. Close the Property Editor.

Your dialog box should now appear as illustrated in Figure 1-42.

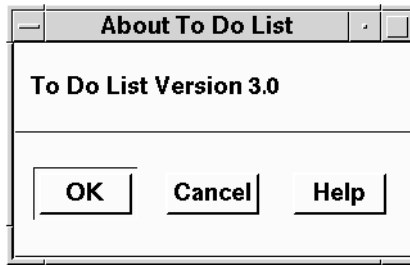



Figure 1-42 Updated MsgBoxDialog

Next you will create an instance of the MsgBoxDialog component.

1. Click on the MsgBoxDialog to select it.
2. Select Selected Objects⇒Instance.

The mouse pointer changes to an upper-left corner shape .

3. Drag and draw the instance of the MsgBoxDialog on the To Do List interface. An instance of `msgBoxDialog1`, called `msgBoxDialog1Instance1` is created. It is this instance that will be used as the target object for the behavioral connection with the Help menu item.

Note: The newly created instance is not drawn on your screen. However, it is visible and selectable from the Browser.

For more information on working with instances, refer to Chapter , “”.

Next you will open the MenuBar Editor to change the Help menu item’s `LabelString` and define its functionality.

1. Double click on `applWindow1`’s menu bar to open the MenuBar Editor.

The MenuBar Editor opens, as illustrated in Figure 1-43.

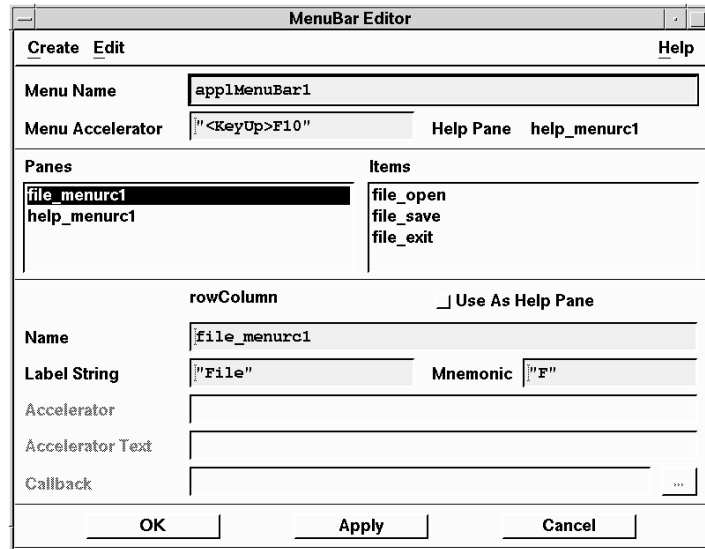


Figure 1-43 The MenuBar Editor

2. From Panes, click on help_menurc1.
3. From Items, click on help_about1.
4. Click in the LabelString field and change the value to “About To Do List”.
5. Open the Text Editor for the Callback field and type in the following code:

```
VisualInterface_Manage(msgBoxDialog1Instance1,
&UxEnv);
```

This code establishes the functional connection between the Help menu item and the instance of msgBoxDialog1.

6. Click on Ok to close the Text Editor.
7. In the MenuBar Editor, click on Ok to apply your changes and close the MenuBar Editor window.
8. Save your project.

You have now completed defining the functionality of your interface.

Step #10: Testing Your Interface

Your interface is now ready to test. UIM/X provides a Test mode that enables you to test your interface instantly, without having to generate code, compile and link the program.

1. Click on the Test icon in the Project Window icon bar, as shown in Figure 1-44.



Figure 1-44 Test Toggle Selected

The Palette, the Property Editor, and any other open editors disappear from your screen.

2. Try all the different features of the interface:
 - Click on the PushButtons to enter, modify and delete tasks.
 - Use the Scale to set their priorities.
 - Pull down the File and Help menus, and try each of their options.
 - Notice that if you try to exit the program, the Application Window's default `DeleteResponse` property intercepts the call and shows a message instead of allowing you to exit.
 - Pop up the Help About dialog.
3. When you are done testing your interface, click on the Design icon. The Palette and the Property Editor reappear on your screen.

Note: If anything in your interface did not work properly in Test mode, go back to the point where you created or changed that object, and make sure that you did everything correctly.

Do not type any commas in your tasks; commas interfere with the relatively simple formatting of the tasks in the List Area.

Step #11: Generating Code

Now that your interface is designed and tested, you can generate its code.

1. Choose the File⇒Generate Project Code command from the Project Window or select the Generate Project Code icon  from the icon bar.

The Generate Code Options dialog box appears, as shown in Figure 1-45. Notice that the default name `ToDo.mk` is already present for the makefile.

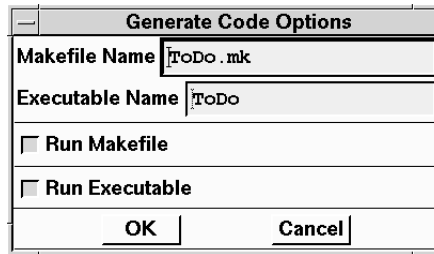


Figure 1-45 Generate Code Options Window

2. Click the Run Makefile and Run Executable check buttons.
3. Click OK.

In a few seconds, your code is generated. Then the code is compiled and linked and the executable is run. You can follow this process by watching the Messages area of the Project Window.

When a message says Done, the code generation and compilation is complete.

Note: You can do all this in one step simply by clicking the Run icon.

Exiting from UIM/X Novice Mode

To exit from UIM/X novice mode, follow these steps.

1. Choose the File⇒Exit command from the Project Window.

Note: If the Project Window is active, you can also exit from the program by pressing Alt + F4, or Alt+F, then X, or by double-clicking on the Window menu button.

2. Click OK to confirm that you want to exit.

Your interface, any open editors, the Palette, and the Project Window all disappear from your screen.

Where Do You Go From Here?

Building this sample interface has shown you all of the important features of the novice mode of UIM/X. You can probably think of many ways to extend this sample, a few of which are suggested below. It's easy to think of new features to add to an existing program; the trick is to make sure these are features your end users really need.

Add More Feedback and Error-Checking

For instance, you could build more feedback into the interface, so that when the end user clicks Delete, a dialog box pops up to ask if they are sure they really want to delete.

You could build in more error-checking, so that when the end user selects File⇒Exit, you check whether the current To Do List has been saved since it was last changed. If not, you could pop up a warning dialog box, or automatically save before you exit.

Add More Functions and Control For the End User

You could add more code to automatically sort the To Do List by priority, so the end user could see their most important tasks at the top of the list.

You could create Open and Save FileSBoxDialogs so the end user could control where their To Do List files are saved and what they are named.

Add More Constraints

You could use the Constraint Editor to add more constraints to your interface, so the end user is presented with an interface that resizes properly.

Add More On-line Help

You could build in some on-line help, by creating more Help items and popping up information dialog boxes with On-line information.

Build Your Own Projects

You can also start to use the novice mode of UIM/X to work on some of your own interface-building projects.

BUILDING YOUR FIRST PROJECT*Where Do You Go From Here?*

And if you are curious about the differences between the novice mode of UIM/X and full UIM/X, see Chapter 13, “Beyond the Basics”. This chapter explains why you might want to consider moving up to the full range of features available in UIM/X.

Basics of UIM/X Novice Mode

2

Overview

This chapter describes all the basics of using the novice mode of UIM/X, including the Project Window, the Palette, and the pop-up menus provided.

What Is UIM/X Novice Mode?

UIM/X is a leading graphical user interface (GUI) builder for UNIX. It includes a full suite of tools for building sophisticated interfaces. UIM/X novice mode is a simpler introduction to UIM/X. After you learn the basics, you can move on to the full-fledged version of UIM/X and all its advanced features.

Using UIM/X novice mode, you can:

- Drag and drop “objects” to create interfaces
- View and change object properties, such as their color, size, or font
- Assign “behavior” to most objects
- Create associated menus and dialog boxes
- Automatically generate source code
- Test the behavior of the objects in your interface

Using UIM/X novice mode to develop user interfaces can save programmers and non-programmers alike many hours of time, effort, and expense.

Starting UIM/X Novice Mode

To Start UIM/X novice mode

1. Open a terminal window, if you do not already have one open.
2. At the UNIX shell prompt, type

```
uimx -novice -language ansic &
```

Note: The `-language` option instructs UIM/X to use ANSI C mode. By default UIM/X starts in C++ mode.

If your `PATH` variable does not provide the full path to the UIM/X executable, you will have to specify it when you run the program:

```
/usr/uimx2.9/bin/uimx -novice -language ansic &
```

After a brief pause, the UIM/X copyright window appears.

3. Click OK to clear the copyright window.

In seconds, the Project Window appears, as shown in Figure 2-1.

The Palette also appears; it is described later on in this chapter.

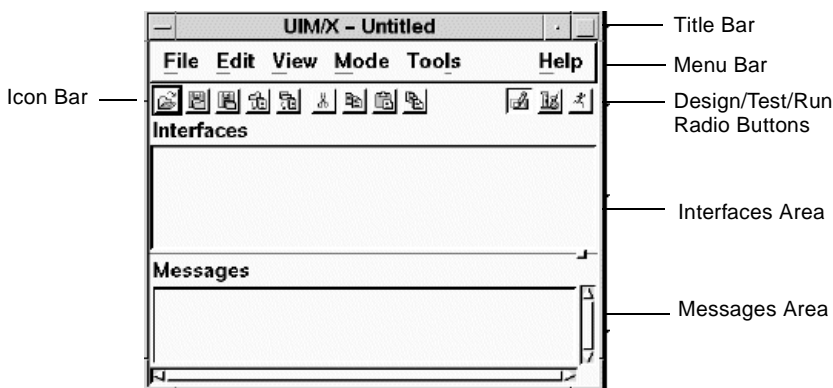


Figure 2-1 Main Areas of the Project Window

Parts of the Project Window

Here are some brief notes on each part of the Project Window:

- *Title Bar*: Displays the name of the project you are currently building.
- *Menu Bar*: Provides six pull-down menus: File to open and save an interface and generate code; Edit to work with objects; View to show and hide interfaces and projects; Mode to select Design, Test, or Run mode; Tools to open the various editors and redisplay the Palette; and Help to see online help.
- *Icon Bar*: Provides quick access to some of the most commonly used Menu Bar commands. From left to right, these icons are: Open, Save Project, Save Interfaces, Generate Project Code, Generate Interface Code, Cut, Copy, Paste, and Duplicate. Bubble help is provided for all icons.

- *Design/Test/Run Radio buttons*: Switch between Design to design a dialog box; Test to test its behavior; and Run to generate, compile, and run your code. Design is the default setting.
- *Interfaces Area*: Displays an icon for each interface in your current project.
- *Messages Area*: Displays any messages generated by UIM/X. You can use a pop-up menu here to clear these messages.

Using the Project Window

You can move, resize, minimize, maximize, or close the Project Window, just like any other window. You can also adjust the size of its panes with the sash. Once the novice mode of UIM/X is running, you can either create an interface, or load an existing interface (.i) file created with the novice mode of UIM/X.

From the Project Window, you can:

- Get online help
- Switch between Design, Test, and Run mode
- Save your work
- Open the editors
- Hide and show interfaces
- Exit from the program

Remember that an *interface* is a window or dialog box you are building in UIM/X. Each of these tasks is described briefly in the following sections.

Getting Online Help

You can get an online introduction to UIM/X by choosing Help⇒Overview from the Project Window. You can see a copyright box showing which version of UIM/X you are using by choosing Help⇒About UIM/X.

Online help with respect to the icons in the Project Window icon bar is presented in the form of “Bubble” help. That is, when the mouse pointer is positioned directly over any of the icons, the icon’s purpose is displayed in bubble form, as depicted in Figure 2-2.

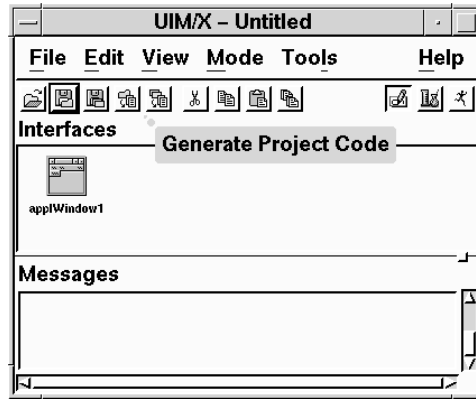


Figure 2-2 Bubble Help for Icons

Using the Design, Test, Run Icons

You can use UIM/X for the three different modes of GUI development: Design, Test, and Run. To switch modes, click the icons in the icon bar of the Project Window, as shown in Figure 2-3. Design is the default mode, since you first have to design an interface before you can test or run it.

Using Design, Test, and Run, there is no need to open another window; you can do everything from within UIM/X.



Figure 2-3 Design, Test, Run Icons

Using Design Mode



You use Design mode to create an interface, add objects to it, then size and position each object. In Design mode, your interface appears on your screen exactly as it will be seen by the end user. You can adjust the properties of any object with the Property Editor, and provide callbacks to give your objects the behavior you need.

Using Test Mode

You use Test mode to test your interface and make sure it works correctly.

Using Run Mode

You use Run mode to generate, compile, and run the code for your project. The novice mode of UIM/X generates code according to the language mode selected. Once your GUI is built and tested, you can generate the code for it


from the Project Window. To generate code, you can either click the Run icon , click the Generate Project Code icon , or select the Generate Project Code option from the File menu.

For more details, see Chapter 10, “Generating Code.”

Saving Your Work

When you work with any software, you should save your work often. Remember that a *project* in the novice mode of UIM/X includes all your open windows and dialog boxes. Saving a project saves all your work in one step.

To Save Your Work

1. Choose the File⇒Save Project command or click on the Save Project icon  from the Project Window icon bar.

The File Selection dialog box appears, with a default file name of `Untitled.prj`, as shown in Figure 2-4.

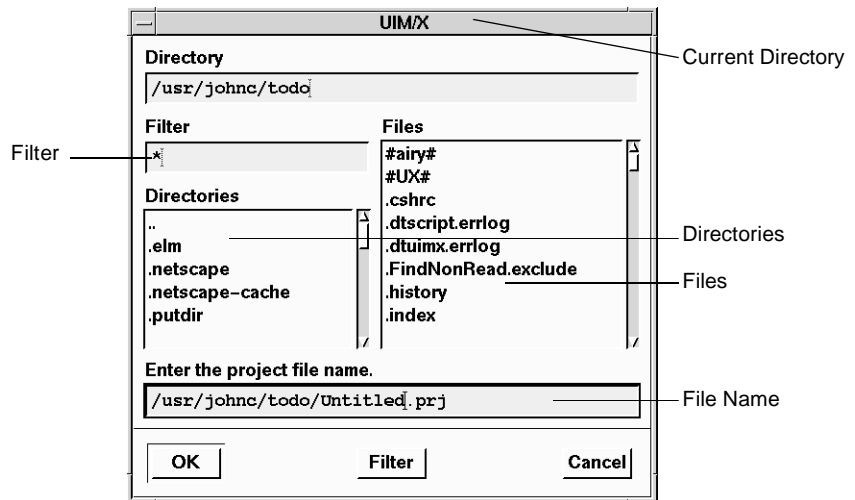



Figure 2-4 File Selection Dialog Box

2. Click OK to accept the default file name and path, and save your project.
OR
2. Click in the Directories box to provide a different path, and type in a new name in the File Name box.
3. Then click OK.

Your project is saved in the specified file. For the rest of your session, File⇒Save Project and the Save Project icon  save the latest version of your project in the same file.

Opening the Editors

To help save you time, a number of visual editors are provided with the novice mode of UIM/X. You can open several of these editors from the Project Window. You can open all the remaining editors from one of these, called the Property Editor.

Opening the Property Editor

You can use the Property Editor to view and revise object properties, as described in Chapter 4, “Viewing and Changing Properties.” You can open the Property Editor from the Project Window, and in several other ways.

To Open the Property Editor from the Project Window

1. Choose the Tools⇒Property Editor command.

The Property Editor appears, ready for you to load an object into it.

From the Property Editor, you can open the Color Viewer, the Color Editor, the Color Map, the Font Viewer, the Callback Editor, and the Icon Viewer.

Opening the Program Layout Editor

You can use the Program Layout Editor to read and revise the makefile for your project, as described in Chapter 9, “Managing Your Projects.” You can open the Program Layout Editor from the Tools menu in the Project Window.

To Open the Program Layout Editor

1. Choose the Tools⇒Program Layout Editor command from the Project Window.

The Program Layout Editor appears, loaded with the makefile for your project.

Opening the Menu Editor

You can use the Menu Editor to create and revise menu bars and option menus in your interface, as described in Chapter 5, “Building Menus.” There are two slightly different versions of the Menu Editor, one for pull-down menus and another for option menus. The appropriate editor opens, depending whether you are working on a pull-down menu or an option menu. You can open the Menu Editor from the Project Window, and in other ways.

To Open the Menu Bar Editor

1. Double-click on any pull-down menu label in the menu bar in an interface.

The Menu Bar Editor appears, ready for you to revise the menu bar.

To Open the Option Menu Editor

1. Select an existing Option Menu in an interface.
2. Select Tools⇒Menu Editor from the Project Window menu bar.

The Option Menu Editor appears, ready for you to edit the option menu.

Opening the Declaration Editor

You can use the Declaration Editor to view the includes, defines, and global for your project, as described in Chapter 12, “Programming in UIM/X.”

To Open the Declaration Editor

1. Select any interface in the Project Window.
2. Choose the Tools⇒Declaration Editor command from the Project Window.

The Declaration Editor opens.

Opening the Connection Editor

You can use the Connection Editor to establish a behavioral connection between a source and target object, as described in Chapter 6, “Adding Connections”.

To Open the Connection Editor

1. Select any object on your interface.
2. Choose the Tools⇒Connection Editor option from the Project Window menu bar.

The Connection Editor opens.

Opening the Constraint Editor

You can use the Constraint Editor to define form constraints graphically without needing to know the numerous Motif form constraint properties, as described in Chapter 7, “Adding Constraints”.

To Open the Constraint Editor

1. Select an object on your interface.
2. Select the Tools⇒Constraint Editor command from the Project Window menu bar.

The Constraint Editor opens.

Hiding and Showing an Interface

Remember that an *interface* is a window or dialog box you are building in UIM/X. You can hide any interfaces to clear space on your screen.

To Hide an Interface

1. Click on an interface to hide in the Project Window Interfaces area.
2. Choose the View⇒Hide Interfaces command from the Project Window.

The selected interface disappears from your screen.

To Show an Interface

1. Click on the icon for a hidden interface in the Interfaces area.
2. Choose the View⇒Show Interfaces command from the Project Window.

If the interface was hidden, it reappears on your screen. If the interface was hidden by other windows, this brings it to the top of your open windows.

Deleting an Interface

You can delete any interface you no longer want. Always make sure to select the correct interface to delete, since there is no way to “undo” a deletion.

To Delete an Interface

1. Click on the interface to delete in the Interfaces area of the Project Window.
2. Choose the Edit⇒Delete command from the Project Window.
A message box asks you to confirm your deletion.
3. Click OK to delete.
Your selected interface is deleted from your project. Its icon disappears from the Interfaces area of the Project Window.

Hiding and Showing a Project

You can hide an entire project to clear space on your screen

To Hide a Project

1. Choose the View⇒Hide Project command from the Project Window.
All interfaces in your project disappear from your screen.

To Show a Project

1. Choose the View⇒Show Project command from the Project Window.
If the project was hidden, it reappears on your screen. If the project was hidden by other windows, this brings it to the top of your open windows.

Using the Palette

The Palette also appears each time you start the novice mode of UIM/X, as shown in Figure 2-5. You can move, resize, minimize, maximize, or close the Palette, just like any other window.

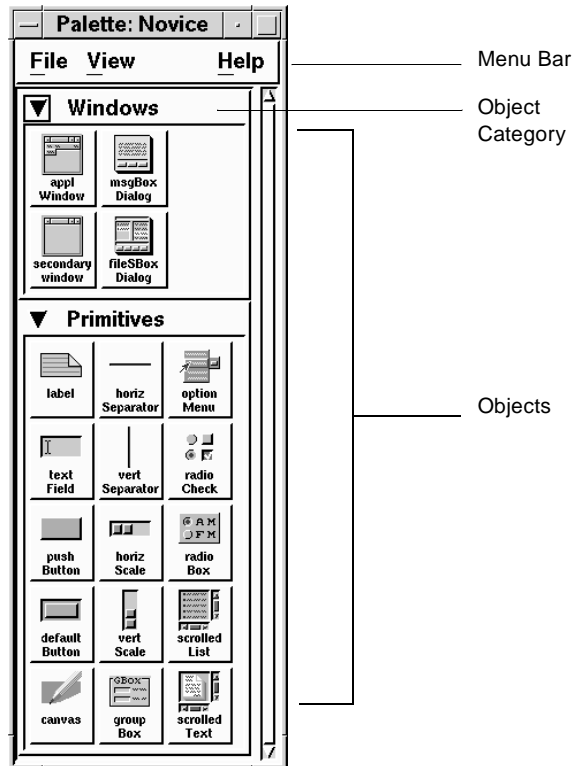


Figure 2-5 Palette

By default, each object is shown as an icon with a name. From the Palette, you can hide the Palette, change how objects are shown, and collapse or expand an object category.

The Palette is a toolbox of objects you can use to build interfaces. Remember that an *object* is a building block you can use to build an interface with UIM/X. To use these objects, you simply drag and drop them from the Palette into your interfaces. The available objects are grouped into two categories in the Palette:

- Windows
- Primitives

Hiding and Showing the Palette

You can hide the Palette after you finish adding objects to your interface. This will clear space on your screen. You can show the Palette if you need it again.

To Hide the Palette

1. Choose the File⇒Close command from the Palette.

The Palette disappears from your screen.

To Show the Palette

1. Choose the Tools⇒System Palette command from the Project Window.

The Palette reappears on your screen. If the Palette was hidden by other windows, this brings it to the top of your open windows.

Reducing the Size of the Palette

You can reduce the size of the Palette in two ways:

- Changing your view to see icons only or names only
- Collapsing object categories

After doing one or both of these, choose the View⇒Adjust Height command from the Palette to resize it and save space on your screen.

Changing Your View of the Palette

By default, each object in the Palette is shown as an icon with a name. You can change your view of the Palette with the View menu, as shown in Figure 2-6. At the start, most people find the Name and Icon view to be the most helpful. However, should you choose to display icons only, bubble help is still provided for all icons.

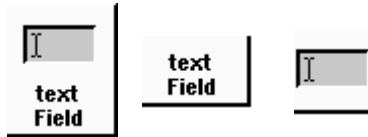


Figure 2-6 Three Views of an Object: Name and Icon, Name Only, and Icon Only

Collapsing and Expanding Object Categories

You can use the expand arrow to collapse and expand either category of object. This enables you to resize the Palette to better fit in the available space on your screen. If you collapse both categories of the Palette, and choose View⇒Adjust Height, the Palette appears as shown in Figure 2-7.

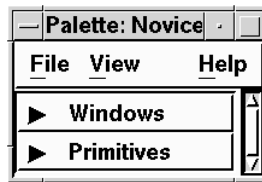


Figure 2-7 Collapsed Categories in the Palette

To Collapse or Expand a Category on the Palette

1. Click on the expand arrow for a category on the Palette.

If the category was collapsed, it expands. You may need to choose View⇒Adjust Height to show all its objects.

If the category was expanded, it collapses.

Using the Pop-up Menus

UIM/X novice mode provides three main pop-up menus:

- Message pop-up menu, in the Messages area of the Project Window,
- Selected Interfaces pop-up menu, in the Interfaces area of the Project Window,
- Selected Objects pop-up menu, in the interface itself, the Constraint Editor, and the Browser.

You use the Message pop-up menu to clear all the messages from the Messages area of the Project Window.

You can use the Selected Interfaces pop-up menu to show, hide, or delete one or more selected interfaces.

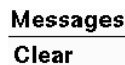
You can use the Selected Objects pop-up menu to cut, copy, paste, duplicate, or delete a single selected object; to align, arrange, duplicate, or delete several selected objects; or to recreate your current interface. These same options are available from the Edit menu in the Project Window.

You can also use the Selected Objects pop-up menu to open the Property Editor, Connection Editor, Constraint Editor, Declaration Editor, Browser, or Menu Editor. These options are also available from the Tools menu in the Project Window.

To Use the Message Pop-Up Menu

1. Point to the Messages area of the Project Window, and press the Menu mouse button.
The Messages pop-up menu appears, as shown in Figure 2-8.
2. Choose the Messages⇒Clear Window command.

All the messages disappear from the Messages area of the Project Window.



The image shows a small rectangular pop-up menu with a white background and a thin border. It contains two text items: 'Messages' at the top and 'Clear' below it. Both items are left-aligned and appear to be in a standard sans-serif font.

Figure 2-8 Messages pop-up menu

To Use the Selected Interfaces Pop-Up Menu

1. Select one or more interfaces in the Interfaces area of the Project Window.
2. Press and hold down the Menu button.
The Selected Interfaces pop-up menu appears, as shown in Figure 2-9.
3. Choose any command from the pop-up menu.

The command is applied to the selected interfaces.

Selected Interfaces	
<u>S</u> how	
<u>H</u> ide	
<u>D</u> elete	

Figure 2-9 Selected Interfaces Pop-up Menu

To Use the Selected Objects Pop-Up Menu

1. Select one or more objects in your current interface.
2. Press and hold down the Menu button.

The Selected Objects pop-up menu appears, as shown in Figure 2-10.

3. Choose any command from the pop-up menu.

The command is applied to the selected objects.


Selected Objects (pushButton1)	
<u>T</u> ools ✓	
<u>C</u> ut	Ctrl+X
<u>C</u> opy	Ctrl+C
<u>P</u> aste	Ctrl+V
<u>D</u> uplicate	
<u>A</u> lign	✓
<u>A</u> rrange	✓
<u>D</u> elete	
<u>R</u> ecreate	
<u>I</u> nstance	

Figure 2-10 Selected Objects Pop-up Menu

Exiting from UIM/X

There are a number of ways you can exit from the novice mode of UIM/X.

To Exit from UIM/X novice mode

1. Select the File⇒Exit command from the Project Window.
OR
1. Double-click on the Project Window Control-menu box: 
OR
1. Press Alt+F and then X with the Project Window active
OR

1. Press Alt+F4.

In each case, a dialog box asks you to confirm that you really want to exit.

1. Click OK to exit.

The Project Window, the Palette, any open editors, and all your interfaces disappear from your screen, and you exit from UIM/X.

Working with Objects

Overview

This chapter defines an object, and describes each object supported by the novice mode of UIM/X. Experienced Motif programmers can skip this chapter; programmers new to Motif should read it carefully.

What Is an Object?

An object is a building block you can use to build an interface with UIM/X. Each object is made up of basic elements such as lines, borders, and shadows, all carefully designed to create a consistent look and feel.

An object is like a prefabricated window for a house. Most builders today use prefabricated windows, instead of creating them from wood, glue, and glass. This saves time, and helps them produce consistent-looking work. A builder could make his own windows, but what would he gain? It's the same for programmers: you could make your own objects, but what would you gain?

Each object has an intended purpose. Some objects, such as Separators, are simple objects you can use for visual design elements. Others, such as PushButtons, are more complex, programmable objects, which can accept a label or an icon and be activated to run a piece of code known as a *callback*.

The novice mode of UIM/X supports two different types of objects: Motif widgets and compound objects. A Motif widget is an object whose appearance and behavior precisely follows the *OSF/Motif Style Guide*. The novice mode of UIM/X supports a number of popular members of the Motif widget set.

What Is a Compound Object?

A compound object consists of several Motif widgets combined into one object for your convenience. While these compound objects are not official Motif widgets, they still follow the *OSF/Motif Style Guide*.

If a regular Motif widget is like a prefabricated window, a compound object is more like a prefabricated wall section. A compound object provides the same benefits as a regular object: it saves you time building it yourself, and ensures that you get consistent results. And it does even more than a regular object.

UIM/X novice mode provides the following compound objects:

- ApplicationWindow
- SecondaryWindow
- RadioBox
- GroupBox

Each of these compound objects has its own special features. For example, an ApplicationWindow contains a built-in menu bar. The GroupBox is a visual design element for grouping related user interface controls.

The Two Categories of Objects

The Palette displays two categories of available objects:

- Windows
- Primitives

Windows are complete windows, while Primitives are objects you can place into these windows. Each category plays a different role in your interface, as described in the following sections.

The Windows Category

The Windows category includes these four compound objects:

- ApplicationWindow
- SecondaryWindow
- MessageBoxDialog
- FileSBoxDialog

The Application Window

An ApplicationWindow is a compound object to get you started quickly. As shown in Figure 3-1, an ApplicationWindow includes a built-in title bar, a menu bar with File and Help pull-down menus, and an empty area where you can add more objects. The first ApplicationWindow you create is automatically named `app1Window1`, the second is `app1Window2`, and so on. You can have one, several, or no ApplicationWindows in a project.

An ApplicationWindow is a “parent” that can “manage” other objects which become its “children”. This means that after you place an ApplicationWindow on your screen, you can add other objects (but *not* other windows) to it. These objects are “adopted” as children of the ApplicationWindow.

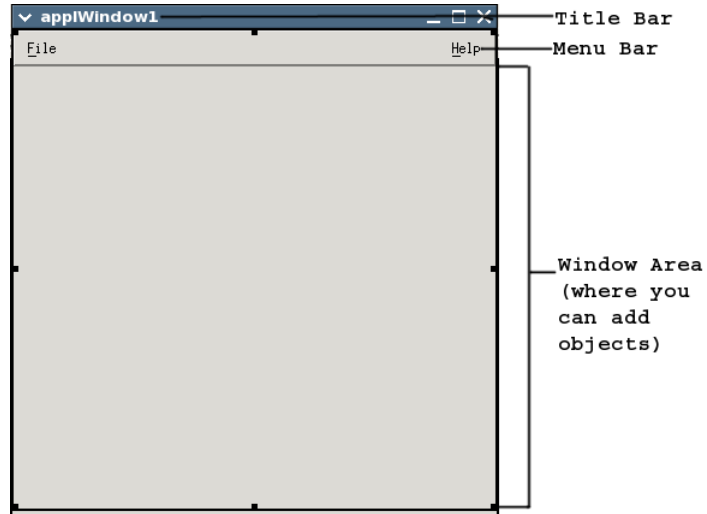


Figure 3-1 Application Window

The Secondary Window

A `SecondaryWindow` is a compound object that looks and acts like an `ApplicationWindow` with no menu bar. You can *not* add a menu bar to a `SecondaryWindow`; if you need a menu bar, use an `ApplicationWindow`. The first `SecondaryWindow` you create is named `secondWindow1`, the next is `secondWindow2`, and so on. You can drag and drop other objects (but *not* windows) from the Palette onto a `SecondaryWindow`, and it will adopt them. You can have one, several, or no `SecondaryWindows` in a project.

The Message Dialog Box

You can create a `MsgBoxDialog` and place it anywhere on your screen. Use this dialog box to display messages to end users in a format they are already familiar with.

As shown in Figure 3-2, a `MsgBoxDialog` contains a message area above some built-in `PushButton`s for OK, Cancel, and Help. You can control the font used to display your message and `PushButton`s. You can also set the style of the dialog box, to make it appear as a standard error, information, question, or warning message box.

When you create a `MsgBoxDialog`, it is *not* attached to your other windows, and “floats” by itself until you attach it with callbacks. For example, you may want to set up your application so that pushing a `Delete PushButton` pops up a `MsgBoxDialog` asking end users if they really want to delete. When an end user clicks `OK`, you can hide this dialog box and complete the operation.

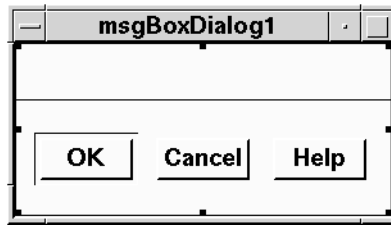


Figure 3-2 Message dialog box

The File Selection Dialog Box

You can create a `FileSBoxDialog` and place it anywhere on your screen. Use this dialog box to enable end users to select a file to open, view, modify, save, load, or delete, using a standard interface they already know.

As shown in Figure 3-3, a `FileSBoxDialog` contains the standard text fields for `Filter`, `Directories`, `Files`, and `Selection`, and built-in `PushButton`s for `OK`, `Filter`, and `Cancel`. You can control the font used to display the `Labels`, the text in the `TextFields`, and the `PushButton`s. You can also set the default filter pattern(`*`), and the style and title of the dialog box. You can *not* add any other objects to this dialog box, since it is already considered complete.

When you create a FileSBoxDialog, it is *not* attached to your other windows, and “floats” by itself until you attach it with callbacks. For example, you may want to set up your menus so that selecting File⇒Open, File⇒Save, or File⇒Delete pops up a FileSBoxDialog. Otherwise, keep this dialog box hidden.

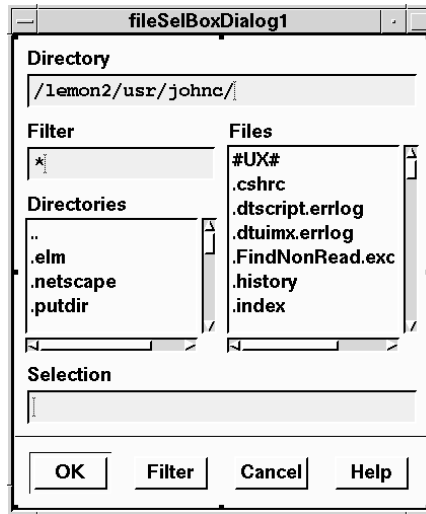


Figure 3-3 File Selection dialog box

Window Objects and Purposes

Each of the four Window objects supported by the novice mode of UIM/X is shown in Table 3-1, along with its name, suggested uses, and how the end user activates it in your interface. Windows are important building blocks of your interfaces, so you should be aware of their similarities and differences.

There is something else to remember about windows. After you create a window or dialog box, you can move or resize it, either by dragging it with the compass or resize pointers, or more precisely by setting its size and location properties with the Property Editor. Do *not* move a window or dialog box with

its title bar, or resize it with its resize handles. This will *not* change the object permanently; the next time you open, show, or reload that window or dialog box, it will return to its previous size.

Table 3-1 Windows Supported by UIM/X Novice Mode

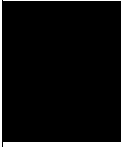



Name	Icon	Suggested Uses
Application Window		<p>Purpose: to serve as the main window in your project, complete with a menu bar.</p> <p>Use the ApplicationWindow by dragging and dropping Primitive objects from the Palette into it. Create menu items and new pull-down menus as required.</p>
Secondary Window		<p>Purpose: to serve as a secondary window in your project, or the main window if you do <i>not</i> need a menu bar.</p> <p>Use a SecondaryWindow by dragging and dropping Primitive objects from the Palette into it.</p>
Message Dialog Box		<p>Purpose: to display a message to an end user using a standard dialog box.</p> <p>Use a MsgBoxDialog to show the end user a message in a standard format. The MsgBoxDialog already includes an area for your message, a Separator, and three PushButtons labelled “OK”, “Cancel”, and “Help”.</p> <p>To activate a MsgBoxDialog, click one of its PushButtons.</p>

Table 3-1 Windows Supported by UIM/X Novice Mode (Continued)

Name	Icon	Suggested Uses
File Selection Dialog Box		<p>Purpose: to enable an end user to select a file using a standard dialog box.</p> <p>Use a FileSBoxDialog to enable the end user to select a file to open, view, modify, save, load, or delete. Since this is a standard dialog box, the end user is already familiar with it. All the features of the standard FileSBoxDialog are supported, including Filter. You can <i>not</i> add any other objects to a FileSBoxDialog, since it is already considered complete.</p> <p>To use a FileSBoxDialog, click on the Directories and Files list boxes until you see your desired file, then double-click on the file name.</p>

The Primitives Category

All the remaining objects in the Palette are in the Primitives category. Some Primitive objects form a visual element in your design, like a Separator. Some display static information, such as a Label. And some Primitives interact with the end user, like a PushButton.

You can add any Primitive object to an ApplicationWindow, a SecondaryWindow, a MsgBoxDialog, and a GroupBox.

Unlike the windows, a Primitive can *not* have any children. If you move another object on top of a Primitive, it overlaps; it is *not* “adopted” as a child. The only exceptions to this rule are:

- the GroupBox, as already noted, which will accept all other Primitive objects as children, and
- the RadioBox, which will accept Radio Check objects as children.

You can *not* add a Primitive to a FileSBoxDialog, or place a Primitive anywhere else on your screen.

Primitive Objects and Purposes

Each of the Primitive objects supported by the novice mode of UIM/X is shown in Table 3-2, along with its name, suggested uses, and how the end user activates it in your interface. You should be well aware of their similarities and differences.

Table 3-2 Primitives Supported by UIM/X Novice Mode (Sheet 1 of 5)




Name	Icon	Suggested Uses
Label		<p>Purpose: to display text or an icon.</p> <p>Use a Label to provide an identifying name or icon to part of your interface. Use a Label plus Separator(s) to visually “block out” part of your interface. A Label is always shown in the Foreground color.</p> <p>A Label cannot be activated.</p>
Text Field		<p>Purpose: to accept, display, or edit one line of text.</p> <p>Use a TextField to accept one line of typed input from the end user. You can also use a TextField to display one line of read-only text.</p> <p>To activate a TextField, click in it and then type, or double-click to select all the existing text and then type to replace it.</p>
Push Button		<p>Purpose: to activate an operation.</p> <p>Use a PushButton to activate each of the main functions of your interface. A PushButton can contain either a text label or an icon to indicate its purpose. Use clear icons or commonly-understood labels for PushButtons, such as “OK” or “Cancel”. Always align multiple PushButtons in a row.</p> <p>To activate a PushButton, click it.</p>

Table 3-2 Primitives Supported by UIM/X Novice Mode (Sheet 2 of 5)





Name	Icon	Suggested Uses
Default Button		<p>Purpose: to activate the most likely operation.</p> <p>Use a DefaultButton for the most likely operation in an interface, such as “OK”. You can only have one DefaultButton per window.</p> <p>To activate a DefaultButton, press Return with that interface active. You can also click it like a normal PushButton.</p>
Canvas		<p>Purpose: to display one or more graphics.</p> <p>Use a Canvas to provide an area for an end user to create, revise, or view graphics.</p> <p>A Canvas cannot be activated by the end user.</p>
Horizontal Separator		<p>Purpose: to visually separate one horizontal area of an interface from another.</p> <p>Use a HorizSeparator as a visual design element to help organize your interface.</p> <p>A HorizSeparator cannot be activated.</p>
Vertical Separator		<p>Purpose: to visually separate one vertical area of an interface from another.</p> <p>Use a VertSeparator as a visual design element to help organize your interface.</p> <p>A VertSeparator cannot be activated.</p>

Table 3-2 Primitives Supported by UIM/X Novice Mode (Sheet 3 of 5)


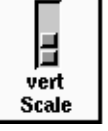
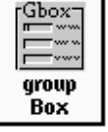

Name	Icon	Suggested Uses
Horizontal Scale		<p>Purpose: to set or display a value in a range by sliding left and right.</p> <p>Use a HorizScale to enable the end user to adjust a graduating value.</p> <p>To activate a HorizScale, drag its slider left or right between 0 and 100% of the specified range. If you change the ProcessingDirection property to reorient the scale to a VertScale, you may have to adjust its Height and Width.</p>
Vertical Scale		<p>Purpose: to set or display a value in a range by sliding up and down.</p> <p>Use a VertScale to enable the end user to adjust a graduating value.</p> <p>To activate a VertScale, drag its slider up or down between 0 and 100% of the specified range. If you change the ProcessingDirection property to reorient the scale to a HorizScale, you may have to adjust its Height and Width.</p>
Group Box		<p>Purpose: to create a visual grouping in your interface.</p> <p>Use a GroupBox to “block out” part of your interface for a certain purpose.</p> <p>A GroupBox cannot be activated.</p>
Option Menu		<p>Purpose: to provide a choice from among two or more possibilities.</p> <p>Use an OptionMenu to enable the end user to choose from a given list of possibilities. To design an option menu, use the OptionMenu Editor, which appears when you double-click on an OptionMenu.</p> <p>To activate an OptionMenu, press and hold down its button, then select your choice from the list.</p>

Table 3-2 Primitives Supported by UIM/X Novice Mode (Sheet 4 of 5)





Name	Icon	Suggested Uses
Radio Button/ Check Button		<p>Purpose: to add more choices to a RadioBox (as a RadioButton) OR to toggle an option between on and off (as a CheckButton).</p> <p>Depending where you create this object, it becomes either a new RadioButton added to an existing RadioBox, or a new CheckButton.</p> <p>If you create this object on top of an existing RadioBox, a new RadioButton is added at the bottom of the RadioBox. If you create this object in an empty part of a window, a new CheckButton is added.</p> <p>To activate a RadioButton, click it. It is selected, while the previously-selected button in the RadioBox is deselected. To activate a CheckButton, click it. It toggles between checked and unchecked.</p> <p>Note that changing some properties (including Alignment, Height, Width, X, or Y) has no effect on a RadioButton in a RadioBox, because the RadioBox manages all the buttons it contains.</p>

Table 3-2 Primitives Supported by UIM/X Novice Mode (Sheet 5 of 5)

Name	Icon	Suggested Uses
RadioBox		<p>Purpose: to select one option among several mutually-exclusive options.</p> <p>Use a RadioBox to provide an either/or choice between two or more options.</p> <p>By default, a RadioBox has two RadioButtons. You can add more choices to a RadioBox by creating a new RadioButton on top of it. The new RadioButton is added at the bottom of the RadioBox.</p> <p>To change the order of the RadioButtons in a RadioBox, drag each button except the one you want to be first out of the RadioBox. Then drag each button back into the RadioBox in the proper order. Do not drag every RadioButton out of the RadioBox, because this deletes the RadioBox.</p> <p>To activate a RadioBox, click on one of its buttons. The previously-selected choice is deselected.</p>
Scrolled List		<p>Purpose: to display a list of items.</p> <p>Use a ScrolledList to display a read-only list of items. Vertical and horizontal scroll bars appear, which you can use if required.</p> <p>Select an item from a ScrolledList by clicking on it.</p>
Scrolled Text		<p>Purpose: to accept, display, or edit more than one line of text.</p> <p>Use a ScrolledText object to accept more than one line of typed input from the end user. You can also use a ScrolledText object to display more than one line of read-only text. Vertical and horizontal scroll bars appear, which you can use if required.</p> <p>To activate a ScrolledText object, click in it and then type. You can also double-click to select a word or triple-click to select a line, and then type to replace the selected text.</p>

Creating an Application Window

You often start a new interface with an `ApplicationWindow`. This `ApplicationWindow` becomes the parent for all the Primitive objects you add to it.

You can create an `ApplicationWindow` in any one of three ways:

- Drag and draw, so you can size and position the `ApplicationWindow`
- Drag and drop, which creates an `ApplicationWindow` at a default size
- Double-clicking, which creates an `ApplicationWindow` at a default size

To Create an Application Window Using Drag and Draw

1. Click on the `ApplicationWindow` in the Palette.
Notice that the mouse pointer changes to an upper-left corner shape: Γ
2. Press and hold down the Select button where you want the top left corner of your `ApplicationWindow` to be located on your screen.
3. While holding down the Select button, drag the mouse down and to the right to “draw” the new `ApplicationWindow`, as shown in Figure 3-4.
4. Release the mouse button.

The program creates an `ApplicationWindow` at the size and location that you specified. By default, this object is called `applWindow1`.

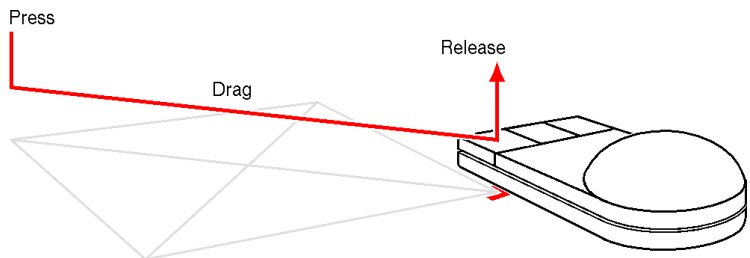


Figure 3-4 Creating an Application Window Using Drag and Draw

To Create an Application Window Using Drag and Drop

1. Point to the `ApplicationWindow` in the Palette, then press and hold down the Adjust button.
Notice that an outline of the `ApplicationWindow`'s default size appears, and the mouse pointer changes to a compass pointer.

2. Still holding down the Adjust button, drag the outline to your screen, and then release the mouse button.

The program creates an `ApplicationWindow` at a default size on your screen.

To Create an Application Window by Double-Clicking

1. Double-click on the `ApplicationWindow` in the Palette.

An `ApplicationWindow` appears at a default size and location on your screen.

You can move or resize an `ApplicationWindow` at any time, using the compass or resize pointers, or the Property Editor. Remember *not* to move an `ApplicationWindow` with its title bar, or resize it with its resize handles. This does *not* permanently move or resize the window; the next time you open, show, or reload the window, it returns to its previous location and size.

Adding Objects to an Application Window

After you create an `ApplicationWindow`, you can add Primitive objects to it. You can add objects using either drag and drop, or drag and draw. As with the `ApplicationWindow`, using drag and drop creates an object at a default size, which you can resize later, while using drag and draw enables you to size and position each object you create.

You can add objects in any order, and move them around and resize them until your interface looks just the way you want it to.

After you place an object in an `ApplicationWindow`, you can cut, copy, paste, duplicate, move, resize, or delete that object. You can also use any of the built-in editors to change any of the properties that determine an object's appearance and behavior.

Selecting Objects

After you place an object in an `ApplicationWindow`, you can act on it in various ways. First you select an object, then you act on it. You can also select more than one object, and then act on all your selected objects at once.

To Select an Object

1. Click on an object in an `ApplicationWindow`.

Selection handles appear around the selected object.

To Select Multiple Objects with the Control Key

1. Hold down the Control key and click on all the objects you want to select.

Selection handles appear around every object you click.

To Select Multiple Objects with the Mouse

1. Point above and to the left of the objects you want to select.
2. Press and hold down the Select button, and drag the pointer. The pointer changes to a black O, and a marquee (dashed box) appears.
3. Continue dragging the pointer to surround all the objects you want to select with the marquee, as shown in Figure 3-5.
4. Release the mouse button.

Selection handles appear around every object inside the marquee.

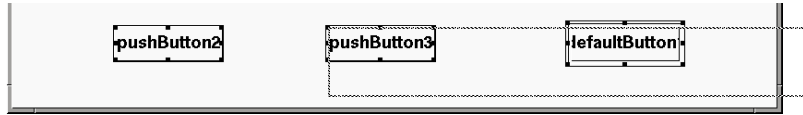


Figure 3-5 Selecting Multiple Objects with the Mouse

You can control the sensitivity of the marquee by adjusting the `minimumSweepSize` resource in your Application Defaults. The default setting is 7 pixels; which means that once you move the mouse more than 7 pixels with the Select button held down, the marquee will appear.

Working with Objects

After you select an object in an `ApplicationWindow`, you can cut, copy, paste, duplicate, or delete that object. These commands work as you expect.

Cutting an object removes it from your interface and places it in UIM/X's Clipboard. Copying an object puts a copy of it in the Clipboard. You can paste the last object you cut or copied back into your interface. Duplicating an object copies it right in your interface. Deleting an object removes it from your interface without letting you paste it back. You can only cut, copy, or paste one object at a time, but you can delete or duplicate multiple objects. You can only act on objects in your interface, not in the Palette.

The Selected Objects pop-up menu provides the same editing options as the pull-down Edit menu in the Project Window.

To Cut or Copy an Object

1. Select one object you want to cut or copy.
2. Choose the `Edit⇒Cut` or `Edit⇒Copy` command from the Project Window or select their respective icons.


Your selected object is cut or copied as you specified.

To Duplicate Objects

1. Select one or more objects you want to duplicate.
2. Choose the Edit⇒Duplicate command from the Project Window or select its corresponding icon.

Your selected object is duplicated.

To Paste a Cut or Copied Object

1. Click in your interface, where you want to paste the object.
2. Choose the Edit⇒Paste command from the Project Window or select the Paste icon  from the icon bar.

The last object you cut or copied is pasted into your interface.

To Delete Objects

1. Select one or more objects you want to delete.
2. Select the Edit⇒Delete command from the Project Window, and click OK in the dialog box to confirm.

Your selected objects are deleted.

The Nine Regions of an Object

You can move or resize the ApplicationWindow or any object in it. When you move or resize an object, the position of the mouse pointer is important, since UIM/X divides each object into nine invisible regions, as shown in Figure 3-6.

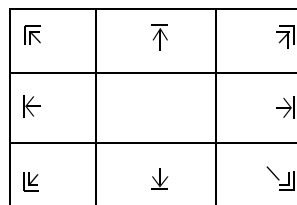





Figure 3-6 Nine regions of an object, and their resize pointers

To see the nine regions of an object, point to any corner of the object and press the Adjust button. The grid that appears is called the “resize grid”. Depending on which region of the object you point to when you press the Adjust button, you see a different resize pointer, as shown in Figure 3-6. Each resize pointer enables you to perform a different function, as listed in Table 3-3.

You can use the central region (5) of the resize grid, which displays the compass pointer, to move a selected object to a new location. You can use the other eight regions to stretch or shrink a selected object to a new size. Always use the resize grid to resize an `ApplicationWindow`, not its resize handles; any changes you make with the resize handles will not be saved.

Table 3-3 Functions of the Resize Pointers

Pointer Shape	Purpose
	To change the object's height and width.
	To change the object's height.
	To change the object's width.

Moving Objects

Using the center of the resize grid, you can move an `ApplicationWindow` and all the objects in it, or any selected object in your interface.

To Move an Object

1. Point to the center of any object in your interface.
2. Press and hold down the Adjust button.

The mouse pointer changes to a compass pointer. This means you can now move the object. (If you see the resize grid, release the button and press again, closer to the center.)

3. Drag the selected object to a new location, then release the mouse button.

The selected object moves, as shown in Figure 3-7.

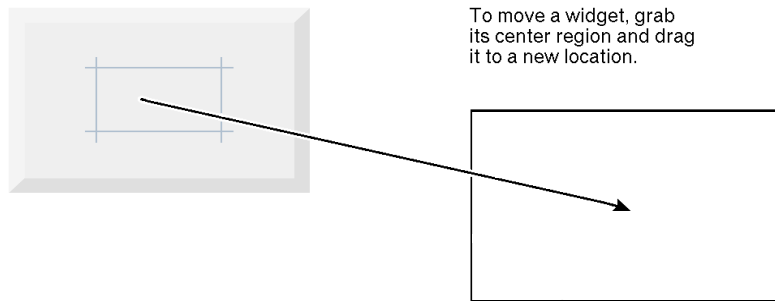


Figure 3-7 Moving an Object

To Move Multiple Objects

1. Select all the objects you want to move.
2. Point to the center of one of the selected objects, then press and hold down the Adjust button.
3. When you see the compass pointer and the outline, drag the compass pointer to move all the objects, then release the mouse button.

Resizing Objects

You can resize an `ApplicationWindow`, or any object in it. You can resize an `ApplicationWindow` by itself to create more space in it; or you can select all the objects in an `ApplicationWindow`, and resize them all together.

To Resize an Object

1. Point to one of the object's resize regions, such as the lower-right corner.
2. Press and hold down the Adjust button.
The mouse pointer changes to a resize pointer, and the resize grid appears; this means you can now resize the object. (If you see the compass pointer instead, release the button and press again, further away from the center.)
3. Drag the resize pointer to resize the object, then release the mouse button.

The object reappears larger or smaller, as shown in Figure 3-8.

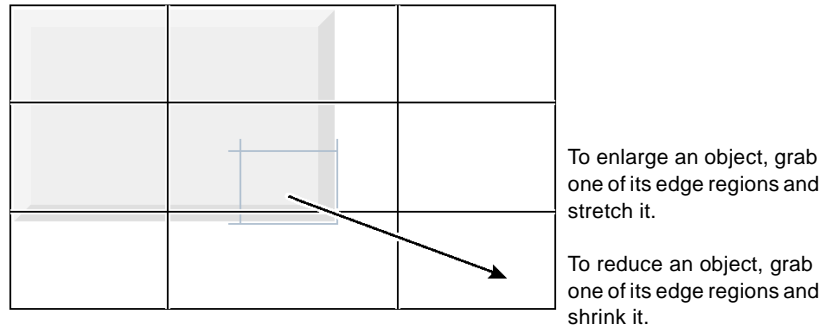


Figure 3-8 Resizing an object

To Resize Multiple Objects

1. Select all the objects you want to resize.
2. Point to the corner of one of the selected objects, then press and hold down the Adjust button.
3. When you see the resize pointer and the resize grid, drag the resize pointer to resize all the objects, then release the mouse button.

Aligning Objects

You can select any number of objects from your interface and align them to make your design more visually pleasing. This is especially important for lining up a row of push buttons. There are six alignment icons in the Align submenu, as shown in Figure 3-9.

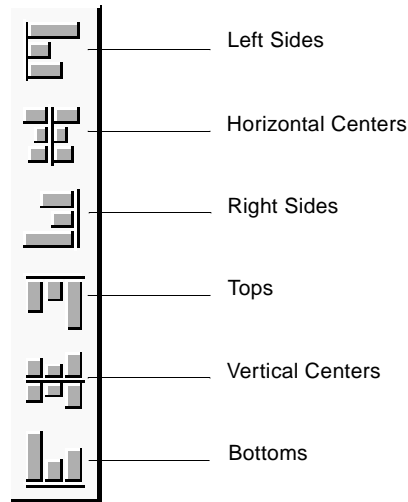


Figure 3-9 Align submenu icons

To Align Objects

1. Select all the objects you want to align.
2. Choose the Edit⇒Align command from the Project Window.

OR

2. Press the Menu button, and choose the Selected Objects⇒Align command.
3. From the submenu, choose an alignment icon.

The selected objects reappear, aligned as you specified.

Arranging Objects

You can arrange objects, like a row of PushButtons, to make your design more visually pleasing. You can control the size of the space between objects, and the size of the space between the objects and the margin of their parent. There are six arrangement icons in the Arrange menu, as shown in Figure 3-10.

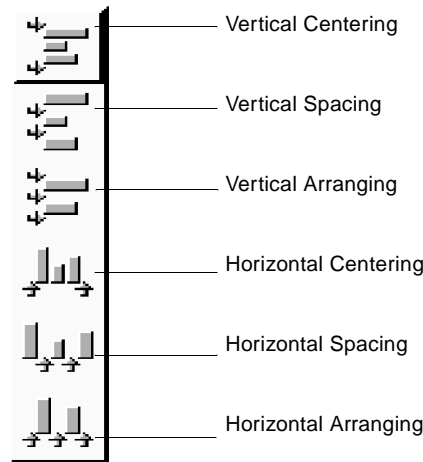


Figure 3-10 Arrange Menu Icons

- **Vertical Centering**
When objects are placed in a vertical column, this option causes the space between the highest object and the upper edge of its parent to be equal to the space between the lowest object and the lower edge of its parent.
- **Vertical Spacing**
When three or more objects are placed in a vertical column, this option causes the spaces between them to be of equal sizes.
- **Vertical Arranging**
This option is a combination of Vertical Spacing and Vertical Centering.
- **Horizontal Centering**
When objects are placed in a horizontal row, this option causes the space between the left most object and the left edge of its parent to be equal to the space between the right most object and the right edge of its parent.

To Arrange Objects

- Horizontal Spacing
When three or more objects are placed in a horizontal row, this option causes the spaces between them to be of equal sizes.
- Horizontal Arranging
This option is a combination of Horizontal Spacing and Horizontal Centering.

1. Select all the objects you want to arrange.
2. Choose the Edit⇒Arrange command from the main window.

OR

2. Press the Menu mouse button, and choose the Selected Objects⇒Arrange command.
3. From the submenu, choose an arrangement icon.

The selected objects reappear, arranged as you specified.

Object Properties

The novice mode of UIM/X supports a certain set of *properties* for each object in the Palette. These properties determine the size, color, label, font, and other characteristics for each object. These supported properties include a subset of those available in Motif, featuring the most frequently-used properties.

For more information on each supported property, see Appendix B, “Object Properties.”

Viewing and Changing Properties

4

Overview

This chapter describes how to use the Property Editor and all the related viewers/editors provided with the novice mode of UIM/X, including the Color Viewer, the Color Editor, the Color Map, the Font Viewer, and the Icon Viewer. These tools are designed to help you see and change object properties.

<i>Property Editor</i>	<i>page 105</i>
<i>Color Viewer</i>	<i>page 113</i>
<i>Color Editor</i>	<i>page 118</i>
<i>Color Map</i>	<i>page 121</i>
<i>Font Viewer</i>	<i>page 122</i>
<i>Icon Viewer</i>	<i>page 125</i>

The other editors provided with UIM/X novice mode are not covered in this chapter. The Menu Editor is covered in Chapter 5, “Building Menus”. The Connection Editor is described in Chapter 6, “Adding Connections”. The Constraint Editor is described in Chapter 7, “Adding Constraints”. The Program Layout Editor is described in Chapter 9, “Managing Your Projects”. The Declaration Editor is described in Chapter 12, “Programming in UIM/X.”

For a hands-on example of using many of these editors to build an actual interface, see Chapter 1, “Building Your First Project”.

For more details on specific properties available in the novice mode of UIM/X, see Appendix B, “Object Properties.”

The Editors Provided

The novice mode of UIM/X provides a full range of editors for viewing and changing object properties. All of these editors, how to access them, and their intended purposes are listed in Table 4-1. All editors have certain similarities. You can move, resize, maximize, minimize, or close each editor, the same as any other window.

Table 4-1 Editors

Name	Purpose	How to Open this Editor
Property Editor	To display and change any available property values.	Double-click an object in an interface; or choose Tools⇒Property Editor in the Project Window; or choose Property Editor from the Selected Objects pop-up menu.
Color Viewer	To preview and select a color for an object.	Click on the Background, Foreground, ListBackground, MenuBackground, or TextBackground property in the Property Editor.
Color Editor	To mix a custom color for an object.	Choose Edit⇒Edit Color in the Color Viewer.
Color Map	To grab any of 256 colors for an object.	Choose Edit⇒Color Map in the Color Viewer.
Connection Editor	To establish a behavioral connection between a source and target object	Select any object, press the Menu mouse button, and choose Selected Objects⇒Tools⇒Connection Editor.
Constraint Editor	To define form constraints.	Select Tools⇒Constraint Editor from the Project Window menu bar or use the Menu mouse button to choose Selected Objects⇒Tools⇒Constraint Editor.
Declaration Editor	To view the includes, defines, and global variables for your interface.	Choose Tools⇒Declaration Editor from the Project Window.

Table 4-1 Editors

Name	Purpose	How to Open this Editor
Font Viewer	To view and select a font for an object	Click on the <code>ButtonFontList</code> , <code>FontList</code> , <code>LabelFontList</code> , <code>MenuFontList</code> , or <code>Text FontList</code> property in the Property Editor.
Icon Viewer	To view and select an icon for an object.	Click on the <code>IconPixmap</code> or <code>LabelPixmap</code> property in the Property Editor.
Menu Editor	To create or revise pull-down or option menus for an interface.	Double-click on a pull-down menu; or choose <code>Tools⇒Menu Editor</code> from the Selected Objects pop-up menu.
Program Layout Editor	To edit the makefile for a project.	Choose <code>Edit⇒Program Layout</code> in the Project Window.

The Property Editor

The Property Editor displays, and enables you to change, all the property values for any selected object in an interface. From the Property Editor, you can also open the Color Viewer, the Color Editor, the Color Map, the Font Viewer, the Icon Viewer, and the Text Editor.

You can *not* open the Menu Editor, the Program Layout Editor, the Connection Editor, or the Constraint Editor from the Property Editor.

Steps in Using the Property Editor

There are six basic steps to using the Property Editor:

1. Opening the Property Editor.
2. Loading an object into the Property Editor.
3. Changing an object's properties, including its color, font, and icon.
4. Adding callbacks to provide behavior to objects.
5. Applying your changes to the object.
6. Closing the Property Editor.

Each of these steps is described in the rest of this chapter.

Opening the Property Editor

You can open the Property Editor in two different states:

- Loaded with the properties for a selected object
- Empty, with no object loaded

Once open, you can drag and drop any object from an interface into the Property Editor to load that object's properties.

To Open the Property Editor With an Object Loaded

Double-click on any object in an interface.

OR

Select any object, press the Menu button, and then choose the Selected Objects⇒Property Editor command.

OR

With an object selected, choose Tools⇒Property Editor from the Project Window's menu bar.

The Property Editor appears, with the selected object loaded.

To Open the Property Editor With No Object Loaded

With no object selected, choose the Tools⇒Property Editor command from the Project Window.

The Property Editor appears, ready for you to load an object into it.

If the Property Editor was open but hidden by other windows, this brings it to the top of your open windows.

About the Property Editor

The Property Editor consists of five main areas, as shown in Figure 4-1.

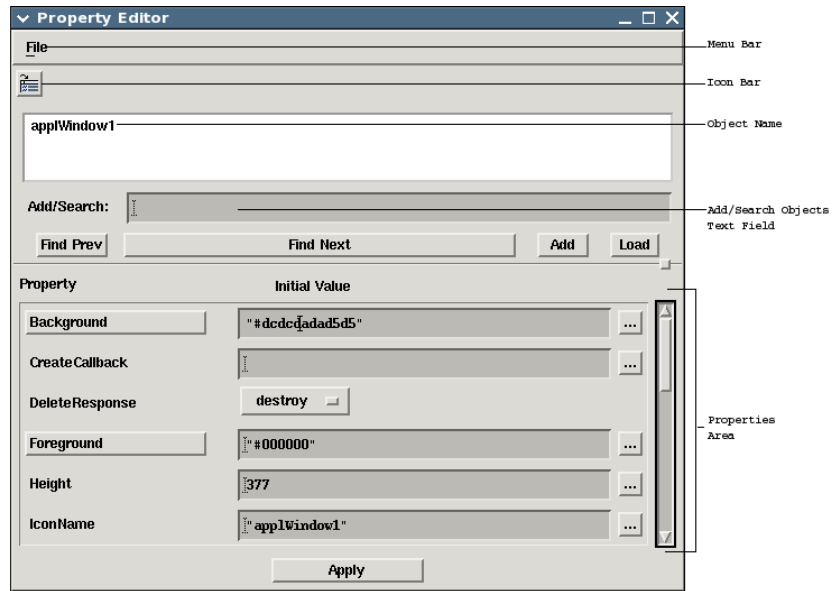


Figure 4-1 Main Areas of the Property Editor

- *Menu Bar:* Provides a File menu to load an object, and reset or close the Property Editor.
- *Icon Bar:* Provides an icon for loading objects into the Property Editor.
- *Object Name:* Shows the name of the object currently loaded into the Property Editor. Enables you to replace the current object with another by dragging and dropping the new object into this area.
- *Add Object Text Field:* Enables you to type an object's name and press Return to load its properties.
- *Properties Area:* Displays the current object's properties in alphabetical order. If applicable, you can click on the property name to open a specialized editor. In some cases, the property value is an option menu you can adjust. Where there is no option menu, an editor button appears beside the property value. Clicking on the button opens the Text Editor to revise that property's value.

Loading an Object into the Property Editor

Once the Property Editor is open, there are six ways to load an object into it:

- Double-click on an object
- Drag and drop an object into the Object Name area
- Use the Property Editor File menu
- Type the object's name in the Add Object area
- Use the Selected Objects pop-up menu, as previously described
- Select an object and click on the Property Editor's load icon.
- Select an object and choose Tools⇒Property Editor from the Project Window menu bar.

Remember that any changes you do not apply to your previous object before loading a new object *are not* saved. You will see a message box which gives you the chance to cancel and apply your changes before loading a new object into the Property Editor.

To Load an Object by Double-Clicking

1. Double-click on any object in an interface.

To Load an Object Using Drag and Drop

1. Point to any object in an interface, press and hold down the Adjust button. An outline of that object appears. (If you see the resize grid, click again closer to the center of the object.)
2. Drag the object's outline into the Object Name area of the Property Editor.
3. Release the mouse button.

To Load an Object Using the File Menu

1. Select any object in an interface.
2. Select the File⇒Load command from the Property Editor.

To Load an Object by Typing its Name

1. Click in the Add Object area of the Property Editor.
2. Type in the object's name and press Return.

To Load an Object Using the Selected Objects Popup Menu

1. Select any object in an interface.
2. Use the Menu mouse button to select Selected Objects⇒Tools⇒Property Editor.

To Load an Object Using the Load Icon

1. Select any object in an interface.
2. Click on the Property Editor's load icon.

Viewing an Object's Properties

With an object loaded in the Property Editor, you can see all its available properties listed in alphabetical order. Different properties are available for each different object. You can see the current value for a property in two ways:

- As a value in a text field, often inside quotation marks (as in "XYZ")
- As the current selection on an option menu

Properties that require a color name, font name, or icon name appear as buttons you can click. Clicking one of these buttons opens the Color Viewer, Font Viewer, or Icon Viewer to help you compose a new name. Many properties have an editor button you can click to open the Text Editor to revise the property's value. Otherwise, you can set the property value with an option menu.

Changing an Object's Property

You can change the value for a property in one of four ways:

1. Typing into a text field
2. Making a different selection from an option menu
3. Clicking the property button and using the appropriate editor
4. Clicking the editor button and using the Text Editor

In each case, after you change a property, the visible value changes to the new value you specified and a change bar appears next to the affected property. After you click the Apply button in the Property Editor, the object is updated to reflect your change.

To Change a Property in a Text Field

1. With an object loaded, scroll to the property you want to change.
2. Double-click in the text field for that property.
3. Type in the new value for that property.
Make sure to include the quotation marks (" ") if required.
4. Click the Apply button.

To Change a Property from an Option Menu

An option menu lists the possible values for a property, as shown in Figure 4-2.

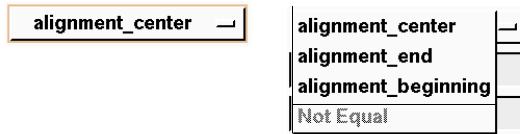


Figure 4-2 Typical Option Menu, Closed and Open

1. With an object loaded, scroll to the property you want to change.
2. Click on the option menu button, and select a new value.
3. Click Apply.

To Change a Property with an Editor

1. With an object loaded, scroll to the property you want to change.
2. Click on the property button.

The appropriate editor appears.

3. Use the editor to change the property, and then click OK.
4. Click Apply in the Property Editor.

To Change a Property with the Text Editor

1. With an object loaded, scroll to the property you want to change.
2. Click on the editor button.

The Text Editor appears, with the property value in a scrolling text field.

3. Double-click in the text field.
4. Type in the new value for that property. Make sure to include the quotation marks (") if required.
5. Click Apply in the Property Editor.

Adding Callbacks to Objects

A *callback* is piece of code you use to make an object behave in a certain way when a specific event occurs. You can use a callback, for example, to make something happen in an interface when the end user clicks on a Push Button. Without any callback for a Push Button, pressing it accomplishes nothing.

You add or change the callbacks for an object using the Callback Editor, as shown in Figure 4-3. You open the Callback Editor from the Property Editor. The novice mode of UIM/X supports eight callbacks that appear as properties in the Property Editor, as follows:

- `ActivateCallback`, which runs when you click on an object such as a Push Button or Text Field
- `CancelCallback`, which runs when you click Cancel in a dialog box
- `CreateCallback`, which runs when an object is created; every object supports this callback to enable you to do any special processing before an object is realized
- `DragCallback`, which runs when you move a slide in a Scale object
- `HelpCallback`, which runs when you click the Help button in a File Selection Dialog or Message Dialog
- `OKCallback`, which runs when you click OK in a dialog box
- `SingleSelectionCallback`, which runs when you select a line in a Scrolled List object
- `ValueChangedCallback`, which runs when an object's current value changes

For more details on any callback, see Appendix B, “Object Properties.”

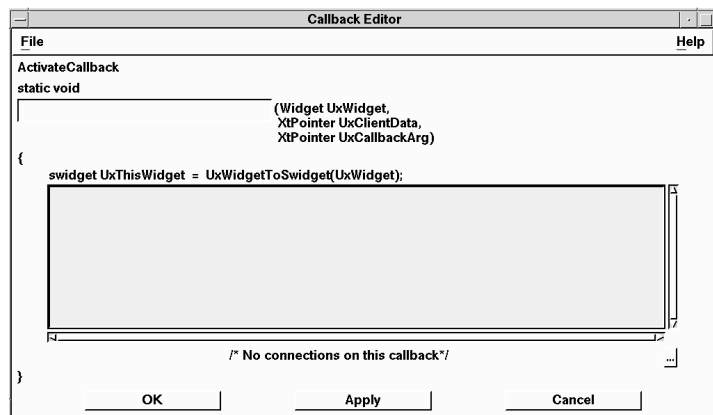


Figure 4-3 Callback Editor

To Add a Callback for an Object

1. Open the Property Editor with an object loaded.
2. Click on the button beside the appropriate callback property.

The Callback Editor appears, as shown in Figure 4-3.

3. Click in the text window and type the code for your callback.
4. When you finish typing your callback, review it carefully, and then click OK.

The Callback Editor disappears. The start of your callback code is visible in the text field for the callback in the Property Editor.

5. Now click Apply in the Property Editor.

Your callback is now applied to your selected object.
Your code does not run until you test it in Test mode.

Resetting the Property Editor

You can reset the Property Editor to clear the current object's values and the Add Object text field.

1. Select the File⇒Reset command from the Property Editor.

The Object Name and Add Object areas are cleared.

Any changes you do not apply before resetting the Property Editor *are not* saved. You will see a message box which gives you the chance to cancel your action and apply your changes before resetting.

Applying Your Changes to an Object

After changing any property, you must click the Property Editor's Apply button to save your changes. If you have a specialized editor open, such as the Color Viewer, Color Editor, Callback Editor, Font Viewer, Icon Viewer, or Text Editor, you must first click the Apply button in the specialized editor, and then click the Apply button in the Property Editor as well.

Any properties you have changed but not yet applied are marked with a change bar in the Property area. If you are changing several properties for the same object, you do *not* need to click Apply after each one, but you *must* click Apply after making your last change.

If there are any errors in the new property values, an error message will appear in the Messages Area of the Project Window. Any property with an incorrect value will be marked with an "X" in the Property area, as shown in Figure 4-4.

To Reset the Property Editor



Figure 4-4 “X” for an Incorrect Property Value

If you try to load a new object, or to reset or close the Property Editor without first clicking Apply, you will see a message box that gives you the chance to cancel your action and apply your changes before continuing.

Closing the Property Editor

After you finish changing object properties, and make sure to apply your changes, there are a number of ways to close the Property Editor. You can close the Property Editor either by:

1. Choosing its File⇒Close command
OR
2. Double-clicking its Window menu button
OR
3. Pressing Alt+F, then C
OR
4. Pressing Alt+F4

Any changes you do not apply before closing the Property Editor *are not* saved. You will see a message box which gives you the chance to cancel and apply your changes before closing.

The Color Viewer

Five available object properties require color names: Background, Foreground, ListBackground, MenuBackground, and TextBackground. With the program’s built-in color tools, you can produce a full spectrum of different colors to use for these properties.

The Color Viewer provides six saved colors, plus a database of all the color names available in your system to choose from. You can also “grab” a color from anywhere else on your screen. From the Color Viewer, you can also access the Color Editor, which enables you to mix your own custom colors, and the Color Map, which provides a palette of 256 more colors to choose from.

In UIM/X, you can type in color names in two possible formats:

- Any color name recognized by your server and enclosed in quotation marks, such as "MediumSlateBlue"
- Any RGB value with hexadecimal digits for the R, G, and B values, starting with a number sign (#) and enclosed in quotation marks, such as "#6a6a4d4d8f8f"

Opening the Color Viewer

You open the Color Viewer from the Property Editor, with an object loaded.

To Open the Color Viewer

1. Make sure the Property Editor is open with an object loaded.
2. Scroll to `Foreground`, `Background`, `ListBackground`, `MenuBackground`, or `TextBackground`.
3. Click on the property's button.

The Color Viewer appears, with the current color for that object's property as the Selected Color.

When the Color Viewer is open, the associated property's button becomes insensitive. If the Color Viewer is open but hidden by other windows, click the editor button beside the property to move the Color Viewer to the top of your open windows.

About the Color Viewer

The Color Viewer consists of six main areas, as shown in Figure 4-5.

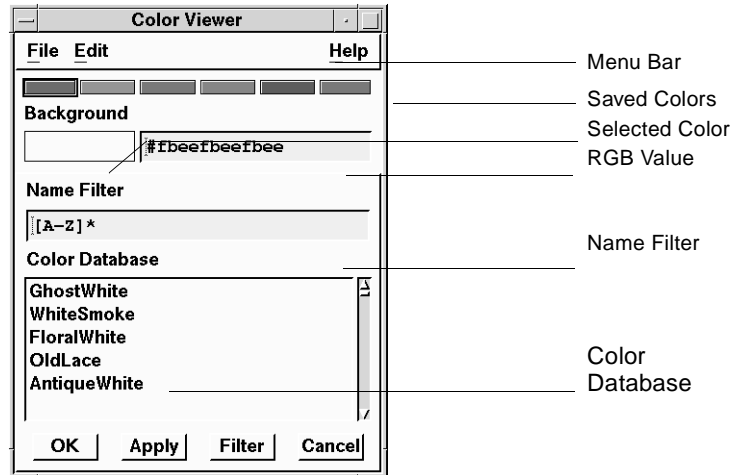


Figure 4-5 Main Areas of the Color Viewer

- *Menu Bar*: Provides two pull-down menus: File to close the Viewer; and Edit to open the Color Editor or Color Map, or grab a color.
- *Saved Colors*: Displays six saved colors to choose from, and enables you to save your own colors during your current session.
- *Selected Color*: Displays the current color and its name. This color name is copied into the Property Editor when you click Apply or OK.
- *RGB Value*: Displays the RGB value of the Selected Color.
- *Name Filter*: Enables you to type in a color name pattern, and then click on the Filter button to list any colors in the Color Database that match this pattern. The default Name Filter is [A-Z]*.
- *Color Database*: Displays any available colors that match the Name Filter, and enables you to click any name to make it the Selected Color.

Selecting a Color with the Color Viewer

There are three ways to select a color with the Color Viewer:

1. Clicking one of the six colors in the Saved Colors area
2. Selecting a color name from the Color Database
3. “Grabbing” a color from anywhere on your screen

From the Color Viewer, you can access both of the other color tools available. You can open the Color Editor to mix a new custom color, or open the Color Map to select from a palette of 256 existing colors.

Once you have selected a color, it appears in the Selected Color area, with its name and RGB value shown.

To Select a Saved Color

Click one of the six colors in the Saved Colors area. The color you clicked becomes the Selected Color, with its name and RGB value shown.

To Select a Color from the Color Database

1. Double-click in the Name Filter field.
2. Type in a new color name pattern you want to match.

For example, the pattern `*Grey` will match all the color names that end in "Grey", such as `DarkGrey`, `BlueGrey`, and so on.

3. Click on the Filter button.

All the available color names that match your pattern appear in the Color Database area.

4. Click on any name in the Color Database.

The color you clicked becomes the Selected Color, with its name and RGB value shown.

Grabbing a Color

“Grabbing” means selecting a color from your screen by clicking on it. This is probably the fastest way to select a color, provided you see the one you want somewhere on your screen.

To Grab a Color from Anywhere on your Screen

1. Make sure the color you want is visible somewhere on your screen. It can be in a different interface, or a different window.
2. Choose the Edit⇒Grab Color command.

The mouse pointer changes to the grab pointer.

3. Click on any color in any window on your screen.

The color you grabbed becomes the Selected Color, with its name and RGB value shown.

Saving a Color

You can mix a color and then save it to use for other objects during your current session. You can save up to six of your own colors in the Saved Colors area. These colors are *not saved* when you exit from the program. Each time you start UIM/X novice mode, the six *original* saved colors appear by default.

To Save a Color

1. Click on a Saved Color you want to replace.

This selects the Saved Color and makes it the Selected Color. The Saved Color you clicked appears pressed in, like a push button.

2. Edit the Selected Color, using the Color Editor, the Color Database, or by grabbing a color.
3. Click on another Saved Color.

Your new color replaces the Saved Color you clicked on in step 1, and the Saved Color you clicked becomes the Selected Color.

Your new saved color is *not* saved when you exit from the program.

Using the Apply and OK Buttons

After you select a color, clicking on the Apply button copies the RGB value of the color to the property you are editing, and keeps the Color Viewer open. Clicking OK does the same as clicking Apply, but closes the Color Viewer.

Remember that you still have to click on the Apply button in the Property Editor to apply the new color to your object.

Closing the Color Viewer

After you finish selecting colors, there are a number of ways to close the Color Viewer:

1. By clicking OK
OR
2. Choosing its File⇒Close command
OR
3. Double-clicking its Window menu button
OR

4. Pressing Alt+F, then C
OR
5. Pressing Alt+F4

The Color Editor

If you cannot find an existing system color that you like, you can mix a custom color with the Color Editor.

The Color Editor enables you to mix colors using three different color scales:

- HSI (Hue/Saturation/Intensity)
- RGB (Red/Green/Blue)
- CMY (Cyan/Magenta/Yellow)

You can use any of these scales, and toggle between them, as you prefer.

Opening the Color Editor

You open the Color Editor from the Color Viewer, with an object loaded. If the Color Editor is open, but hidden by other windows, you can bring it to the top by requesting it again. By default, the Color Editor appears with the HSI and RGB color scales active.

To Open the Color Editor

1. Make sure the Color Viewer is open with an object loaded.
2. Choose the Edit⇒Edit Color command from the Color Viewer.

The Color Editor appears, with the Selected Color shown as the Original Color with its RGB number at the top.

About the Color Editor

The Color Editor consists of five main areas, as shown in Figure 4-6.

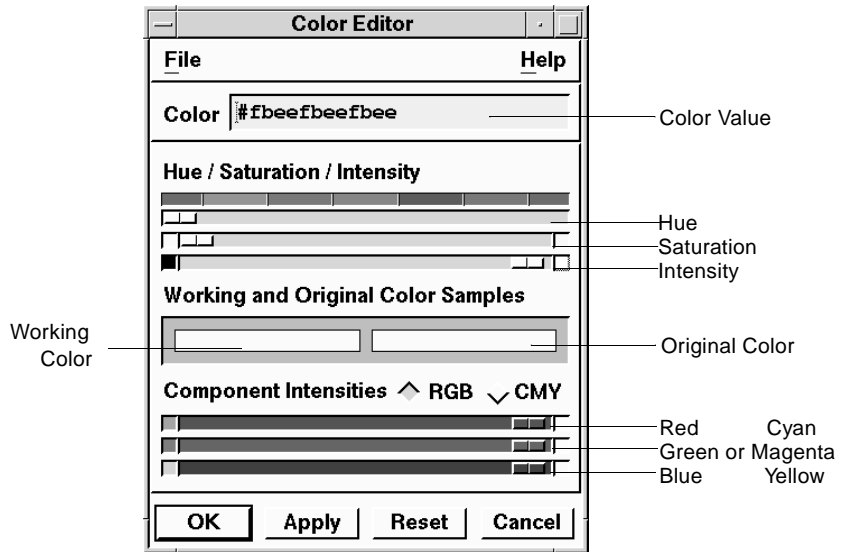


Figure 4-6 Main Areas of the Color Editor

- *Color Value*: Shows the RGB value of the Selected Color in the Color Viewer, changing to reflect your changes to that color.
- *Hue/Saturation/Intensity*: Provides three sliders used to adjust the HSI scale.
- *Working Color*: Displays a sample of your original color, changing to reflect your changes to that color.
- *Original Color*: Displays the original Selected Color in the Color Viewer.
- *RGB or CMY*: Provides three sliders used to adjust the RGB or CMY scales. You can click the radio button to toggle between scales at any time.

Mixing a Color

To mix a color, you start with an Original Color from the Color Viewer. The color you mix is your Working Color.

To Mix a Color with the Color Editor

1. In the Color Viewer, select a color close to your desired color.
2. Choose the Edit⇒Edit Color command.
3. In the Color Editor, move any of the sliders to mix your desired color.

You can use all three scales, and switch between RGB and CMY as you mix. The RGB value of your Working Color is updated automatically.

4. When the Working Color is the one you want, click Apply in the Color Editor. The Working Color becomes the Selected Color in the Color Viewer.
5. Click Apply in the Color Viewer.
6. Click Apply in the Property Editor.

To Reset the Color Editor

1. To reset your Working Color to your Original Color, click Reset *before* you click Apply.

Using the Apply and OK Buttons

After you mix a color, clicking Apply makes your Working Color the Selected Color in the Color Viewer, and keeps the Color Editor open. Clicking OK does the same as clicking Apply, but closes the Color Editor.

Remember that you still have to click on Apply in the Color Viewer and the Property Editor to apply the new color to your object.

Closing the Color Editor

After you finish mixing colors, there are a number of ways to close the Color Editor:

1. By clicking OK
OR
2. By choosing its File⇒Close command
OR
3. By double-clicking its Window menu button
OR

4. By closing the Color Viewer (which automatically closes the Color Editor)
OR
5. By pressing Alt+F, then C
OR
6. By pressing Alt+F4

The Color Map

The Color Map provides a palette of 256 colors, from which you can select any color simply by “grabbing” or clicking on it. This is probably the fastest and easiest way to select a color, provided you find the one you want. You can grab any color you see on the Color Map. You can also grab any color you see anywhere on your screen, using the Color Viewer, as described above.

The Color Map consists of a menu bar, a palette of up to 256 different colors, and an OK button, as shown in Figure 4-7.

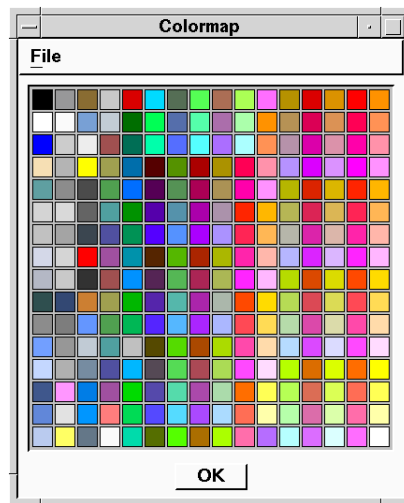


Figure 4-7 Color Map

To Grab a Color from the Color Map

1. Choose the Edit⇒Color Map command from the Color Viewer.
The Color Map is displayed.
2. Choose the Edit⇒Grab Color command from the Color Viewer.
3. Click on any color in the Color Map.

4. Click OK to close the Color Map.

The color you grabbed becomes the Selected Color in the Color Viewer, with its name and RGB value shown.

To close the Color Map without grabbing a color, click OK.

The Font Viewer

Five available object properties require a font name: `ButtonFontList`, `FontList`, `LabelFontList`, `MenuFontList`, and `TextFontList`. The Font Viewer enables you to choose any font available on your system, and see a sample of your choice on your screen. Remember that you should use as few different fonts as possible for visual consistency in your interface.

In UIM/X, font names are given as strings composed of three parts:

1. A short name or family name
2. A font (style) name, if required
3. A size and character set, if required

For instance, a typical short name for a font is "lucidasans-bold-18". Since this name gives a style (bold) and a size (18), it is already complete.

A typical family name for a font is "-linotype-helvetica". The full name for this font could be

```
"-linotype-helvetica--medium-r-normal-13-100-100-100-p-72-iso8859-7".
```

This kind of string is quite difficult to recall and type in by memory. Using the Font Viewer, you can compose even the longest font name by picking from lists on your screen.

Opening the Font Viewer

You open the Font Viewer from the Property Editor with an object loaded.

To Open the Font Viewer

1. Make sure the Property Editor is open with an object loaded.
2. Scroll to the appropriate property.
3. Click on the property button.

The Font Viewer appears.

When the Font Viewer is open, the property button becomes insensitive.

If the Font Viewer is open but hidden by other windows, click the editor button beside the property to move the Font Viewer to the top of your open windows.

About the Font Viewer

The Font Viewer is divided into five main areas, as shown in Figure 4-8.

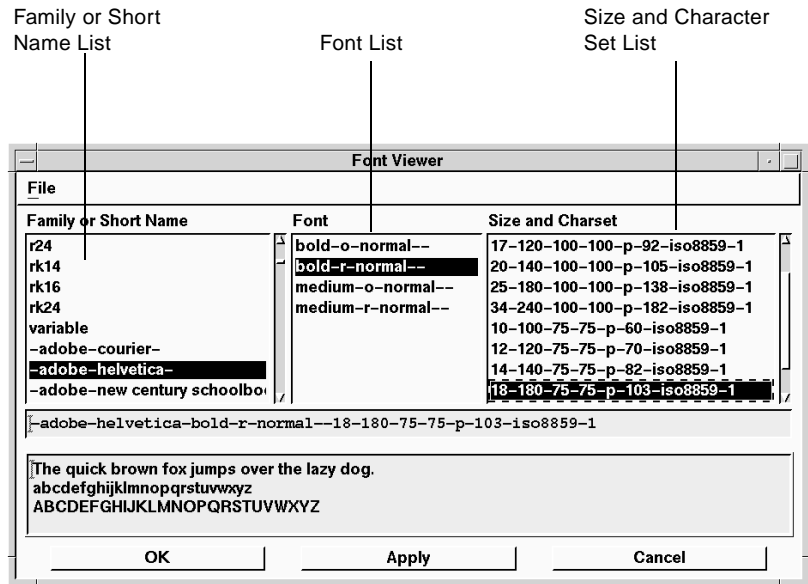


Figure 4-8 Main Areas of the Font Viewer

- *Family or Short Name List:* Lists your available fonts, either as short names or family names. Enables you to click any item from this list. If you select a font family name, the styles for that family appear in the Font List, and its sizes and character sets appear in the Size and Charset List.
- *Font List:* Lists the available font styles (such as medium and bold) for a selected family, and enables you to click any item from this list.
- *Size and Character Set List:* Lists the available sizes and character sets for a selected family, and enables you to click any item from this list.
- *Selected Font:* Displays the font name composed from your selections. This font name is copied to your object's property when you click Apply or OK.
- *Font Display Area:* Displays a sample of your selected font. You can also type in this area yourself and see your typing displayed in the selected font.

To Select a Font

1. Click on a font name in the Family or Short Name list.
2. If you selected a font family, click on a font style from the Font List and a font size or character set from the Size and Charset List.
3. Check how your font appears in the Font Display Area.

If you need any special characters, you can make sure these are available in your chosen font by trying to type them here.

4. When you are satisfied with your selection, click Apply.

The name of your Selected Font is copied to the object's property.

Using the Apply and OK Buttons

After you select a font, clicking on the Apply button copies the name of the font to the object's property, and keeps the Font Viewer open. Clicking OK does the same as clicking Apply, but closes the Font Viewer.

Remember that you still have to click Apply in the Property Editor to apply the new font to your object.

Closing the Font Viewer

After you finish choosing a font, there are a number of ways to close the Font Viewer:

1. By clicking OK
OR
2. By choosing its File⇒Close command
OR
3. By double-clicking its Window menu button
OR
4. By pressing Alt+F, then C
OR
5. By pressing Alt+F4

The Icon Viewer

Two available object properties require an icon name: `IconPixmap` and `LabelPixmap`. An icon name is the complete path name to a selected icon file (sometimes called a “pixmap” or “bitmapped graphic”), suitable for use with an object. With the Icon Viewer, you can locate, preview, and select an icon to use for an object.

Opening the Icon Viewer

You open the Icon Viewer from the Property Editor.

To Open the Icon Viewer

1. Make sure the Property Editor is open with an object loaded.
2. Scroll to the `IconPixmap` or `LabelPixmap` property.
3. Click on the property name button.

The Icon Viewer appears.

When the Icon Viewer is open, the property button becomes insensitive. If the Icon Viewer is open but hidden by other windows, click the editor button beside the property to bring it to the top of your open windows.

About the Icon Viewer

The Icon Viewer consists of five main areas, as shown in Figure 4-9.

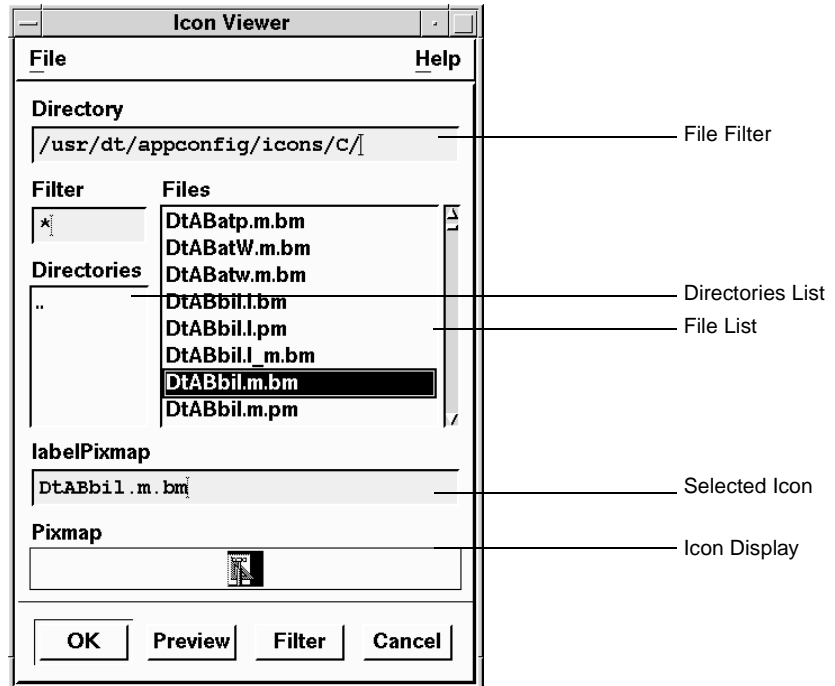


Figure 4-9 Main Areas of the Icon Viewer

- *File Filter:* Enables you to type in a file filter and then click on the Filter button to control what subdirectories appear in the Directories List and what files appear in the Files List.
- *Files List:* Displays the files in the current directory. You can click on a file name to preview it in the Icon Display.
- *Directories List:* Displays the subdirectories in the current directory. You can click on a directory to see its files in the Files List.
- *Selected Icon:* Displays the file name of the selected icon. This file name is copied to the object's property when you click OK.
- *Icon Display:* Displays the selected icon file. When the Icon Viewer is first opened, this area displays the message `No Icon Yet`.

To Select an Icon

1. Click on a directory name, if required, to locate the icon.
2. Click on the file name of the icon you want to see.

Or type the icon's name in the Selected Icon text field, and click on the Preview button.

3. Preview your icon in the Icon Display area.
4. When you have selected the icon you want, click OK.

The file name of your selected icon is copied to the object's property.

Closing the Icon Viewer

After you finish viewing icons, there are a number of ways to close the Icon Viewer:

1. By clicking OK
OR
2. By choosing its File⇒Close command
OR
3. By double-clicking its Window menu button
OR
4. By pressing Alt+F, then C
OR
5. By pressing Alt+F4

Building Menus

Overview

This chapter describes how to use the Menu Editor provided with the novice mode of UIM/X to create menus for your interfaces. Using the Menu Editor, you can create two standard types of menus, as shown in Figure 5-1:

- Pull-down menus, that drop down from the menu bar
- Option menus, that pop up from an option menu button

You cannot create pop-up or cascading menus with UIM/X novice mode. You can use either Push Buttons or Separators in pull-down menus and option menus. The Menu Editor enables you to quickly see and change all the properties associated with these menus.

For a hands-on example of using the Menu Editor to help build an interface, see Chapter 1, “Building Your First Project.”

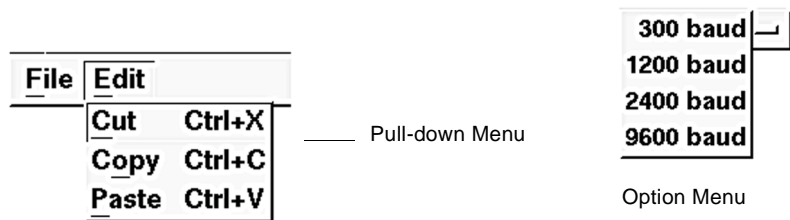


Figure 5-1 Two Types of Menus You Can Create with UIM/X novice mode

The Parts of a Menu

As shown in Figure 5-2, a menu consists of the following parts:

- One *Cascade Button* OR *Option Menu Button*, used to open the menu
- One *Pane*, used for the background
- Two or more *Items*, used for the menu options
- One *Mnemonic* per item (if desired)
- One *Accelerator* per item (if desired)

When you create a pull-down menu, it is automatically assigned a Cascade Button. You can give the Cascade Button any label, which is displayed in the menu bar. Pointing to the Cascade Button and pressing the Select button pulls down the associated menu.

When you create an option menu, it automatically receives an Option Menu Button. When you click on the Option Menu Button, the menu opens and stays open. When you click anywhere else on the option menu, the menu opens and stays open as long as you hold down the Select button.

When they are open, both menus display a background Pane. Each Pane is a RowColumn object that contains a list of Items or menu options.

A Mnemonic is a character you can press on the keyboard to open a pull-down menu or activate a menu option, such as E for Edit or C for Cut. A mnemonic must occur in the label for the pane or item, since it is underlined in the label.

An Accelerator is a keyboard shortcut you can use to activate a menu option, such as Ctrl+X for Cut or Ctrl+C for Copy. An accelerator need not occur in the label for the pane or item itself.

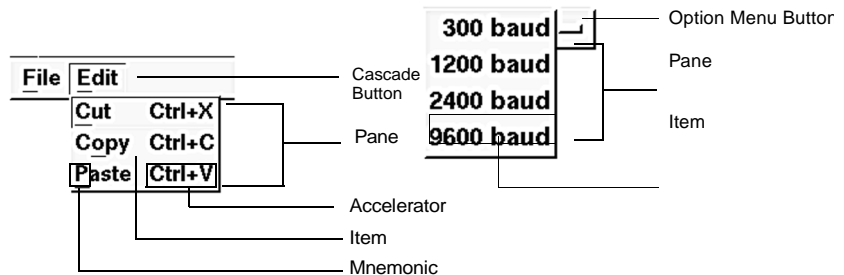


Figure 5-2 Parts of a Menu

Tips on Creating Menus

Well-designed menus can make an interface simpler for both beginners and experienced end users. To be effective, your menus must be well-planned, designed, and tested. Here are some basic tips on creating effective menus.

- Keep all menu labels short, simple, and logical.
- Use standard menu names that users are familiar with, including “Edit”, “Tools”, “Options”, “Search”, “Window”, “Special”, “View”, or “Format”.
- Label any option that leads to a dialog box with “...” as in “Open...”
- Group all similar tasks or functions in the same menu.
- Use separators to group similar items within the same menu.
- Add mnemonics and accelerators so end users can activate options from their keyboards if they prefer.
- Do *not* use a mnemonic or accelerator for any option that could delete the end user’s work. This saves users from losing their work by accident.
- Set option menus to the most likely values before shipping an interface.

After you create your menus, you can give them behavior by writing the appropriate callbacks in the Menu Editor.

Using Pull-down Menus

A pull-down menu can provide either global commands, such as options to set, or specific actions to perform on selected objects, such as cut, copy, or paste. The Application Window provides two pull-down menus ready for you to use: File and Help. You may have these two or more pull-down menus in a menu bar. If you do not want any pull-down menus in an interface, use a Secondary Window, not an Application Window.

Using Option Menus

An Option Menu presents a set of mutually-exclusive choices; in other words, a list of possible options from which the end user chooses one. Option Menus often provide a list of possible values, such as baud rates for a port. You can have no, one, or several Option Menus in an interface.

The last value you select while testing or using an Option Menu remains set after you generate code or exit (unless you insert code to reset it). In effect, it remains the default. You may want to select the most likely option for any Option Menus before shipping an interface to any end users.

Objects in Menu

Like everything else in an interface, a menu is built from objects. The menu bar consists of Cascade Buttons and Row Columns. Each Row Column consists of Push Buttons and Separators, which are its children. The Option Menu also includes Push Buttons and Separators as its children.

Objects in Pull-down Menus

When you add a new pull-down menu to the menu bar, the program creates another Row Column object to hold the new pane. Each pane is assigned a unique name made up from the menu name, the letters *rc* for RowColumn, and a number showing the order of its creation. These numbers are assigned consecutively across interfaces within a project.

For example, the File pane in your first menu bar is called `file_menurc1`, and the Help pane is `help_menurc1`. If you create a second interface, its File pane is called `file_menurc2`, and its Help pane `help_menurc2`. The first pull-down menu you add to `applWindow1` is `applMenuBar1_p3`. Then if you add a pull-down menu to `applWindow2`, it is `applMenuBar2_p3`, and so on.

Adding a new pull-down menu automatically creates a new Cascade Button, which is assigned a unique name. The name and properties for the Cascade Button are not available for viewing or editing in UIM/X novice mode, since these are automatically managed by the menu object.

Objects in Option Menus

When you create an Option Menu, a new pane is added to manage the vertical arrangement of the items and to display the current menu item. Unlike a menu bar, an Option Menu can only contain one pane, so the `Create⇒Pane` option is unavailable. The pane is the child of the Option Menu, while the menu items are the children of the pane.

Adding an Item to a Menu

When you add a new item to a menu pane, the item is assigned a unique name made up from the pane name, the letter *b* for button, and a number indicating the order of its creation: `file_menurc1_b1`, `file_menurc1_b2`, `file_menurc2_b1`, `optionMenu_rc1_b2`, `optionMenu_rc1_b3`, and so on.

Steps in Using the Menu Editor

There are six basic steps to using the Menu Editor:

1. Opening the Menu Editor.
2. Creating a pane (for pull-down menus *only*), and creating all the items.
3. Setting the properties for each pane and item.
4. Adding callbacks if required.
5. Applying your changes to save them.
6. Testing the new menu, and revising it if necessary.

Opening the Menu Editor

There are two slightly different versions of the Menu Editor, one used for pull-down menus and another for option menus. The appropriate editor opens automatically, depending whether you are working on a pull-down menu or an Option Menu. You can have several instances of the Menu Editor open at once.

To Open the Menu Bar Editor

1. Double-click on the name of any pull-down menu in a menu bar.

The Menu Bar Editor appears, loaded with the menu bar for the Application Window.

To Open the Option Menu Editor

1. Double-click on any Option Menu in an interface.

The Option Menu Editor appears, loaded with the selected Option Menu.

About the Menu Editor

The Menu Editor consists of five main areas, as shown in Figure 5-3.

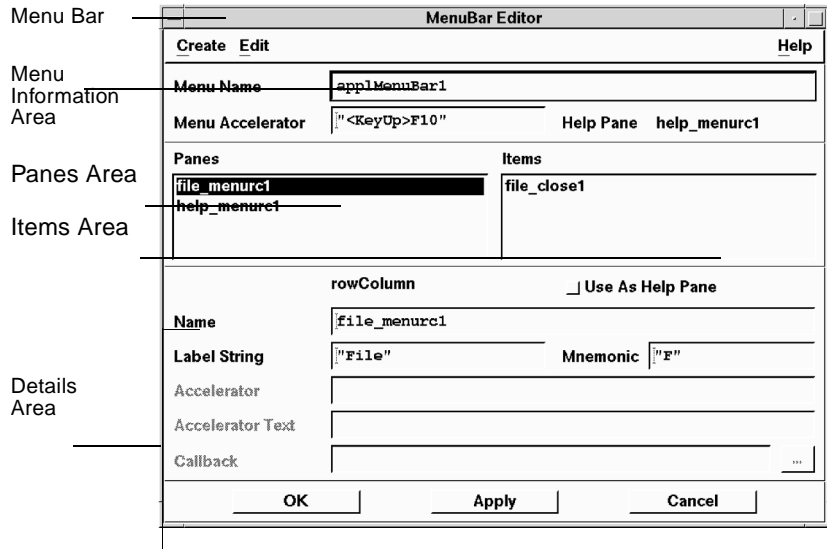


Figure 5-3 Main Areas of the Menu Editor

- *Menu Bar:* Provides two pull-down menus: Create to create a pane or item; and Edit to duplicate or delete a pane or item.
- *Menu Information Area:* Enables you to type in certain properties of the menu. This area changes slightly, depending on whether you are editing a pull-down menu or an option menu.
- *Panes Area:* Lists the panes in the current menu. A menu bar has several panes, one for each pull-down menu. An option menu has only one pane. You can select any pane to see its properties in the Details area.
- *Items Area:* Lists the items in the current pane. You can select any item to see its properties in the Details area.
- *Details Area:* Displays properties of the current pane or item: Name, Label String, Mnemonic, Accelerator, Accelerator Text, and Callback.

Creating a Pull-down Menu

Using the Menu Editor, you can add a custom pull-down menu to the menu bar, with whatever label and items you prefer.

To Create a Pull-down Menu

1. Double-click on the name of any pull-down menu in a menu bar.
The Menu Bar Editor appears, loaded with the menu bar for the Application Window, as shown in Figure 5-3.
2. Choose the Create⇒Pane command.
A new pane appears in the Panes area, called `applMenuBar1_p3`.
3. Click between the quotes (" ") in the Label String text field, and type in the name for your new menu, exactly as it should appear in the menu bar.
4. Choose the Create⇒Item command, and then choose Push Button.
A new item appears in the Items area, called `applMenuBar1_p3_b1`.
5. Type in a Label String, Mnemonic, Accelerator, and Accelerator Text property for the new item as required.
6. Repeat steps 4 and 5 to create all the items you want for your new pull-down menu. Add Separators as required.
7. Click on the Apply button.

The new menu label appears on the menu bar, as shown in Figure 5-4. You can add the callbacks for this menu later on.

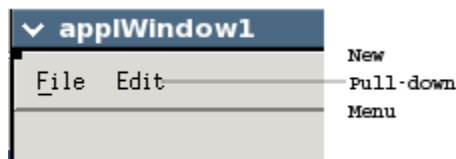


Figure 5-4 Your New Pull-down Menu

Creating an Option Menu

You can drag and drop, or drag and draw, an Option Menu from the Palette to an interface like any other object. You can give your Option menu whatever items you prefer, and set it to your chosen default.

To Create an Option Menu

1. Add an Option Menu to an interface.
2. Double-click on the new Option Menu.
The Option Menu Editor appears, loaded with the new Option Menu.

3. To label the Option Menu, click between the quotes (" ") in the Label String text field at the top, and type in your label. (This step is optional.)
4. Click on the first item called `optionMenu_pb1`.
5. Adjust the Label String, Mnemonic, Accelerator, and Accelerator Text properties of the item as required.
6. Choose the Create⇒Item command, then choose Push Button.

A new item appears in the Items area, called `optionMenu_rc1_bn`.
7. Adjust the Label String, Mnemonic, Accelerator, and Accelerator Text properties of the new item as required.
8. Repeat steps 6 and 7 to create all the items you want for your Option Menu. Add Separators as required.
9. Click on the Apply button.

The Option Menu reappears in the interface, relabelled if you specified in Step 3, as shown in Figure 5-5. You can add the callbacks for this Option Menu later on.



Figure 5-5 Your New Option Menu

Using Mnemonics

A mnemonic is an underlined character in the label for a menu pane or item, such as the P in Print. Providing a mnemonic enables the end user to open a pull-down menu and choose any item with the keyboard instead of the mouse.

To activate a mnemonic, the end user presses that key along with the appropriate Meta key. Depending on the end user's keyboard, this could be Extend char+*mnemonic*, Alt+ *mnemonic*, or Compose Character+*mnemonic*.

Pressing the mnemonic for any pull-down menu, such as *Meta key*+F for File, opens the menu. With the menu open, the end user can press the mnemonic for any menu item without using the Meta key, such as O for Open. You cannot provide a mnemonic to open an Option Menu; the end user must always click on the Option Menu Button to do so. With the Option Menu open, the end user can press the mnemonic for any menu item without the Meta key.

To Create a Mnemonic for a Pane or Item

1. With the Menu Bar Editor open, select the pane or item to receive a mnemonic.
2. Click between the quotes (" ") in the `Mnemonic` text field, and type a single letter that occurs in the label for that pane.
3. Click on the Apply button.

Using Accelerators

An accelerator is a key combination that activates a menu item without opening the related menu. You can also use an accelerator to open a menu. An accelerator is displayed in the menu to the right of the related option, to show which keystroke(s) you can press to activate it, such as `Ctrl+X` for Cut. You can provide an accelerator for any Push Button in a menu.

Not every option needs an accelerator; in fact, any option that deletes the end user's work should *not* have one. Not giving an accelerator for a destructive option avoids the chance of a user accidentally pressing the keys to activate it.

To create an accelerator for a menu or menu item, you need to set both the `Accelerator` and `AcceleratorText` properties. The `Accelerator` property specifies the event to associate with the button, specified as a standard X keysym. For example, the accelerator `Shift+F3` is given as the string `Shift<Key>F3`.

The `AcceleratorText` property specifies a text string to display next to the related menu item. For example, for the accelerator `Shift+F3`, the `AcceleratorText` string could be given as `"Shift+F3"`. This property only displays the related text; it does not assign that accelerator. Always test your accelerators to make sure that both properties are set.

To Create an Accelerator

1. With the Menu Editor open, select the pane or item to receive an accelerator.
2. Click between the quotes (" ") in the `Accelerator` text field and type a standard X keysym string to represent the accelerator.
3. Click between the quotes (" ") in the `AcceleratorText` field and type the accelerator string to be displayed with that menu item.
4. Click on the Apply button.

When you open the menu, the accelerator appears as you specified.

Duplicating a Pane or Item

You can use the Duplicate command to save time setting up items with several properties in common. Duplicating a pane or item copies all the property values displayed in the Menu Editor. A duplicated pane appears to the right of the existing panes in the menu bar, and to the left of Help. A duplicated item appears below the item you duplicated.

You can also use the Duplicate command along with Delete to rearrange your menu bar and items, as described in the next sections.

To Duplicate a Pane or Item

1. Select the pane or item to duplicate.
2. Choose the Edit⇒Duplicate command.

The selected pane or item is duplicated as you specified.

Deleting a Pane or Item

Deleting a pane or item removes it permanently from an interface. There is no way to “undo” your deletion, or paste the deleted pane or item back in. Always select the correct pane or item to delete. When you delete a pane or item, the previous pane or item in the list is selected. Each pane must contain at least one item, so you cannot delete the last item from a pane and then apply your changes.

You can also use the Delete command along with Duplicate to rearrange your menu bar and items, as described in the next sections.

To Delete a Pane or Item

1. Select the pane or item to delete.
2. Choose the Edit⇒Delete command.
You see a message asking you to confirm your deletion.
3. Click OK.

The pane or item is deleted as you specified. There is no way to undo.

Rearranging Your Menu Bar

Always plan your menu bar before you create it. By default, each new pull-down menu is added to the right of the existing menus in the menu bar, and to the left of Help. If you need several pull-down menus to appear in a certain order from left to right, you can create them in the same order.

Or, after your menus are created, you can rearrange them using the Duplicate and Delete commands. For example, suppose you have a menu bar that includes File, A, B, C, Help. You want to rearrange it as File, A, C, B, Help; in other words, you want to move menu C to the left of menu B. To do so, you can:

- Duplicate menu B (yielding File, A, B, C, B, Help)
- Delete the original menu B (yielding File, A, C, B, Help)

You can also switch any menu with the Help menu at the far right. You can therefore move any pull-down menu anywhere on the menu bar.

To Rearrange the Menu Bar

1. With the Menu Bar Editor open, select the pane you want to move right.
2. Choose the Edit⇒Duplicate command.
A duplicate of the pane appears to the right of the last pane you added.
3. Select the original pane you duplicated, choose the Edit⇒Delete command, and click OK to confirm.
The pane you duplicated is deleted.
4. Repeat steps 2 and 3 until the menu bar is rearranged as you want.

To Make Any Menu a Help Menu

Menu panes are placed from left to right on your menu bar in the order in which they were created. The only exception to this rule is the Help pane, which is always placed on the far right of the menu bar.

However, you can make any menu pane a Help menu.

1. With the MenuBar Editor open, select the pane you want to occupy the Help menu's position.
2. Click on the Use as Help Pane check button.
3. Click on the Apply button.

The menu bar reappears with your selected pane in the Help menu's former location. The sequence of menu panes is reordered such that the Help pane's position now corresponds to the order in which it was created.

Rearranging Your Menu Items

You can control the order of the items in a pull-down or option menu as you create them. You can create an item to be displayed below any existing item, or at the top of the menu. The items in a menu are displayed in the same order they occur in the Items list.

To Add an Item Below an Existing Item

1. Select the item you want to see *above* the new item you want to create.
2. Choose the Create⇒Item command, then choose an item.
3. Adjust the Label String, Mnemonic, Accelerator, and Accelerator Text properties of the new item as required.
4. Click on the Apply button.

When you open the menu, your new item appears *below* the specified item.

To Add an Item at the Top of a Menu

1. Select the item at the top of the menu.
2. Choose the Edit⇒Duplicate command.
A duplicate of the top item appears second from the top in the Items list.
3. Re-select the item at the top of the menu.
4. Choose the Create⇒Item command, then choose a button.
Your new item appears second from the top in the Items list.
5. Adjust the Label String, Mnemonic, Accelerator, and Accelerator Text properties of the new item as required.
6. Re-select the item at the top of the menu once again.
7. Choose the Edit⇒Delete command, then click OK to confirm.

When you open the menu, your new item appears at the top of the menu, and the item that was at the top is now second.

Adding Callbacks to Menus

Remember that until you enter a callback for a menu item, selecting it will not do anything. You can add or change the callback for a menu item at any time, using the Callback area in the Menu Editor. This callback code will run whenever the associated menu item is selected. You should always test the behavior of each menu item when you switch to Test mode.

To Add a Callback to a Menu Item

1. Open the Menu Editor with a menu loaded.
2. Click on the item to enter a callback for.

3. Click on the ... button beside the `Callback` text field.
The Text Editor appears, as shown in Figure 5-6.

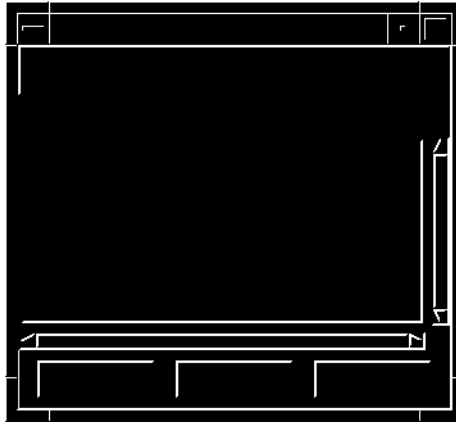


Figure 5-6 Text Editor

4. Click between the braces (`{ }`) and type the C code for your callback.
5. When you finish typing your callback, review it carefully, and then click OK.
The Text Editor disappears. The start of your callback code is visible in the `Callback` text field in the Menu Editor.
6. Now click `Apply` in the Menu Editor.
Your callback is now applied to your selected menu item.

Testing Your Menu

After you create a menu, you should always check to make sure it appears exactly as you expected. You should also test the menu's behavior, to make sure it works exactly as you expected.

To Test a Menu

1. Click on the `Test` radio button in the Project Window.
The Palette and any open editors disappear.
2. Open your new menu.
3. Proofread all menu items.
Ask yourself if the wording, spacing, and grouping of all items is clear, simple, and logical.
4. Check that you provide enough Separators.

5. Check that you provide mnemonics and accelerators for all non-destructive options, if required.
6. If you have callbacks for any menu items, select each menu item and make sure it works as you expected.
7. After you finish testing your menu, click on the Design radio button in the Project Window.

Then revise your menu if required, and repeat your testing from step 1.

Closing the Menu Editor

After you finish creating or revising menus, and apply your changes, there are a number of ways to close the Menu Editor:

1. By clicking OK
OR
2. By double-clicking its Window menu button
OR
3. By pressing Alt+F4.

Remember that any panes or items you do not apply before closing the Menu Editor *are not saved*.

Adding Connections

Overview

The Connection Editor establishes a behavioral connection between a source and target object by automatically creating snippets of callback code based on user-specified criteria.

Each connection is defined as a rule of the form: For *Source Object*, on *Callback*, perform *Method* on *Target Object*.

An example of a simple behavioral connection might be:

- For `pushButton1` (source), on `ArmCallback` (callback), use `setLabel` (method) to display the string “Hello” in `label1` (target).

Steps in Using the Connection Editor

1. Opening the Connection Editor
2. Loading a source
3. Loading a target
4. Defining Connections
5. Modifying Connections
6. Closing the Connection Editor

Each of these steps is described in the remainder of this chapter.

Opening the Connection Editor

You can open the Connection Editor in three different states:

- With a source object loaded,
- With both source and target objects loaded, or
- Empty, with no object loaded.

With a Source Object Loaded

1. Select any object, press the Menu mouse button, and choose the Selected Objects⇒Tools⇒Connection Editor command

OR

2. Choose the Tools⇒Connection Editor option from the Project Window menu bar.

The Connection Editor appears, with the selected object loaded as the source.

With Both Source and Target Objects Loaded

1. Drag the mouse pointer (using the Select mouse button) from a source object to a target object while depressing the Shift key.

A line drawn between the source and target objects appears during this operation to indicate the intended connection.

2. Release the mouse button.

The Connection Editor appears, with the selected objects loaded as source and target, respectively.

With No Objects Loaded

1. With no objects selected, choose the Tools⇒Connection Editor option from the Project Window menu bar.

The Connection Editor appears with all fields empty, ready to load objects into it.

Once the Connection Editor is open, you can proceed to establish functional connections between source and target objects.

About the Connection Editor

The Connection Editor comprises six main areas, as illustrated in Figure 6-1.

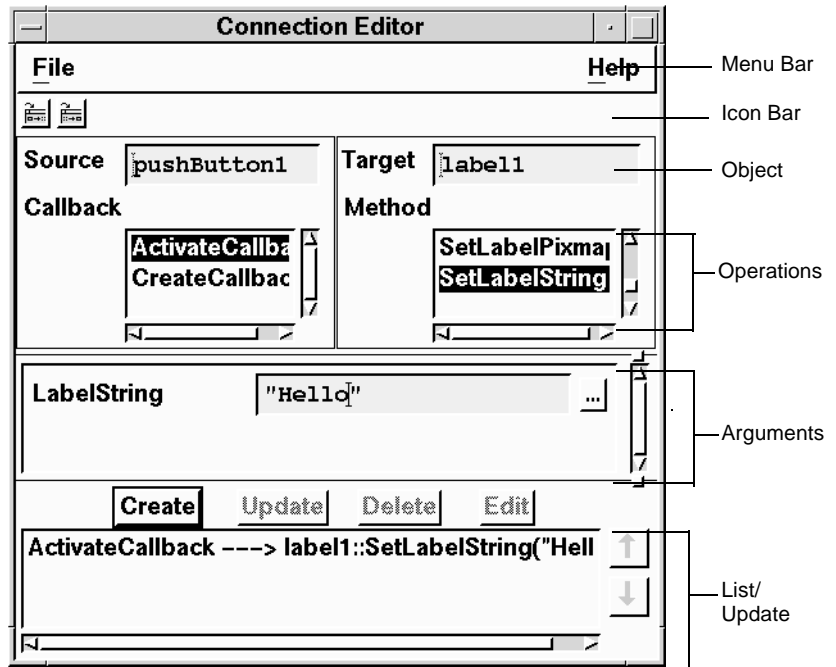



Figure 6-1 Main Areas of the Connection Editor

- *Menu Bar*: Provides a File menu to load source and target objects and reset or close the Connection Editor, and a Help menu to provide online help.
- *Icon Bar*: Provides icons for loading source and target objects.
- *Object area*: Enables selection of Source and Target objects.
- *Operations area*: Enables selection of the type of callback and method.
- *Arguments area*: Displays the values of the arguments for the currently selected method (including the return value, where applicable), and allows the user to change these values. Default values are provided initially.
- *List/Update area*: Provides a display area for the created connections and enables selection for subsequent actions using the Create, Update, Delete, and Edit command buttons.

Loading a Source

If you have opened the Connection Editor with an object selected, the Connection Editor appears with the selected object loaded as the Source. All available callbacks for the object are displayed in a scrolled window directly below the Source field.


If you have opened the Connection Editor with no object selected, or if you wish to select a new source, simply:

1. Select an object from your interface, then choose File⇒Load Source from the Connection Editor menu bar or its corresponding icon  from the icon bar
OR
2. Drag an object from the Browser and drop it in the Source field
OR
3. Type the object name into the Source text field and press Return.

The selected object and its associated callbacks are then loaded into the Connection Editor.

Loading a Target

To load a Target:

1. Select an object from your interface, then choose File⇒Load Target from the Connection Editor menu bar or its corresponding icon  from the icon bar

OR

2. Drag an object from the Browser and drop it in the Target field

OR

3. Type the object name into the Target text field and press Enter.

The selected object and its applicable methods are then loaded into the Connection Editor.

Defining Connections

With both Source and Target loaded, you can begin to define a connection. To define a connection, you must select a Callback and a Method from the Operations area:

1. Click a callback from the Callback scrolled window.
2. Select a method by clicking the method name in the Method scrolled window.

When applicable to the selected method, the default value of the method's associated arguments (including a return value, where applicable) are displayed in their respective text fields in the Arguments area of the Connection Editor. You can change the arguments' values simply by typing in new values or by selecting a (...) button to invoke the text editor for the corresponding argument.

3. Create the connection by selecting the Create command button.

The created connection is displayed and highlighted in the list section of the Connection Editor's List/Update area, for example:

```
CreateCallback--->label1::SetLabel("Hello")
```

To create additional connections, repeat the above steps. Always make sure to complete the procedure using the Create command button. Thus, the Connection Editor will add a new connection to the end of the list rather than replace an existing (highlighted) connection.

To duplicate a connection, highlight the connection and select Create. The connection is duplicated and added to the list.

Modifying Connections

With the connection highlighted, you can perform further operations by selecting a command button in the List/Update area or selecting a different type of callback or method from the Operations area.

To Modify Connections

1. If the connection is not already selected, select it from the list by clicking it;
2. Select a connection and/or method from the Operations area, as earlier described. Edit the argument values if you like.
3. Select the Update command button.

The newly created connection appears in the list section, replacing the previously selected connection.

To Modify Arguments

1. If the connection is not already selected, select it from the list by clicking it and then click the Edit push button.

The current values of the arguments appear in their respective text fields in the Arguments area.

2. Type in a new value for the argument or select the (...) button adjacent to the text field to invoke the text editor.
3. Select the Update command button.

The connection is updated with the new argument value.

To Delete a Connection

1. If the connection is not already selected, select from the list by clicking it.
2. Select the Delete command button.

The connection is erased from the list.

Closing the Connection Editor

After you finish establishing connections, there are a number of ways to close the Connection Editor. You can close the Connection Editor:

1. By choosing its File⇒Close command
OR
2. By double-clicking on its Window menu button

OR

3. By pressing Alt+F, then C

OR

4. By pressing Alt+F4.

Adding Constraints

Overview

The Constraint Editor allows you to define form constraints graphically without needing to know the numerous Motif form constraint properties. It allows you to create interfaces which maintain proportion perfectly when resized either manually or due to a font change.

Steps in Using the Constraint Editor

1. Opening the Constraint Editor
2. Specifying Constraints
3. Applying Constraints
4. Closing the Constraint Editor

Each of these steps is described in the remainder of this chapter.

Opening the Constraint Editor

You can open the Constraint Editor by:

1. Selecting Tools⇒Constraint Editor from the Project Window menu bar
OR
2. Using the Menu mouse button to choose the Selected
Objects⇒Tools⇒Constraint Editor command.

The Constraint Editor appears, with a visual representation of the selected form displayed.

Once the Constraint Editor is open, you can proceed to establish constraints.

About the Constraint Editor

The Constraint Editor comprises four main areas, as illustrated in Figure 7-1.

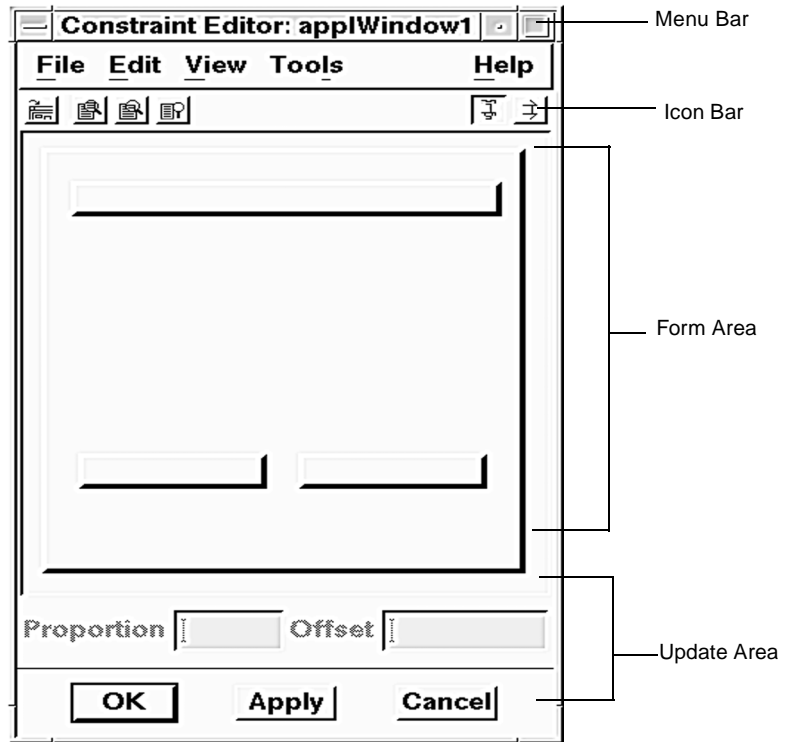


Figure 7-1 Main Areas of the Constraint Editor

- *Menu Bar:* Provides:
 - A File menu, to load an interface and reset or close the Constraint Editor;
 - An Edit menu, to delete or recreate previously defined constraints;
 - A View menu, to provide zoom in, zoom out, and actual size capability;
 - A Tools menu, to select the tools Bolt or Dimension;
 - A Help menu, to provide online help.

- *Icon Bar*: Provides icons to enable quick selection of Tools menu and menu bar options. Bubble help is provided for all icons.
- *Form area*: Displays the interface and provides a working area in which to select objects and define constraints;
- *Update area*: Displays and allows editing of constraint parameters and enables the user to apply or cancel constraints. The Update Area controls comprise:
 - The Proportion text field, which displays and allows modification of Proportion for Dimension constraints;
 - The Offset/Length text field, which displays and allows modification of Offset for Dimension constraints and Length for Bolt constraints;
 - The OK button, which applies constraints and closes the Constraint Editor;
 - The Apply button, which immediately applies constraints and displays them graphically in the Form area; and
 - The Cancel button, which cancels applied values in the Update area and closes the Constraint Editor.

Defining Constraints

You can establish constraints between children on a form or between a child and its parent. The Constraint Editor, however, will not allow you to create a set of contradictory or conflicting constraints.

You define constraints by:

1. Selecting a tool from the Tools menu or from the icon bar;
2. Positioning the mouse pointer over the edge of the object to which the constraint is to be applied; and
3. Clicking the Select mouse button or drawing a connection to another object, depending on the constraint selected.

Selecting a Tool

The type of constraint applied to an object is determined by the tool selected. The tools and their functions are described in the remaining paragraphs.

Tool	Function
Bolt	Used to apply Bolt constraints to selected objects.
Dimension	Used to apply x-axis and/or y-axis Dimension constraints to selected objects.

Bolt

The Bolt constraint allows you to anchor one object to another such that an absolute distance between the objects is maintained during subsequent move and resize operations.

To impose a Bolt constraint:

1. Select the Bolt icon on the icon bar or Tools⇒Bolt from the menu bar.
2. Position the mouse pointer on an edge of the object to be bolted.

When the pointer is over a valid area, the edge of the object will turn white.

3. Press and hold the Select mouse button.

The valid destinations, that is, edges of objects to which the first object can be bolted, turn black.

4. Drag the mouse pointer to a valid destination.

When the mouse pointer is positioned over a valid edge, the edge turns white.

5. Release the mouse button.

The applied constraint is depicted graphically by a bolt symbol, as illustrated in Figure 7-2. The caption of the right-hand text field changes to “Length” and the text field displays the length to be maintained between objects during subsequent move and resize operations.

To change the value of Length, type a new value into the text field and select Apply. The graphics in the form area are immediately updated to reflect the new value.

ADDING CONSTRAINTS

Defining Constraints

Bolt constraints can be established between two objects on a form or between an object and the form itself.

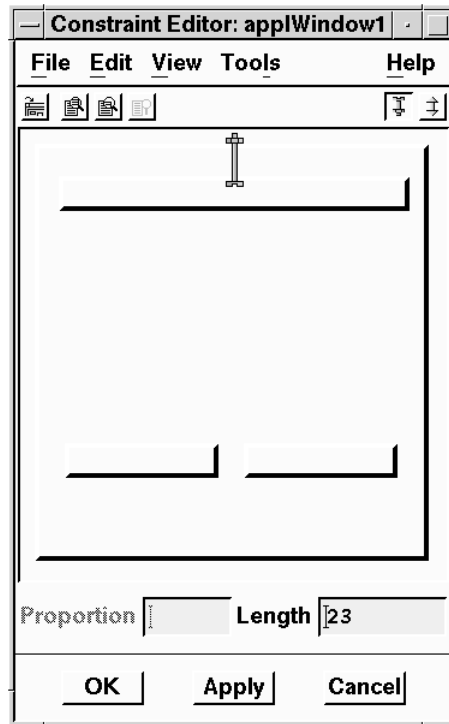



Figure 7-2 Bolt Constraint

Dimension

The Dimension tool allows you to constrain an object such that during subsequent move or resize operations, the position of the selected edge remains fixed at a proportionate distance from the left (y-axis) or bottom (x-axis) edge of the form.

Applying a Dimension constraint to the upper or lower edge of an object imposes a y-axis constraint. Applying a Dimension constraint to the left or right edge of an object imposes an x-axis constraint. Once such a constraint is applied, you cannot move the object from its position on that axis.

To impose a Dimension constraint:

1. Select Tools⇒Dimension from the menu bar or select the Dimension icon  on the icon bar.
2. Position the pointer on an edge of the selected object.

3. When the pointer is over a valid edge, the edge turns white.
4. Press the Select mouse button.

The applied constraint is depicted graphically by a double-headed arrow linking the selected object to the parent, as illustrated in Figure 7-3.

The caption of the left-hand text field changes to “Proportion” and the text field displays the percentage of the form (in the appropriate dimension) occupied by the Dimension arrow.

The caption of the right-hand text field changes to “Offset” and the text field displays the number of pixels to offset the edge from its base position.

To change the values of Proportion or Offset, type new values into their respective text fields and select Apply. The graphics in the form area are immediately updated to reflect the new values.

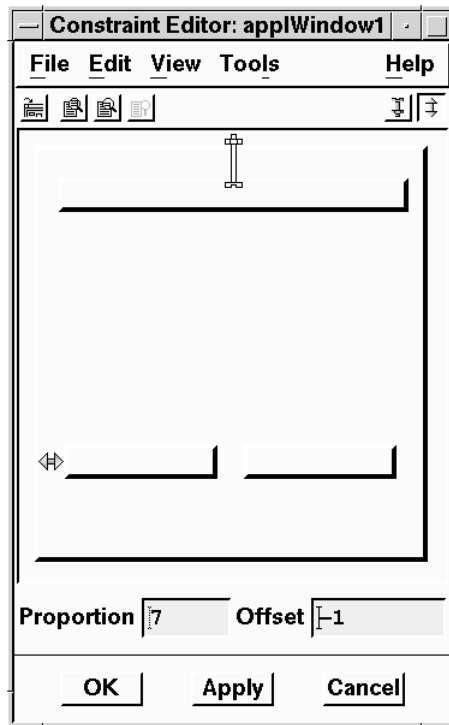


Figure 7-3 Dimension Constraint

Applying Constraints

Selecting the OK push button applies any new values in the Update area and closes the Constraint Editor window.

To apply new values without closing the Constraint Editor, select the Apply push button.

Removing Constraints

You can remove all previously applied constraints simultaneously or remove individual constraints.

To remove all constraints:



1. Select the Edit⇒Select All command from the menu bar or use the Menu mouse button to choose the Selected Objects⇒Select All command.
2. Select the Edit⇒Delete command from the menu bar or use the Menu mouse button to choose the Selected Objects⇒Delete command.


To remove individual constraints:

1. Position the mouse pointer over the graphic symbol for the constraint to be deleted. The constraint symbol turns white.
2. Press the Select mouse button. The constraint symbol appears in its highlighted (selected) color.
3. Select the Edit⇒Delete command from the menu bar or use the Menu mouse button to choose the Selected Objects⇒Delete command.

View Menu Options

The Constraint Editor's View menu provides three options, which are also selectable via their respective icons on the icon bar: Zoom In, Zoom Out, and Actual Size.

The Zoom In  and Zoom Out  features allow you to select a progressively greater factor of magnification or minimization of the form area with each successive selection of the menu option or click of its respective icon.

Actual Size  reverts the viewable interface in the form area to its original size.

These features can be useful in facilitating the manipulation of objects with a small “grab” area (for example, separators), as well as alleviating the need to use the scroll bars to navigate the viewable area of the interface.

The View menu features are provided for editing convenience only and have no effect on the actual size of the selected interface.

Closing the Constraint Editor

Once you have finished adding constraints, there are a number of ways to close the Constraint Editor. You can close the Constraint Editor by:

1. By choosing its File⇒Close command
OR
2. By double-clicking on its Window menu button
OR
3. By pressing Alt+F, then C
OR
4. By pressing Alt+F4.
OR
5. By clicking the OK push button
OR
6. By clicking the Cancel push button.

Browsing Object Hierarchies

8

Overview

In a complex interface, it can be difficult to select a window or shell object that is hidden by its children. To help you work with complex interfaces, UIM/X provides a Browser for editing object hierarchies. The Browser displays objects in a compact format, providing a tree (or outline) view of the objects in an interface, as shown in Figure 8-1.

Using the Browser, you can quickly review your interface, select any object, create any new objects you need, move existing objects up or down the hierarchy, and reassign objects to different parents.

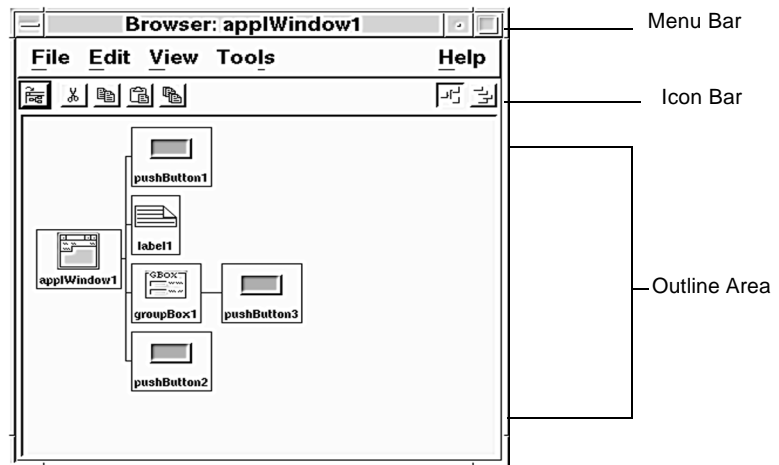


Figure 8-1 Tree View of an Interface in the Browser

About the Browser

The Browser comprises three main areas, as illustrated in Figure 8-1:

- *Menu Bar*: Provides:
 - A File menu, to load interfaces and close or reset the Browser;
 - An Edit menu, from which objects in the interface area may be selected/deselected, cut, copied, pasted, duplicated, and deleted. You can also select the Arrange and Align features from the Edit menu;
 - A View menu, to enable selection of viewing options for the outline area;
 - A Tools menu, to provide access to the various editors supplied with UIM/X;
 - A Help menu, to provide online help.
- *Icon Bar*: Provides icons for quick selection of commonly used Menu Bar commands. Bubble help is provided for all icons;
- *Outline area*: Displays the interface hierarchy and provides a working area in which to perform operations provided by the Menu Bar and Icon Bar.

Steps in Using the Browser

1. Opening the Browser.
2. Loading an interface.
3. Browsing the interface.
4. Viewing instances of Objects
5. Changing view
6. Closing the Browser.

Each of these steps is described in the remainder of this chapter.

Opening the Browser

The Browser can be opened empty or with an interface loaded.

If an object on the interface has been selected, you can open the Browser by:

1. Selecting Tools⇒Browser from the Project Window

OR


2. Choosing the Selected Objects⇒Tools⇒Browser command.

The Browser window appears loaded with the selected interface.

If no object on the interface has been selected, open the Browser as described in Step 1. The Browser window appears empty. You can then proceed to select and load an interface.

Loading an Interface into the Browser

If the Browser window is opened with no interface loaded,

1. Click on your interface, or click on its icon in the Interfaces area of the Project Window.
2. Select File⇒Load from the menu bar of the Browser, or its respective icon  from the icon bar.

The selected interface is loaded into the Browser.

Browsing the Interface

The Browser allows you to view and edit the object hierarchy of your interface as well as perform many of the operations you would normally do by working within the interface itself.

Selecting an object in the Browser is exactly like selecting it in the interface. Operations such as dragging and dropping an object from the Browser to an editor and duplicating objects are equally possible.

The Browser's Edit menu, while furnishing such editing functions as Copy, Cut, Paste, Duplicate, Delete, and Recreate, also provides access to the Arrange and Align features of UIM/X, which facilitate the alignment of objects on the interface.

From the Browser's Tools menu, you can open the Property Editor, Constraint Editor, Menu Editor, Connection Editor, and Declaration Editor, and drag and drop selected objects into them to define object and interface behavior.

Re-parenting an Object

Using the Browser, you can easily reassign an object to a different parent by dragging and dropping.

Figure 8-2 shows a simple interface and its representation in tree view in the browser.

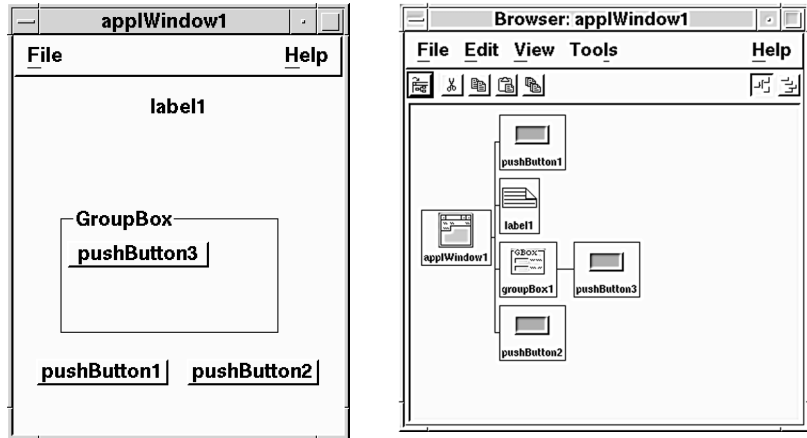


Figure 8-2 A Simple Interface and its Browser Representation

Suppose that you wanted to re-parent `pushButton2`, a child of `applWindow1`, such that it would become a child of `groupBox1`.

In the Browser:

1. Point to `pushButton2` and press and hold the Adjust mouse button.
2. Drag `pushButton2` from its present location on the tree to directly over `groupBox1` and release the mouse button.

The push button is now a child of groupBox1, as depicted in Figure 8-3.

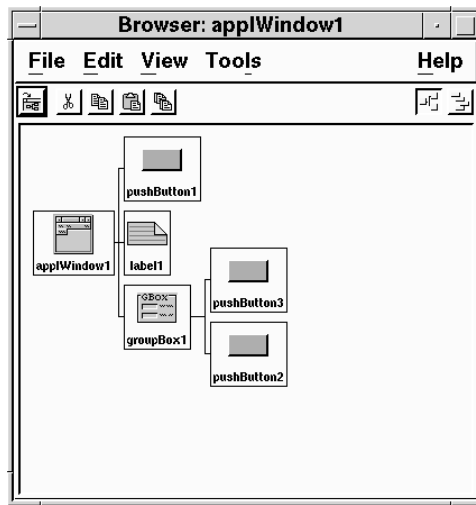


Figure 8-3 The Browser with the Re-parented Push Button

Viewing Instances of Objects

The Browser is particularly useful for viewing and working with instances of objects, since instances are not visible on your interface at design time.

In Step #9 of the tutorial in Chapter 1, you created an instance of a `MsgBoxDialog` that pops up when the Help About menu item is selected. In order to verify that the instance was added to the Application Window, you need simply select your Application Window by clicking on it with the Select mouse button and choose `Tools⇒Browser` from the Selected Objects popup menu.

The Browser will then appear loaded with the Application Window. By scrolling in the Browser window, you will locate the instance, as shown in Figure 8-4.

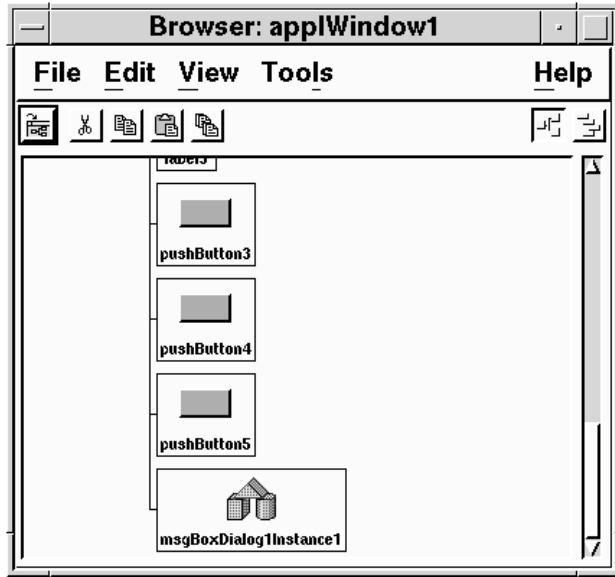




Figure 8-4 Browser Showing Instance of MsgBoxDialog

Changing View

You can change the view of your interface in the outline area of the Browser by choosing from among the View menu options. The View menu allows you to display objects in the hierarchy by name, by icon, or by name and icon, and provides the option to show each object's class.

Alternative views of the hierarchy are available by selecting the Tree icon  or the Outline icon  from the icon bar or by selecting the desired option from the View menu. Figure 8-5 shows an outline view of the Browser loaded with the same interface.

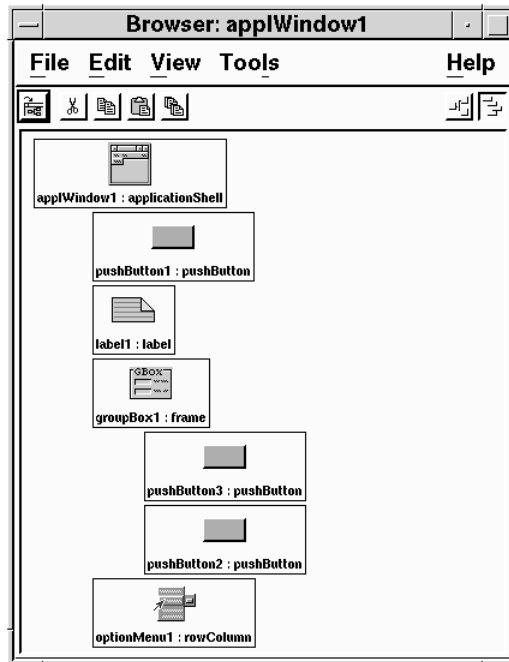


Figure 8-5 The Browser in Outline View

Closing the Browser

When you have finished browsing the object hierarchy, there are a number of ways to close the Browser. You can close the Browser:

1. By selecting the File⇒Close command from the menu bar
OR
2. By double-clicking its Control-menu box
OR
3. By pressing Alt+F4
OR

By pressing Alt+F, then C.

Managing Your Projects

Overview

This chapter describes a project and its files, how to save and manage files, the makefile and how to edit it with the Program Layout Editor.

What Is a Project?

Sometimes a GUI requires only one interface. More often, a complete GUI requires several different interfaces. UIM/X saves each interface as a separate file. You can save and open each interface file by itself. Or you can let UIM/X group all these files together and save your entire GUI as a *project*.

Working with projects provides several advantages:

- Saving all your interfaces in one step
- Opening all your interfaces in one step
- Generating code for all your interfaces in one step
- Automatically generating your application's main program and makefile

In UIM/X, a project file groups together all the files you need for your GUI. A project file lists all the required UIM/X files, including interface files, code files, and a makefile. In most situations, the project file is the only file you need to deal with directly. UIM/X can automatically create, name, save, and open all the other related files as required.

File-Naming Conventions in UIM/X

UIM/X uses the conventions listed in Table 9-1 for naming files.

Table 9-1 File Naming Conventions

Extension	File Type
.c	C code file
.cc	C++ code file
.h	include/header file
.i	interface file
.mk	makefile
.prj	project file

A Typical Project and Its Files

To see how these file-naming conventions apply, consider a typical project with two interfaces, `applWindow1` and `applWindow2`. Saving this project under the name `test.prj` in the *home*/project directory creates the following files in that directory:

- `test.prj` (a project file)
- `applWindow1.i` (the first interface file)
- `applWindow2.i` (the second interface file)

Generating the project's code and compiling it in the same directory creates the following additional files:

- `test` (the executable program)
- `test.c` (the main program)
- `test.mk` (the makefile)
- `applWindow1.h` (the header file for the first interface)
- `applWindow2.h` (the header file for the second interface)
- `applWindow1.cc` (the source file for the first interface)
- `applWindow2.cc` (the source file for the second interface)

Note: If you specified C language mode when starting UIM/X, your code files will have the file extension “.c” instead of “.cc”. By default UIM/X starts in C++ mode.

At the same time, a backup copy of each interface file is stored in the `uxback project_name` subdirectory for safe keeping.

Controlling Where Your Project Files Reside

By default, UIM/X saves all your project files in the same directory. You can override this default and save your project files in any directory you want. For example, suppose your project consists of the following files:

- `test.prj` (the project file)
- `applWindow1.i` (the first interface file)
- `applWindow2.i` (the second interface file)

Saving Your Files

Suppose you want to save the project file in *user/project*, but save the interface files in *user/project/interfaces*. To do so, first select and save each interface in *user/project/interfaces* using the File⇒Save Interfaces As command. Then save the project in *user/project* using the File⇒Save Project As command. The end result will be:

- in *user/project* :
 - test.prj
- in *user/project/interfaces* :
 - applWindow1.i
 - applWindow2.i

The project file refers to the interface files with the relative paths *interfaces/applWindow1.i* and *interfaces/applWindow2.i*. This makes it easier to move or copy the whole project directory tree to another location in the file system and keep the project file references valid.

Generating Code

Next, suppose you want to generate the code for the interfaces in *user/project/interfaces*, but write the main program and makefile in *user/project*. To do so, first select and generate the code for each interface in *user/project/interfaces* using the File⇒Generate Interfaces Code As command. Then generate the project's code in *user/project*, using the File⇒Generate Project Code As command. The end result will be:

- in *user/project* :
 - test.cc
 - test.mk
- in *user/project/interfaces* :
 - applWindow1.cc
 - applWindow2.cc
 - applWindow1.h
 - applWindow2.h

Both .i files will also be backed up in the *user/project/uxbackproject_name* directory.

Using the File Selection Dialog Box

The File Selection dialog box appears when you open or save a project or an interface file in UIM/X. The File Selection dialog box includes five main areas, as shown in Figure 9-1.

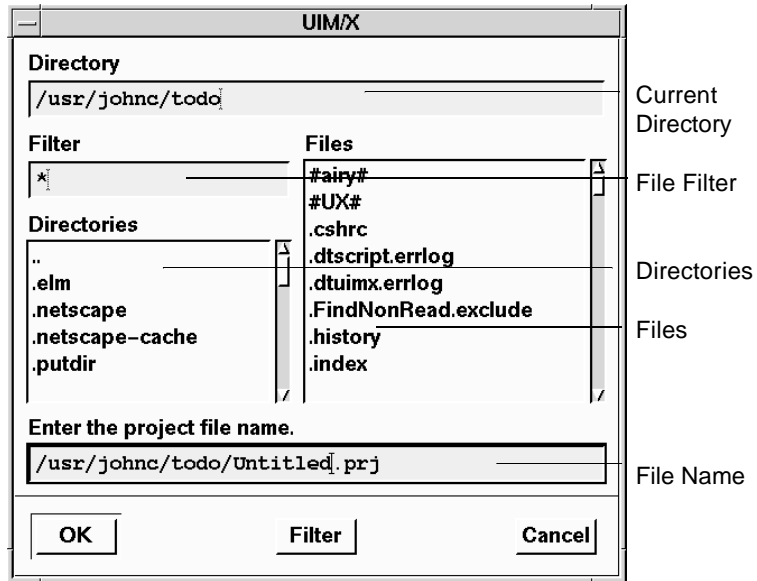


Figure 9-1 Main Areas of the File Selection Dialog Box

- *Current Directory.* Displays and allows selection of the current directory where interface files or projects will be opened or saved.
- *File Filter:* Enables you to type in a pattern used to filter the Directories and Files Lists. After typing your filter, click on the Filter button. The default filter (*) displays all the files in the current directory, while *.prj lists all the projects, *.i lists all the interfaces, and so on.
- *Directories List:* Displays all the directories under the current directory. To display a directory, click on its name, then click on the Filter button.
- *Files List:* Displays all the files in the current directory that match the file filter. Click on a file name to copy its name to the project file name field.
- *File Name Field:* Displays the name of the file to open or save. You can type here, or click a name from the Files List to copy it into this field.

Creating a Project

You can save your current project from the File menu in the Project Window. Saving a project for the first time saves each interface in the Project Window in an individual interface (.i) file and creates the project (.prj) file. Resaving a project saves any changes to any of the interfaces.

To Create a Project

1. Choose the File⇒Save Project As command from the Project Window.
The File Selection dialog box appears.
2. Click in the Directories list to move to your preferred directory, if required.
3. Double-click in the File Name field, and type in your project name, ending in .prj.
4. Click OK.

The project is created as you specified.


Resaving a Project

After you create a project, you can resave it at any time. Resaving a project saves any changes to any of your interfaces.

Always keep your files up to date by saving your work often. Since your project keeps track of all the files generated by the program, always resave your project after you generate code for any interface.

The program does *not* automatically save your current project when you exit. If you try to exit without saving your changes, you see a dialog box showing what you modified since your last save. The dialog box gives you a chance to cancel and save your changes before exiting.

To Resave a Project


1. Choose the File⇒Save Project command or select the Save Project icon  from the Project Window icon bar.

The project and all its interface files are resaved with your latest changes.

Saving Your Interfaces

From the File menu in the Project Window, you can save one or more interfaces, each in its own .i file. You can save an interface with its default name, or any other file name you prefer. All interface names should end in .i.

To Save Interfaces with Default Names

1. Select one or more interfaces to save in the Project Window.
2. Choose the File⇒Save Interfaces command or select the Save Interfaces icon  from the Project Window icon bar.

Each interface is saved with a default name ending in `.i`.

To Save Interfaces with Your Own Names

1. Select one or more interfaces to save in the Project Window.
2. Choose the File⇒Save Interfaces As command from the Project Window.

The File Selection dialog box appears for the first interface.

3. Click in the Directories list to move to your preferred directory, if required.
4. Double-click in the File Name field, and type your file name ending in `.i`.
5. Click OK.

The File Selection dialog box reappears for the next interface.

6. Repeat steps 3 through 5 until all the interfaces are saved.

Each interface is saved with the name you specified.

Always Save For The Future

You can open a project or an interface file with UIM/X, but you can *not* open a program. If you might need to make any changes to your work in the future, you *must* save your work as either an interface or a project file. Some programmers try to skip this step, by building interfaces and generating code, without saving their work as an interface or project file. This is a mistake, because UIM/X cannot read in a program file; it requires either a `.i` file or a `.prj` file. Interface files contain the source code for generating an interface, in a text format you can view with any text editor. Do *not* edit an interface file; it is much simpler to make changes with UIM/X.

Opening a Project or Interface File


From the File Selection dialog box, you can open any existing project or interface file. Opening a project file opens *all* the interfaces that make up that project. Opening an interface opens *only* that one interface.

You can open only one project at once. If you try to open a second project, you see a dialog box that gives you a chance to cancel and save your first project before loading the second. You can open multiple interface files one by one.

MANAGING YOUR PROJECTS


Opening a Project or Interface File

To Open a Project

1. Choose the File⇒Open command or select the Open icon  from the Project Window icon bar.
2. Use the File Selection dialog box to find the proper project (* .prj) file.
3. Click OK.

The specified project and all its interfaces are loaded into the program.

To Open an Interface File

1. Choose the File⇒Open command or select the Open icon  from the Project Window icon bar.
2. Use the File Selection dialog box to find the proper interface (* .i) file.
3. Click OK.

The specified interface file is loaded into the program.

To Open a File From the Command Line

1. Type the following command to load a project:

```
uimxn -file project.prj &
```

OR

2. Type the following command to load an interface file:


```
uimxn -file interface.i &
```

The specified file is loaded into the program.

Modifying an Interface Without Opening Its Project File

You do *not* need to open a whole project file to modify a single one of its interfaces. You can open the interface (* .i) file by itself, revise it as you want, and then save it under the same name. The next time you open that project, the modified interface is automatically opened.

To Modify an Interface Without Opening its Project File

1. Choose the File⇒Open command or select the Open icon  from the Project Window icon bar.
2. In the File Selection dialog box, double-click on the interface (.i) file you want to modify.
3. Click OK.
4. Modify the interface however you want.

5. Choose the File⇒Save Interfaces command from the Project Window.
Your modified version of the interface file is saved.



Reusing Interface Files

You can reuse an interface by sharing it among various projects. This means if you build an interface that you can use in another project, you need not rebuild it, you can simply reuse it. You can reuse an existing interface in several ways: by adding an existing interface to a project, or by renaming it. You can also reuse several interfaces by renaming or duplicating an entire project.

Adding an Interface to a Project

If you save an interface file, and then later you create a project that includes that interface, the interface file remains in its original location, and is referenced by the project file. If you create a new interface for a project, the new interface file is saved in the same directory as the project file by default.

To Add an Existing Interface to a Project

1. Open the project in UIM/X novice mode.
2. Choose the File⇒Open command or its corresponding icon  from the Project Window.
3. In the File Selection dialog box, open the interface (.i) file to add.
4. Choose the File⇒Save Project command or its corresponding icon  from the Project Window.

The project is re-saved under the same name, with the interface file added to it.

Renaming an Interface

You can rename an existing interface by saving it with a new name or directory.

To Rename an Interface File

1. Select the interface to rename.
2. Choose the File⇒Save Interface As command from the Project Window.
3. In the File Selection dialog box, click in the Directories list to move to your preferred directory, if required.
4. Double-click in the File Name field, and type your file name ending in .i.
5. Click OK.

The interface is saved with the new file name you specified.

Renaming a Project

You can rename an existing project by saving it under a new name. Saving a project file under a new name renames only the project file; all your other files keep their existing names.

To Rename an Existing Project

1. With the project open, choose the File⇒Save Project As command from the Project Window.
2. Double-click in the File Name field, and type in a new project name, ending in `.prj`.
3. Click OK.

The existing project is saved under a new name.

Duplicating a Project

To duplicate an entire project, you can simply save the existing project in a new directory. This duplicates all the project files and subdirectories under the new project directory. Files are saved with their absolute pathname, so that files shared by more than one project are not duplicated.

To Duplicate a Project

1. Open the project to duplicate in UIM/X novice mode.
2. Choose the File⇒Save Project As command from the Project Window.
3. Click on the directory to receive your duplicate of the project.
4. Click OK.

A duplicate copy of your project is saved in the new directory.

Using the Program Layout Editor

You have one makefile for each project, named *project.mk*. You can use the Program Layout Editor provided with UIM/X to read and revise the makefile for your project. You may never need to do so, though, since UIM/X can automatically create and revise the makefile for you.

What Is a makefile?

As you know, applications often consist of many source files, each of which may need to pass through preprocessors, assemblers, compilers, and other utilities before being combined with the rest of your program. Forgetting to recompile a module you have changed, or that depends on something else you have changed, can lead to frustrating bugs. On the other hand, recompiling everything just to be safe takes a lot of extra time.

The UNIX `make` command solves this problem. `make` looks at the specified relationship between your files, and their date stamps, and then does what is required, but nothing more, to efficiently generate an up-to-date version of your program code.

With a makefile, you can specify the relationship between files and the processes that generate files, file-to-file dependencies, files that were modified and their impact on other files, and the exact sequence of operations needed to generate a new version of the program. UIM/X novice mode can automatically generate a project's makefile for you. Then you can use the Program Layout Editor to view and modify the makefile if required.

The makefile Template

UIM/X provides a generic makefile template, which you can find in `/usr/uimx2.9/config/make.tem`. This is the template for the makefile created when you generate code with UIM/X. You can view and edit this makefile with the Program Layout Editor if you need to create a custom makefile for your project.

If you do edit the makefile, UIM/X saves the revised makefile as part of your project. From then on, each time you open that project, UIM/X opens your revised makefile, and no longer uses `config/make.tem`.

Using the Makefile Template

The makefile template contains variables such as `$PJ_EXECUTABLE`. These variables are placeholders for information that is filled in by UIM/X when you generate code. For example, the `$PJ_EXECUTABLE` variable is replaced by the executable name you enter in the Generate Code Options dialog box.

Do *not* edit any of these variables or your makefile will not generate properly. You may edit the rest of your makefile as required. For example, you are free to change any of the following:

- libraries (LIBS) and library paths (LIBPATH)
- name of the compiler (ANSI_CC, KR_CC, or CPLUS_CC)
- compile flags (CFLAGS)
- link flags (LD_FLAGS)

To Open the Program Layout Editor

1. Choose the Tools⇒Program Layout Editor command from the Project Window.

The Program Layout Editor appears, loaded with the makefile for your current project.

About the Program Layout Editor

The Program Layout Editor consists of two main areas, as shown in Figure 9-2.

Startup Interface

Makefile Viewing Area

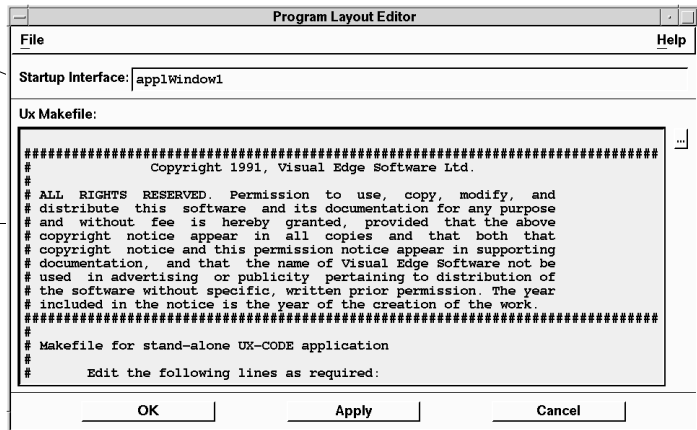


Figure 9-2 Main Areas of the Program Layout Editor

- *Startup Interface Area:* Shows the name of the current startup interface, which defaults to the first interface you created in your project.
- *Makefile Viewing Area:* Shows the makefile for your current project.

To Revise Your makefile

1. Choose the Tools⇒Program Layout Editor command from the Project Window.
2. Click on the button.

The Text Editor opens, loaded with the makefile for your project.

3. Scroll to the lines you want to revise, and make all the changes you want. Repeat until you have made all your changes.
4. Click OK in the Text Editor.
5. Click Apply in the Program Layout Editor.

Your revised makefile is saved as you specified.

To Change Your Startup Interface

1. Select the interface you want as the Startup Interface in the Project Window.
2. Choose the Tools⇒Program Layout Editor command from the Project Window.

The Program Layout Editor appears, with the name of the startup interface.

3. Choose File⇒Load Startup Interface from the Program Layout Editor.

Notice that the startup interface name changes to your selected interface.

4. Click Apply in the Program Layout Editor.

Your selected interface is now the startup interface in your project.

Closing the Program Layout Editor

After you finish editing the makefile, there are a number of ways to close the Program Layout Editor. You can close the Program layout Editor by:

1. Choosing its File⇒Close command
OR
2. Double-clicking its Window menu button
OR
3. Pressing Alt+F, then C
OR
4. Pressing Alt+F4.

Generating Code

Overview

This chapter describes how to generate the code for your project, and a few tips on troubleshooting your generated application. For more information on using code in UIM/X, see Chapter 12, “Programming in UIM/X.”

You can generate code for your whole project, or for selected interfaces only. You can generate code in two ways:

- By clicking the Run button in the Project Window
- By using one of the four available File⇒Generate Code commands
- By using one of the two available File⇒Generate Code icons

When generating code, you can decide to name and run your makefile and executable in the Generate Code Options dialog box, as shown in Figure 10-1.

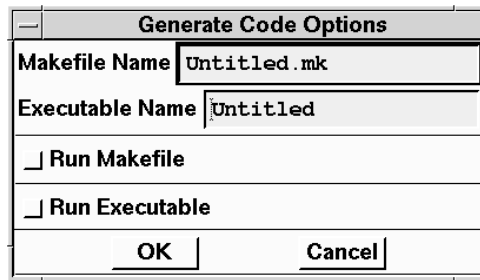


Figure 10-1 Generate Code Options Dialog Box

Generating Code for a Project

When you generate code for a project, the program generates code for every interface in your project. The code is generated according to your entries in the Generate Code Options dialog box.

Note: The type of code generated is dependent on the language mode selected when starting UIM/X. By default UIM/X starts in C++ mode.

To Generate Code for a Project

1. Open the project in UIM/X novice mode.
2. Choose File⇒Generate Project Code As from the Project Window.
The Generate Code Options dialog box appears.
3. Double-click in the Makefile Name text field, and type in a different name if desired.
By default, the makefile is named after the project, with the extension `.mk`, as in `project.mk`.
4. Double-click in the Executable Name text field, and type in a different name if desired.
By default, the executable name is the project name.
5. Click on the Run Makefile button to automatically run the makefile to produce the executable.
6. Click on the Run Executable button to automatically run the executable after it is generated.
7. Click OK.

The code for your project is generated as you specified. The makefile and executable are run as you specified.

Generating Code for Selected Interfaces Only

You can also generate code for specific interfaces only. You can choose any interface in the Project Window, and generate code for that interface only. In this case, no main program file or makefile is generated.

To Generate Code for Selected Interfaces


1. In the Project Window, select one or more interfaces to generate code for.
2. Choose File⇒Generate Interface Code As in the Project Window.
The File Selection dialog box appears.
3. Use the File Selection dialog box to provide the file name and directory for the generated files for your selected interface.

4. Click OK in the File Selection dialog box.
If you selected more than one interface, the File Selection dialog box appears for the next interface.
5. Repeat steps 3 and 4 for each interface.
After you enter the names for all your interface files, the code for your selected interfaces is generated as you specified.

Saving Time Regenerating Code


If you already generated the code for your project or your selected interfaces, you can save a little time when you re-generate by suppressing the dialog box that would otherwise appear.

Using the Generate Project Code Command

If you already generated the code for your project, and are now regenerating the code, you can save time by choosing the File⇒Generate Project Code command or selecting its corresponding icon  from the Project Window icon bar.


This repeats the same functions without presenting the Generate Code Options dialog box yet again. All the files are automatically written under their current names and all your current options are respected.

Using Run Mode

You can regenerate all project code and run your application in one easy step by switching to run mode. Select run mode either by choosing the Mode⇒Run command from the Project Window menu bar or selecting its corresponding icon  from the icon bar.

This repeats the same functions as the Generate Project Code command and automatically runs your application upon successful generation of the code. All the files are automatically written under their current names and all your current options are respected.

Using the Generate Interface Code Command

If you already generated the code for your selected interfaces, and are now regenerating the code, you can save time by choosing the File⇒Generate Interface Code command or selecting its corresponding icon  from the Project Window icon bar.

This repeats the same functions without presenting the File Selection dialog box yet again. All the files are automatically written under their current names and all your current options are respected.

Troubleshooting Your Compiled Application

Here are some common problems you may encounter after compiling an interface built with UIM/X, and what you can do about them.

Unexpected Property Values

During development, your interface inherits property values from the Application Defaults. Your compiled application may reference a different resource file with different settings. That resource file is referenced by the `XtAppInitialize()` function in the main program file.

Unexpected Location of Window or Dialog Box

A window or dialog box may not appear in the location you expected. This is likely because during design, you moved that object with its title bar. The object moves for the moment, but the next time you test or generate code, it returns to its previous location. This is done on purpose so you can quickly move a window or dialog box to clear some space on your screen if it gets crowded.

To move the location of a window or dialog box in your compiled application, always move it with the Adjust button and the compass pointer.

Unexpected Size of Window or Dialog Box

A window or dialog box may not appear at the size you expected. This is likely because during design, you resized that object with its resize handles. The object is resized for the moment, but the next time you test or generate code, it returns to its previous size. As above, this is done on purpose so you can quickly shrink a window or dialog box to clear some space on your screen if it gets crowded.

To resize a window or dialog box in your compiled application, always use the Adjust button and the resize pointers, or adjust it more precisely by pixel using the `height` and `width` properties in the Property Editor.

Unexpected Startup Window

You may start to design a single interface and eventually realize that you need more than one window or dialog box. By default, the first interface you created is the “Startup Interface” shown when you run your compiled application.

You can change the startup interface as described in Chapter , “This chapter describes a project and its files, how to save and manage files, the makefile and how to edit it with the Program Layout Editor..” In the Project Window, select the interface you would like to see when you run your application, choose Tools⇒Program Layout Editor from the Project Window, then choose File⇒Load Startup Interface. Regenerate the project code for your project. Now when you run your application, the interface you selected will appear as the startup interface.

An Introduction to Instances

11

Overview

In general, an application consists of one main interface and many interfaces that pop up as a result of application and user activity. File Selection Boxes and Message Boxes are examples of commonly used dialogs that appear temporarily and behave independently of the main interface. UIM/X simplifies the popping up of independent interfaces using *instances*.

Instances allow you to create independent interfaces, then reuse them in other interfaces. You create the original interface exactly as you would any other, changing properties and adding behavior as desired. To reuse the interface in another, you simply drag and draw an instance of it.

Instances inherit all the properties and behavior of the original. Instances also inherit methods from the `UxInterface` class, which are available for use in the Connection Editor or in callback code.

How to Create an Interface Instance

To create an instance of an interface, the original must first be created and then selected. You can then create an instance of the interface by selecting the Instance option from the Selected Objects popup menu. This is described in further detail in the following example of how to pop up a dialog box.

How to Pop Up a Dialog Box

When designing GUIs, you will often want to pop up a dialog box from your main interface. This section provides a complete example of how to pop up a File Selection Dialog box by pressing a push button. You can apply the same process to popping up a Secondary Window or a Message Dialog.

There are five steps to this process:

1. Creating the Application Window
2. Creating the Push Button
3. Creating the File Selection Dialog box
4. Creating an instance of the File Selection Dialog box
5. Using the Connection Editor to define behavior for the Push Button

Step #1: Creating the Application Window

This involves creating the main interface window.

1. Click on the Application Window icon in the Palette.
2. Press and hold down the Select button where you want the top left corner of the Application Window to be located on your screen.
3. While holding down the Select button, drag and draw the Application Window on your screen.
4. Release the mouse button.

The program creates an Application Window called `applWindow1`, as shown in Figure 11-1.

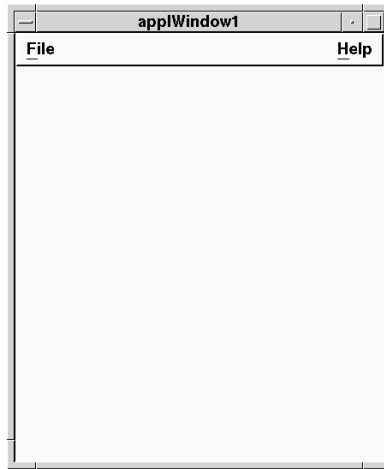


Figure 11-1 Your New Application Window

Step #2: Creating the Push Button

This step involves creating and naming the Push Button.

1. Point to the `pushButton` icon in the Palette.
2. Press and hold down the Adjust mouse button.
The familiar compass pointer and object outline appear.
3. Drag the outline onto your interface.
4. Release the mouse button.
A `pushButton` appears where you specified in your interface.
5. Open the Property Editor and change the `labelString` property to an appropriate label, perhaps "Save".
6. Apply the change and close the Property Editor.

Step #3: Creating the File Selection Dialog Box

This involves creating a dialog box and giving it an appropriate title.

1. Point to the `fileSBoxDialog` icon in the Palette.
2. Press and hold down the Adjust mouse button.
The familiar compass pointer and object outline appear.

3. Drag and draw the outline to the desired size on the desktop.
4. Release the mouse button.

A File Selection dialog box, similar to that depicted in Figure 11-2 appears where you specified on the desktop.

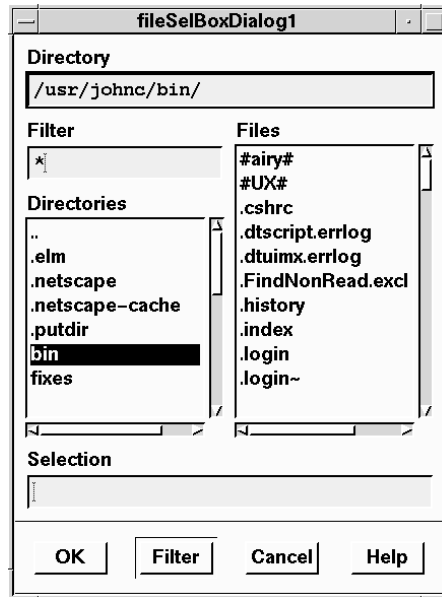


Figure 11-2 File Selection Dialog Box

5. Open the Property Editor and change the `dialogTitle` to an appropriate name, perhaps "Save File".
6. Apply your change and close the Property Editor.

Step #4: Creating an Instance of the File Selection Dialog Box

This involves creating an instance of the File Selection dialog box which will pop up when the push button is selected.

1. Click on the File Selection dialog box, `fileSelBoxDialog1` to select it. An interface must be selected to create an instance of it.
2. Point to the main interface, `applWindow1`.
3. Press and hold the Menu mouse button on the Main Window interface to display the Selected Objects popup menu.

The Selected Objects popup menu appears as shown in Figure 11-3. Notice that the bottom selection is “Instance of fileSelBoxDialog1”.

Selected Objects (applWindow1)	
Tools	—
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Duplicate	
Align	—
Arrange	—
Delete	
Recreate	
Instance of fileSelBoxDialog1	

Figure 11-3 Selected Objects Popup Menu

- Choose “Instance of fileSelBoxDialog1”.
The mouse pointer changes into the corner shape.
- Drag and draw the instance of fileSelBoxDialog1 onto the application window.
Notice that the instance is not visible on the interface. This is expected behavior for parented top-level widgets. You will verify that the instance has been added using the Browser.
- Open the Browser by choosing Selected Objects⇒Tools⇒Browser.
The Browser appears, as shown in Figure 11-4, indicating that your instance has been added.

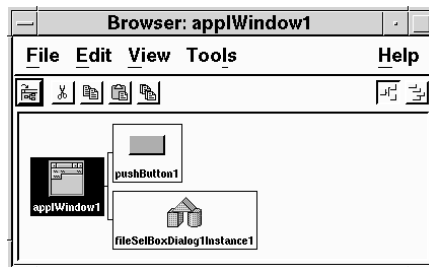


Figure 11-4 The Browser with Instance of File Selection Dialog Box Added

- Leave the Browser open, as it will be required for the next step.

Step #5: Defining Behavior for the Push Button

In this step you will define the connection between the push button and the instance of the File Selection dialog box using the Connection Editor.

1. Click on the push button on your interface to select it.
2. Open the Connection Editor by selecting Selected Objects⇒Tools⇒Connection Editor.

The Connection Editor appears with `pushButton1` loaded as the source object. Notice that all available callbacks for `pushButton1` are listed in the Callback area of the Connection Editor.

3. With the Browser still open, position the mouse pointer on `fileSelBoxDialog1Instance1`, click on it with the Adjust mouse button, then drag and drop it from the Browser to the Target area of the Connection Editor.

The instance's default methods are listed in the Method area of the Connection Editor, as shown in Figure 11-5.

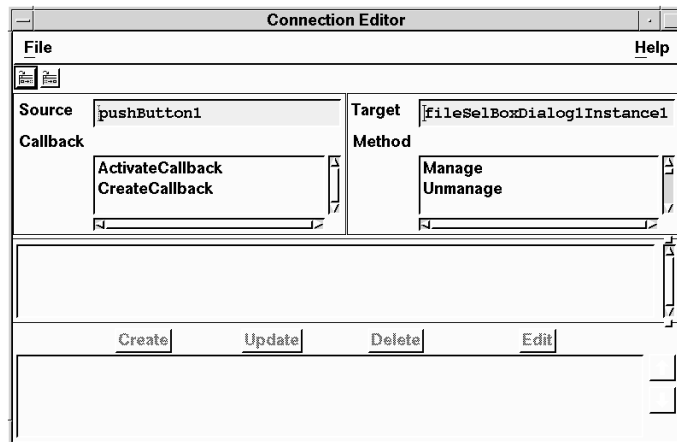


Figure 11-5 The Connection Editor with Source and Target Loaded

4. Click on `ActivateCallback` in the list of callbacks and on `Manage` in the list of methods.

Any parameters or return values available appear in the parameters area.

5. Complete the connection by clicking on `Create`.

The new connection appears in the Connection Editor, as shown in Figure 11-6.

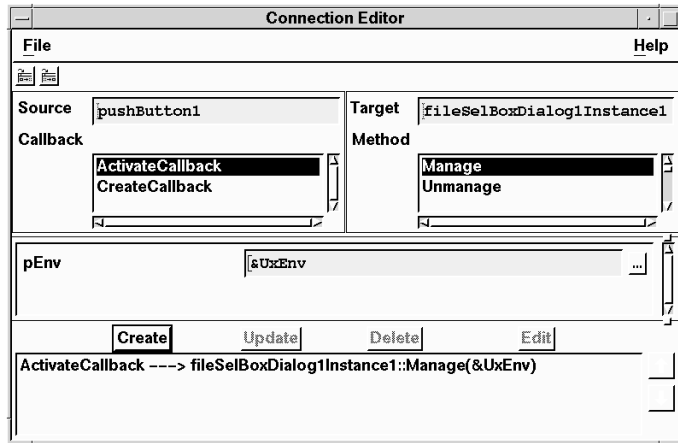


Figure 11-6 The Connection Editor with the Completed Connection

6. Close the Connection Editor
7. Save your work.

Testing Your Interface

When you click on the Save push button during Test, your Save dialog box should appear. When you click OK or Cancel in the dialog box, it should disappear. When you generate code and run your interface, your project should behave the same way.

Programming in UIM/X

12

Overview

This chapter describes the basics of programming in UIM/X, and includes some useful tips and techniques. Non-programmers can safely skip this chapter.

The Ux Convenience Library

As you develop a GUI with UIM/X, you often need to add code to give your objects behavior. You can write this code as regular Xt/Motif code, following its many rigorous rules. But it's faster and easier to use the library of functions provided with UIM/X as the Ux Convenience Library.

The Ux Convenience Library is an easy-to-use set of functions to help specify behavior and manage interfaces generated by UIM/X. This library covers all the most common functions you will likely need, including complex tasks such as converting resources, allocating color map entries, managing children of dialog shells, and handling special cases of geometry. As you develop and test your GUI, the Ux Convenience Library does extensive error checking, notification, and recovery.

Using the Ux Convenience Library means you *do not* have to learn the more complex OSF/Motif function library and programming style. In other words, you can begin specifying simple behavior within minutes, and move on to complex behavior within days.

Setting and Retrieving Property Values

For every property of each Motif widget, the Ux Convenience Library provides a `UxPut` function to set the property value, and a `UxGet` function to retrieve the property value. You use these functions by placing `UxPut` or `UxGet` in front of the property's name. For example, the two functions for the `Width` property are `UxPutWidth()` and `UxGetWidth()`.

The `UxGet` function takes one argument, the name of the object's *swidget*. UIM/X maintains a small C structure for every object you create. A *swidget* is a pointer to this structure. UIM/X uses *swidgets* to identify the objects in an interface. The `UxGet` function returns the current value of the property. For example, the following code determines the `y` value of `canvas1`:

```
int value;  
value = UxGetY(canvas1);
```

The `UxPut` function takes two arguments: the swidget name for the object, and the new value for the property. For example, the following line sets the `Height` of `canvas1` to 120:

```
UxPutHeight (canvas1, 120);
```

Each property has a specific data type, such as `int`, `float`, or `char*`.

Note: The compound objects on the UIM/X novice mode Palette contain certain properties that are not recognized Motif properties. `UxPut` and `UxGet` functions are not available for these non-standard properties. Therefore, to write code using `UxPut` or `UxGet` functions for any compound object, you should use UIM/X. When you load a Novice Mode interface file into Standard Mode UIM/X, you are asked whether to break up its compound objects into normal Motif widgets. In this case, answer yes.

Creating and Displaying Interfaces

When you run your compiled application, only the Startup Interface (as defined in the Program Layout Editor) is displayed. If you add a dialog box or a Secondary Window to your project, these are not displayed unless you explicitly create and display them or use instances, as discussed in Chapter , “In general, an application consists of one main interface and many interfaces that pop up as a result of application and user activity. File Selection Boxes and Message Boxes are examples of commonly used dialogs that appear temporarily and behave independently of the main interface. UIM/X simplifies the popping up of independent interfaces using instances.”. When UIM/X generates code for an interface, it provides an `Interface` function that you can use to create and display that interface. By convention it looks like this:

```
create_InterfaceName ( parent )
```

After you create an interface by calling `create_InterfaceName (parent)`, you can use any of the following three functions from the `Ux Convenience Library` to pop up, pop down, or destroy it. In all three functions, *interface* is the value returned by the object’s `Interface` Function.

UxPopupInterface(swidget *interface*, grabtype *grabtype*)

This function makes an interface visible on the screen once it has been created with its Interface Function. Sometimes an interface pops up other interfaces, creating a cascade of interfaces. The `grabtype` argument allows you to specify how the end user interacts within a cascade of interfaces. The *grabtype* can be:

- `no_grab`, which allows the user to interact with any window on the screen
- `nonexclusive_grab`, which allows the user to interact with any object in the interface cascade, but not with object outside the cascading interfaces
- `exclusive_grab`, which limits the end user to the interface that you are popping up (for example, a popup menu)

UxPopdownInterface(swidget *interface*)

This function takes a visible interface and removes it from the screen. It also removes any grab that the interface has.

UxDestroyInterface(swidget *interface*)

This function destroys an interface. If you try to destroy a static interface during Test, you see a message that the function was called, but the interface is not destroyed.

Using the Declaration Editor

You use the Declaration Editor to include application header files, define macros, and declare variables used in callback code. For an example, refer to [<Xref>Step #8: Adding Declarations and Callbacks](#), in Chapter 1, “Building Your First Project”.

To Open the Declaration Editor

1. Select any interface in the Project Window.
2. Choose Tools⇒Declaration Editor.

The Declaration Editor opens, as shown in Figure 12-1.

To Use the Declaration Editor

1. Open the Declaration Editor for your selected interface.
2. Click on the button to open the Text Editor, and type in the code you need.
3. Click OK in the Text Editor to save your code.
4. Click OK or Apply in the Declaration Editor.

To Close the Declaration Editor

After you finish using the Declaration Editor, you can close it by clicking OK, or by double-clicking on its Window menu button.

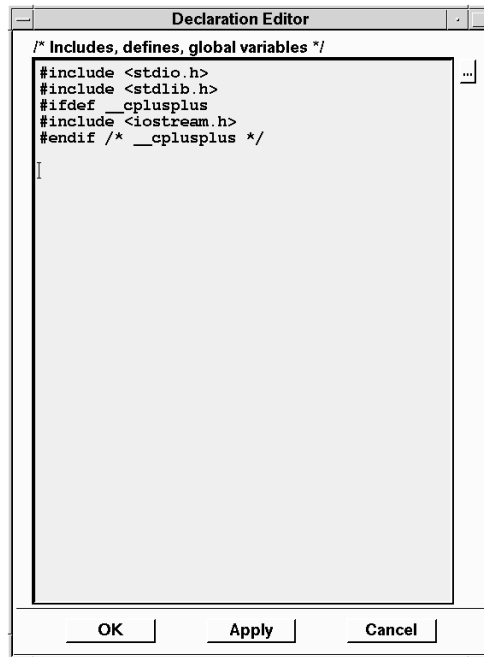


Figure 12-1 Declaration Editor in UIM/X novice mode

Beyond the Basics

13

Overview

UIM/X novice mode introduces you to the basics of UIM/X:

- Building a GUI from a palette of objects
- Setting properties
- Adding behavior by writing callback code
- Testing the interface
- Generating code

Everything you learned in the novice mode of UIM/X applies in UIM/X. You can load and edit novice mode interfaces, and you build interfaces in exactly the same way. The difference is that UIM/X provides additional features and tools.

You can build useful, working applications in UIM/X novice mode. But to build production-quality applications, you must move beyond the basics. The purpose of this chapter is to introduce you to some of the advanced features of UIM/X that make it possible to build professional products.

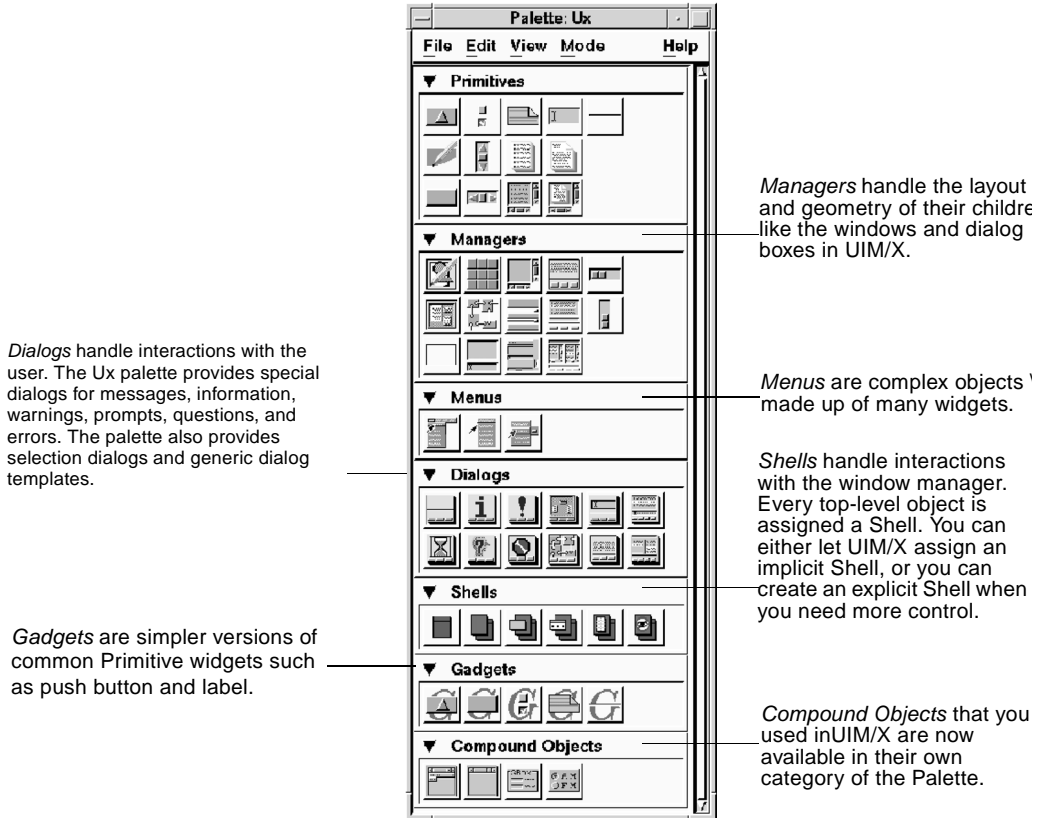


Figure 13-1 Default UIM/X Palette

Building GUIs in UIM/X

UIM/X provides a comprehensive palette of GUI objects. The Ux Palette, shown in Figure 13-1, includes the entire Motif widget set, as well as the UIM/X compound objects.

UIM/X supports all of Motif: every widget, every shell, every gadget, every convenience dialog, and every property. This gives you the flexibility to build interfaces that satisfy demanding requirements.

For example, in UIM/X you can use shell widgets to build dialogs (for errors, cautions, printer control) that force the user to confirm before continuing with a selected action. Using shells gives you control over how your dialogs interact with the window manager (for example, whether or not a dialog can be resized or iconified).

This increased flexibility results in interfaces that are easier to use, but which are more complex. Most interfaces consist of a hierarchy of widgets. Typically, each interface starts with a shell widget. The shell widget has a single child, usually a manager widget such as a form or main window. This manager widget has its own children, some of which are primitive widgets, and some of which are other manager widgets with their own children.

UIM/X also provides a specialized Main Window Editor for building main windows. The Main Window Editor, like the Menu Editor, simplifies the process of building complex object hierarchies.

Setting Properties

In UIM/X, the Property Editor gives you access to every property and every callback of every widget.

This makes UIM/X a great learning tool for newcomers to Motif. You can build sample interfaces using different combinations of widgets. You can try different property values to see what happens. And to find out what triggers a callback, you can put a `printf()` in the callback and switch to Test mode.

But the Property Editor does much more than help you learn Motif. It provides some unique features to speed up GUI development and help build production-quality applications.

Polishing Interface Details

To make it easier to polish the details of an interface by setting properties, the Property Editor provides some special features:

- Widget properties are divided into categories such as Core, Specific, Behavior, and Declaration. This makes it easier to find and edit properties.
- Multiple widgets can be loaded and edited. This allows you to give an entire group of widgets the same value for a set of properties.
- Widgets can be loaded into an open Property Editor, either by drag-and-drop or by using the Automatic Load feature, which loads widgets as soon as they are selected. This allows you to quickly edit the same set of properties on one widget after another.
- The set of available properties can be reduced by hiding properties. For example, if you want to edit only those properties that have been set before, you can hide all properties that still have their default values. Or, if you load multiple widgets, you can hide all properties that have the same value for each widget.

Giving an Interface a Dynamic Initial State

In many applications, the initial state of an interface is *dynamic*. For example, a menu might display the name of the currently loaded file. Or the color of an icon might change to the current drawing color.

If all you could do in UIM/X was enter constant property values, you would not be able to define a dynamic initial state for an interface. But the Property Editor allows you to enter C or C++ code for any property value. So, for example, if you want a color property to depend on the current state of the application, you just enter a C or C++ expression that queries the application and returns the right color.

Note: UIM/X has a built-in Interpreter, so anywhere you can enter a constant value, you can enter a valid C or C++ expression. When you apply the Property Editor, the Interpreter evaluates any C or C++ expressions entered as initial values.

When you enter code for a property value, you can use a global variable, a function, or an argument to the Interface Function (the function that creates the interface). The only requirement is that the expression evaluate to the type expected by the widget for the property.

You may be wondering where these global variables, functions, and Interface Function arguments come from. After all, in UIM/X novice mode, the only place you enter code is in the Callback Editor, and you don't have access to the Interface Function. To build interfaces with dynamic initial states, you need more programming flexibility than UIM/X novice mode offers. In UIM/X, the Declaration Editor gives you this flexibility.

Programming in UIM/X

In UIM/X, you program by writing callback code, which (with the exception of the `CreateCallback`) is triggered by keyboard or mouse input *after* the interface is created and displayed. But what if you need to do something just before the widgets in an interface are created? What if you want to add an argument to the Interface Function so you can use it to set a property value? What if you need to initialize your application, or customize the event loop?

To handle these and other issues, UIM/X provides a complete programming environment. All code associated with a project can be entered in UIM/X editors—you don't ever have to edit the generated code.

Editing Interface Code in the Declaration Editor

In UIM/X, the Declaration Editor gives you full control over the code generated for an interface. Figure 13-2 highlights the purpose of each area of the Declaration Editor.

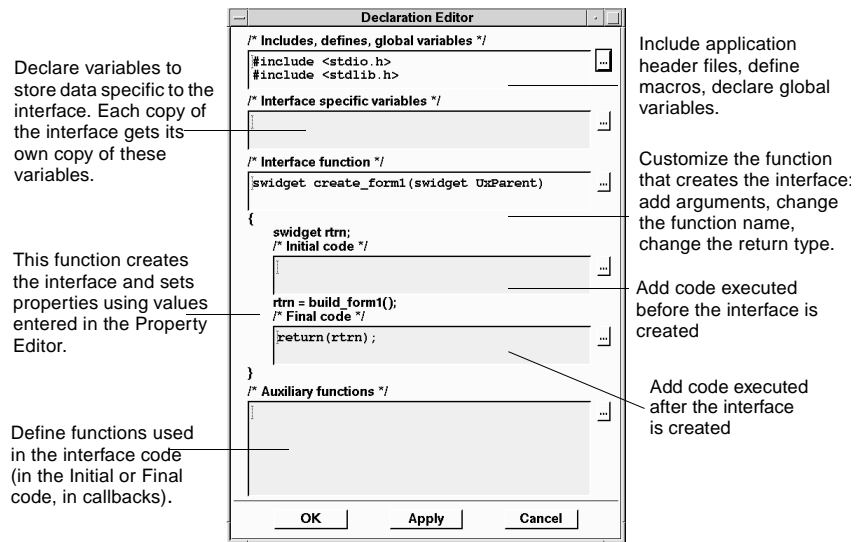


Figure 13-2 UIM/X Declaration Editor

Connecting the Interface to Your Application

The Declaration Editor lets you connect the interface to your application code. For example, suppose you had a database application and you wanted to use values stored in the database as property values.

First, you include the required database header files in the Includes, Defines, Global Variables area of the Declaration Editor. This makes the database functions available throughout the interface code, including callbacks.

Second, in the Interface-Specific Variables area, declare variables to hold the values retrieved from the database. Then use these variables as property values in the Property Editor.

Finally, in the Initial Code area, query the database for the required values and store them in the interface-specific variables. Because values entered in the Property Editor are set when the widgets are created, you have to use the Initial Code area, not the Final Code area. Initial Code is executed before widget creation, while Final Code is executed after widget creation.

Adding Arguments to the Interface Function

One way to create an interface with a dynamic initial state is by using Interface Function arguments as property values. For example, suppose you wanted the color of an interface to be a parameter of the interface (that is, a value that can change each time you create the interface). To do this, you add a color argument to the Interface Function, and then use the argument to set the color property values.

Adding Initial and Final Code

The Initial and Final Code areas let you add code to the body of the Interface Function. By default, the Interface Function calls a `build_interface()` function and returns the value returned by the build function.

The build function is generated by UIM/X. This function creates the widgets in the interface, sets their properties using values entered in the Property Editor, and adds the callbacks defined in the Callback Editor. It then returns the top-level widget in the resulting interface.

The Initial and Final Code areas let you add code you want executed before and after the build function is called. For example, in Chapter 12, “Programming in UIM/X,” you learned how to create and pop up an interface, by first calling the Interface Function and then calling `UxPopupInterface()`.

If you always wanted the Interface Function to create and pop up the interface, you could add a call to `UxPopupInterface()` in the Final Code area:

```
UxPopupInterface(rtrn);  
return(rtrn);
```

Controlling Application Window Behavior

Writing callback code is only one way of adding behavior to an interface. Many applications provide features that cannot be implemented solely through callbacks. For example, a drawing program might provide a window where you can draw lines, rectangles, and circles.

To implement this type of application, you have to be able to specify how the application responds to keyboard and mouse input. UIM/X provides a set of editors for interactively choosing events and linking them to C code actions.

The mechanism that ties events to actions is called a translation table. You use translation tables to connect events in a window to application code (actions). This connection between events and actions is called *application window* behavior.

Editing the Main Program

In UIM/X, the Program Layout Editor gives you access to the main program. This allows you to add application initialization code. For example, you may need to open a database before creating any interfaces. You can also use the Program Layout Editor to customize the main event loop.

Generating Code

In UIM/X novice mode, you generate C or C++ code that uses the Ux Convenience Library. In UIM/X, you have many more choices, as shown in Figure 13-3.

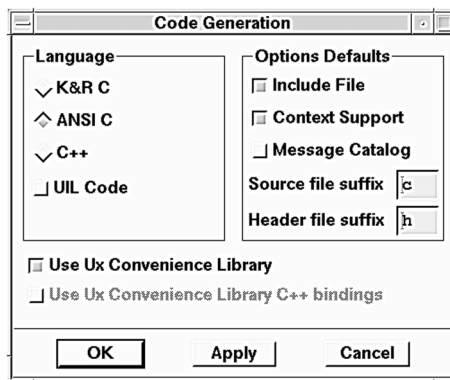


Figure 13-3 Code Generation Options

In UIM/X, you can generate ANSI C, K & R C, C++, with or without UIL code. You can also set a number of default options:

- Whether or not to generate include files.
- Whether or not the generated code should use a context structure to manage multiple copies of an interface.
- Whether or not to generate a complete catalog of all the messages in your interface. If you select this option, strings entered in the Property Editor are stored in a message catalog.
- You can also choose whether you want your generated code to call the Ux Convenience Library functions. By default, this check box is selected. You can deselect this toggle button if you want to call Xt and Xm functions only.

Generating Resource Files

Most X/Motif applications allow for end-user customizing by providing resources that control application appearance and behavior. UIM/X can generate resource files for the interfaces you build.

In the Property Editor, you can specify whether properties are Public, Private, or Default:

- Public properties go into a resource file when you generate code. This allows the end user to customize the application by setting the resource values. UIM/X generates a resource file for each interface.
- Private properties are set in the generated code, and cannot be set by the end-user.
- Default properties use the default values provided by the widget.

What this means is that you can give the end user as much or as little control over application appearance and behavior as you want.

Object-Oriented Programming

UIM/X supports object-oriented programming in either C or C++. It provides an interactive way to build class hierarchies with inheritance, encapsulation, and polymorphism.

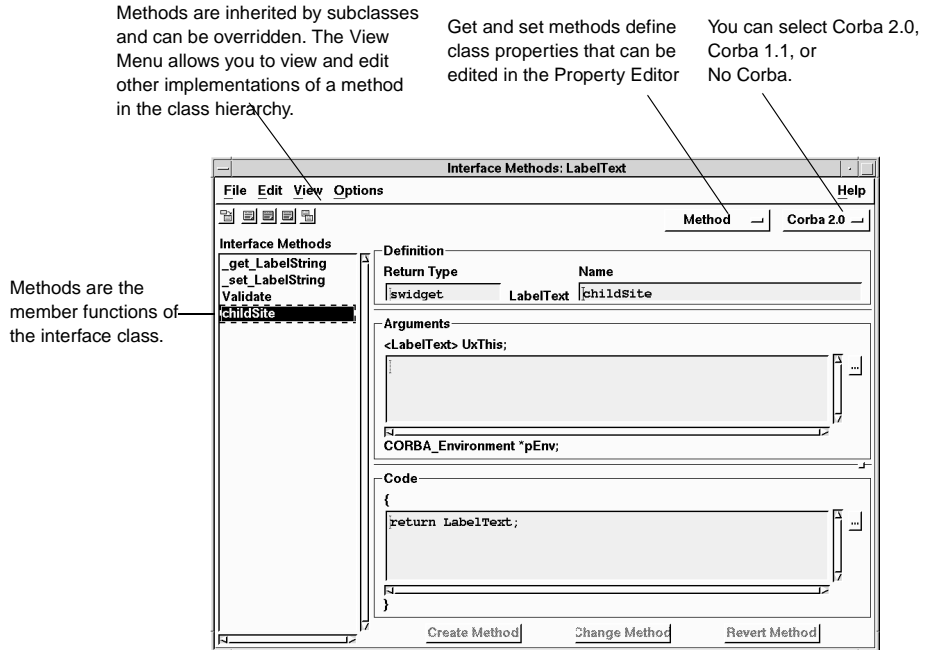


Figure 13-4 Method Editor

You can instantly create a reusable component (a class in C++) simply by dragging an object from the palette or an interface onto the desktop. To define class interfaces, you use a specialized Methods Editor (shown in Figure 13-4). Methods defined in UIM/X work in C or C++, so you can realize the benefits of object-oriented programming whether you program in C or C++.

Programming with C++

UIM/X also provides a set of features that make it easier to program with C++. You can edit C++ class declarations, generate methods as virtual or non-virtual member functions, and generate methods as public, protected or private protected members.

Building Palettes

In UIM/X, you can build your own palettes. This means you can build and distribute palettes that contain all your most useful components (classes). And companies or workgroups can implement an in-house GUI style guide by distributing custom palettes to all their developers.

Non-Visual Objects

Some objects, by their very nature, cannot be represented visually. Files, servers, database objects and data structures, for example, have no graphical interface.

With the Non-Visual Shell, UIM/X provides the structure for developing non-visual object classes. By integrating them into UIM/X you can extend the UIM/X palette to include your new non-visual object classes. By placing an instance of your non-visual object into the palette, you can ensure that its methods are available for use, while remaining uneditable.

For a detailed explanation of working with non-visual objects, refer to Chapter 6 of the *UIM/X Tutorial Guide*.

Effective GUI Design

A large, bold, black letter 'A' is centered within a gray rectangular area in the top right corner of the page.

Overview

This chapter discusses the main principles of effective GUI design, and some common pitfalls to avoid.

The Principles in a Nutshell

An effective GUI has one key purpose: it enables the end user to interact with an application simply and naturally. The best interface is one that is scarcely noticed, one that creates the least distraction between what the end user wants to do and the process of doing it. Always remember that end users just want to get on with their jobs.

The most important design principles for building good GUIs can be summed up as follows:

- Know your end users.
- Give your end users control.
- Use real-world metaphors.
- Be forgiving.
- Use color tastefully.
- Build GUIs that can travel the world.
- Present tasks in logical sequence.
- Avoid common pitfalls.

Each one of these key principles is discussed in more detail in this appendix.

Know Your End Users

Just as comedians or politicians have to know their audience to give a winning performance, developers have to know their end users to create a first-class interface.

What Do Your Users Want?

Most of all, users want to get their work done. Users are busy; they have deadlines, distractions, and pressures. Most users want to work by pointing and clicking, not by typing in command line switches. Whenever you have to make a tough design decision, always ask yourself, “What do my users want?”

Your Users May Be Less Technically Inclined

Remember that your users may be less technically inclined than you are. Do not assume that they know everything you know, or that they will approach your application the same way you do. The inner workings of your program do not concern them as much as getting their work done.

Finding Out More About Your Users

Larger companies with a lot of resources take a formal approach to this question, sponsoring controlled usability studies and focus groups. Smaller firms with fewer resources tend to use informal ways of getting closer to their users: support lines, surveys, or even the occasional lunch meeting. Whatever you can do to get to know your users better, do it. In most cases, they will be delighted to know that someone cares about what they think.

Give Your End Users Control

End users want to be in control of the tools they use to do their jobs. You can give your end users more control in several ways.

For one, be flexible. Provide several ways to perform the same task: through a menu, a keyboard accelerator, or by dragging and dropping an object. Allow end users to configure your application to suit themselves and the way they work. Group commonly used functions in the most accessible locations, and hide those that are used less often.

Use Real-World Metaphors

Most good user interfaces in some way reflect the real world. For instance, push buttons can be pushed in, and sliders can be moved. Many of the objects supported by UIM/X are based on metaphors from the real world. Let your end users manipulate these items as if they were physical objects.

But remember that your chosen metaphors must relate to the experience of your end users. Every profession has its own set of traditions and symbols. For instance, an icon of a brush could mean painting to a graphic designer, but to a crime-lab chemist, a brush could mean gathering samples. A grid could suggest a spreadsheet to an accountant, but to a microelectronics engineer a grid might suggest a circuit board layout. Once again, knowing your end users can help you select the most appropriate metaphors for them.

Be Forgiving

Remember that end users are human beings, prone to making mistakes. A good GUI is forgiving, and encourages end users to explore without doing any damage to their work.

Anticipate Likely Errors

Try to anticipate the likely errors your end users may make, and provide some way to undo them. Always give your end users a chance to think twice before deleting any of their work. One way to do this is to provide a Cancel push button on every dialog box, and an Undo feature in the appropriate menu.

Encourage End Users to Explore

Many end users want to explore a GUI. They want to pull down every menu and look at every option without necessarily wanting to perform every available action. Give end users a chance to cancel any serious operation before it affects their work. Encouraging end users to explore without risk enables them to learn faster and be more effective with your application.

Use Color Tastefully

Color is one of the most powerful ways to communicate, and all modern GUIs should take advantage of it. In general, your goal should be to design a pleasing color scheme that remains pleasing over time.

Create a Pleasing Color Scheme

To create a pleasing color scheme, do not use too many bright or dark shades in the same window. Do not use colors that clash harshly for foregrounds and backgrounds. If you are designing for a business user, make your color scheme dignified, not dazzling.

There are also technical reasons for not using too many colors in one interface: your end user's system may not support as many colors as yours does, so that they run out of "color space" before you do, and cannot display all your colors.

Use the Same Colors You Find in Nature

Another rule of thumb is to use the same colors that nature uses, in the forests, in the flowers, in the animal kingdom, and in the oceans. In fact, the natural world is probably where we get our instinctive feeling for what makes a tasteful color scheme.

Remember That Color Is Subjective

Color is highly subjective, so there are no absolutes. In the fashion world, a color may be "in" one year and "out" the next. A color that helps a person from one region relax can make a person from somewhere else feel agitated. Color has emotional, cultural, religious, and political associations you may not be aware of. These associations are especially critical if your application is intended for end users in more than one region of the world.

Do Not Rely Completely on Color

Do not rely completely on colors for emphasis. Remember that approximately 10% of the male population in the western world is color-blind, and that some end users have gray-scale monitors. Use the size, shape, and placement of objects to differentiate between them, as well as their colors. Many developers design first in black and white, and then add colors later.

Build GUIs That Can Travel the World

If you are designing a user interface to be distributed around the world, make sure it's ready for world travel.

Do Not Use Culturally Specific Elements

Be careful to avoid making any culturally specific references that may be confusing or insulting in other cultures. For example, telephones, electrical outlets, and mailboxes look different in different countries. Icons based on these objects may be puzzling in other countries. Some hand gestures considered friendly in one culture are insulting in others. Some common computer terms such as *abort* or *execute* may offend people with different beliefs. Do not risk confusing or upsetting end users elsewhere in the world: avoid using any culturally specific icons, graphics, or terms.

Make Your Messages Easier to Translate

Here are some guidelines for writing messages which will make a translator's job easier:

- Use articles, such as *the* and *a*.
- Use the present tense and imperative verbs, such as *choose* and *check*.
- Use complete sentences as often as possible.
- Avoid *and/or*, since this construction does not exist in all languages.
- Minimize your use of contractions, such as *don't* and *couldn't*.
- Avoid acronyms that are not already in world-wide use.

Allow for Various Information Conventions

Addresses, dates, currencies, measurements, and time are given in various formats in different regions. Allow for various conventions. Avoid using fixed-length fields, labels, or formats that cannot accommodate other conventions. Keep in mind that most other languages require more characters than English to convey the same message. Allow room for text strings in menus, windows, and dialog boxes to expand if they are translated.

Present Tasks in a Logical Sequence

Make sure your push buttons and menus present the tasks to be done in a logical sequence. There are several logical ways to present tasks, such as:

- Chronological order, from first to last.
- Frequency of occurrence, from the most common to the most rare.
- Complexity, from the easiest to the most difficult.
- Familiarity, from the most familiar to the most unexpected.

Whichever method you choose, use it consistently.

Avoid Common Design Pitfalls

Here are some other common design pitfalls to avoid:

- Not reducing the “busy-ness” of screens.
- Not revising your designs based on feedback from end users.
- Not keeping menus short (five to seven items is optimal).
- Not paying attention to fine details, such as spacing.
- Not hiding programming details from end users.

Object Properties

B

Overview

The novice mode of UIM/X supports a certain set of *properties* for each object in the Palette. These properties determine the size, color, label, font, and other characteristics for each object. This appendix defines each object property available in the novice mode of UIM/X.

These supported properties include a subset of those available in Motif, featuring the most frequently-used properties. The following sections describe:

- Property value data types available
- Special strings required for colors, fonts, and icons
- Seven core properties available for each object
- Additional properties available for certain objects

You do not need to memorize all these properties and their options to be productive with the novice mode of UIM/X. As you work with objects, these properties will quickly become more familiar to you.

Property Value Data Types

UIM/X supports a number of property value data types, as listed in Table B-1. You can see and edit each of these data types in the Property Editor.

Table B-1 Property Value Data Types

Data Type	Description
Integer	An integer.
Float	An integer, or a floating-point number.
String	A string constant enclosed in quotation marks ("string").
Boolean	Either <code>true</code> or <code>false</code> , as listed in an option menu. For example, <code>Sensitive</code> can be either <code>true</code> or <code>false</code> .
Enumerated Type	A number of possible string values, as listed in an option menu. For example, the <code>Alignment</code> property can be either <code>alignment_end</code> , <code>alignment_beginning</code> , or <code>alignment_center</code> .

Special String Types

The novice mode of UIM/X supports three special string types for specifying colors, fonts, and icons, as listed in Table B-2. To make entering these special strings easier, three specialized editors are provided. Each editor helps you compose a different type of string using a graphical interface. These specialized editors are the Color Viewer, the Font Viewer, and the Icon Viewer, as described in Chapter 4, "Viewing and Changing Properties".

You can also enter a special string by typing it into the Text Editor; be sure to follow exactly the proper format or you will see an error message.

Table B-2 Special String Types in UIM/X

Special String Type	Description	How To Change This String
Color	Any color name enclosed in quotation marks and recognized by your server, such as "MediumSlateBlue"; or any RGB value with hexadecimal digits for the R, G, and B values, such as: "#6a6a4d4d8f8f". Five object properties require a color name: Background, Foreground, ListBackground, MenuBackground, and TextBackground.	Click on the property name button to open the Color Viewer from the Property Editor, or use the Text Editor to type in a color name.
Font	Any font name enclosed in quotation marks and recognized by your server. Five object properties require a font name: ButtonFontList, FontList, LabelFontList, MenuFontList, and TextFontList.	Click on the property name button to open the Font Viewer from the Property Editor, or use the Text Editor to type in a font name.
Icon	Any icon file name enclosed in quotation marks and recognized by your server. Two object properties require an icon name: IconPixmap and LabelPixmap.	Click on the property name button to open the Icon Viewer from the Property Editor, or use the Text Editor to type in an icon name.

The Seven Core Properties for Each Object

All objects in the novice mode of UIM/X (except for the Radio Box) have the same seven core properties available, as listed in Table B-3. You can change any of these properties either by moving or resizing the object, or by using one of the built-in editors. For details, see Chapter 4, “Viewing and Changing Properties”.

The `X`, `Y`, `Width`, and `Height` properties are set when you first create an object. Moving or resizing the object changes the values of these four properties accordingly. The `Background` and `Foreground` colors of an object default to your current colors. The `Sensitive` property is set to `True` by default.

Table B-3 Core Properties for Each Object in UIM/X Novice Mode

Property	Meaning	How To Change This Property
X	The horizontal offset of the object's left side, in pixels from the left side of the window (for a Primitive object), or from the left of the display (for a Window).	Move the object left or right, or resize the object, or use the X text field in the Property Editor.
Y	The vertical offset of the object's top, in pixels from the top edge of the window (for a Primitive object), or from the top of the display (for a Window).	Move the object up or down, or resize the object, or use the Y text field in the Property Editor.
Width	The width of an object in pixels.	Resize the object, or use the Width text field in the Property Editor.
Height	The height of an object in pixels.	Resize the object, or use the Height text field in the Property Editor.

Table B-3 Core Properties for Each Object in UIM/X Novice Mode

Property	Meaning	How To Change This Property
Background	The background color of an object.	Click the Background button to open the Color Viewer from the Property Editor.
Foreground	The foreground color of an object.	Click the Foreground button to open the Color Viewer from the Property Editor.
Sensitive	Whether an object responds to input from an end user (given as true or false).	Use the option menu for Sensitive in the Property Editor.

Additional Properties Available for Certain Objects

Beyond the seven core properties common to all objects, each object has a certain set of additional properties available. You can view and change any of these properties with the Property Editor. Each property takes a certain default value when an object is first created. You can see the default value for any property by loading a newly-created object into the Property Editor. The

purpose of each property and how to change its value are listed in Table B-4.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 1 of 10)

Property	Explanation	How to Change
Activate Callback	Code that provides behavior to an object when it is activated. This callback is supported by the Push Button, Default Push Button, and Text Field.	Enter or revise the code for <code>ActivateCallback</code> in the Property Editor.
Alignment	An option menu with three choices for aligning an object's label, <code>alignment_center</code> (centered), <code>alignment_end</code> (right-justified), or <code>alignment_beginning</code> (left-justified). This value is pre-set to center. This property is supported by the Label, Push Button, Default Push Button, Group Box, and Radio Button/Check Button.	Use the option menu for <code>Alignment</code> in the Property Editor.
Button FontList	The font used to display all Labels for all Push Buttons in a dialog box. This property is supported only by the Message and File Selection dialog boxes.	Click the <code>ButtonFontList</code> button to open the Font Viewer from the Property Editor.
Cancel Callback	Code that provides behavior to an object when the Cancel Push Button is pushed. This callback is supported only by the Message and File Selection dialog boxes.	Enter or revise the code for <code>CancelCallback</code> in the Property Editor.
Create Callback	Code that provides behavior to an object when it is created. <code>CreateCallback</code> is intended for any special processing you need to do before an object is realized. This callback is supported by every object.	Enter or revise the code for <code>CreateCallback</code> in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 2 of 10)

Property	Explanation	How to Change
Default Button Shadow Thickness	The thickness of the shadow around the Default Push Button. This value is pre-set to 1, matching the Motif standard.	There is no need to change this value.
Delete Response	<p>An option menu with three choices for how to respond if an end user closes a window:</p> <p><code>destroy</code> (destroy a Secondary Window, or destroy and exit an Application Window), <code>unmap</code> (hide the window), and <code>do_nothing</code>.</p> <p>Any Close/Delete events are blocked by the program in Design and Test modes.</p> <p>This value defaults to <code>Destroy</code>. This property is supported only by the Application and Secondary Windows.</p>	Use the <code>DeleteResponse</code> option menu in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 3 of 10)

Property	Explanation	How to Change
Dialog Style	<p>An option menu with five choices to set how a dialog box interacts with applications:</p> <p><code>dialog_modeless</code> (dialog box need <i>not</i> be cleared before any other interactions),</p> <p><code>dialog_primary_application_modal</code> (dialog box <i>must</i> be cleared before <i>some</i> other interactions in its parent window),</p> <p><code>dialog_full_application_modal</code> (dialog box <i>must</i> be cleared before <i>some</i> other interactions in its application),</p> <p><code>dialog_system_modal</code> (dialog box <i>must</i> be cleared before <i>any</i> other interactions in <i>any</i> applications), and</p> <p><code>dialog_work_area</code>.</p> <p>This value defaults to <code>dialog_modeless</code>. This property is supported only by the Message and File Selection dialog boxes.</p>	Use the DialogStyle option menu in the Property Editor.
Dialog Title	<p>The title to display in the title bar of a dialog box. This value is pre-set to null. This property is supported only by the Message and File Selection dialog boxes.</p>	Use the DialogTitle text field in the Property Editor.
Directory	<p>The default directory to show in the File Selection dialog box. This defaults to the current directory in the window where you started UIM/X.</p>	Use the Directory text field in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 4 of 10)

Property	Explanation	How to Change
Drag Callback	Code that provides behavior to a Scale when an end user drags the slider from its current position. This callback runs almost continuously whenever an end user moves a slider. This callback is supported only by the Horizontal and Vertical Scales.	Enter or revise the code for Drag Callback in the Property Editor.
EditMode	An option menu with two choices to indicate how a ScrolledText object can be changed by the user: <code>single_line_edit</code> or <code>multi_line_edit</code> .	Use the EditMode option menu in the Property Editor.
FontList	The font used to display text in an object. This property is supported by every object that can accept text, namely Label, Text Field, Push Button, Default Push Button, Horizontal and Vertical Scales, Radio Button/Check Button, Scrolled List, and Scrolled Text.	Click the Font List button to open the Font Viewer from the Property Editor.
HelpCallback	Code that provides the behavior for the Help button in a File Selection Dialog or Message Dialog.	Use the HelpCallback text field or click the ... button to open the Text Editor from the Property Editor.
IconName	The name to use for a window icon when it is minimized. IconName defaults to Name unless you provide another value. This property is supported only by the Application and Secondary Windows.	Use the IconName text field in the Property Editor.
IconPixmap	The pixmap to use for a window icon when it is minimized. This property is supported only by the Application and Secondary Windows.	Click the Icon Pixmap button to open the Icon Viewer from the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 5 of 10)

Property	Explanation	How to Change
Initial State	An option menu with five choices to set the initial state of a window: NormalState, DontCareState, ZoomState, IconicState, and InactiveState. This value defaults to NormalState. This property is supported only by the Application and Secondary Windows.	Use the Initial State option menu in the Property Editor.
Items	A list of all the items to display in a Scrolled List, separated by commas as follows: item1, item2, item3. This property is supported only by the ScrolledList.	Click the ... button to open the Text Editor from the Property Editor.
Label FontList	The font used to display all Labels in a dialog box. This property is supported only by the Message and File Selection dialog boxes.	Click the Label FontList button to open the Font Viewer from the Property Editor.
Label Pixmap	The pixmap to display in an object. For the pixmap to show, the object's LabelType property must be set to pixmap. This property is supported by the Label, Push Button, Default Push Button, Group Box, and Radio Button/Check Button.	Click the Label Pixmap button to open the Icon Viewer from the Property Editor.
Label String	The text to display in an object. For the text to show, the object's LabelType property must be set to string. This property is supported by the same objects as LabelPixmap and LabelType.	Use the Label String text field in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 6 of 10)

Property	Explanation	How to Change
Label Type	An option list with two choices to set which type of label to show in an object: string or pixmap. This property is supported by the same objects as LabelPixmap and LabelString.	Use the LabelType option menu in the Property Editor.
List Background	The color of the text area and scroll bars in a Scrolled List. Background sets the area between the scroll bars and the text area. Foreground sets the color the text is displayed in. This property is supported only by the Scrolled List.	Click the List Background button to open the Color Viewer from the Property Editor.
Maximum	The maximum value for a Horizontal or Vertical Scale.	Use the Maximum text field in the Property Editor.
Menu Background	The background color for all menus in an Application Window. Since no other windows can have menus in the novice mode of UIM/X, this property is supported only by the Application Window.	Click the Menu Background button to open the Color Viewer from the Property Editor.
Menu FontList	The font used for all menus in an Application Window. Since no other windows can have menus in the novice mode of UIM/X, this property is supported only by the Application Window.	Click the Menu FontList button to open the Font Viewer from the Property Editor.
Message String	The text message to display in a Message dialog box. This property is supported only by the Message dialog box.	Use the Message String text field in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 7 of 10)

Property	Explanation	How to Change
Message Dialog Type	An option menu with seven choices to set the type of Message dialog box: <code>dialog_error</code> (shows a large error sign), <code>dialog_information</code> (shows large i), <code>dialog_message</code> (with Push Buttons), <code>dialog_question</code> (shows head with Q), <code>dialog_template</code> (no Push Buttons), <code>dialog_warning</code> (shows large !), or <code>dialog_working</code> (shows hourglass). This value is pre-set to <code>dialog_message</code> . This property is supported only by the Message dialog box.	Use the Message DialogType option menu in the Property Editor.
Minimum	The minimum value for a Horizontal or Vertical Scale.	Use the Minimum text field in the Property Editor.
Name	The name assigned to an object when you create it. This name is assigned in the format <code>ObjectNameX</code> where <code>X</code> = the number of existing objects of the same type+1. This property is supported by every object. Note that Name is <i>not</i> the same as Title, which is the name shown in the title bar of a window.	Use the Name text field in the Property Editor.
OKCallback	Code that provides behavior to an object when the OK Push Button is pushed. This callback is supported only by the Message and File Selection dialog boxes.	Enter or revise the code for OKCallback in the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 8 of 10)

Property	Explanation	How to Change
Pattern	<p>The final character in the filter shown in the Filter text field in a File Selection dialog box. This defaults to the wildcard “*”.</p> <p>This property is supported only by the File Selection dialog box.</p>	<p>Use the <code>Pattern</code> text field in the Property Editor.</p>
Processing Direction	<p>An option menu with four choices to indicate where the Maximum falls in a Horizontal or Vertical Scale:</p> <p><code>max_on_top</code> (Vertical Scale only), <code>max_on_bottom</code> (Vertical Scale only), <code>max_on_left</code> (Horizontal Scale only), <code>max_on_right</code> (Horizontal Scale only).</p>	<p>Use the <code>Processing Direction</code> option menu in the Property Editor.</p>
Separator Type	<p>An option menu with seven choices to set the appearance of a Horizontal or Vertical Separator:</p> <p><code>single_line</code>, <code>double_line</code>, <code>single_dashed_line</code>, <code>double_dashed_line</code>, <code>no_line</code>, <code>shadow_etched_in</code> (line carved in), or <code>shadow_etched_out</code> (line raised out).</p> <p>This defaults to <code>shadow_etched_in</code>.</p> <p>This property is supported only by the Horizontal and Vertical Separators.</p>	<p>Use the <code>Separator Type</code> option menu in the Property Editor.</p>
Set	<p>An option menu with two choices to set the state of a Radio Button or Check Button: <code>true</code> (selected), or <code>false</code> (deselected).</p>	<p>Use the <code>Set</code> option menu in the Property Editor.</p>

Table B-4 Additional Properties Available for Certain Objects
(Sheet 9 of 10)

Property	Explanation	How to Change
ShowValue	An option menu with two choices to set whether to show the current value for a Horizontal or Vertical Scale: <code>true</code> (show value), or <code>false</code> (hide value).	Use the ShowValue option menu in the Property Editor.
Single Selection Callback	Code that provides behavior to a Scrolled List when an end user selects one of its lines. This property is supported only by the Scrolled List.	Enter or revise the code for Single Selection Callback in the Property Editor.
Text	The actual text contained in a Text Field or Scrolled Text object. This text is displayed in the Foreground color and the FontList font. This property is supported only by the Text Field and Scrolled Text.	Use the Text text field, or click the ... button to open the Text Editor from the Property Editor.
Text Background	The color of the text area and scroll bars in a Scrolled Text object. Background sets the area between the scroll bars and the text area. Foreground sets the color the text is shown in. This property is supported only by the Scrolled Text object.	Use the Color Viewer, opened from the Property Editor.
Text FontList	The font used to display all text in Text Fields in a dialog box. This property is supported only by the Message and File Selection dialog boxes.	Click the Text FontList button to open the Font Viewer from the Property Editor.
TextString	The character string representing the path name displayed in a File Selection Dialog's text field.	Use the TextString text field, or click the ... button to open the Text Editor from the Property Editor.

Table B-4 Additional Properties Available for Certain Objects
(Sheet 10 of 10)

Property	Explanation	How to Change
Title	The text to display in the title bar for a window. Note that <code>Title</code> is <i>not</i> the same as <code>Name</code> , which is a name for each object assigned by the program. <code>Title</code> defaults to <code>Name</code> unless you provide another value. This property is supported only by the <code>Application</code> and <code>Secondary Windows</code> .	Use the <code>Title</code> text field in the Property Editor.
Title String	A title to display with a <code>Horizontal</code> or <code>Vertical Scale</code> . This title is displayed in the <code>Foreground</code> color and the <code>FontList</code> font. This property is supported only by a <code>Horizontal</code> or <code>Vertical Scale</code> .	Use the Property Editor.
Value	The current value of the slider's position in a <code>Horizontal</code> or <code>Vertical Scale</code> . This value must equal the <code>Minimum</code> or <code>Maximum</code> , or fall somewhere between these two values. This property is supported only by the <code>Horizontal</code> and <code>Vertical Scale</code> .	Use the <code>Value</code> text field in the Property Editor, or move the slider in a <code>Horizontal</code> or <code>Vertical Scale</code> .
Value Changed Callback	Code that provides behavior to an object when the object's current value changes. This property is supported by the <code>Radio Button/Check Button</code> , <code>Text Field</code> , <code>Scrolled Text</code> , and <code>Horizontal</code> and <code>Vertical Scales</code> .	Enter or revise the code for <code>Single Selection Callback</code> in the Property Editor.

B

Additional Properties Available for Certain Objects

Index

Symbols

"X" to show incorrect value in Property Editor 31, 112
.Xdefaults file 16
... button (in Property Editor) 46, 48, 107, 109

A

accelerator
 creating 137
 definition 40, 137
 purpose 40, 130
Accelerator property (in Menu Editor) 135, 136, 137
Accelerator Text property (in Menu Editor) 135, 136, 137
ActivateCallback 46, 48
ActivateCallback property 43, 111, 226
Add Object area (in Property Editor) 107
adding
 callbacks to menu 140
 menu items 140
 object to interface 14
Adjust Height command (in Palette) 73, 74
Adjust mouse button xii
Align command 19, 99
Align menu 19, 99
aligning
 objects 19
aligning objects 98, 100
Alignment property 222, 226
Alt key xi, 136
Alt+F, C (to exit from UIM/X) 60, 76
application defaults xiii, 38, 94

Application Window
 default name 92
 moving 186
 resizing 186
ApplicationWindow
 as compound object 80
 creating 5
 default name 6, 191
 default size 93
 double-clicking to create 93
 drawing 92
 moving 85, 186
 purpose 85
 resizing 11, 85, 186
 using 85
Apply button
 in Color Editor 120
 in Color Viewer 117
 in Font Viewer 124
 in Property Editor 112
applying constraints 158
Arrange command 21, 101
Arrange menu 101
arranging objects 100

B

Background property 29, 35, 113, 222, 224
black O pointer 94
bolt constraint 155
Browser
 icon bar in 162
 in UIM/X 161
 loading an interface 163
 opening 163
ButtonFontList property 38, 122, 226

Index

C

- C++ 211, 212
- Callback Editor 46
- callbacks
 - adding to menu 140
 - definition 43, 80, 110
 - supported by UIM/X 111
- CancelCallback property 43, 111, 226
- Canvas 88
- Cascade Button 130, 132
 - purpose 40
- categories of objects 81
- CDE
 - and Dt UIM/X v
- changing
 - object colors 35
 - object height or width 10, 96
 - object properties 109–110, 224
 - Startup Interface 181
 - your view of Palette 74
- CheckButton 90
- Clear Window command 75
- clientAutoPlace resource 1
- Clipboard (in UIM/X) 94
- closing
 - Color Editor 120
 - Color Map 121
 - Color Viewer 117
 - Declaration Editor 200
 - Font Viewer 124
 - Icon Viewer 127
 - Menu Editor 142
 - Palette 73
 - Property Editor 113
 - UIM/X 60, 76
- closing the Constraint Editor 159
- CMY (in Color Editor) 118, 119
- code
 - generating 183–187
 - generating options in UIM/X 210
 - in callbacks 43
- collapsing categories 74
- color
 - changing 35
 - grabbing from Color Map 121
 - grabbing from screen 116
 - mixing with Color Editor 37
 - naming conventions 114
 - saving in Color Viewer 117
 - selecting with Color Viewer 116
- Color Database (in Color Viewer) 36, 115, 116
- Color Dialog
 - browsing 163
- Color Editor
 - closing 120
 - CMY in 118
 - Color Value in 119
 - HSI in 118, 119
 - mixing colors with 120
 - opening 104, 118
 - Original Color in 119
 - overview 118
 - purpose 104
 - resetting 120
 - RGB values in 118, 119
 - Working Color in 119
- Color Map 104, 121
- Color Map command (in Color Viewer) 121
- Color Value (in Color Editor) 119
- Color Viewer
 - closing 117
 - Color Database in 36, 115, 116
 - menus in 115
 - Name Filter in 115, 116
 - opening 35, 104, 114
 - overview 113
 - purpose 104
 - RGB values in 115
 - Saved Colors in 115, 116
 - Selected Color in 115
 - selecting color with 116
- Common Desktop Environment *See* CDE
- compass pointer 10, 11, 92, 96

- Compose Character key 136
- compound objects
 - definition x, 80
 - limitation of 199
- Connection Editor
 - closing 148, 167
 - loading a source 146
 - loading a target 146
 - opening 70
- connections
 - defining 147
 - modifying 148
- Constraint Editor
 - closing 159
 - icon bar in 154
 - introduction 26
 - opening 70, 152
 - steps in using 152
 - view menu options 158
- constraints
 - applying 26, 158
 - bolt 155
 - defining 154
 - dimension 156
 - removing 158
 - types of 155
- Control key
 - selecting multiple objects with 19
- Control key, selecting multiple objects with 93
- copying an object 94
- CreateCallback property 43, 111, 226
- creating
 - accelerator 137
 - ApplicationWindow 5, 92–93
 - effective menus 131
 - interfaces 199
 - mnemonic 136
 - object with drag and draw 92
 - object with drag and drop 92
 - option menu 135
 - project 174
 - pull-down menu 135
 - cutting an object 94
 - Cyan/Magenta/Yellow *See also* CMY
- D**
- dashed box (marquee) when selecting multiple objects 18
- Declaration Editor 200, 201
 - opening 70
- Declarations command (in Project Window) 70, 200
- default
 - in option menu 132
- DefaultButton 88
- DefaultButtonShadowThickness property 227
- defining connections 147
- defining constraints 154
- Delete command (in Menu Editor) 138, 139, 140
- Delete command (in Project Window) 71
- DeleteResponse property 59, 227
- deleting
 - interface 71
 - menu panes or items 138
 - object 95
- Design phase 67
- Design, Test, Run icon 67
- Design, Test, Run radio box 66
- Details Area (in Menu Editor) 134
- dialog boxes
 - error-trapping in 61
 - moving 186
 - popping up 190–195
 - resizing 186
- DialogStyle property 228
- DialogTitle property 192, 228
- dimension constraint 156
- Directories List
 - in File Selection dialog box 173
 - in Icon Viewer 126
- Directory property 228
- double-clicking
 - on Window menu button to exit 76, 113, 117,

Index

- 120, 124, 127, 142
 - to load object into Property Editor 32, 106
 - to select text in field 32
- drag and draw
 - creating object with 6, 14, 93
- drag and drop
 - creating object with 14, 15, 93
 - to load Property Editor 108
- DragCallback property 43, 111, 229
- Dt UIM/X
 - and CDE v
- Duplicate command (in Menu Editor) 138, 139, 140
- Duplicate command (in Project Window) 17
- duplicating
 - menu panes or items 138
- duplicating an object 95

E

- Edit Color command (in Color Viewer) 36, 118, 120
- Edit menu (in Project Window) 94
- EditMode property 229
- Enter key *See also* Return key
- error message 31
- Exit command (in Project Window) 76
- exiting from UIM/X 76
- expand arrows (in Palette) 74
- expanding categories (in Palette) 74
- Extend char key 136

F

- Family or Short Name List (in Font Viewer) 39, 123
- File Filter
 - in File Selection dialog box 173
 - in Icon Viewer 126
- File Name Field (in File Selection dialog box) 173
- File Selection dialog box 13, 68, 173
- Files List
 - in File Selection dialog box 173
 - in Icon Viewer 126

- FileSBoxDialog 86
- font
 - naming conventions 123
 - selecting 124
- Font Display area (in Font Viewer) 123
- Font List (in Font Viewer) 39, 123
- Font Viewer 104
 - Apply button in 124
 - closing 124
 - Family or Short Name List in 123
 - Font Display area in 123
 - Font List in 123
 - opening 38, 104, 122
 - overview 122
 - purpose 104
 - Selected Font in 123
 - Size and Char Set List in 123
- FontList property 38, 122, 222, 229
- Foreground property 29, 35, 87, 113, 222, 224

G

- Generate Code Options dialog box 183
- Generate Project Code command (in Project Window) 60
- Grab Color command (in Color Viewer) 116
- grab pointer 116
- grab type argument 200
- grabbing
 - color from Color Map 121
 - color from screen 116
- GroupBox 80, 89

H

- handles 8
- Height property 29, 224
- hiding
 - interface 70
 - Palette 73
 - project 71
- Horizontal Separator 135, 136
- HorizScale 14, 89
- HorizSeparator 14, 22, 86, 88

- HSI (in Color Editor) 118, 119
- Hue/Saturation/Intensity *See also* HSI
- I**
- icon
 - selecting 127
 - viewing 126
- Icon Bar
 - bubble help 66
 - in Browser 162
 - in Constraint Editor 154
 - in Project Window 6, 65
- Icon Display (in Icon Viewer) 126
- Icon Viewer
 - closing 127
 - Directories List in 126
 - File Filter in 126
 - Files List in 126
 - Icon Display in 126
 - opening 104, 125
 - overview 125
 - purpose 104
 - Selected Icon in 126
- IconName property 229
- IconPixmap property 125, 229
- incorrect property value error 112
- InitialState property 230
- interface
 - changing Startup 181
 - definition x
 - deleting 71
 - hiding 70
 - showing 70
- Interface function 199
- interfaces
 - creating and displaying 199
- Interfaces Area (in Project Window) 66
- Item command (in Menu Editor) 135, 136, 140
- items
 - naming conventions 132
 - purpose 40, 130
- Items Area (in Menu Editor) 134
- Items property 230
- L**
- Label 14, 86, 87
- Label String property 191
- LabelFontList property 38, 122, 230
- LabelPixmap property 125, 222, 230
- LabelString property 230
- LabelString property (in Menu Editor) 135, 136
- LabelType property 231
- ListBackground property 35, 113, 231
- Load command (in Property Editor) 108
- Load Startup Interface command (in Program Layout Editor) 181
- loading object into Property Editor 108
- M**
- makefile
 - definition 179
 - revising 181
- Makefile Viewing Area (in Program Layout Editor) 181
- marquee (dashed box), when selecting multiple objects 18
- Maximum property 231
- menu
 - adding callbacks to 140
 - creating 135
 - objects 132
 - parts of a 130
 - testing 141
- menu bar
 - rearranging 139
 - revising 40
- Menu Editor
 - closing 142
 - Details Area 134
 - Items Area 134
 - Menu Information Area 134
 - menus in 134
 - opening 69, 104, 133

Index

- overview 129
- Panes Area 134
- purpose 104
- steps in using 133
- two different versions 133
- Menu Information Area (in Menu Editor) 134
- menu items
 - adding 140
 - creating 135
 - deleting 138
 - duplicating 138
 - rearranging 140
- Menu mouse button xii
- menu pane
 - creating 135
- menu panes
 - deleting 138
 - duplicating 138
- MenuBackground property 35, 113, 231
- MenuFontList property 38, 122, 231
- menus
 - in Color Viewer 115
 - in Menu Editor 134
 - in Property Editor 107
 - tips on creating effective 131
- message 31
- Message pop-up menu 75
- MessageDialogType property 232
- Messages area (in Project Window) 66
- MessageString property 231
- Meta key 136
- Minimum property 33, 232
- minimumSweepSize resource 94
- mixing colors with Color Editor 37, 120
- mnemonic
 - creating 136
 - definition 40, 136
 - purpose 40, 130
- Mnemonic property (in Menu Editor) 135, 136, 137
- modifying connections 148
- Motif
 - presumed familiarity with 79
 - widget set 80
- Motif widget
 - definition x
- Motif widget set 204
- mouse
 - Adjust button xii
 - Menu button xii
 - Select button xii
 - usage xii
- mouse buttons, naming conventions for vii
- mouse pointer
 - black O 18, 94
 - compass pointer 11, 92, 96
 - grab pointer 116
 - resize pointer 10, 95
 - upper-left corner shape 6, 92
- mouse, selecting multiple objects with 94
- moving
 - multiple objects 19
- moving an object 15, 96
- MsgBoxDialog 85
- multiple objects
 - aligning 19
 - moving 19, 97
 - resizing 98

N

- Name Filter (in Color Viewer) 115, 116
- Name property 224, 232
- naming conventions
 - colors 114, 222
 - file names 170
 - fonts 122, 222
 - icons 222
 - menu items 132
 - menu options xi
 - menu panes 132
 - mouse buttons vii
 - Return key xi
 - shell prompts xi

O

object

- adding to interface 14
- adding to window 93
- categories 14, 74, 81
- changing properties 109–110, 224
- children and parents 86
- colors, changing 35
- compound, definition 80
- copying 94
- cutting 94
- definition x, 80
- deleting 95
- double-clicking to load into Property Editor 106
- duplicating 95
- height, changing 10, 96
- loading into Property Editor 108
- moving 15, 96
- nine regions of 10, 95
- pasting 95
- properties 109–110, 224
- resizing 15, 96, 97
- viewing properties 109
- width, changing 10, 96

Object Name area (in Property Editor) 107

object-oriented programming 211

objects

- aligning 98, 100
- arranging 100
- in menus 132
- moving multiple 97
- resizing 10
- resizing multiple 98
- selecting multiple 18, 19, 93

OKCallback property 44, 111, 232

online help 61, 66

- bubble help 66

Open command (in Project Window) 176

opening 104

- Color Editor 104, 118

- Color Map 121

- Color Viewer 35, 104, 114

- Connection Editor 70, 104

- Constraint Editor 70

- Declaration Editor 70, 200

- Font Viewer 38, 122

- Icon Viewer 104, 125

- Menu Editor 69, 104, 133

- Property Editor 69, 104, 106

- Text Editor 107

option menu 109, 110

- adding 132

- creating 135

- purpose 130

- setting default 132

- tips on creating effective 131

OptionMenu 89

Original Color (in Color Editor) 37, 119

OSF/Motif Style Guide x, 80

P

Palette

- building 213

- categories in 14, 81

- changing your view 74

- closing 36

- collapsing categories 74

- definition 14

- expand arrows in 74

- expanding categories 74

- hiding 73

- in UIM/X 204

- reducing the size of 73

- showing 73

- using 72–74

Palette command (in Project Window) 73

pane

- naming conventions 132

- purpose 40, 130

Pane command (in Menu Editor) 132, 135

Panes Area (in Menu Editor) 134

pastings a cut or copied object 95

Index

- Pattern property 233
- pointer *See alsomouse* pointer
- pop-up menus
 - Message 75
 - Selected Interfaces 75
 - Selected Objects 17, 76, 94
- Primitives category (in Palette) 86
- ProcessingDirection property 233
- Program Layout command (in Project Window) 69
- Program Layout Editor 69, 178–182
- project
 - creating 174
 - default file name 68
 - definition x, 170
 - generated files 171
 - hiding 71
 - purpose 170
 - resaving 174
 - showing 71
- Project Window
 - Design, Test, Run radio box in 66
 - Icon Bar in 6, 65
 - Interfaces Area in 66
 - Menu Bar in 65
 - Messages area in 66
 - Title Bar in 65
 - using 66–71
- properties
 - Accelerator 135, 136, 137
 - Accelerator Text 135, 136, 137
 - ActivateCallback 43, 111, 226
 - Alignment 222, 226
 - Background 29, 35, 113, 222, 224
 - ButtonFontList 38, 122, 226
 - CancelCallback 43, 111, 226
 - CreateCallback 43, 111, 226
 - DefaultButtonShadow Thickness 227
 - DeleteResponse 59, 227
 - Dialog Title 192
 - DialogStyle 228
 - DialogTitle 228
 - Directory 228
 - DragCallback 43, 111, 229
 - EditMode 229
 - FontList 38, 122, 222, 229
 - Foreground 29, 35, 87, 113, 222, 224
 - Height 29, 224
 - IconName 229
 - IconPixmap 125, 229
 - InitialState 230
 - Items 230
 - Label String 191
 - LabelFontList 38, 122, 230
 - LabelPixmap 125, 222, 230
 - LabelString 230
 - LabelType 231
 - ListBackground 35, 113, 231
 - Maximum 231
 - MenuBackground 35, 113, 231
 - MenuFontList 38, 122, 231
 - MessageDialogType 232
 - MessageString 231
 - Minimum 33, 232
 - Mnemonic 135, 136, 137
 - Name 224, 232
 - OKCallback 44, 111, 232
 - Pattern 233
 - ProcessingDirection 233
 - Sensitive 224
 - SeparatorType 233
 - Set 233
 - ShowValue 234
 - SingleSelectionCallback 44, 111, 234
 - supported for all objects 224
 - Text 234
 - TextBackground 35, 113, 234
 - TextFontList 38, 122, 234
 - Title 235
 - TitleString 235
 - unique to each object 225
 - Value 235
 - ValueChangedCallback 235

- Width 29, 224
- X 29, 224
- Y 29, 224
- Properties Area (in Property Editor) 107
- property data types 199, 222
- Property Editor
 - Add Object area in 107
 - Apply button in 112
 - closing 113
 - in UIM/X 205
 - loading object into 108
 - menus in 107
 - Object Name area in 107
 - opening 69, 104, 106
 - overview 105
 - Properties Area in 107
 - purpose 29, 104
 - resetting 112
 - restricted to one instance only 104
- Property Editor command (in Project Window) 69, 106
- Property Editor command (in Selected Objects pop-up menu) 106
- property values
 - retrieving 198
 - setting 198
- property values, entering incorrect 112
- pull-down menu
 - adding 132
 - creating 40, 135
 - tips on creating effective 131
- PushButton 14, 86, 87

R

- RadioButton 80, 91
- RadioButton 90
- rearranging
 - menu bar 139
 - menu items 140
- Red/Green/Blue *See also* RGB
- removing constraints 158
- resaving project 174

- Reset command (in Property Editor) 112
- resetting
 - Color Editor 120
 - Property Editor 112
- resize grid 10, 95, 97
- resize pointer 10, 95, 97
- resizing
 - an object 15, 96, 97
 - ApplicationWindow 11
 - objects 10
- resources
 - setting xiii
- retrieving property values 198
- Return key xi
- Return key, with DefaultButton 88
- RGB values
 - Color Editor 118, 119
 - Color Viewer 36, 37, 115
- RowColumn object 40, 130, 132
- RowColumn object (in pull-down menu) 132
- Run icon 67

S

- Save Interface As command (in Project Window) 177
- Save Interface command (in Project Window) 175
- Save Project As command (in Project Window) 174
- Save Project command (in Project Window) 12, 68, 174
- Saved Colors (in Color Viewer) 115, 116, 117
- saving
 - color in Color Viewer 117
 - your work in UIM/X 12, 68
- ScrolledList 14, 91
- ScrolledText 91
- SecondaryWindow 85
 - as compound object 80
 - moving 186
 - resizing 186
- Select mouse button xii
- Selected Color (in Color Viewer) 115

Index

- Selected Font (in Font Viewer) 123
- Selected Icon (in Icon Viewer) 126
- Selected Interfaces pop-up menu 75
- Selected Objects pop-up menu 17, 76, 94
- selecting
 - color 116
 - font 38, 124
 - icon 127
 - multiple objects with Control key 19, 93
 - multiple objects with mouse 18, 94
 - object 8, 93
- selection handles 8, 93
- Sensitive property 234
- SeparatorType property 233
- Set property 233
- setting
 - default item in option menu 132
- setting property values 198
- showing
 - interface 70
 - Palette 73
 - project 71
- ShowValue property 234
- SingleSelectionCallback property 44, 111, 234
- Size and Character Set List (in Font Viewer) 39, 123
- starting UIM/X 68
- Startup Interface 199
 - changing 181
 - definition 181
- Startup Interface Area (in Program Layout Editor) 181
- swidget
 - definition 198

T

- Test phase 67
- testing
 - menus 141
- Text Editor 109, 110, 200
 - opening 107

- Text property 234
- TextBackground property 35, 113, 234
- TextField 14, 87
- TextFontList property 38, 122, 234
- Title property 235
- TitleString property 235
- Tools menu 73, 106
- typography in this guide xi

U

- UIM/X
 - callbacks supported 111
 - Clipboard in 94
 - exiting from 76
 - pop-up menus in 75
 - saving your work 12, 68
 - starting 68
- upper-left corner pointer 92
- Use as Help Pane check button (in Menu Editor) 139
- Ux Convenience Library 198, 199, 211
- UxDestroyInterface () 200
- UxGet functions 198
- UxPopdownInterface () 200
- UxPopupInterface () 200
- UxPut functions 198

V

- Value property 235
- ValueChangedCallback property 235
- VertScale 89
- VertSeparator 86, 88
- View menu (in Palette) 74
- viewing
 - fonts 123
 - icons 126
 - objects 74

W

- Width property 29, 224
- Window menu button
 - double-clicking to exit 60, 76, 113, 117, 120,

124, 127, 142

Windows category (in Palette) 81

Working Color (in Color Editor) 37, 119

X

X property 29, 224

Xt/Motif code

 alternative to 198

Y

Y property 29, 224

