OpenZinc

Designer

Version 1

OpenZinc
Designer

OpenZinc Application Framework
Version 1.0
OpenZinc Software

Preface

With OpenZinc Designer, OpenZinc offers the tightest integration available

between an interactive design tool and the supporting class library.
Most Windows developers use a resource tool to help create their program
interface. Resource tools are language- and library-dependent by design, and
therefore they are inadequate to access all the features of a given class
library. This inadequacy results in a fragmented approach to application
development, with isolated user functions and nonspecific documentation.
Consequently, the application developer is saddled with the sometimes pain-
ful details of integrating code with both the class library and the resource
tool. In contrast, the seamless integration of OpenZinc Designer and OpenZinc Appli-
cation Framework contrasts sharply with this *a la carte* approach.
As remarkable proof that OpenZinc's class library is flexible and robust, OpenZinc
built the Designer completely with OpenZinc objects. The equally remarkable
result is that with the Designer, you can work visually with OpenZinc objects—
and with all other features of the library as well. You simply drag windows
and window objects from the menu or button bar, and drop them on the

screen. And since OpenZinc Designer allows us to work with OpenZinc objects, we can port all our OpenZinc Designer output to all compilers and operating platforms that OpenZinc supports.

This approach contrasts sharply with that taken by traditional resource-builders and source-code generators. These tools lock developers into API, platform, and language dependencies, which hinder flexibility.

OpenZinc Designer provides a better way. Let's look at how you can use this powerful tool to expand your horizons.

To help you do this, we've divided this book into four sections: introduction, tutorials, reference, and appendices. The introduction provides a guided tour of the Designer, explaining how to access its functionality and features. The tutorials, based on an application called **MOVIE,** provide a detailed explanation of how to build a complex application with the Designer. The reference documents each option and window of the Designer, explaining what each does and how to use it. And the appendices section explains how to build the Designer and its components, making it easy to add in components later.

Conventions used in this book

This manual uses the following conventions:

TABLE 1. Conventions

Italics	identify arguments, variables, and pointers in function and method prototypes.
Bold	identifies file and directory names, and OpenZinc class and member function names.
`Constant width text`	identifies programming examples and command line or shell output.
`C:`	is the command line DOS prompt, which you can access from inside Windows

2 *OpenZinc Designer's Support Editors* 33

section two **OpenZinc Designer** tutorials

section three **OpenZinc Designer** reference

10 *File Options   195*

15 *Control Objects   275*

16 *Selection Objects   303*

17 *Other Objects 315*

18   *Help Options   335*

19   *Image Editor   339*

20 *Help Editor   383*

21 *Message Editor   395*

22 *Defaults Editor   409*

23 *String Editor   447*

section four **OpenZinc Designer** appendices

xxx *OpenZinc* *Designer*

section one

OpenZinc Designer

introduction

Introducing OpenZinc Designer

OpenZinc™ Designer is an interactive design tool that allows us to create and

edit components of user interfaces in OpenZinc programs, including windows and window objects, images and icons, help prompts and windows, strings used by our application, and application language and locale default information. In this chapter we will learn the basics of using OpenZinc Designer.

Running OpenZinc Designer

OpenZinc Designer runs on all environments supported by OpenZinc Application Framework including: DOS, Windows, OS/2, Macintosh, OSF/Motif, Curses, and NEXTSTEP. The Designer is located in the **/OpenZinc/BIN** directory and has the name **DESIGN.EXE** or **?DESIGN.EXE,** where the question mark indicates the environment. For example, the OS/2 designer is called **ODESIGN.EXE.** To launch OpenZinc Designer in most operating environments (i.e., DOS, Motif, NEXTSTEP, and Curses) that use command lines, simply type

```
C:> design <Enter>
```

For Windows, type

```
C:> wdesign <Enter>
```

For Windows NT, type

```
C:> ndesign <Enter>
```

For Unicode, type

```
C:> udesign <Enter>
```

For OS/2, type

```
C:> odesign <Enter>
```

In environments with a graphical user interface, such as Macintosh, double-click on the OpenZinc Designer icon. If you have trouble launching OpenZinc Designer, please see *Getting Started with OpenZinc Programming* for more information on how to run an application.

OpenZinc Designer has five main components:

• the Window Editor;
• the Image Editor;
• the Help Editor;
• the Message Editor; and
• the Defaults Editor.

Only the Window Editor should appear as a full window on the screen at startup—all other editors should be visible as minimized icons at the bottom-left portion of the screen. In this chapter, we'll discuss the Window Editor, and leave the other editors for the next chapter.

The Window Editor has four visible areas: title, pull-down menu, button bar and status bar. Each of these areas is described below.

The title area is located at the top of the window. It contains information about the Window Editor as well as the name of the file we are working on.

The menu bar is located just below the title bar at the top of the window. It contains the options necessary for OpenZinc Designer file operations and for creating window resources. Selecting some menu items causes an action to take place immediately, while selecting others causes a related window to appear, from which more options are available. Menu items that cause another window to appear are distinguished by ellipses (...). A brief explanation of each menu item follows:

File. This menu consists of options that control the creation of files and allow exiting from the Designer. The selectable items on this menu are:

- **New...**
- **Open...**
- **Save**
- **Save As...**
- **Close**
- **Delete...**
- **Preferences...,** and
- **Exit.**

Edit. This menu consists of options that edit or control the operation and presentation of objects within an application. The edit options are:

- **Object...**
- **Cut**
' **Copy**
- **Paste**
- **Delete**
- **Move**
- **Size**
- **Group,** and
- **Ungroup.**

Window. This menu consists of options that control the creation of window resources within the current file. The selectable items are:

- **Import...**
- **Export...**
- **Create**
- **Load...**
- **Store**
- **Store As...**
- **Clear**
- **Clear All**
- **Delete...,** and
- **Test....**

Object. This menu presents the objects, divided into four groups, that can be created with the Designer. The four groups presented in the first level pull-down menu are:

- **Input**
- **Control**
- **Selection,** and
- **Other.**

Selecting one of these items causes another menu to appear which contains the actual window objects of that group.

Help. This menu provides a list of the following selectable help contexts:

- **Index...**
- **File**
' **Edit**
- **Window**
- **Object,** and
- **About Window Editor.**

All of these menu items are discussed in more detail later in this book.

The button bar   The button bar presents some of the available window objects within OpenZinc Designer. It is designed to allow you to easily select these items with a mouse and then attach them directly to your current resource window. When one of the objects is selected, its name appears in the **place object** field on the status bar, where it remains until it is attached to a window, or until another object is selected from the button bar. The object is attached to a resource by positioning the mouse cursor on the desired location and clicking the mouse button.

The button bar is not available in text mode.

The Status bar   The status bar displays some information about the current object and about

the state of the current process. The following fields are present:

type. Indicates what the current object is.

name. Displays the string identification of the current object.

pos. Indicates the position, in cell coordinates, of the current object. If the current object is attached to a parent window, its position is relative to that parent window.

size. Indicates the size, in cells, of the current object (width by height).

place object. Indicates the object that has been most recently selected from the button bar (or from the **Object** options menu) that is ready to be placed on a resource window.

Help

In addition to the four major areas, we can request help at any time by select-ing **Help** from the pull-down menu or by pressing the system-dependent key that invokes the help system (<F1> under DOS and Windows, for example).

How to start

Once you have entered OpenZinc Designer and the Window Editor, the following steps can be followed for creating a basic application:

1. Open a new file for the application by selecting **File** I **New...** Select the drive and directory to which the file is to be saved, and enter a name for the file at the **Filename** prompt. If all of the information is correct, select the **OK** button.

2. Create a new resource by selecting **Window** I **Create.** A generic window that can be moved and sized will appear on the screen.

3. Attach the desired objects to the window:

- Select the objects with the mouse directly from the button bar, or select them from the **Object** menu options.

- Position the cursor in the window at the desired location and press the left mouse-button.

4. Edit the objects:

- Call the information notebook by double clicking on the object itself, or double click on the resource window and then select the object (by dou-ble-clicking on its name) from within the **Subobjects** folder.

- Change the default information by positioning the cursor on a field, pressing the left mouse-button, and entering the new information. Flags are toggled by clicking on the associated option.

  Notice that there are several different folders in which default informa-tion is listed. You can change folders by clicking on the tab of the desired folder. When all of the necessary information is entered for the object, select the **OK** button of the current folder.

5. Save the current window resource by selecting **Window** I **Store As...** Enter a name for the window at the **Name** prompt. Select the **OK** button to close the **Store As...** window and store the resource.

6. Save the current file by selecting **File** I **Save.**

7. Test the resource by selecting **Resource I Test...** and interacting with the objects. When you are done testing it, select the **Exit Test** button.

8. To add other resources to the current file, repeat steps 2 through 7.

Creating a simple window

Let's familiarize ourselves with the some of the window objects that can be created using OpenZinc Designer. First, we'll create a simple window. Then we'll add a prompt object, a string field, a button field, and a spin control object.

Creating the resource window and its objects

Create a resource window by selecting **Window I Create** from the pull-down menu. Once selected, a medium-sized window will appear at the center of the screen. This window has a border, maximize and minimize buttons, a system button, and a title.
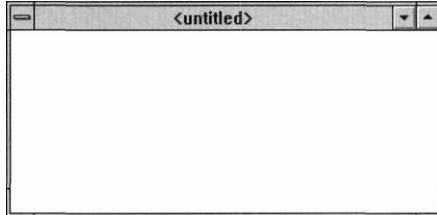
Let's place the prompt object on the window. Here's how:

1. Move the mouse cursor over the **Prompt** object in the button bar.

2. Select the prompt's bitmapped button by pressing the left mouse-button. Notice that once you select **Prompt,** the status bar has the word **Prompt** in the **place object** field.

3. Place the new prompt object in the resource window by moving the mouse cursor over the resource window, and then

4. Click the left mouse-button on the desired location.
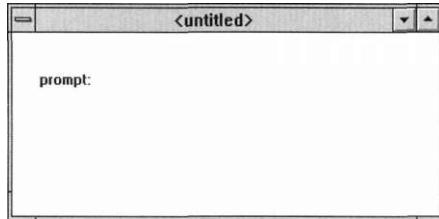
Follow the same procedure to create a string object. The string object's bit-mapped-button is located on the top-left part of the button bar.

Select the string's button, then place the object by clicking the left mouse-button inside the resource window.
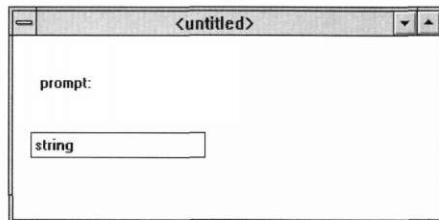
Select a button object for placement in the window.

Create and place it by selecting the button's bitmapped-image from the button bar, and then by clicking the left mouse-button at any position inside the resource window.
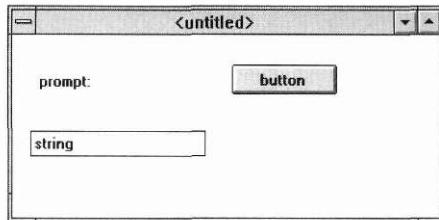
The final object that we will create is a spin control. The spin control is located on the second line of the button bar and is the first bitmapped button you see.

Place the spin control anywhere inside the resource window.
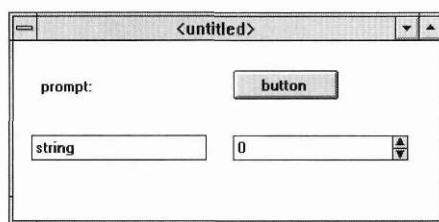
Creating and placing window objects using the Designer is simple, but it is not the only method. Later, we will discuss some of those ways.

Information inside of objects

Each object has information, such as default data, location, and settings, encapsulated inside of it. We can view or edit this information by double-clicking the object, which shows its information in notebook form. The information notebook contains information unique to that object on its first page, and generic information on its subsequent pages.

Prompt information

For example, if we double-click on the prompt object, the prompt's information notebook appears:
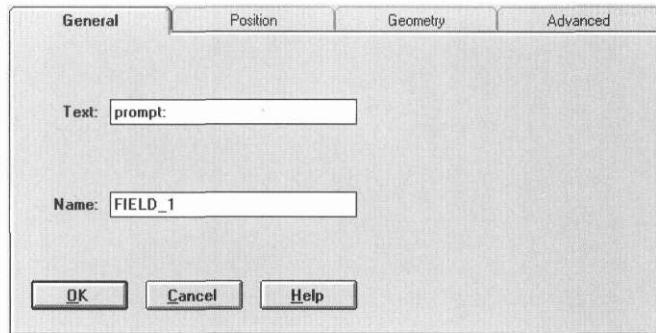
The prompt information notebook contains four pages. They are:

- **General**
- **Position**
- **Geometry;** and
- **Advanced**

Let's first examine the **General** page.

General. The **General** page contains only information associated with the prompt, namely the prompts **Text** and **Name.** You should see the word *prompt:* in the **Text** field. This is the default text that is connected with our prompt when we create and place it on the resource window. Near the second field you should see the title **Name.** This associated data field contains the programming name of our prompt object, in our case, **FIELD_1.**

Position. The **Position** page is presented when we click the left mouse-button over the **Position** tab.
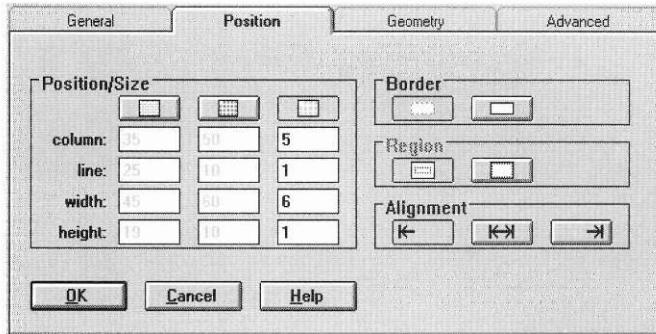
Position has four areas of interest: **Position/Size, Border, Region,** and **Alignment. Position/Size** contains several buttons, prompts, and numbers. Each number describes the location and position of the prompt on the resource window, including pixel, mini-cell, and cell coordinate information. The **Border** area tells us whether the object will be drawn with a surrounding border. The **Region** area controls how the object uses its region; whether it expands to fill the space of its parent or uses the position and size information shown in the **Position/Size** area. The **Alignment** area tells us whether the text is left, center, or right justified within the field's region.
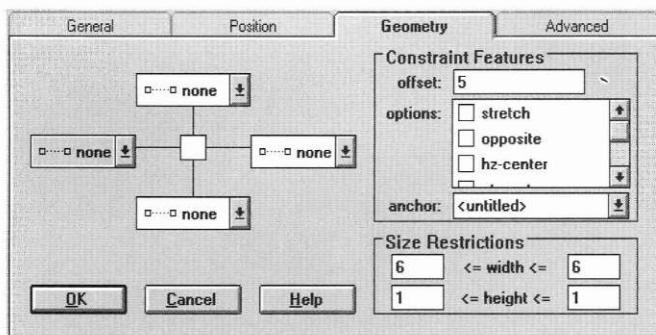
Geometry. This page has three main object groups. The first group, located at the top-left portion of the notebook page, defines the object's geometry constraints. The top-right portion of the notebook page allows us to modify the constraint information identified by the first region. The bottom-right

portion of the notebook page contains information that restricts the prompt object's height and width within its parent window. We will discuss geometry management in greater detail during a future tutorial.
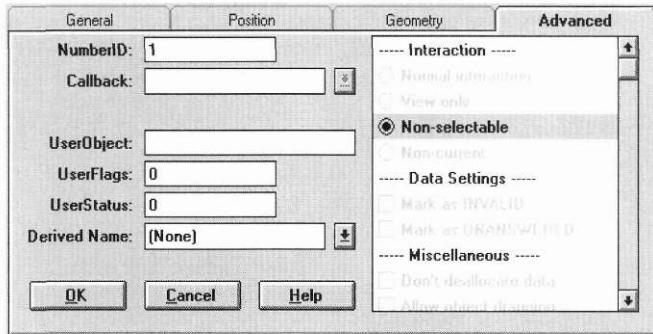
Advanced. This page contains the object's advanced features and data options. In brief, it provides a place for user information—where we can change the programming, or identification, number associated with our object—-and a list that contains many advanced features and options, useful when programming advanced objects. We'll examine more features associated with this page in later tutorials.

Now let's exit the prompt's information notebook. One way to exit the prompt notebook is to select the **OK** button. Besides exiting the notebook, this saves the data we changed when viewing the notebook pages. We can cancel the operation without saving changes by pressing **Cancel.**

Further, if we need help with any of the pages in the notebook, we could press **Help,** which would cause the Designer to display a help window sensitive to the context of the application. This means if we press the **Help** button in the advanced page, we will receive help on the advanced portion of our object. Similarly, if we press **Help** on the general section of the notebook, help information will be presented for the object's general features.

Let's leave the prompt information notebook by pressing **Cancel.**

String information

Let's look at the information notebook for the string object. Invoke the string information notebook by double-clicking on the resource window's string object.
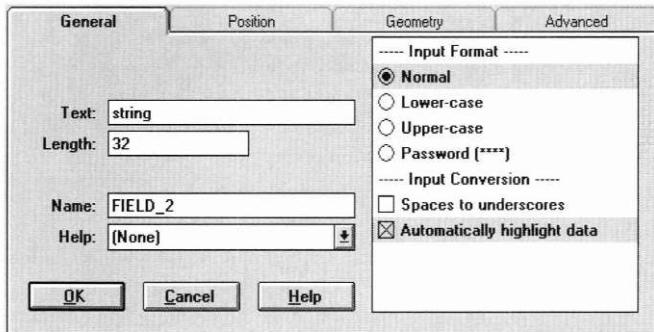
The general page of the string object contains the same **Name** and **Text** field that were visible on the prompt's general information page. In addition, however, new **Length** and **Help** fields appear on the page. The notebook's **Text** field shows the default data for our string object. The **Length** field indicates the maximum length of text that can be entered into our string. The default value for string data is 32 characters. The **Name** field, as described for the prompt object, is the string identifier associated with our object, in this case a string. The **Help** field contains the help context name to associate with our string object. This field will be described later in this tutorial.

The right side of the general page lists many formats that can be selected with our string object. These formats include: normal input, lowercase, uppercase, and password. In addition, we can convert spaces to underscores when we edit the field or we can highlight the string data when we begin editing the string data. We will discuss these features more extensively in later tutorials. For now, we just want to become familiar with the location of the fields on the notebook.

Take a few minutes to browse through these pages and to examine the information associated with string objects. Once you have finished browsing this notebook press the **Cancel** button.

Button Information

Let's now look at the button object and its associated information.
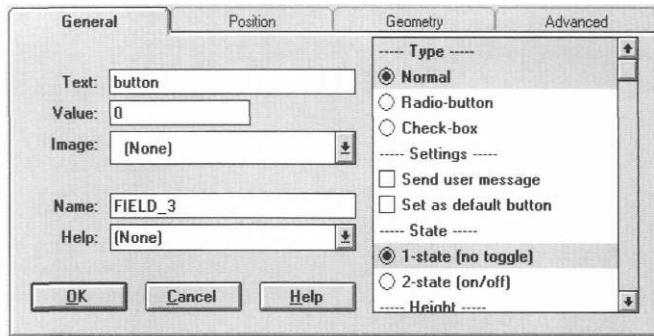
The button object has text, value, image, name, and help fields, as well as many button flags. All of these features can be selected to change the operation and appearance of a button on the screen. In time, you will find that most of these features are beneficial to your programming efforts. Take a moment to view this information.

Spin control Information

The final object whose information we will look at is the spin control object.
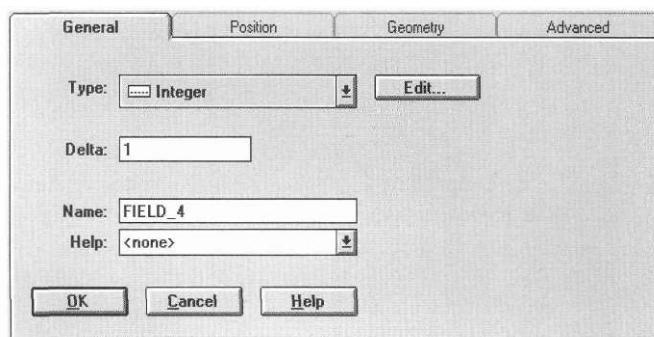
The bottom two fields **Name** and **Help** look like fields from the other objects that we created. However, the **Type, Edit,** and **Delta** fields are unique to spin controls. By default, the spin-control object contains a 16-bit integer value, but its information notebook allows us to change its default data type to a date, time, real, or bignum type.

We've accessed the information associated with our prompt, string, button, and spin control objects. Each object contains unique information on its first information notebook page, and generic information on its subsequent notebook pages. We've also created a simple window with simple objects, which we'll use as a springboard to learn new parts of the Designer.

Creating lists and grouped objects

We just learned how to use the Designer to access the information associated with certain window objects. Now we're going to learn how to use the Designer to create lists and groups. Move the simple window we just created to the bottom of the screen, and create another window by selecting **Window I Create** from the pull-down menu. Another untitled window will immediately appear on the screen.

We will create two objects for this window using the drag-and-drop capability of OpenZinc Designer. To create the vertical list using drag and drop, do the following:

1. Move to the vertical list button on the button bar.

2. Click down on the left mouse-button, then, while holding down the left mouse-button

3. Drag the object to the resource window.

4 . Release the left mouse-button.

OpenZinc Designer automatically places the object inside the resource window.
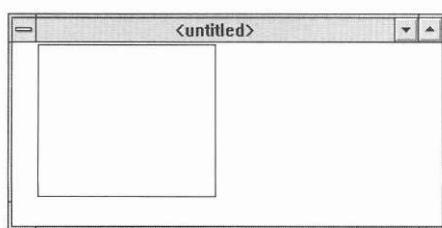
Now let's do the same with a group object. Click down on the group's button, drag the object to the resource window, and release the left mouse-button. This places the group object.
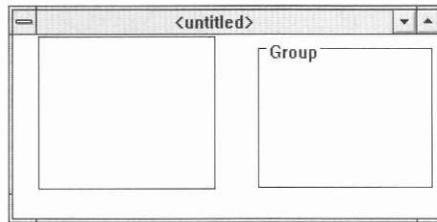
Let's now look at some other ways that objects can be created and placed into lists and groups. First, recall how we created and placed objects in the first simple window. We selected an object by clicking the left mouse-button while positioned over the object's button on the button bar. Let's do this now with a string object; click on the string button, release it, then move the mouse cursor over the list object and reclick the left mouse-button. This places a string object inside of the list.

At this point, we could go back to the string button, reclick the button from the button bar and reclick inside the list. This would allow us to slowly create and place string objects in the list—but OpenZinc Designer offers a faster way of creating and placing additional string objects. This method simply requires us to click the right mouse-button while positioned inside a resource window. When you click the right mouse-button the **place object** field on the status bar updates to reflect that a new string object can be added. When we click the left mouse-button, the place object goes away, but the string appears in our list. We can place another string object by simply reclicking the right-mouse and then clicking the left mouse-button inside the list.

Another way to create objects—this one a little slower but more intuitive for finding objects if we are not familiar with the Designer—is with the pull-down menu. The **Object** menu item has several object categories—**Input, Control. Selection,** and **Other**—which contain groups of particular object types. Though we can also access these objects through the button bar, object categories allow us to more easily find the objects we want. For instance, the

Input category contains all the objects related to inputting information; objects including string, formatted string, text, date, time, bignum, integer, and real input objects.
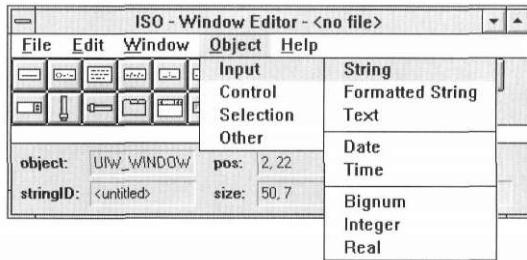
Getting back to object creation, now add a set of radio buttons to the group object by clicking on the **Object** pull-down menu. Then move down to the **Control** item and select **Radio Button** from the pop-up menu. Now the words **Radio Button** appear in the **place object** field.
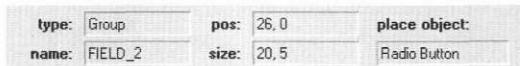
We can now place a radio button inside our group object by moving the mouse cursor over the group object and by clicking the left mouse-button.

Place three more radio buttons inside your group by doing any of the following:

- Selecting a radio button from the pull-down menu in the **Object** category,
- Selecting the radio button image from the button bar, or by
- Clicking the right mouse-button to reactivate the radio-button place object.
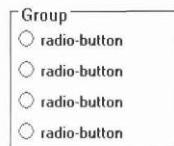
List information   Let's look at one more OpenZinc Designer feature through the list information notebook. Move the mouse cursor over the list object and double-click with the left mouse-button. At this point the vertical list notebook appears. Notice

the same four categories that we had with the simple objects, namely: **General, Position, Geometry,** and **Advanced.** In addition, however, we now have a new category called **Subobjects.** Select the subobjects page by clicking the **Subobjects** tab with the left mouse-button. There are four areas in the subobjects notebook page. The first is called **Objects,** the second **Directories,** the third is a group of support buttons, and the fourth is a combo box, located on the bottom-right portion of the page.
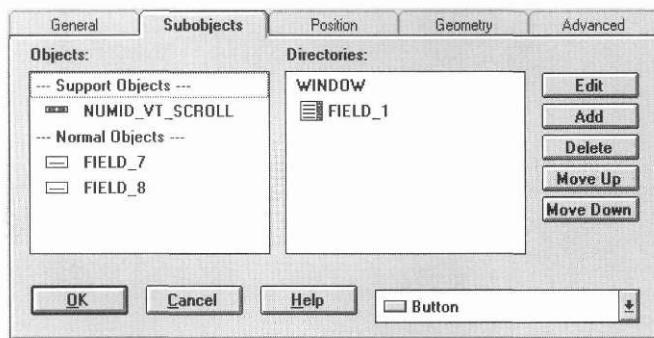
Let's first look at the items in the objects list. The first item in the list is an image of a scroll bar with the word **NUMID_VT_SCROLL.** In addition, the list contains small pictures of the string objects inside our edit list. We can bring up their information notebooks by:

• double-clicking the left mouse-button on any of the objects in this list, or by

• scrolling to the desired list item and pressing <Space>, or by

• scrolling to the desired list item and selecting the **Edit** button, located on the right side of the notebook page.

Now let's bring up the string information notebook. Double-click on the first string object in the list. A new window appears for the string object, just underneath the vertical list notebook.
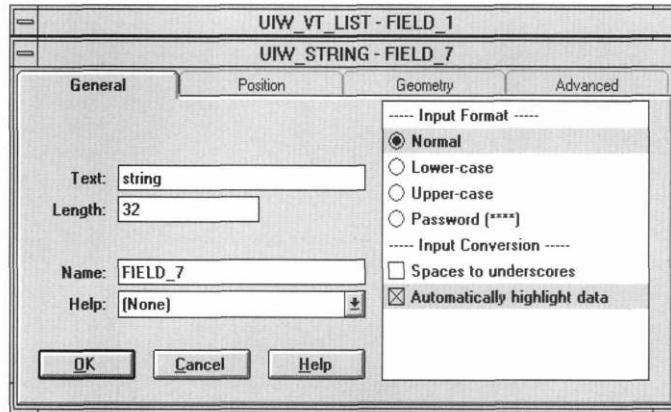
Browse the string notebook information, then exit by selecting the **Cancel** button. Control now returns to the vertical list notebook.

The support buttons, located on the right side of the notebook page are used to edit objects, add and delete objects, or to move objects up or down in the objects list. The directory portion of the notebook shows us our current edit object, the vertical list, and the parent hierarchy of the object back to the untitled resource window.

The bottom-right portion of the notebook page allows us to create additional objects of a specified type. The default type of object for vertical list is **Button.** We can add a new **Button** object by selecting the **Add** button located on the right side of the notebook page. We will examine these features later in our tutorial.

Group Information

Now exit from the vertical list notebook by pressing **Cancel.** Bring up the group notebook by double-clicking on the group object in the resource window. (Be careful as you double-click on the group not to select one of the radio buttons inside the group, as doing so would open the radio button information notebook.)
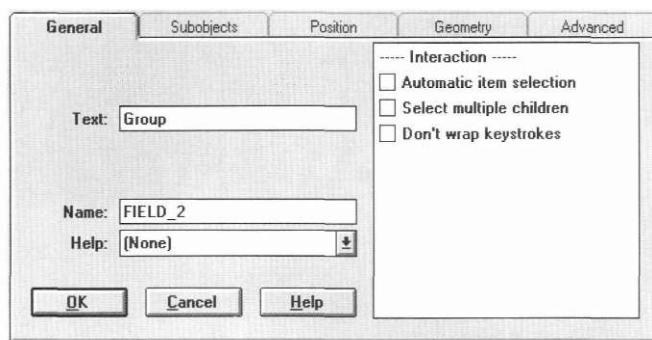
The group notebook contains the same five categories that were available with the vertical list. If we click on the **Subobjects** tab, we view the same information page seen in the vertical list, but under the **Objects** category we see button images instead of strings. These are the four radio buttons that we inserted into the group. Exit from the group notebook by selecting **Cancel.**

Take a moment to browse the list and group objects, then move the resource window to the bottom-right portion of the screen next to our first resource window.

Creating advanced window objects

Let's now create a window with advanced objects. Select the **Window I Create** option to create a new window. We are going to place three objects on this window; a pull-down menu, a button bar, and a status bar. First, create the pull-down menu using one of the methods discussed earlier, then place

the pull-down menu inside the resource window. Next, create and place a button bar inside of the resource window. Finally, create and place a status bar inside the resource window.

```
┌──────────────────────────────────────────┐
│ ▭          <untitled>              ▼  ▲    │
├──────────────────────────────────────────┤
│ item                                       │
├──────────────────────────────────────────┤
│                                            │
│                                            │
│                                            │
├──────────────────────────────────────────┤
│                                            │
└──────────────────────────────────────────┘
```
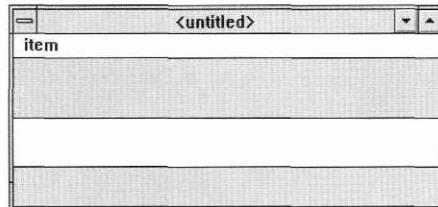
Let's place a few pull-down items in the pull-down menu by selecting the pull-down item option from the button bar then by clicking the left mouse-button inside the pull-down menu. (Remember that the right mouse-button reactivates the last object you created). Next create several buttons to go inside the button bar. Do this by selecting the button item from the button bar and placing it inside the button bar. Finally, place several strings inside the status bar. This is done by selecting the string from the button bar and then by placing it inside the status bar object.

```
┌──────────────────────────────────────────┐
│ ▭          <untitled>              ▼  ▲    │
├──────────────────────────────────────────┤
│ item   item   item                         │
│ ┌────────┐ ┌────────┐                      │
│ │ button │ │ button │                      │
│ └────────┘ └────────┘                      │
│                                            │
├──────────────────────────────────────────┤
│ string            │ string                 │
└──────────────────────────────────────────┘
```
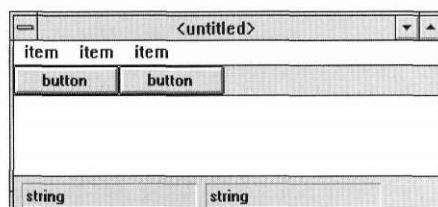
Button bar Information

Let's take a moment to look at the button bar's edit window. Invoke the button bar notebook by double-clicking on any region outside the button objects' regions, but inside the button bar's region.
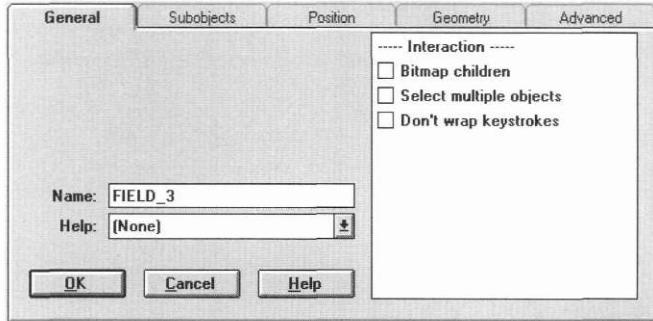
Notice the five notebook categories; **General, Subobjects, Position, Geometry,** and **Advanced.** Information contained in the button bar edit window is similar to that contained in the vertical list and group objects we created earlier. Once you have finished browsing the button bar information, exit by selecting **Cancel.**

Subobject Information

The method used to bring up edit information for the pull-down menu is a little more complex. In most environments, this object is considered a support or decorative object, so we cannot invoke its information notebook by simply clicking inside the object's region. This information notebook must be called through the Window Editor. To do this, bring up the **General** information notebook by double-clicking anywhere inside the edit window.
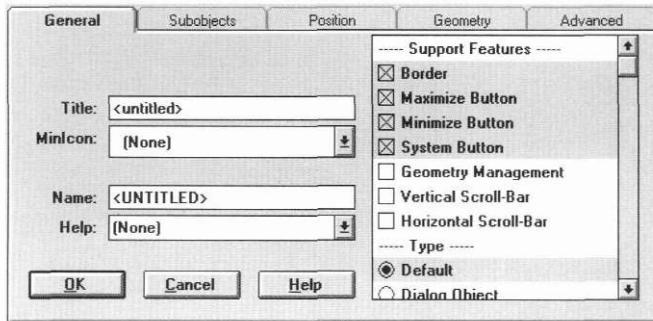
Now move to the **Subobjects** page. On the left side of the page under the **Objects** category there is an image associated with the pull-down menu. In addition, on the right-side of the notebook page, under the **Directories** category, there is another field associated with the pull-down menu. The left-side

list allows us to directly invoke the object's information notebook. The right-side list, however, allows us to traverse through the window object hierarchy and view any of the objects that we have placed in our resource window. Double-clicking over items on the right-side list does not bring up their associated information notebook, it simply traverses down into that object's view. For example, double-click on the pull-down menu item located in the right-side list.

General	**Subobjects**	Position	Geometry	Advanced

Objects:

--- Support Objects ---	
--- Normal Objects ---	
🖹 FIELD_2	
🖹 FIELD_5	
🖹 FIELD_6	

Directories:

<untitled>	
🖹 FIELD_1	
🖹 FIELD_2	
🖹 FIELD_5	
🖹 FIELD_6	

[Edit] [Add] [Delete] [Move Up] [Move Down]

[OK] [Cancel] [Help] 🖹 Pull-down Item ▼
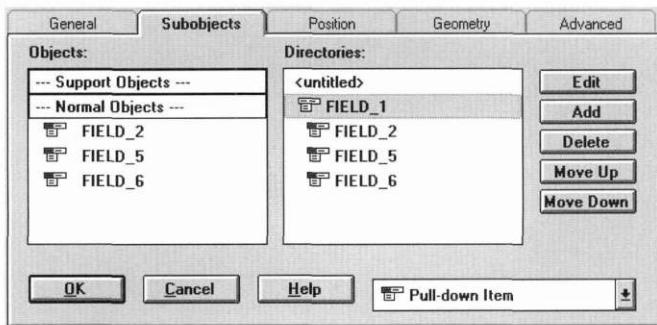
Inside the pull-down menu's view, we can select any of the pull-down items from the left-side **Objects** list by double-clicking on the item.

Let's briefly revisit the **Add** feature associated with this notebook page. On the left-side, we see the pull-down items created earlier in the tutorial. In addition, the word *Pull-down Item* appears inside the combo box, located on the bottom-right portion of the page.

We can now add a pull-down item to the pull-down menu by pressing **Add.** Once the **Add** button is selected, an additional pull-down item appears in the object list. These are actual window objects and are as real as the other pull-down items created through the main Window Editor.

Let's delete some of the pull-down items by selecting an item in the object list and then by pressing the **Delete** button. When we delete an item from the object list it is removed from our view. We can delete additional items by continuing to select the **Delete** button.

The **Move Up** and **Move Down** buttons, located on this notebook page, allow us to move items either up or down in the object list, thereby changing their position and the user movement on and off the items. To move an item up in the list, do the following:

1. Select the item from the object's list.

2. Press the **Move Up** button on the right side of the notebook page, or type <Ctrl + up-arrow> from the keyboard. (To move an object down follow the same process but select **Move Down** or type <Ctrl + down-arrow>.)

After you have browsed this notebook, exit back to the main Window Editor by selecting **Cancel** from any notebook page. Now, prepare for the next tutorial section by moving the pull-down window to the bottom-right portion of the screen, next to the other two windows we created previously in this tutorial.

*Using the **Edit** command*

We will now shift our focus from creating window objects to editing and placing objects. Let's give ourselves some objects to edit by creating a new window and creating several prompt and string objects. Place them randomly on the window, but keep the prompts on the left side of the window and the strings on the right side of the window.
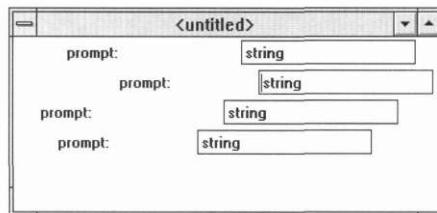
We can group objects for modification by using the **Edit I Group** option located in the pull-down menu, or by:

1. Pressing the <Ctrl> key on the keyboard, while

2. Pressing the left mouse-button, then by

3. Dragging the mouse cursor over the objects that you want to place in the group, shown by a overlapping rectangle, and then by

4. Releasing the mouse button once all desired objects have been included in the group.

For example, use the **Edit I Group** command to select the prompt fields in our window. First, select the **Edit** I **Group** option, click the left mouse-button in the top-left portion of our window, and then drag the mouse cursor over the prompts that we have created in the window. Finally, release the left mouse-button. The new edit group forms a shaded rectangle behind the four prompts.
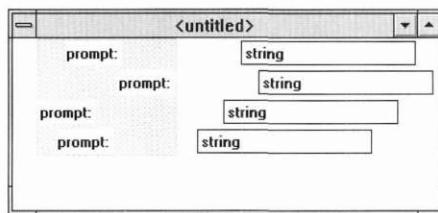
We can modify the position of these prompts by pressing down on the group with the left mouse-button and then by moving them with the mouse drag operation. In addition, we can move the group by selecting the **Move** and **Size** commands located under the **Edit** option. Conceptually, the shaded window area acts as a new group or object. All of the prompts within the shaded region are now considered subobjects of the edit group.

Let's change the left justification of all the prompts inside our edit group. Double-click the left mouse-button anywhere inside the edit group.
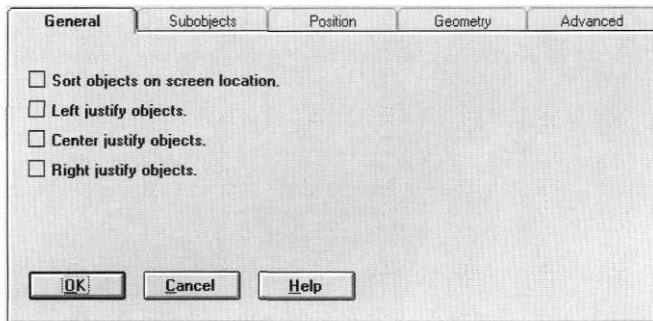
We make all of the prompts in our edit group left justified by selecting the **Left-justify objects** option in the notebook and then by pressing **OK.**
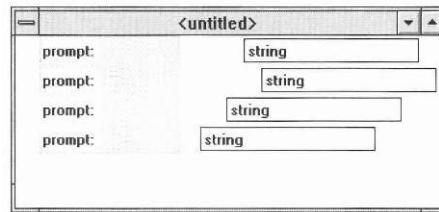
In addition to left justification, we can center- or right justify the objects inside our edit group using the same method, but by selecting the appropriate option. To remove the edit group, select the **Edit I Ungroup** option from the pull-down menu. Once the edit group has been removed, each object returns to an autonomous state—it can be moved, sized or edited individually.

Group the string objects using only the keyboard and mouse by simultaneously pressing the <Ctrl> key and the left mouse-button, then by dragging the left mouse-button and the associated XOR rectangle over the string objects. Be careful not to begin your group operation over the prompt objects—this would cause some of the prompts to be included in the string group. Left justify these objects by double-clicking on the edit group, then by selecting the **Left-justify objects** option and by pressing **OK.**

Move the string group close to the prompts by pressing the left mouse-button over the edit group, then by dragging the objects near the prompt objects.
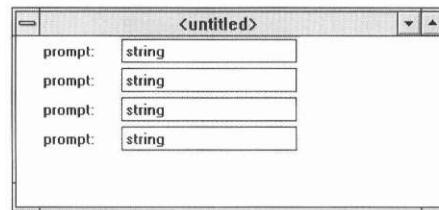
Now group both the prompts and the strings together to examine the **Edit I Cut, Edit I Copy** and **Edit I Paste** commands. Do this by pressing the left mouse-button over the prompt area, by dragging the mouse to include the string objects from the string area and then by letting-up on the left mouse-button.
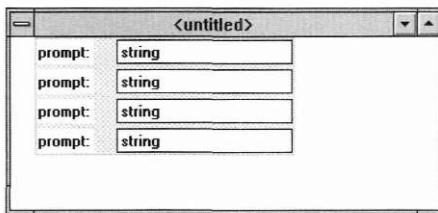
Let's first examine the **Edit I Copy** option. We can copy the edit group we selected by choosing **Edit I Copy** from the pull-down menu and then selecting **Edit I Paste.** Once you choose this option, notice the **place object** status indicator contains the new string *Edit Group.* We now have a copy of the edit group in the copy/paste buffer. To paste the edit group, press the left mouse-button inside the edit window.
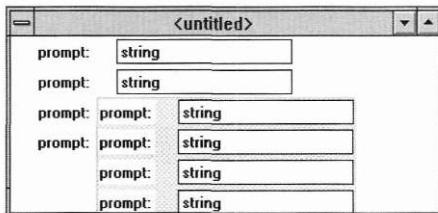
We can create additional copies of the edit group, by pressing the right mouse-button to reactivate the place object, then by pressing the left mouse-button to create a new edit group.

Each time you paste an edit group, the focused edit group changes. OpenZinc Designer restricts users to only one edit group per window. This is always the last edit group defined or created using any of the edit commands.

Let's remove the edit group that we just created, by selecting the **Edit I Delete** option from the pull-down menu.

The **Edit I Cut** option is similar to **Edit I Copy,** except that the edit group is removed from the screen. The contents of the edit group are moved directly into the paste buffer. Retrieval of the paste buffer is accomplished in the same manner as described previously in this tutorial.

We have now looked at some of the basic features associated with the **Edit** option. Prepare for the next section by removing the edit group.

File options

Let's give each of our four windows the names *SIMPLE, LIST, PULL-DOWN,* and *GROUP.* Bring back the first window into the middle of the screen by pressing the left mouse-button while positioned over the window's title, and by dragging the window to the center of the screen. Double-click inside the window's region to bring up the information notebook.
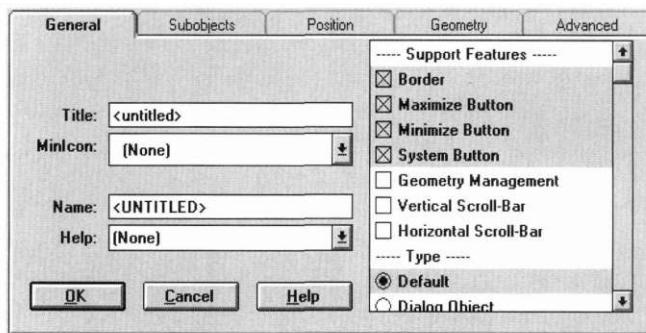
Type in the word

```
Simple
```

in the **Title** field. Then move to the **Name** field, and enter the word

```
SIMPLE
```

Save both the title and name by selecting **OK.** Now bring the list edit window to the middle of the screen. Change its name by double-clicking on the window, typing the word

```
List
```

into the **Title** field, and

```
LIST
```

into the **Name** field, then select **OK.** Change the name of the pull-down menu window by bringing the window to the middle of the screen, invoking it's associated information notebook and by entering the word

    Pull-Down Menu

into the **Title** field, and

    PULL_DOWN

into the **Name** field. Finally, change the name of our edit group window by following the same procedures and entering the name

    Group

into the **Title** fields, and

    GROUP

into the **Name** fields. All of our windows now have unique names and are ready to be saved out to a file. Save the information to a file named **TEMP.DAT** by selecting the **File** I **New** option, then by entering the name

    TEMP.DAT

into the field just under the **Filename** prompt, then by selecting **OK.**

After creating the file, all of the windows on the screen can be saved by selecting the **File** I **Save** command. Notice that two windows appear during the save operation—a **.CPP** status window, and an **.HPP** status window, which show the state of our save operation. The first status window is entitled *Generating the CPP file.*
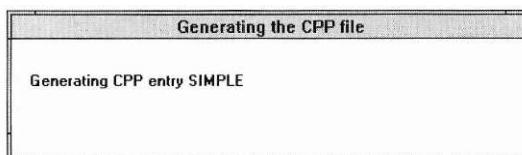
The second window is entitled *Generating the HPP File.*
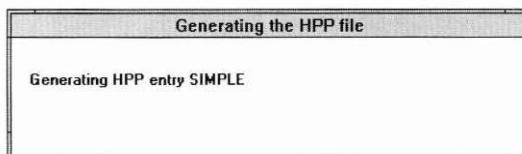
Each time a window is stored, the status information for the window changes. For example, saving our windows will cause the status window to update *SIMPLE, LIST, PULL-DOWN,* and *GROUP*

The file **TEMP.DAT** contains the four windows *SIMPLE, LIST, PULL-DOWN,* and *GROUP* that we created during our edit operation. Selecting the **File I Save** operation also causes two other files to be created—**TEMP.CPP,** and **TEMP.HPP**

TEMP.CPP contains entries used by programmers when the windows are used in an application program. In particular, this file contains a jump table and a user information table. **TEMP.HPP** gives us programming hooks that tell the name and index of a particular field inside our edit window. We'll discuss the details of using **TEMP.CPP** and **TEMP.HPP** in the next tutorial.

We will come back to our edit windows in a few moments. Let's first prepare for the next section of this tutorial by selecting the **Window I Clear All** option from the pull-down menu. When you select **Window I Clear All,** all of the edit windows are removed from the screen, though they still exist in the **TEMP.DAT** file.

OpenZinc Designer's Support Editors

In the last chapter, we learned about the Window Editor, the main editor of OpenZinc Designer that allows us to create and edit windows. In this chapter, we'll learn about the support editors: the Image Editor, the Help Editor, the Message Editor, the Defaults Editor, and the String Editor. These are what allow us to edit bitmaps and icons, create help contexts, create run-time loadable strings, and internationalize applications. If you've closed the **.DAT** file we created in the last chapter, open it again, since this tutorial will modify the objects we created.

Key Concepts
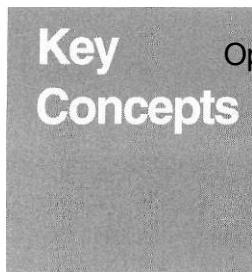
OpenZinc Designer's support editors

creating and editing bitmaps and icons

creating and editing help contexts

basics of globalization using OpenZinc Designer

Image   Editor

The first support editor is the Image Editor, located in the bottom-left portion of the screen at the Designer's launch time. Open the Image Editor by double-clicking on the image edit icon or by single-clicking the image edit icon and then selecting the **Restore** option from the pop-up menu that appears above the Image Editor's icon.
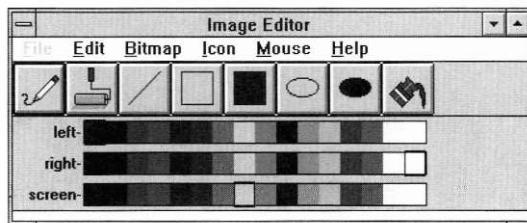
Main components

The Image Editor's interface has four main components:

- **the title;**
- the pull-down menu;
- the button bar; and
- the color bars.

Title. The title, which contains the words *Image Editor*, is located on the top portion of the window like the Window Editor.

Pull-down menu. The pull-down menu, which appears just under the title, controls the operation of the Image Editor. However, it does not control the operation of the Window Editor. In particular, we have the same **File** option available as the Window Editor, except this option is grayed out. (Since the Image Editor is a subcomponent of the Window Editor, the option is not selectable in the Image Editor—the Window Editor performs all file operations.) The other options in the pull-down menu—**Edit, Bitmap, Icon, Mouse,** and **Help**—allow image editing.

Button bar. The button bar, located underneath the pull-down menu, contains buttons which represent draw operations.

Color bars. The final interface component is a set of three color bars. The first color bar shows colors that are associated with the left mouse-button. The second shows colors associated with the right mouse-button. And the third shows screen colors. We can change the current color selection with the mouse by selecting color bar options.

Learning to use the Image Editor

Let's begin learning to use the Image Editor. Create a bitmap image by selecting the **Bitmap I Create** option in the pull-down menu.
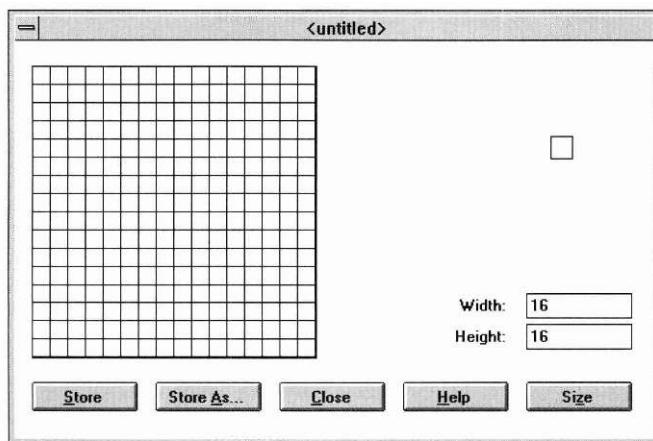
The bitmap image window has four areas—the drawing field, the actual image, the image size, and support buttons. The expanded image is located on the left side of our window. The actual image is located on the right side of the window and shows how the bitmap will appear on the screen. On the bottom-right side of the window are two numbers that contain the bitmap width and height. The support buttons, **Store, Store As..., Close, Help,** and **Size** allow us to store, close, resize, or obtain help about the bitmap image.

**Drawing an
ellipse**

Let's draw an ellipse in the bitmap. Move the bitmap window to the bottom of the screen, so the Image Editor's patterns are visible. Select the blue color from the left color bar. Then select the unfilled ellipse button. To draw the ellipse in the bitmap window, do the following:

1. Click while positioned in the top-left edit cell.

2. Drag the mouse cursor to the bottom-right portion of the edit bitmap.

3. Release the mouse button.
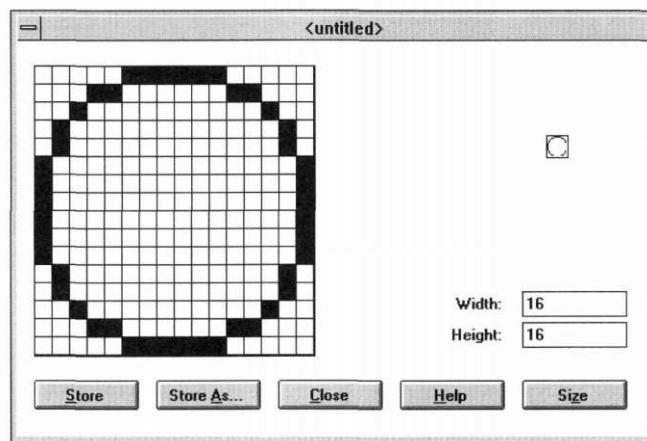
Now let's fill the ellipse with the light blue color. Use the right mouse-button to select the light blue color from the right color bar. Use the left mouse-button to select the color fill pattern, which is the right-most button on the bar

located in the Image Editor control window. Fill the interior of the ellipse by clicking the right mouse button while positioning the mouse inside the ellipse.
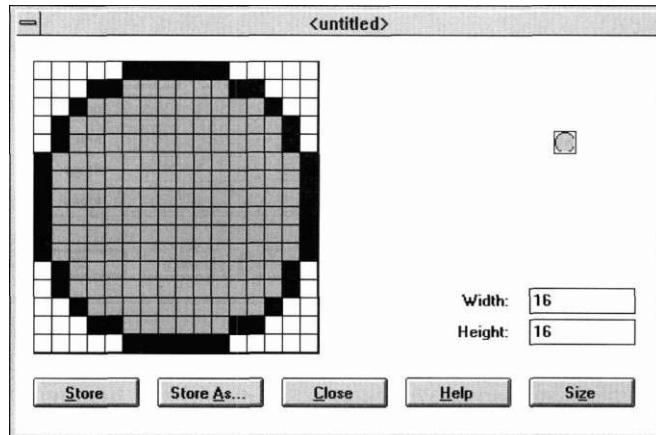
Filling the bitmap

Let's fill the outer four corners of the bitmap with a screen background color, Use the left mouse-button to select yellow from the screen color bar. We still have the desired **fill** option selected as our pattern. Move the mouse cursor over the top-left portion of the image without overlapping the ellipse and click the left mouse-button. Do the same thing for the top-right, bottom-right, and bottom-left areas of the edit bitmap. You should now have a bitmap with three separate colors, yellow on the corners, blue on the outer ellipse, and light blue on the inside of the ellipse. Store this bitmap image by selecting the **Store** As command and by entering the name

```
circle
```

into the Store As window.
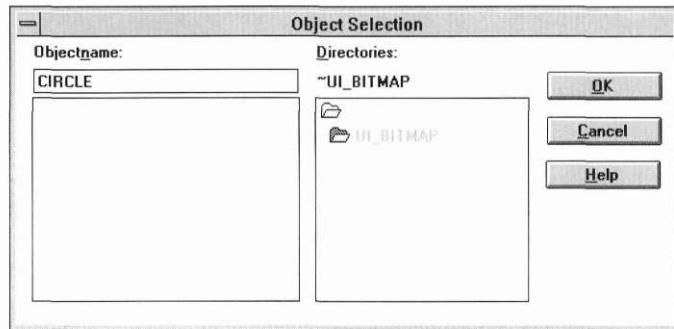
Now store the circle in the TEMP.DAT file by selecting the OK button.

Creating an icon image

Now let's create an icon image. First, close the *circle* bitmap by selecting the **Close** button. Then select the **Icon I Create** option.
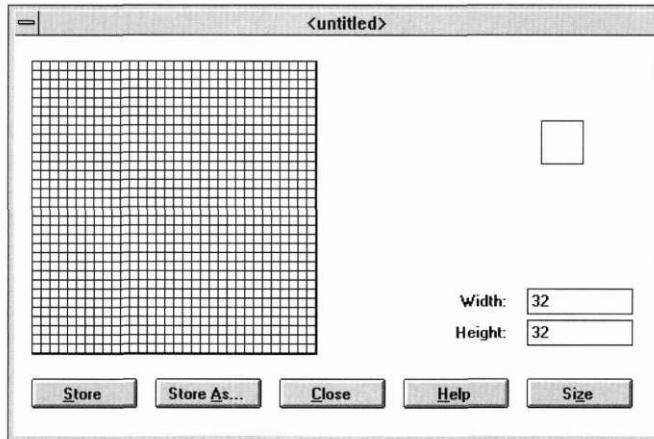
The default width and height for icon images is 32, whereas the default height and width for bitmap images is 16. Let's draw a small rectangle and an overlapping ellipse for our icon image. To draw these objects, first select the filled rectangle button from the button bar. Select a left color by pressing the left mouse-button inside the left color bar over the desired color. Move back to the icon image and press the left mouse-button just inside the icon image. Drag the image to approximately half the window size, then release the left mouse-button.
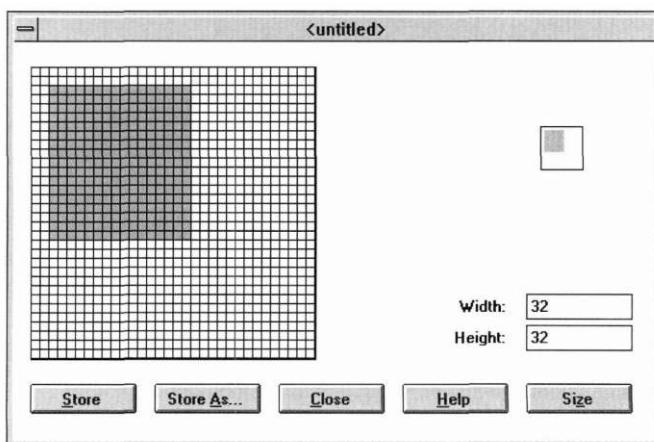
Select a new color, then select the filled ellipse as your pattern. At around the three-fourth mark of the right-bottom portion of your rectangle, begin drawing the ellipse. Release the left mouse-button once your ellipse pattern has filled half the image.
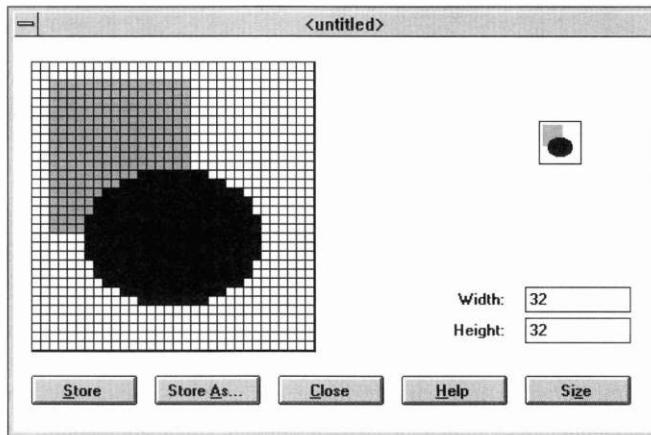
Drawing a bounding rectangle

Draw a bounding rectangle on the image by selecting the unfilled rectangle pattern and by selecting the color *black* from the left color bar. Begin the rectangle by selecting the top-left pixel, dragging the mouse cursor to the bottom-right corner of the image, and releasing the mouse-button.
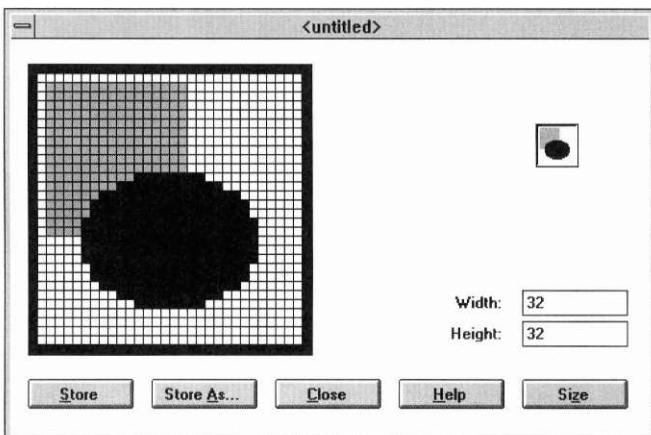
Save this icon image as *rectangle*. To do so, select the **Store As** button, enter the name

```
rectangle
```

and press **OK.** Close this window by selecting the **Close** button.

**Viewing the
Stored images**

To view the names of the images we just stored, go into the bitmap pull-down item and select the **Bitmap I Load** option.
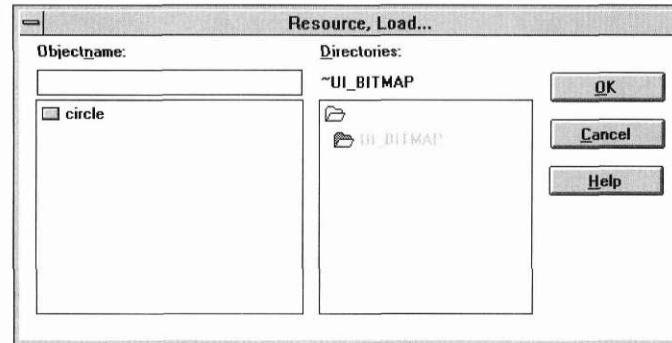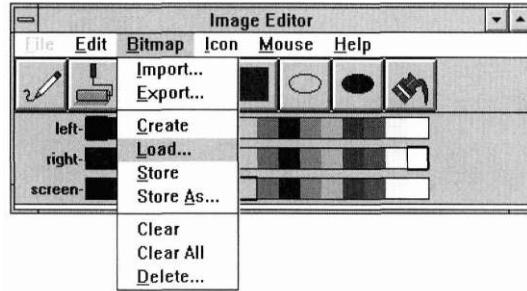
The bitmap named *circle* appears in our bitmap list. Press **Cancel** to close the window, then select **Icon I Load** to view the icon list. The bitmap and icon images are now available to the Window Editor. Save your work by returning to the Window Editor and selecting the **File** I **Save** option. Then

minimize the Image Editor by selecting the minimize button from the image window. Finally, reload the simple window from TEMP.DAT by selecting the Window I Load command from the Window Editor's pull-down menu.
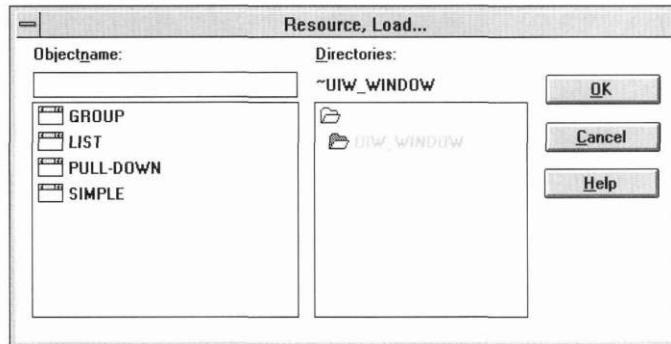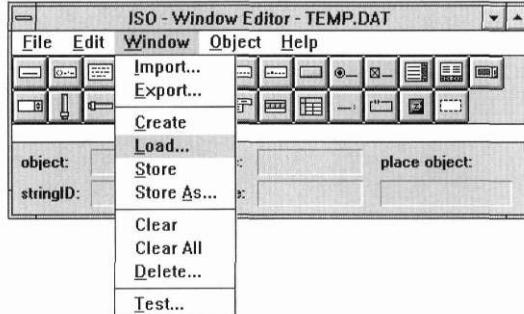
Select the simple window by double-clicking on the *SIMPLE* list item or by entering the name

```
SIMPLE
```

under Objectname and by selecting OK. The simple window is now visible on the screen.

Incorporate the bitmap and icon images into the edit window by first invoking the button's information notebook. (Remember, to select the button notebook, move to the button object, then double-click the mouse button.)
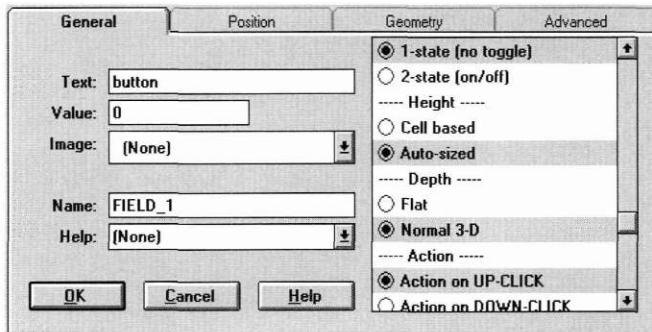
Select the *circle* image by first moving down to the image field. Then click the left mouse-button on the down-arrow of the combo-box, and select the circle image from the pull-down list. The circle image is now selected and the button information notebook reflects the image change.
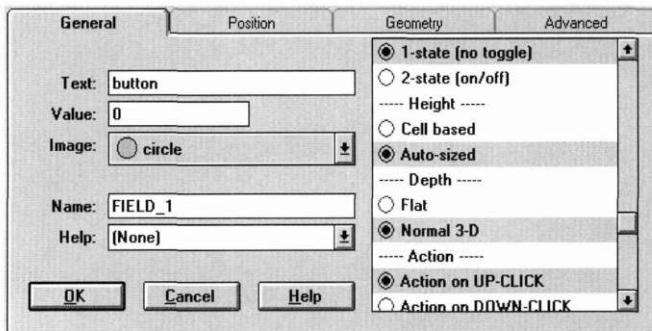
Save the changes by pressing OK. The edit window is automatically updated to show the bitmapped button.
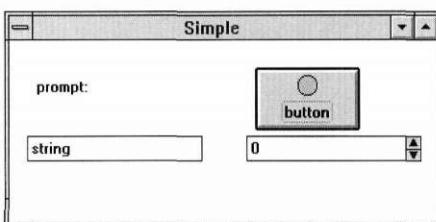
Associating the icon with the window

Let's associate the icon image with our simple window. Double-click on the window to bring up its information notebook. Move down to the **MinIcon** field, bring up the image list, then select the *rectangle* image from the available items. Save this change by pressing **OK.**

View the changes made for the minimized icon and the bitmapped button by selecting the **Window I Test** feature.
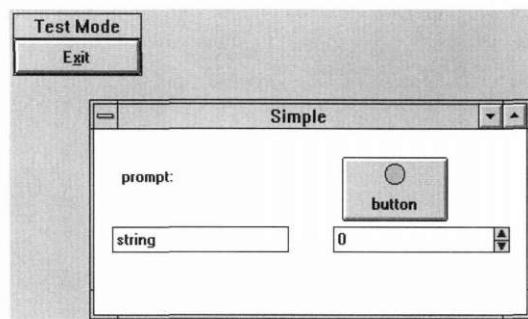
To view the minimized icon, press the minimized button from the simple window.
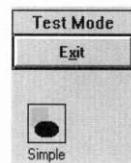
To exit test mode, press the **Exit** button on the **Test Mode** window. We're now finished with our tutorial on using the Image Editor.

We have now examined various methods used to create bitmap and icon images and how to insert those images into our simple edit window. Take a moment to browse the Image Editor and the options available through the Image Editor and the Window Editor.

Help   Editor

The Help Editor will be the final area of focus in this tutorial. We'll leave extensive coverage of the Message and Defaults Editors to a later chapter, though we will introduce them in a moment.

```
┌─┬──────────────────────────────────────────────────────┬───┬───┐
│ ─│                     Help Editor                      │ ▼ │ ▲ │
├──┴──────────────────────────────────────────────────────┴───┴───┤
│ File   Context   Help                                            │
│                                                                  │
│                                                                  │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```
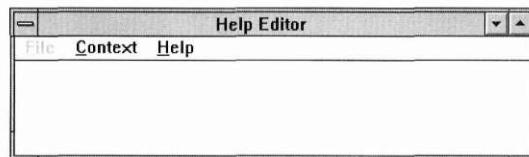
Creating a help context

Like the Image Editor, the Help Editor is also a subcomponent of the Window Editor; it too contains a grayed-out **File** option. The Help Editor allows us to create a *help context*, which is specific help information tied to a particular object. Select the **Context I Create** command to begin creating a help context.

```
┌─┬──────────────────────────────────────────────────────┬───┬───┐
│ ─│                     Help Editor                      │ ▼ │ ▲ │
├──┴──────┬───────────────────────────────────────────────┴───┴───┤
│ File   │Context  Help                                            │
│  ┌──────┴──────┐                                                 │
│  │ Import...   │                                                 │
│  │ Export...   │                                                 │
│  ├─────────────┤                                                 │
│  │ Create...   │                                                 │
│  │ Load...     │                                                 │
│  │ Store       │                                                 │
│  │ Store As... │                                                 │
│  ├─────────────┤                                                 │
│  │ Clear       │                                                 │
│  │ Clear All   │                                                 │
│  │ Delete...   │                                                 │
│  └─────────────┘                                                 │
└──────────────────────────────────────────────────────────────────┘
```

```
┌─┬────────────────────────────────────────────────────────────┐
│ ─│                        <untitled>                          │
├──┴─────────────────────────────────────────────────────────────┤
│                                                                 │
│   Title: ┌──────────────────────────────────────────────────┐ │
│          └──────────────────────────────────────────────────┘ │
│ Message: ┌────────────────────────────────────────────────┬─┐ │
│          │                                                │▲│ │
│          │                                                ├─┤ │
│          │                                                │ │ │
│          │                                                │ │ │
│          │                                                │ │ │
│          │                                                ├─┤ │
│          └────────────────────────────────────────────────┴▼┘ │
│                                                                 │
│      ┌────────┐  ┌──────────┐  ┌────────┐  ┌────────┐          │
│      │ Store  │  │ Store As.│  │ Close  │  │ Help   │          │
│      └────────┘  └──────────┘  └────────┘  └────────┘          │
└─────────────────────────────────────────────────────────────────┘
```
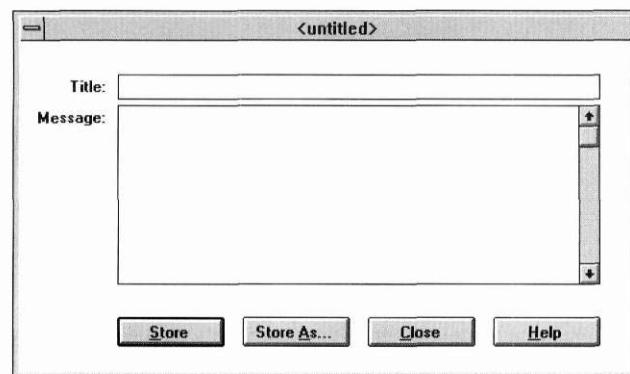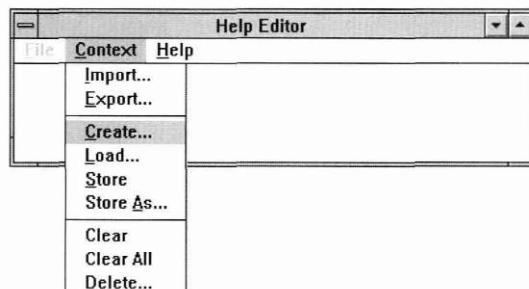
Type the words

```
Simple Help
```

into the title field. Type the words

```
Simple help for a simple button
```

into the message field.

```
┌────────────────────────────────────────────────────┐
│ ═  │              <untitled>                         │
├────────────────────────────────────────────────────┤
│                                                      │
│  Title:  │Simple Help                              │ │
│                                                      │
│  Message: │Simple help for a simple button    │ ▲ │ │
│           │                                    │   │ │
│           │                                    │   │ │
│           │                                    │   │ │
│           │                                    │   │ │
│           │                                    │ ▼ │ │
│                                                      │
│     ┌────────┐ ┌──────────┐ ┌────────┐ ┌────────┐   │
│     │ Store  │ │ Store As │ │ Close  │ │  Help  │   │
│     └────────┘ └──────────┘ └────────┘ └────────┘   │
└────────────────────────────────────────────────────┘
```
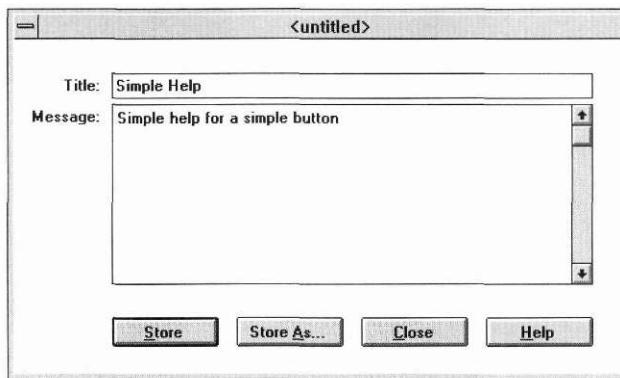
Store this information by pressing the **Store** As button, and entering the word

```
SIMPLE^HELP
```

and then by selecting OK. Close this window and create a new help context by reselecting the **Context I Create** command. Enter the words

```
General Help
```

into the title field. Enter the words

```
General help for our Application
```

into the message field.
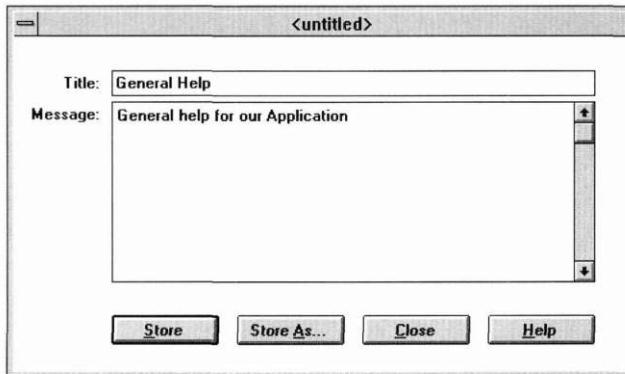
Save this help context by selecting **Store** As, by entering the name

```
GENERAL_HELP
```

and by pressing **OK.** Remove this window by pressing the **Close** button.

Connecting the help context to the help system

We have now created two help contexts that can be used by the Window Editor. Now we need to connect them to the button object. Minimize the Help Editor and then open the button information window by double-clicking on the **Edit** menu item. Move down to the **Help** combo-box and select *S1MPLE_HELP* from the pull-down list. Press **OK** to save the changes.
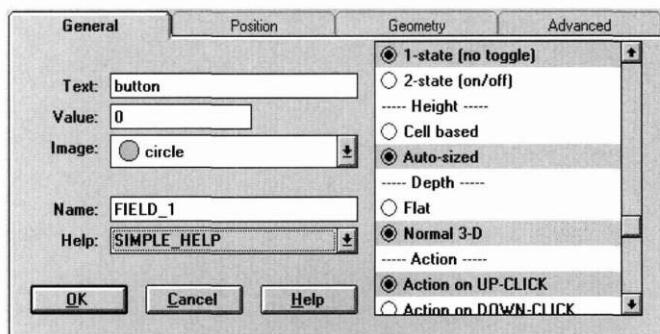
Now bring up the edit information notebook for the main window. Move down to the **Help** field and select *GENERAL JiELP* from the list. Save the changes by pressing **OK.** We can now view our help changes by reentering

the Window I Test option. Once you are in test mode move to the button field and press the help key (<F1> in DOS and Windows). This invokes the specific help associated with our button object.
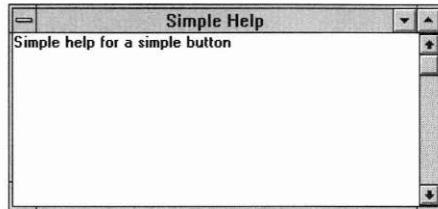
Close the help window by selecting the Close option from the system button. To view the general help associated with this window, move to any other field and press the help key.
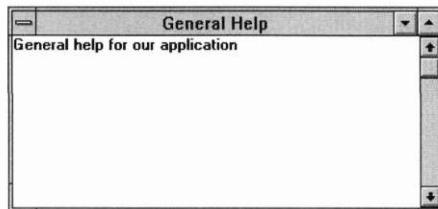
Exit the test mode by selecting Exit from the test mode window. Take a few moments to browse the Help Editor and its available options.

Message, Defaults, and String Editors

There are three remaining pieces of the Designer: the Message Editor, the Defaults Editor, and the String Editor, which we use for internationalizing applications. Briefly open the Defaults Editor by double-clicking on the **Defaults Editor** image icon. Select **Language I Create** to create a window in the English language:

Browsing the Defaults Editor

We can modify many pieces of information associated with an object. For instance, an object's **Day** tab contains *short days, abbreviated days,* and *long days.* If we wanted to translate the window to Spanish, we could edit each of these fields and change the text to the appropriate language.
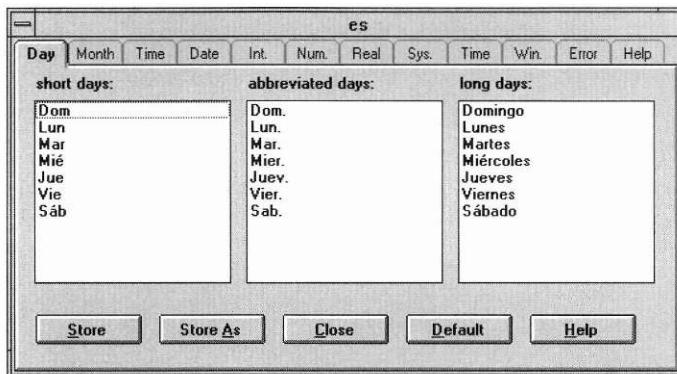
In a similar manner, we could change each of the tabbed items—month, am/pm specifiers, date, integer, number, real, system button, time, win, error, and help. Once we change this information, we can store the window with the same method used by the Image and Help Editors. Let's close this notebook and look at a locale notebook by selecting **Locale I Create.**
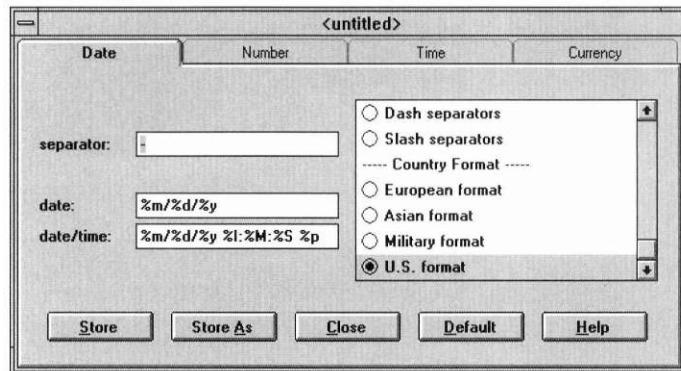
The locale notebook contains information, such as date, number, time, and currency values, for a specific geographic area, which we can save to an environment independent **.DAT** file. Go ahead and browse the Defaults Editor, then invoke the Message Editor by double-clicking on the message icon.

```
┌─────────────────────────────────────────────────────┐
│ ─            Message Editor            ▼ ▲           │
│   File   Message   Help                              │
│                                                       │
│                                                       │
│                                                       │
└─────────────────────────────────────────────────────┘
```
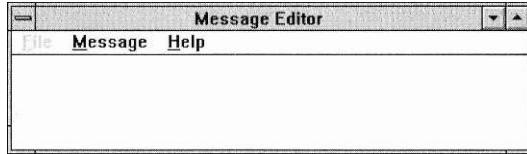
Browsing the
Message Editor

The Message Editor ties program identifiers, which are **const** number declarations, with strings assigned at run time. For example, an application may use the word *Exit.* Rather than programming a specific hard-coded string into the program code, we can associate a logical number with the word *Exit.* In the application we use the raw number with a **ZIL_LANGUAGE::-GetMessage()** function to get, rather than enter, the actual string.

Create a message table by selecting the **Message I Create** option from the pull-down menu.

```
┌─────────────────────────────────────────────────────┐
│ ─                   <untitled>                        │
│  ┌──────────────────────────────────┐   ┌────────┐  │
│  │                                  │   │  Edit  │  │
│  │                                  │   ├────────┤  │
│  │                                  │   │  Add   │  │
│  │                                  │   ├────────┤  │
│  │                                  │   │ Delete │  │
│  │                                  │   ├────────┤  │
│  │                                  │   │Move Up │  │
│  │                                  │   ├────────┤  │
│  │                                  │   │Move Down│ │
│  └──────────────────────────────────┘              │
│   ┌──────┐  ┌─────────┐  ┌───────┐  ┌──────┐       │
│   │ Store│  │Store As..│  │ Close │  │ Help │       │
│   └──────┘  └─────────┘  └───────┘  └──────┘       │
└─────────────────────────────────────────────────────┘
```
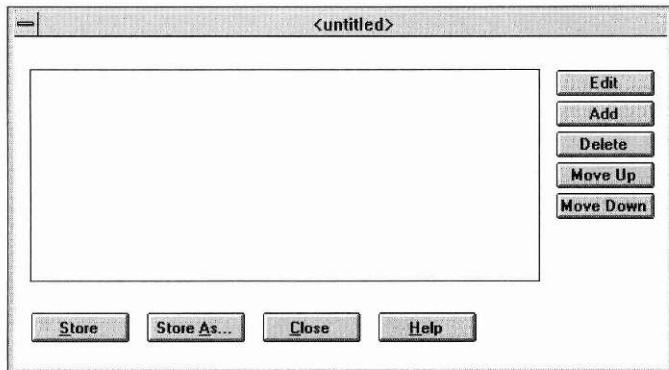
Messages can be added to, deleted from, or moved in our message table by selecting the support buttons located on the right side of the window. Take a few moments to browse the Message Editor.

Browsing the

String Editor   The String Editor provides access to Unicode character sets, allowing you to translate the text of your application into language characters not available from a standard keyboard. You must be in Unicode mode to use the String Editor. Invoke it by double clicking on the **String Editor** image icon, or, in DOS and Windows, by pressing <F12>when positioned on a text field.
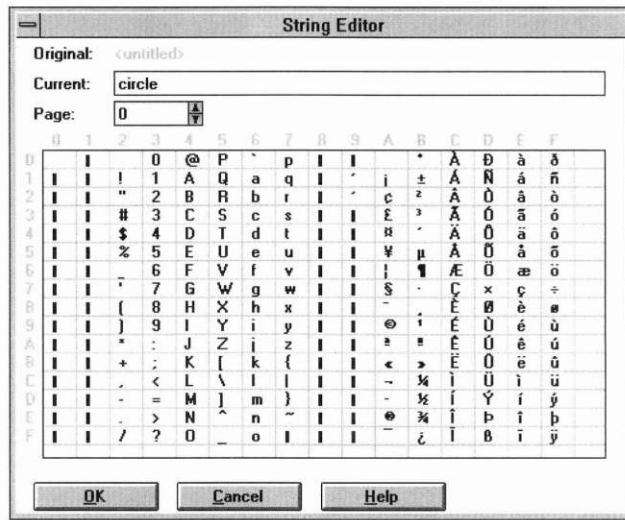
Browse through the character sets by either entering a page number into the **Page** field, or by clicking its spinner arrows with the mouse. Notice that if you select a character with the mouse, that character will appear in the **Current** field. In this manner, new character strings are formed.

Exiting the Designer

To exit the Designer, select the **File I Exit** option. If no changes have been made since the last save, the following dialog box appears:
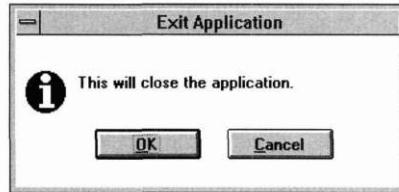
If any changes have been made to the file, the program will ask you to save the **.DAT** files again.

Press the **OK** button in the exit window to exit the application.

And we're done!

section two

OpenZinc Designer

tutorials

Chapter 3

Writing an Application

T

JL his tutorial details compiling and running an application in OpenZinc. If you are not familiar with OpenZinc or OpenZinc Designer, see the appropriate sections of *Getting Started with OpenZinc Programming* or *OpenZinc Designer*.

In the first part of this chapter, we will look at the finished application to see what we will be doing. In the second part of the chapter we begin writing the application by putting together the basic framework required to bring up the Movie Control Window.

Key Concepts

examining **MOVIE**

learning its components

beginning a complex application

Movie Components

In the next few chapters, we'll discuss the steps to build the **MOVIE** application. Each chapter discusses how to implement some major component of **MOVIE,** and the following diagram represents how these components fit together.

FIGURE 1. **The relationship of MOVIE's components**
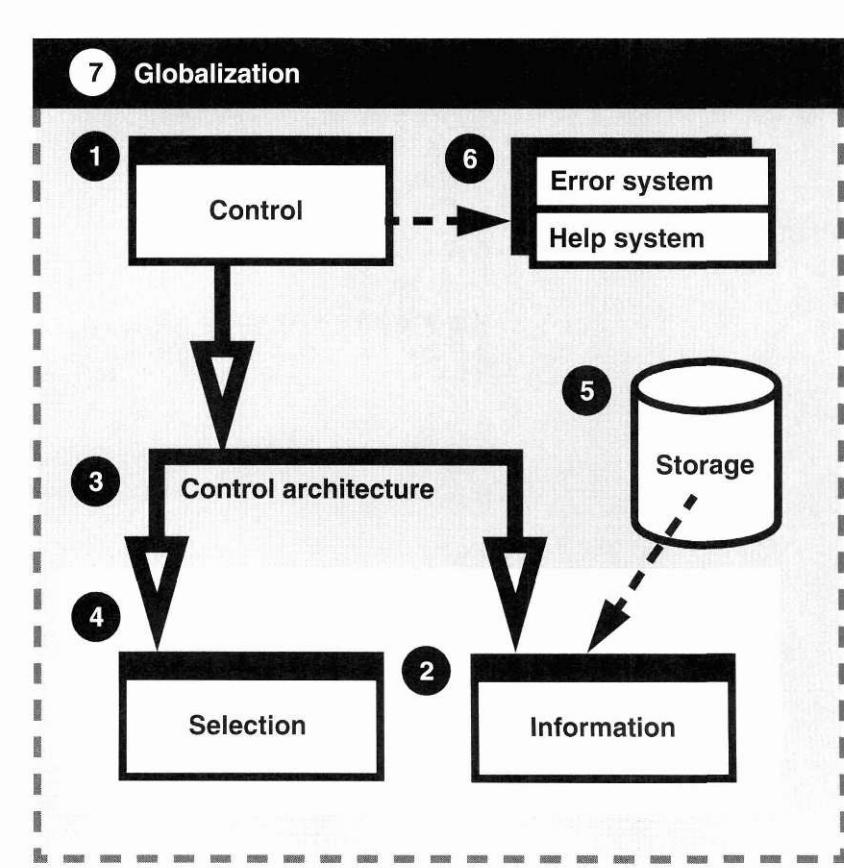
The following table describes what we will do in the chapters of the **MOVIE** tutorial. Each row in the table corresponds to a step in the diagram. And each chapter will contain the piece of the diagram that represents the current step.

TABLE 2. Steps in the MOVIE application

Step	Chapter	App	Description
1	Chapter 3—Writing an Application Movie	**MOVIE1** Control Window	Putting together the framework for the
2	Chapter4—Designing Dialog Windows	**MOVIE2** the	Designing dialog windows, but leaving implementation for later. These windows will display informa- tion about the movie, as well as allow us to select a movie.
3	Chapter 5—Architect- ing the Control	**MOVIE3**	Implementing the architecture to control the other components of the application.
4	Chapter 6—Deriving Support Modules Movie Selection and Movie Informa	**MOVIE4**	Fleshing out the functionality of the tion windows that we left in part 2.
5	Chapter 7—Loading and Storing Data °	**MOVIE5**	Giving **Movie** the ability to store data in a flat file.
6	Chapter 8—Making Movie Robust	**MOVIE6**	Making **Movie** robust by implement- ing error and help systems
7	Chapter 9—Generat- ing an International- ized Application	**MOVIE7**	Globalizing **Movie** to display its interface in French and German as well as English.

You will achieve the best results if you complete the tutorials in order, as each provides a foundation for the next. If you jump in the tutorials in the middle, you will likely find yourself confused.

Each **MOVIE** component in the tutorial contains **.CPP, .HPP,** and **.DAT** files for user-defined objects and for persistent objects. We generate the code for user-defined objects, and OpenZinc Designer generates the code for persistent objects. Since we use two types of storage in **MOVIE,** we use two separate **.DAT** files—**MOVIE.DAT** for data storage, and **P_MOVIE.DAT** for per- sistent object storage.

Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 3. **Components of the MOVIE application**

Type of file	Name of file	Description of file
User-provided files	**MOVIE.CPP**	The main program
	MOVIE.HPP	Class definitions, identifications, and messages
	MOVIE. DAT	User data storage
Designer-generated files	**P_MOVIE.CPP**	Code for tying Designer objects to our program
	P_MOVIE.HPP	Identifications and help contexts
	P_MOVIE.DAT	Persistent object storage

Running the MOVIE application

Now that we've discussed what we're going to do in the next few chapters, let's take a moment to view the application as it appears in its finished state.
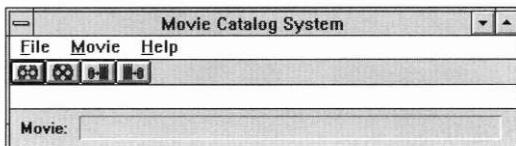
Running the program in its finished state

The source code for **MOVIE,** which contains the program in its finished state, is located in **/OpenZinc/TUTOR/MOVIE. (MOVIE** is exactly equivalent to **MOVIE7,** which we'll write in the last **MOVIE** tutorial.) Go ahead and compile the source code and run the executable. To start the program in command-line environments, type the word

```
movie
```

If you need to enter the full path, type

```
c:\OpenZinc\tutor\movie
```

For systems that require an iconized program, double-click on the icon to bring up the movie application.

You should see the following window on the screen:
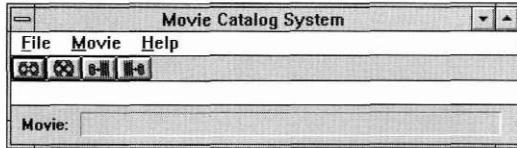
Here's the way the program works. The movie cataloger allows users to generate new movie information records or to view previously created records. The main window contains a title field, a pull-down menu, a button bar, and a status bar. The title contains the name of our application, **Movie Catalog System.** The pull-down menu has three options: **File, Movie,** and **Help.**

File allows us to exit the application or to receive help about the cataloging system. **Movie** allows us to either create, load, store, or delete movie records from the information catalog. **Help** brings up help about the movie catalog system or about the individual options that are available during the application's operation.

The button bar contains four bitmapped buttons. These buttons give us the same options as those in the **Movie** section of the pull-down menu, namely, the ability to do all of the following:

1. Create a new movie record.
2. Delete an existing record.
3. Load a record from the catalog library.
4. Store the current movie record into the movie catalog.

Working with the program

Let's view some of the movies that have previously been entered into the cat-
alog system. Select Movie I Load from the pull-down menu. Then we can
browse through some of the functionality of our program.
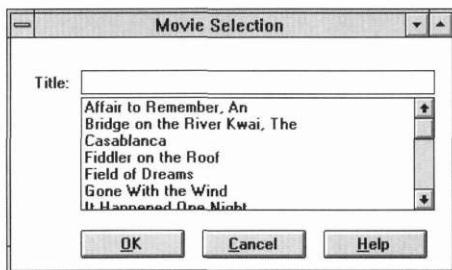
Movie Selection   The Movie Selection dialog lets us view all of the movies currently entered
into the system by presenting documented movies in the vertical list. We can
view information about a movie by clicking on the desired item, then by
pressing OK. For example, if we do this with *Affair to Remember, An,* we
will see the movie information record for this movie.
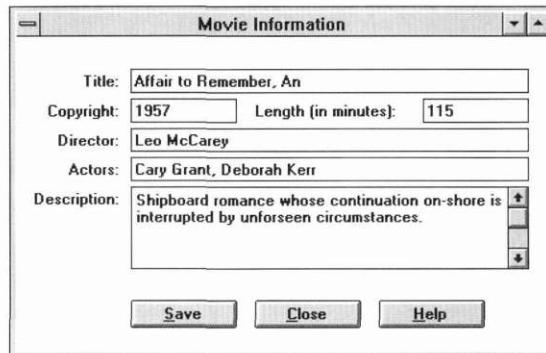
Movie   The Movie Information record presents information about the movie we
Information   have selected, such as the title, copyright date, length of the movie (in min-
utes), name of the movie director, list of the most prominent actors and
actresses, and a short description of the movie. In addition, there are three
buttons at the bottom of the record that allow us to save the record into the

movie catalog, to close the movie record window, or to receive help. Take a moment to view the information associated with *An Affair to Remember*. Once you have finished, close the window by pressing **Close.**

Movie Create

New movie records are created by selecting **Movie I Create** from the pull-down menu.
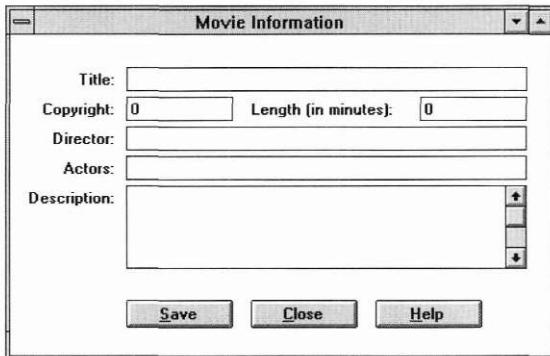
The Movie Create record contains the same fields we saw when the **Load** operation was selected, except that all the fields are blank. We can enter data into the movie information window by moving to each field and then by entering the appropriate data. Let's experiment by adding a new record for the movie *Vertigo*. To do this:

1. Select **Movie I Create** from the pull-down menu.

2. Move to the **Title** field and enter the name

   Vertigo

3. Move to the **Copyright** field and enter the date

   1958

4. Move to the **Length** field and enter the value

   68

5. Move to the **Director** field and enter the name

   Alfred Hitchcock

6. Move to the **Actors** field and enter the names,

   James Stewart

   and

   Kim Novak

Move to the **Description** field and enter the sentence,

```
Haunting dream-like thriller. A great motion picture that
demands multiple viewings.[1]
```
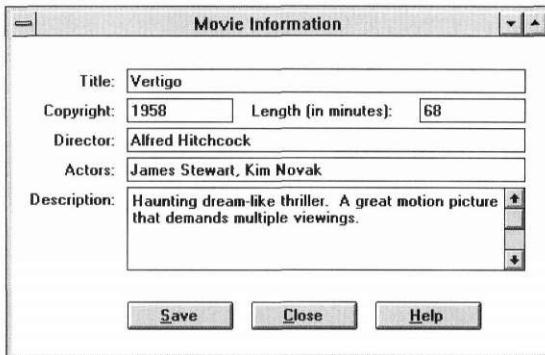

We can save the movie information by selecting the **Save** button from the movie record, by selecting the **Movie I Save** option from the pull-down menu, or by clicking in the button bar while positioned over the save button, which contains a picture of movie reels and an arrow pointing to a cylindrical disc.

Movie Delete

We can delete movies from the catalog by selecting **Movie I Delete** from the pull-down menu, and then by moving to the desired video item, clicking on the selection so that the movie's title appears in the title field, and then selecting **OK.** (We can also double-click the selection to delete it.)
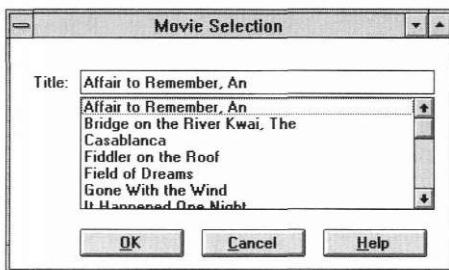
Take a few moments to browse its contents, then exit the application by selecting the **File I Exit** option and by pressing **OK** on the exit application window.

---

[1] .*Leonard Maltin's Movie and Video Guide,* 1994.

Application components

Let's look at the **MOVIE** application and the code and data we need to generate an executable. While we're doing that, we'll examine the architectural concepts of creating applications with OpenZinc Designer.

When we store windows with OpenZinc Designer, three files are generated: **<filenamo.DAT, <filename>.CPP,** and **<filename>.HPP** When we write an application that uses OpenZinc's persistent objects, which are stored to disk and retrieved later, the associated **.CPP** files must be included with our own C++ code files to produce the application. The following diagram shows the relationship of these components and how we combine them to generate an application.

FIGURE 2. Creating a OpenZinc application
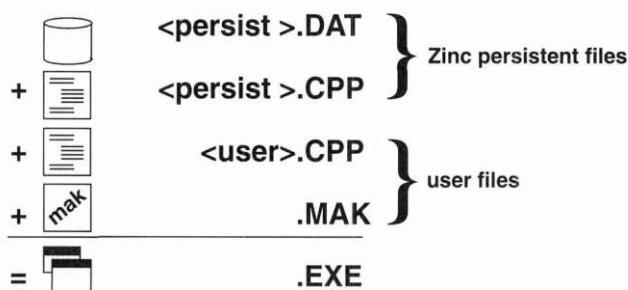
OpenZinc Designer automatically generates a **P_MOVIE.DAT, P_MOVIE.HPP,** and **P_MOVIE.CPP** file. But we will create two additional source code files that will control the internal flow of our application—these are **MOVIE.HPP** and **MOVIE.CPP**

MOVIE.HPP

MOVIE.HPP will contain all the class definitions and messages that will be used in the **MOVIE** application. These definitions include the code for **MOVIE_CONTROL, MOVIE_SELECTION,** and **MOVIE_INFORMATION** classes. The class messages section will contain special identifications—integer values that will signal the program to create, load, store and delete movie records.

```
//      class definitions
Class MOVIE_CONTROL : public UIW_WINDOW
{
public:
```

```
    static ZIL_ICHAR *_pathName;
    static ZIL_ICHAR *_exitName;
    static ZIL_STORAGE *_dataFile;
    static ZIL_STORAGE_READ_ONLY *_intlStorage;
    static ZIL_ICHAR _movieName[64];
    static ZIL_LANGUAGE *_errorMsgTable;
    MOVIE_CONTROL(void);
    ~MOVIE_CONTROL(void);
    virtual EVENT__TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
      ZIL_OBJECTID objectID = ID_DEFAULT);
private:
    static EVENT_TYPE Exit(UI_DISPLAY *, UI_EVENT_MANAGER *,
      UI_WINDOW_MANAGER *windowManager);
    EVENT_TYPE MovieCreate(const UI_EVENT &event);
    EVENT_TYPE MovieDelete(const UI_EVENT &event);
    EVENT_TYPE MovieLoad(const UI_EVENT &event);
    EVENT_TYPE MovieStore(const UI_EVENT Sevent);


class MOVIE_SELECTION : public UIW_WINDOW
{

public:
    static ZIL_ICHAR *_pathName;
    static ZIL_ICHAR *_allObjects;
    MOVIE_SELECTION(ZIL_STORAGE_READ_ONLY *file,
      ZIL_USER_EVENT request);
    virtual EVENT_TYPE Event(const UI_EVENT Sevent);
private:
    ZIL_USER_EVENT request;


class MOVIE_INFORMATION : public UIW_WINDOW
{

public:
    static ZIL_ICHAR *_pathName;
    MOVIE_INFORMATION(ZIL_ICHAR *name = ZIL_NULLP(ZILICHAR));
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void Load(const ZIL_ICHAR *name,
      ZIL_STORAGE_READ_ONLY *file,
      ZIL_STORAGE_OBJECT_READ_ONLY *object =
        ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY),
      UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),
      UIJCTEM *userTable = ZIL_NULLP(UI_ITEM));
    virtual void Store(const ZIL_ICHAR *name,
      ZIL_STORAGE *file = ZIL_NULLP(ZIL_STORAGE),
      ZIL_STORAGE_OBJECT *object = ZIL_NULLP(ZIL_STORAGE_OBJECT),
      UIJETEM *objectTable = ZIL_NULLP(UI_ITEM),
      UIJCTEM *userTable = ZIL_NULLP(UI_ITEM));
```

```
//       class messages
const ZIL_USER_EVENT OPT_HELP= 10000;
const ZIL_USER_EVENT OPT_MOVIE_CREATE= 10001;
const ZIL_USER_EVENT OPT_MOVIE_DELETE= 10002;
const ZIL_USER_EVENT OPT_MOVIE_LOAD= 10003;
const ZIL_USER_EVENT 0PT_M0VIE_ST0RE= 10004;
const ZIL_USER_EVENT OPT_MOVIE_OK= 10005;
const ZIL_USER_EVENT OPT_RESET_SELECTION= 10006;
```

MOVIE.CPP

MOVIE.CPP will contain four sections.

1. The first section will define all of the information and code needed to open and control the application.

2. The second section contains the Movie Control Window that we viewed first on the screen, as well as its annotated code.

3. The third section contains the code necessary to use the movie selection window that we viewed earlier in the tutorial.

4. The fourth and final section contains all the programming code for the movie information window, including code needed to create the record information window, to save the information record, and to load or store the contents of individual movie records.

Our makefile will combine these source modules to produce object code, then generate the movie executable, **MOVIE.EXE** under DOS and Windows. (Makefiles for other platforms are included in the **MOVIE** directory.)

We will look more closely at all of the program code later in this tutorial, but for now, take some time to become familiar with it, as we'll refer to it throughout the entire **MOVIE** series of tutorials.

Writing MOVIE1's Movie Control Window

Now that we've seen the completed application, let's begin writing it. First, let's look at the Movie Control Window's **.DAT** information and **.CPP** code. The goal in this section is to get something—anything—to run!

Components Of MOVIE1

In Figure 1 on page 58, we discussed the components of the **MOVIE** application we'll be working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE1,** the first component of **MOVIE.** The diagram below shows the component we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
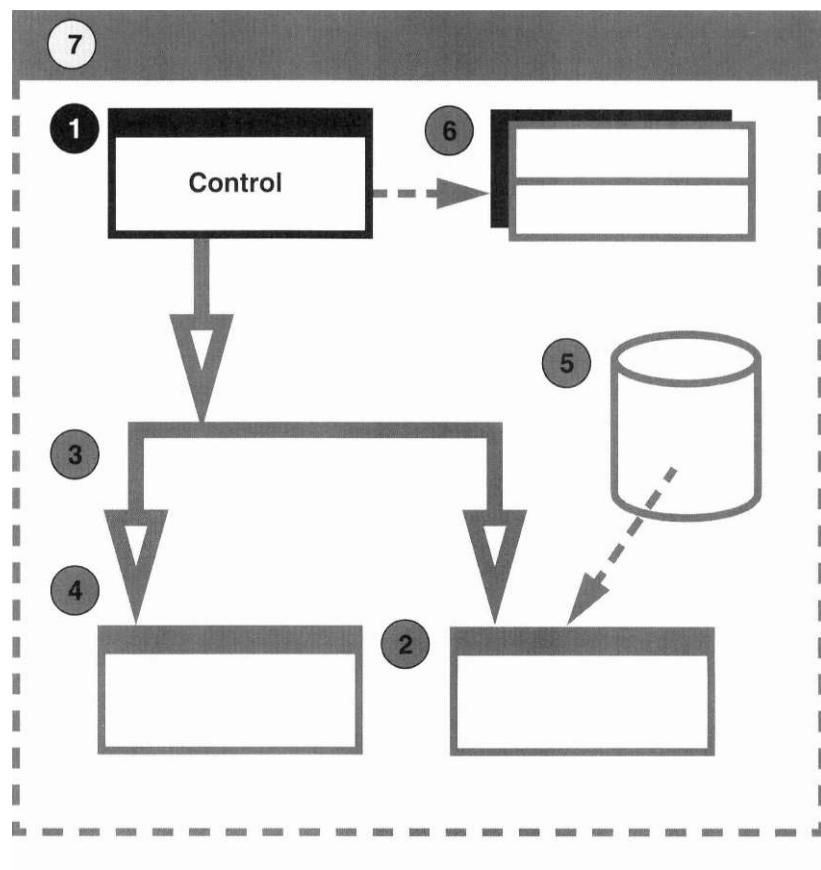
Source files

Below is a list of source files we'll be working with in this part of the tutorial.

TABLE 4. Components of MOVIE1

Type of file	Name of file	Description of file
User-pro-vided files	**MOVIE1.CPP**	The main program
	MOVIE1.HPP	Class definitions, identifications, and messages
Designer-generated files	**P_MOVIE1.CPP**	Code for tying Designer objects to our program
	P_MOVIEl.HPP	Identifications and help contexts
	P_MOVIEl.DAT	Persistent object storage

Creating the window

We will first create a portion of the controlling window with OpenZinc Designer, then write a short program that creates and loads this window into our application.
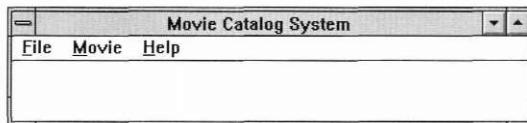
Enter the Designer to create the application Movie Control Window. Create a new window by selecting the **Window I Create** option from the pull-down menu. Next, add a pull-down menu by clicking on the pull-down button, then by placing the pull-down menu inside the edit window. This creates a pull-down menu with one pull-down item.
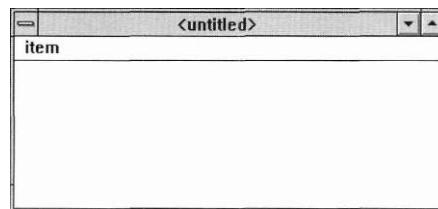
Now let's create the **File, Movie,** and **Help** items on the pull-down menu. We already have one pull-down item on the menu, so we need to add two more:

1. Select the pull-down item from the button bar and click inside the pull-down menu. This places the second pull-down item, which will eventually be the **Movie** item, in the menu.

2. Reactivate the pull-down item option by pressing the right mouse-button.

3. Place the last pull-down item, which will eventually be **Help,** in the pull-down menu.

Change the names of these pull-down items:

1. Open the window's information notebook,

2. Move down to the **Subobjects** notebook page,

3. Double-click the pull-down menu image, **FIELD_1,** from the **Directories** list. This gives us access to the objects attached to the pull-down menu.

4. Edit the information associated with each item in the object's list.

For example, to edit the information associated with the first pull-down item, double-click on FIELD_2 in the object list. Then move to the **Text** field and enter

    &File

(The '&' designates the F character as a hot key for that object. When designating the hot key character, put the '&' just before the appropriate character.)

Change the name of the file item by moving to the **Name** field and entering the word

    FILE_OPTION.

Save these changes by pressing OK.
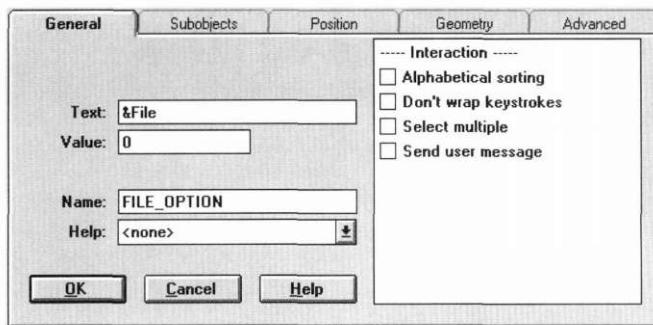
Next, open the information notebook for FIELD_3.

Enter the word

    SMovie

into the text field and

    MOVIE_OPTION

into the name field. Press OK to save the changes.
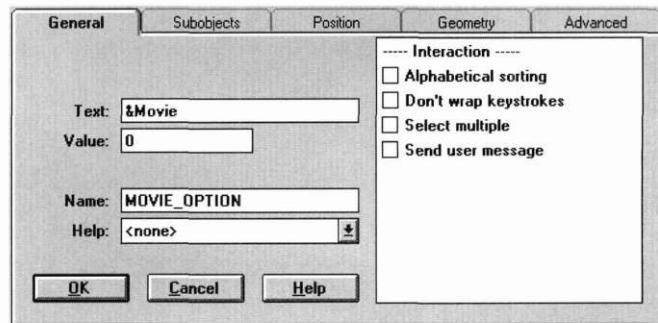
Follow the same procedure with the last menu item by entering the text

    &Help

and name

    HELP_OPTION

into the item's information notebook. Select OK to save the changes to the pull-down item. Then select OK on the window's information notebook to save the changes to the window and exit the information windows.
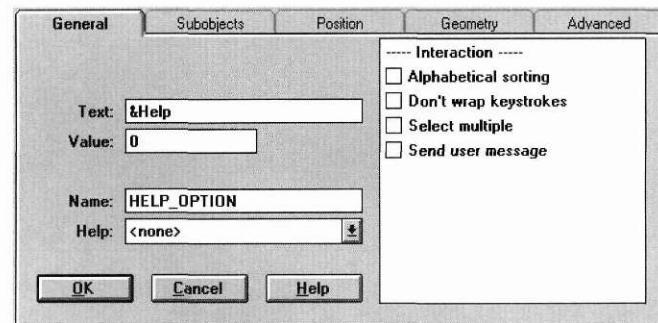
When you return to the untitled edit window, three pull-down items should be visible: **File, Movie,** and **Help.**

—	Movie Catalog System	▼ ▲
File Movie Help		
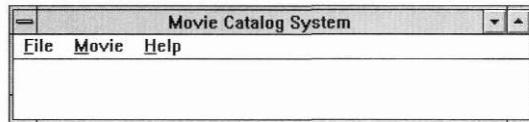
Now let's size the window by moving the mouse cursor over the bottom-right corner of the window so that the mouse is positioned over the window's border, then by clicking and dragging the window's border to the desired size—we want a window width of 60 and height of 4. The window's size can be viewed on the status bar in the **pos** and **size** fields; the position and size of the window are given in cell coordinates.

type:	Window	pos:	26, 8	place object:	
name:	<UNTITLED>	size:	60, 4		
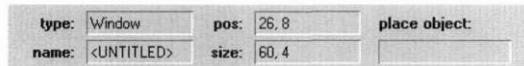
Let's now enter three subobjects into the **File** pull-down item.

1. Double-click on the edit window and then select the **Subobjects** note-book page.
2. Double-click on the pull-down menu in the **Directories** list.
3. Double-click on the pull-down item labeled *OPTION_FILE* in the **Directories** list. This gives us access to the objects attached to the pull-down item.

General	Subobjects	Position	Advanced

Objects:

--- Support Objects ---
--- Normal Objects ---

Directories:

MOVIE
  📇 FIELD_1
    📇 FILE_OPTION

[Edit]
[Add]
[Delete]
[Move Up]
[Move Down]

[OK] [Cancel] [Help] [Pop-up Item ▼]
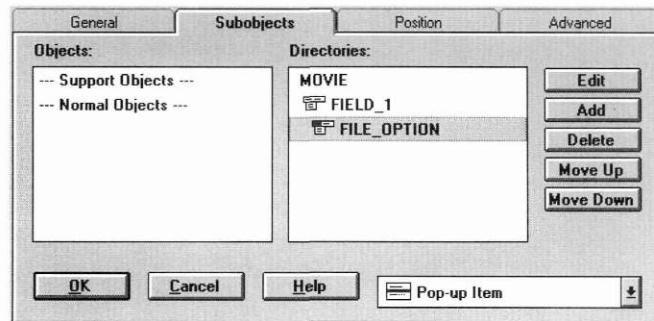
At this point, the directory list shows the edit hierarchy of our pull-down item, and the object list contains a list of all the added **Support** and **Normal** objects, though no object should be visible yet. Add three pop-up items to the objects list by pressing the **Add** button three times. Pressing the **Add**

button adds an instance of the type of object currently selected in the object combo box in the bottom-right corner of the notebook page. Only pop-up items can be added to a pull-down item.
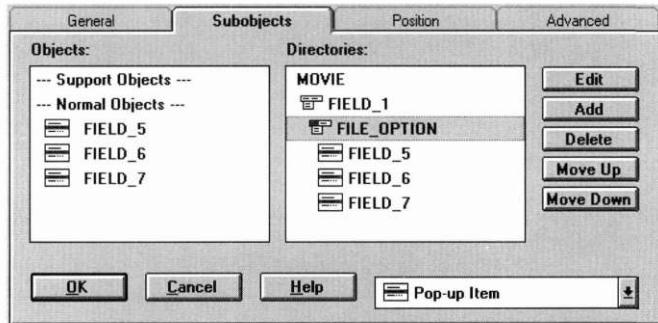
Edit the first item, FIELD_5, by double-clicking on the field item in the objects list, then by entering the word

```
&About...
```

in the text field and

```
ABOUT_OPTION
```

in the name field.
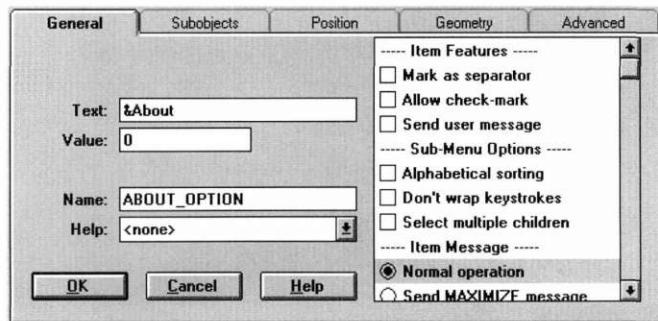
Save these changes, then edit FIELD_6. Identify this item as a menu separator by deleting all the text from the text field and by entering

```
FILE SEP1
```

into the name field. (You could also select the separator option from the options list.)
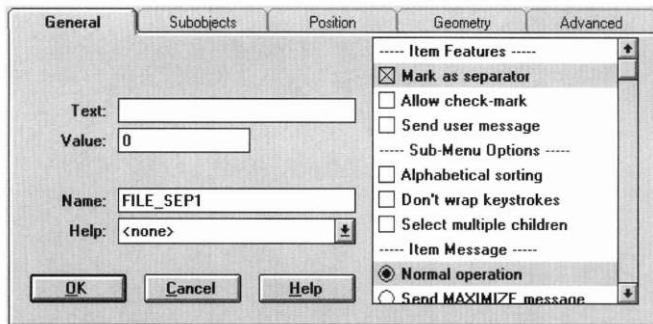
Select FIELD_7 and enter the name

```
E&xit
```

into the text field and

```
EXIT_OPTION
```

into the name field. Save the changes by pressing OK and return to the main edit window. Let's take a moment to enter a title and name with the pull-down menu and with the edit window. Bring up the information notebook and enter the words

```
Movie Catalog System
```

in the title field and

```
MOVIECONTROL
```

in the name field.
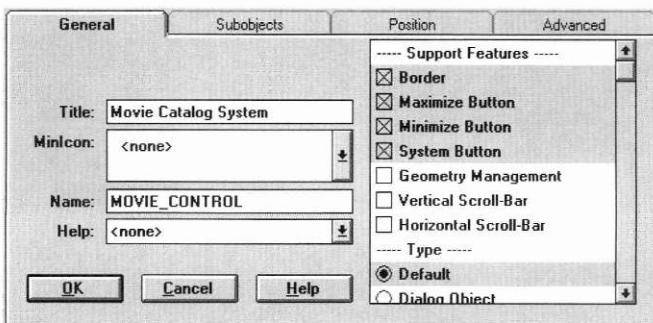
Move to the **Subobjects** notebook page and double-click on the pull-down menu inside the objects list. Change the name of this field to *PULL_DOWN_ MENU.* Press **OK** to save the changes.

We now have a simple window with associated pull-down and pop-up items. Let's save the window by selecting **File I New** and entering the name

```
P_M0VIE1.DAT
```

under the **Filename** prompt and by pressing **OK.**
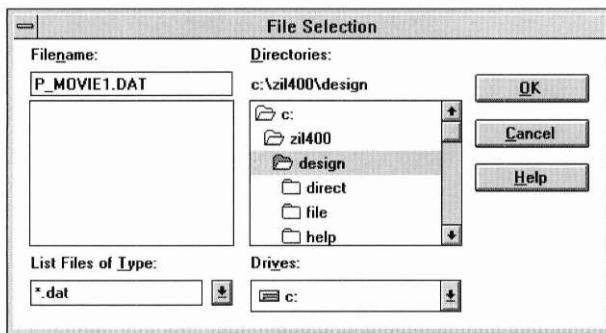
Select **File I Save** from the pull-down menu. The window is saved to the **P_ MOVIEl.DAT** file. Finally, exit the Designer by selecting **File I Exit.**

MOVIE1 .CPP   Let's now look at the code modules used to run our application. First, let's look at the main control loop used to bring up the Movie Control Window and to process user input. The main control loop is located in **MOVIE1.CPP**

```cpp
#include <ui_win.hpp>

int UI_APPLICATION::Main(void)
{
  UI_APPLICATION::LinkMain();
  *windowManager + new UIW_WINDOW("p_moviel.dat~MOVIE_CONTROL");
  UI_APPLICATION::Control();
  return (0);
}
```

The first part of the program contains include files necessary to initialize OpenZinc Application Framework and to launch our movie program. The next part, **UI_APPLICATION::Main()**, loads the program. **LinkMain()** is needed to make sure a **main()** function definition is included in our applica-

tion. The next line is used to retrieve **MOVIE_CONTROL** from the **.DAT** file, and then to add the window into the window manager. The next line, **UI_APPLICATION::Control()**, sends control to OpenZinc Application Framework, which processes all of the user responses, then returns control to our module, once we have finished viewing the main Movie Control Window. The final line returns the value 0 to the main application control loop, which, in turn, returns 0 to the operating system.

So all we need is six lines of code to run our simple application. Take a moment to examine this code and review the purpose behind each line.

P_MOVIE1.CPP   When OpenZinc Designer writes **P_MOVIEl.DAT,** it generates an associated **.CPP,** and **.HPP** file. **P_MOVIEl.CPP** contains three sections:

1. Include directives,

2. pointers to user functions, and

3. pointers to OpenZinc object constructors.

The first section gives directives that load OpenZinc Application Framework objects and the programming information associated with the Movie Control Window we just created.

```
#include <ui_win.hpp>
#define USE_DERIVED_OBJECTS
#include "p_moviel.hpp"
```

The next section contains a user information table. The user-table normally contains information that has been entered by a programmer in the Designer but is not yet used by our application. Thus the table contains only an end-of-table indicator.

```
static UI_ITEM _userTable[] =
{
  { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
```

The final section contains a table with object descriptions. At this point, we have only created a few objects—a border, maximize and minimize buttons, several pop-up items, a pop-up menu, several pull-down items, a pull-down menu, system button, title, and a window. As you browse this table, you will see unique object, name, and constructor references to the objects used by the Movie Control Window. These references tell OpenZinc how to construct and link specific persistent objects into our application.

```
static UI_ITEM _objectTable[] =
{
```

```
                              ID_BORDER, ZIL_VOIDF(UIW_BORDER::New),
                              UIW_BORDER::_className, 0 },
                              ID_MAXIMIZE_BUTTON, ZIL_VOIDF(UIW_MAXIMIZE_BUTTON::New),
                              UIW_MAXIMIZE_BUTTON::_className, 0 },
                              ID_MINIMIZE_BUTTON, ZIL_VOIDF(UIW_MINIMIZE_BUTTON::New),
                              UIW_MINIMIZE_BUTT0N::_className, 0 },
                              ID_P0P_UP_ITEM, ZIL_V0IDF(UIW_P0P_UP_ITEM::New),
                              UIW_P0P_UP_ITEM::_className, 0 },
                              ID_POP_UP_MENU, ZIL_VOIDF(UIW_POP_UP_MENU::New),
                              UIW_POP_UP_MENU::_clas sName, 0 },
                              ID_PULL_D0WN_ITEM, ZIL_VOIDF(UIW_PULL_DOWN_ITEM::New),
                              UIW__PULL_DOWN_ITEM: :_className, 0 },
                              ID_PULL_DOWN_MENU, ZIL_VOIDF(UIW_PULL_DOWN_MENU::New),
                              UIW__PULL_DOWN_MENU: :_className, 0 },
                              ID_SYSTEM_BUTTON, ZIL__VOIDF(UIW_SYSTEM_BUTTON::New),
                              UIW_SYSTEM_BUTTON::_className, 0 },
                              ID_TITLE, ZIL_VOIDF(UIW_TITLE::New),
                              UIWJTITLE::_className, 0 },
                              ID_WINDOW, ZIL_VOIDF(UIW_WINDOW::New),
                              UIW_WINDOW::_className, 0 },
                              ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
```

Makefile

MOVIE1.CPP and **P_MOVIEl.CPP** are both compiled and linked to produce an executable. The makefile you will use to generate this application depends on the type of compiler and operating system you are currently running. In this manual we will show a simplified Borland 4.0 Windows version of our makefile.

```
# MOVIE1 program makefile
#    make -fbtcpp400.mak windows (makes the Windows movie program)
# Be sure to update your TURBOC.CFG and TLINK.CFG files to include
the OpenZinc paths, e.g.:
#    -I.;C:\OpenZinc\INCLUDE;C:\BC4\INCLUDE
#    -L.;C:\OpenZinc\LIB\BTCPP400;C:\BC4\LIB
#        Windows compiler options
WIN_CPP=bcc
WIN_LINK=tlink
WIN_RC=rc
WIN_CPP_OPTS=-c -de -ml -01 -x  RT  Vf -WE -w
WIN_RC_0PTS=-k
WIN_LINK_0PTS=/c /C /Twe /x
WIN_OBJS=cOwl
WIN_LIBS=win_zil mathwl import cwl
 .cpp.obw:
   $(WIN_CPP) $(WIN_CPP_0PTS) -o$*.obw {$< }
#        Windows
windows: wmovie1.exe
```

```
wmoviel.exe:  moviel.obwp_moviel.obw
   $(WIN_LINK)  $(WIN_LINK_0PTS)  @&&!
$(WIN_OBJS) -Hnoviel.obw+p_moviel.obw
$*,   ,$(WIN_LIBS),wmovie.def

   $(WIN_RC)  $(WIN_RC_OPTS)  wmovie.rc  $<
```

Take a moment to view the appropriate makefile information for this application. (Makefiles for each compiler and environment supported by OpenZinc are in this directory.)

Conclusion

The main components of our application are the **P_MOVIEl.DAT** and the **P_M0VIE1.CPP** and **MOVIE1.CPP** files, which are compiled to produce the movie application. When launched, the movie program loads **MOVIE_ CONTROL** from **P_M0VIE1.DAT** and then displays the information.

In the next chapter, we're going to learn how to use OpenZinc Designer to design the interface components of **MOVIE.**

Designing Dialog Windows

in this tutorial, we're going to learn how to design dialog windows for our movie application. We will not do anything with how the program works, only how it looks. (Remember, in the first four tutorial chapters, we discuss **MOVIE**'s architecture. We'll fill in the holes in "Loading and Storing Data" on page 127.)

Key Concepts	creating interfaces
	working with bitmaps

Working with MOVIE2

Components of MOVIE2

In Figure **1** on page 58, we discussed the components of the **MOVIE** application we've been working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE2,** the second component of **MOVIE.** The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
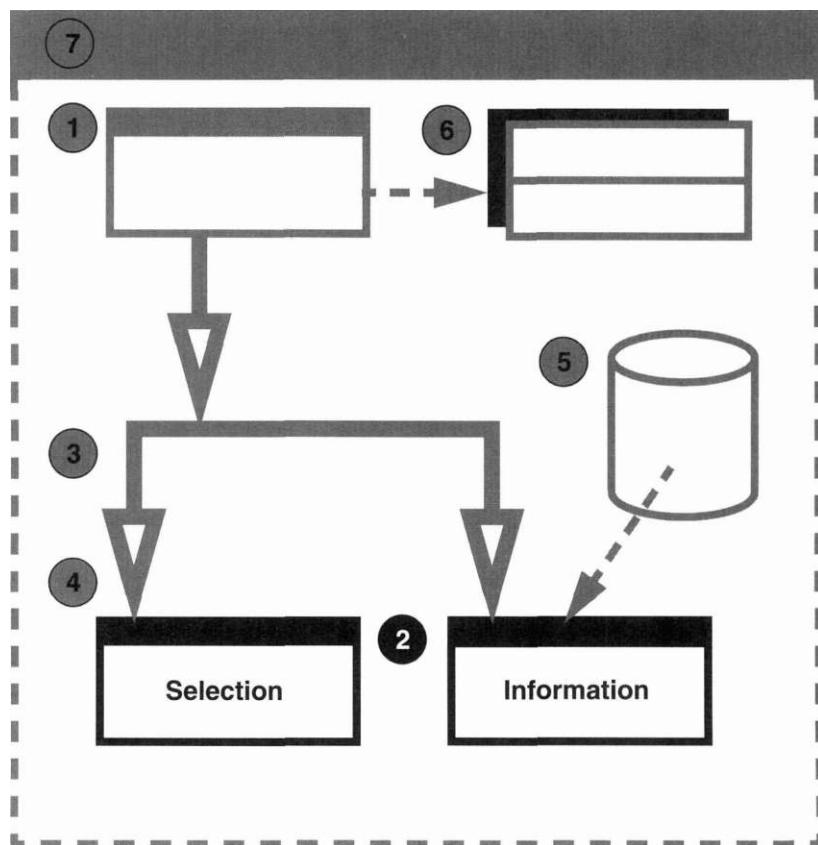
Source files

Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 5. **Components of MOVIE2**

Type of file	Name of file	Description of file
User-pro-vided files	**MOVIE2.CPP**	The main program
	MOVIE2.HPP	Class definitions, identifications, and messages
Designer-generated files	**P_MOVIE2.CPP**	Code for tying Designer objects to our program
	P_MOVIE2.HPP	Identifications and help contexts
	P_MOVIE2.DAT	Persistent object storage

What we'll do

We will first complete the movie and help options located in the pull-down menu. Then we will create the Movie Selection and Movie Information windows, but without implementing any of their functionality.

But first, let's take a moment to discuss how we will learn how to create the remaining items in our Movie Control Window.

For example, under the **File** option we created three items, **About...,** a menu separator, and **Exit.** And we described in the last chapter how we created and edited the information.

We will use the same process for the rest of our application, but we won't describe many of the basic operations, such as how to invoke the information window, how to position the cursor in the fields, and how to select options from the window button bar. If you get confused or lost during the tutorial, take a few minutes to return to the previous tutorial, then examine the steps used to edit the information. To help, we will supply numerous pictures that show you the proper state of the edit objects.

Finishing the Movie Control Window

Let's take a few minutes to finish creating the interface of the main Movie Control Window. First, open OpenZinc Designer. Then reload the Movie Control Window.

1. Select **File I Open.**

2. Enter the filename

    P_M0VIE1.DAT

    and press **OK.**

3. Select **Window** I **Load.**

4. Then select **MOVIE_CONTROL** from the file list.

Now let's add items to the **Movie** options.

1. Double-click on the edit window.

2. Select the subobjects category.

3. Double-click on the pull-down menu item in the **Directories** list,

4. Double-click **MOVIE_OPTION** in the **Directories** list, then

5. Select **Add four** times. This process adds four items to the **Movie** option.
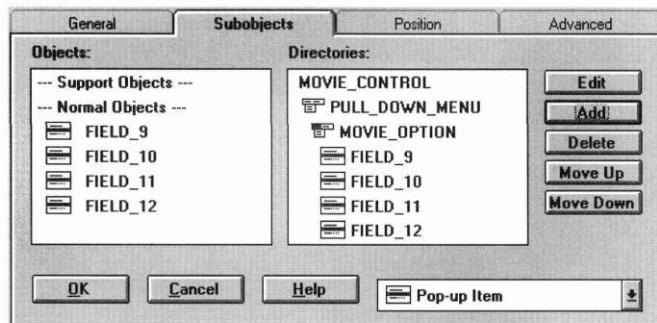
We will associate the **Create, Load, Store,** and **Delete** text with the new items in the object list. Edit the first option by double-clicking on the item. Enter the text

    &Create

and the name

    CREATE_OPTION

into the general information. Close this notebook and then invoke the second item's notebook information. Enter

```
&Load...
```

as the text and

```
LOAD_OPTION
```

as the name. On the third item, enter

```
&Store
```

as the text and

```
STORE_OPTION
```

as the name. In the final object item, enter

```
&Delete...
```

as the text and

```
DELETE_OPTION
```

as the name.
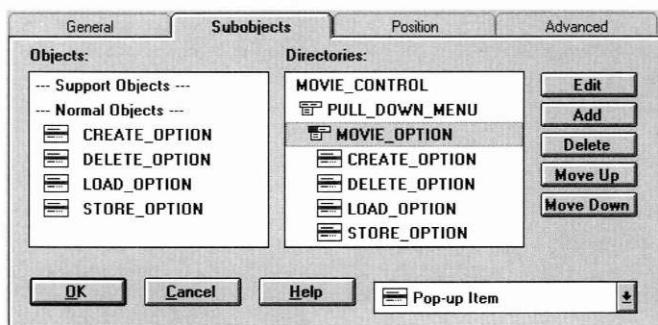
Now move into the **Help** category. First, double-click the pull-down menu in the **Directories** list, then double-click on **HELP_OPTION.**
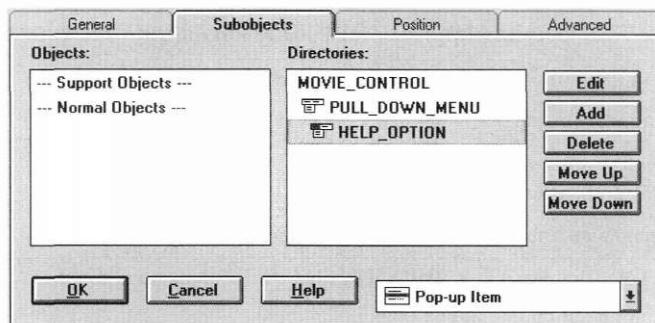
Add four pop-up items to the object's list by pressing the **Add** button four times. The information we'll add to these items is **File, Movie,** a line separator, and **About Movie Catalog.** Edit the items in the same manner as you did the movie options, but enter the text as follows:

1. In the first item enter the text

    ```
    File
    ```

    and name

    ```
    HELP_FILE
    ```

2. In the second item enter the text

    ```
    Movie
    ```

    and the name

    ```
    HELP_MOVIE
    ```

3. In the third item delete all text to create a separator item, then enter

    ```
    HELP_SEPl
    ```

    in the name field.

4. In the last item, enter

    ```
    About Movie Catalog
    ```

    in the text field, and

    ```
    HELP_MOVIE_CATALOG
    ```

    in the name field.

Save all the changes by pressing the OK button until you return to the main edit window.
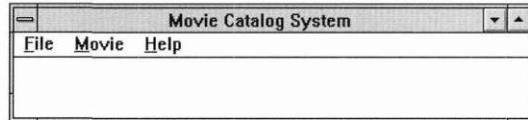
Your main Movie Control Window now has all the category and item information needed for the pull-down menu.

Creating a tool bar

Now that we've created the interface of our application, we can move on to creating a tool bar for the movie application. Create a tool bar by selecting the tool bar item from the Window Editor's button bar, and then by placing the item inside the edit window.
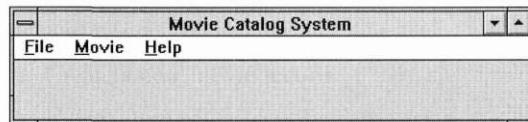
Add four buttons to the tool bar by selecting the button item from the Window Editor's button bar and by creating four buttons inside the movie window's tool bar.
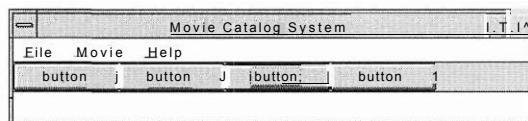
Size each button in the tool bar to a width of 4 and a height of 1. Remember that the size and position are indicated in the Window Editor's status bar.
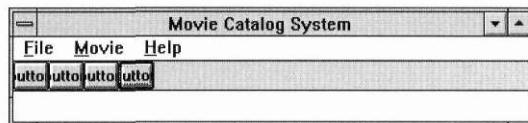
Importing bitmaps

Rather than creating bitmap images, let's use the Image Editor's import feature. Bring up the Image Editor by double-clicking on its icon. Then select **Bitmap I Import** from the Image Editor.
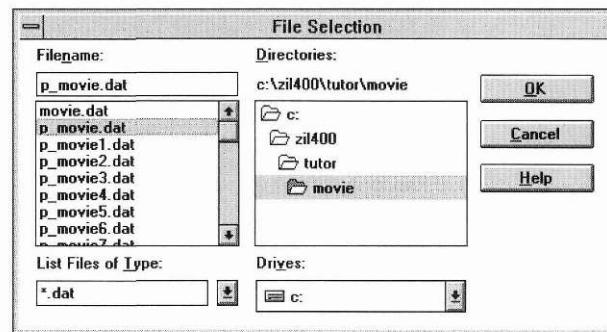
Select the item

```
P MOVIE.DAT
```

from the filename field and press **OK.** Select each item in the object list by moving to the item and clicking the left-mouse button while positioned over the item. (The selected item will appear shaded in the list when it is selected.)
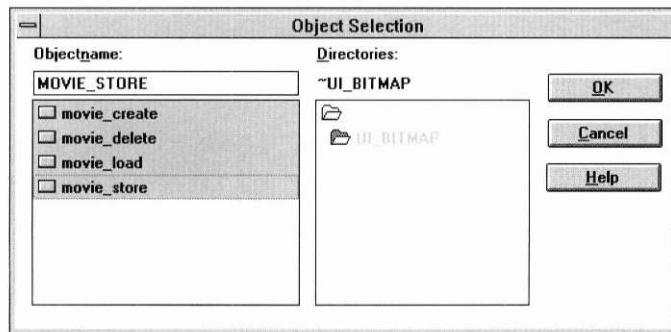
The bitmapped images are imported once the **OK** button is selected. (When the Image Editor imports a bitmap image, the window's main status field will show the image that is being imported. Once the import process is complete, the storage selection window is removed from the display and control returns to the Image Editor.) We can now view the bitmap images by selecting **Bitmap I Load.**
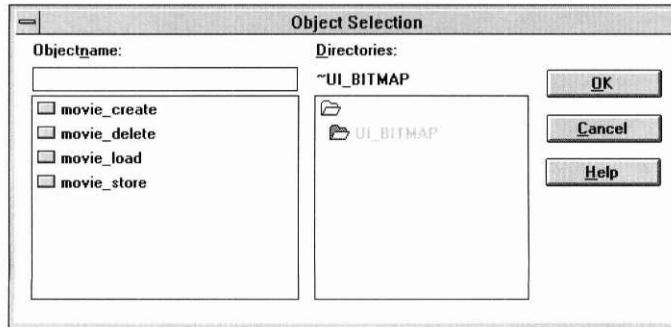
The load procedure tells us that four bitmap images are available: **movie_create, movie_delete, movie_load,** and **movie_store.** Return to the Window Editor by removing the Load Window and then by minimizing the Image Editor's Movie Control Window.

Editing the tool bar buttons

We can now edit the information associated with our four tool bar buttons, The four buttons will be used to represent the create, delete, load, and store options, respectively. Invoke the information notebook for the first button in the tool bar. This will be designated **CREATE_BUTTON.**

Remove the default text of this button. Move down to the image field and select the **movie_create** bitmap image from the combo box's pull-down list. Enter the name of the field as **CREATEJBUTTON.** Choose the **Auto sized** feature from the button's option list. Save the changes by pressing **OK.**
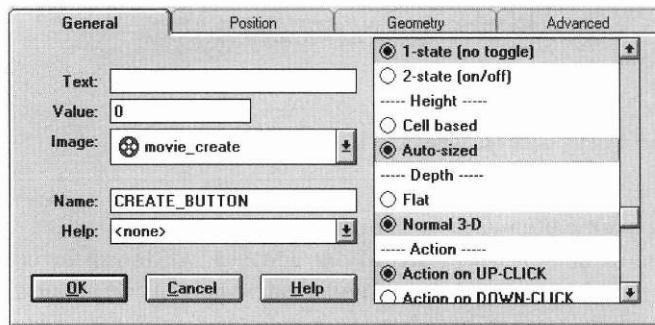
Follow the same process for the next button by deleting the text, then by selecting **movie_delete** as the button image, then by entering the name

```
DELETE BUTTON
```
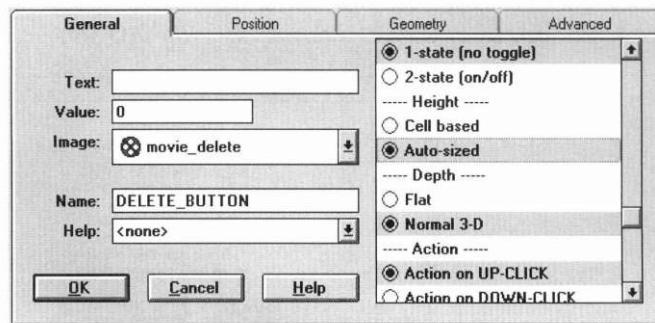
Edit the next button so that it contains the **movie_load** image, named **LOAD_BUTTON.**
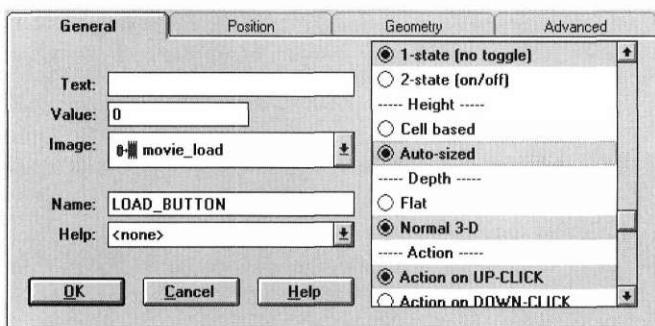
The final button should contain the image **movie_store** and the name **STORE BUTTON.**
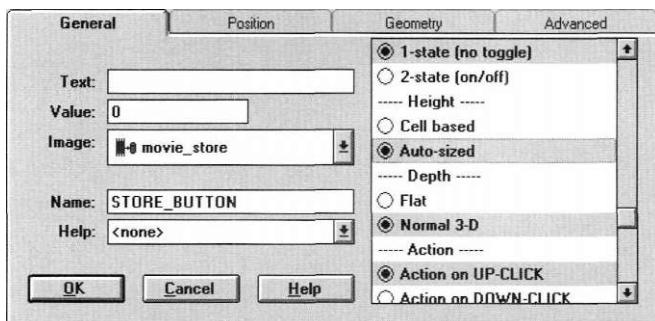
We now have all the presentation features for the Movie Catalog window. Save these changes by selecting **File I Save** from the Window Editor.

Browsing the window

Let's see what the Movie Catalog window looks like by selecting **Window I Test.** At this point, the movie catalog system should appear on the screen.
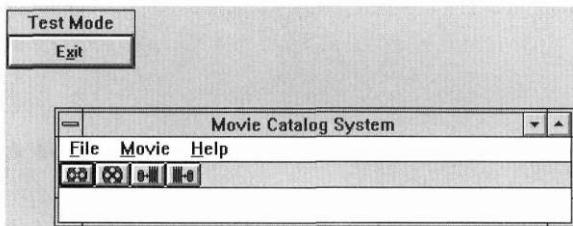
Take a minute to browse the window information. Under the **File** option you should see the **About...** and **Exit** items as well as a menu separator.
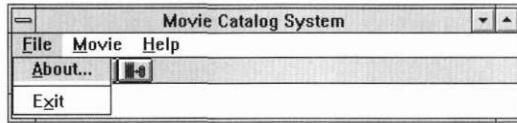
Under the **Movie** option, you should see the options **Create, Load..., Store,** and **Delete....**
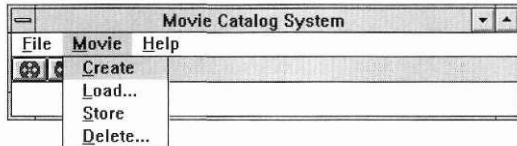
Under the **Help** option you should see the options: **File. Movie,** a menu separator, and **About Movie Catalog.**
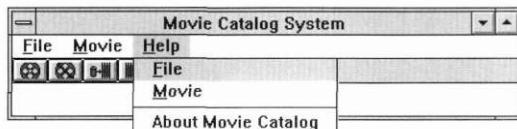
If we exited the Window Editor now, we could compile and run the same code produced in the last section; it would present the same information we just viewed in test mode. Recall that we invoked the Movie Information window by reading a **UIW_WINDOW** object. We have done nothing to modify the operation of our application—just its interface.

Creating the Movie Selection window

Now that we've created the Movie Catalog window, let's create the Movie Selection window. We just saved the Movie Catalog window, so clear it from the screen by selecting **Window I Clear.** Then create another window by selecting **Window I Create.** Remember, the Movie Selection window will be used to select a movie from the catalog system. Therefore, we need a win-

dow that contains the title of the movie we want to select, a list that tells us the available movie options, and three buttons that allow us to select a movie, cancel out of the selection process, or to obtain help about the Movie Selection window.
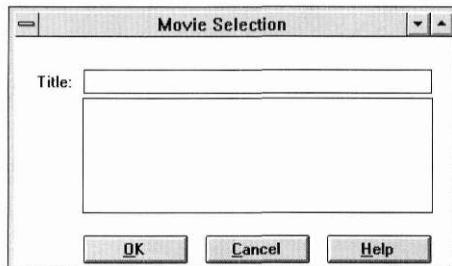
Let's first change the title and name associated with the window that we just created. In the general window notebook enter the title **Movie Selection** and the name **MOVIE_SELECTION.**
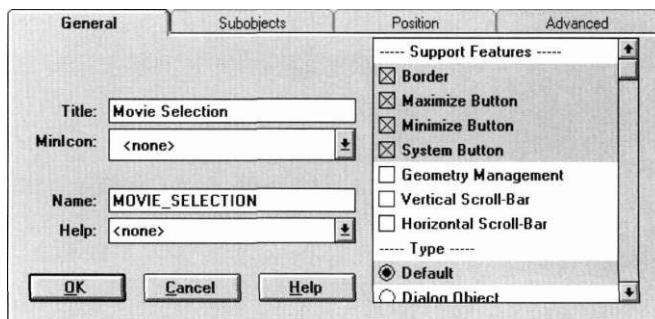
Prepare the window by creating a prompt field for the title, a string field for the title, a vertical list field for the movie selections, and three buttons for **OK, Cancel,** and **Help.** Your edit window should look like this:
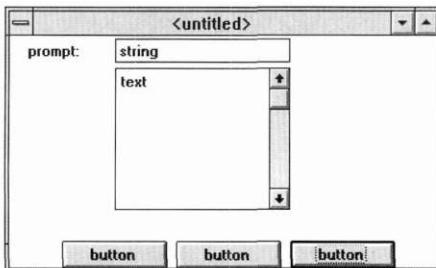
Change this window to be a dialog window by invoking the window's information notebook, then by selecting **Dialog object** as the default from the options list (located at the right-side of the notebook page.)

Change the text associated with the prompt to

```
Title:
```

Now, set the name of the prompt to be

```
SELECT_PTITLE
```

(The **P** represents "prompt.") Next, edit the string field. Remove the default text by pressing the <delete> key, then change the length of data to 64, and change the name of the field to

```
FLD_SELECT_TITLE
```

Change the name of the vertical list to

```
FLD_SELECT_SELECTION
```

Change the text of the first button to

```
&OK
```

and the name of the button to

```
SELECT_OK
```

Change the text of the second button to

```
&CANCEL
```

and the name to

```
SELECT_CANCEL
```

Change the text of the final button to

```
&HELP
```

and the name to

```
SELECT HELP
```
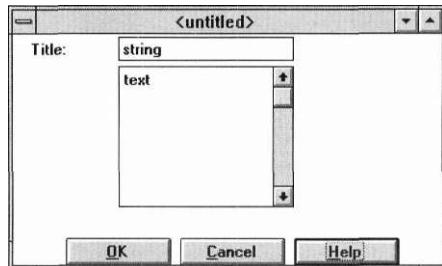

Now let's modify the position and size of all the fields in the window. First, size the Movie Selection window to be width 52 and height 9. Next, position the title prompt to be at position 2,1. Refer to the **size:** and **pos:,** fields on the status bar to validate the position and size of the window and its window objects. Position the title string to be at position 8,1 with a width of 40. Position the vertical list directly under the title field at 8,2 and size it to be of width 40 and height 3. Position the **OK** button to be at position 8,6 with the size of 12,1. Position the **Cancel** button to be at position 22,6 and size 12,1. Move the **Help** button to position 36,6 with the size of 12,1. Your Movie Selection window now looks like this:
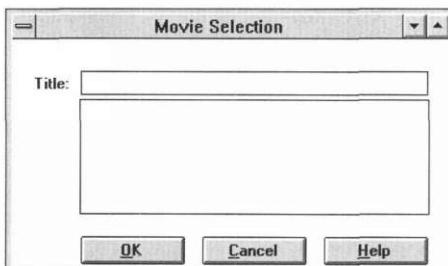
Save the information for this window by selecting **Window I Store** from the Window Editor's pull-down menu, then remove the window from the screen.

Creating the Movie Information window

Now that we've finished the Movie Selection window, let's create the Movie Information window.
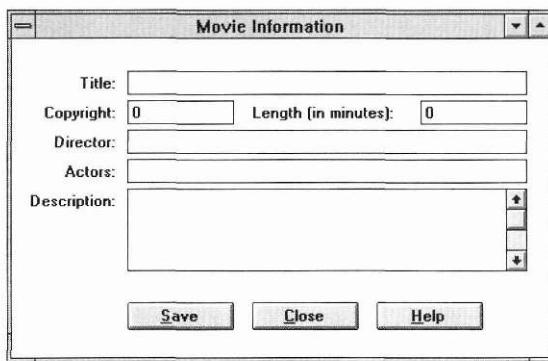
We'll start out by describing how to create the prompt and field information for each item in the movie information record.

1. The **Title Information** field has a prompt with the text *title* and the name *INFO_PTITLE* right justified at the location (5,1). The size of the title should be (7,1).

2. The **Title Information** is a string field located at position (13,1) with size (45,1). It has blank text a length of 128 and a name *FLD_IN FO_TITLE.*

3. The **Copyright** prompt is located at position **(1**,2) and has the text *Copyright:* and name *INFO_PCOPYRIGHT.*

4. The **Copyright** field is an integer field located at position (13,2) with the size (12,1), value 0, and name *FLD_INF0_C0PYR1GHT.*

5. The **Length** prompt is located at position (27,2) contains the text *length (in minutes):,* and name *FLD_PLENGTH.*

6. The **Length** field is an integer field located at (46,2) with size (12,1), and has the value 0, and name *FLD_INFO_LENGTH.*

7. The **Director** field has a prompt located at position 2,3 with size (10,1). The text is *Director:,* and the name is *INFO_PDIRECTOR.*

8. The **Director** field is a string field located at position (13,3) with size (45,1). The text is blank length 128, with the name *FLD_INFO__DIRECTOR.*

9. The **Actor** prompt has the text *Actors:* and name *INFO_PACTORS.*

10. The **Actor** field is a string field located at position **(13,4)** with size **(45,1).** It has blanked-out text and a maximum length of **128** and the name *FLD_INFO_PACTORS.*

11. The **Description** prompt has the text *description:* and name *INFO_PDESCRIPTION.*

12. The **Description** field is of type text and is located at position **(13,5)** with a size of **(45,3).** It has no text, but it has a maximum length of **1024** and identifying name *FLD_INFO_DESCR1PTION.*

13. The **Save** button is located at position **(13,8)** with size **12,1** and has the text *&Save,* and the name *INFO_SAVE.*

14. The **Close** button is located at position **(27,8)** with size **(12,1**) and has the text *&Close* with name *INFO_CLOSE.*

15. The **Help** button is located at position **(41,8)** with size **(12,1)** and has the text *&HELP* with name *INFO_HELP.*

We have now completed the creation of the Movie Information record. Save this information by selecting **File I Save** from the Window Editor's pull-down menu, then exit the Designer.

Updating the source code

We have now created and saved the interfaces of our three windows: **MOVIE_CONTROL, MOVIE_SELECTION,** and **MOVIE_INFOR-MATION.** Let's go back into our source code and change it so that the program can display all three windows to the screen. We do this by adding two additional window creation lines—one for the Movie Selection window, and one for the Movie Information window.

```
// Include the appropriate directives.
#include <ui_win.hpp>

int UI_APPLICATION::Main(void)
{
  UI_APPLICATION::LinkMain();
  // Add the Movie Control Windows and process user responses.
  *windowManager
    + new UIW_WINDOW("p_movie2.dat~MOVIE_CONTROL")
    + new UIW_WINDOW("p_movie2.dat~MOVIE_SELECTION")
    + new UIW_WINDOW("p_movie2.dat~MOVIE_INFORMATION");
  UI_APPLICATTON::Control();
  return (0);
}
```

Now recompile and run the application. Your screen should have all three windows that we created in OpenZinc Designer.
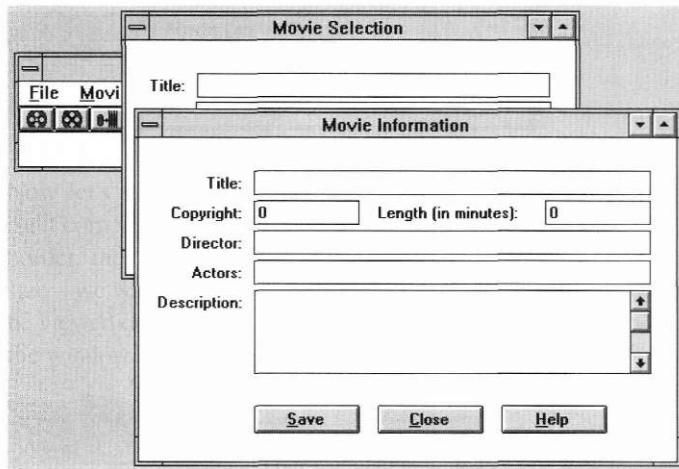
You can move through the windows and browse the information, but no control has been built into the application; we will do this in the next tutorial. To exit the application, remove all three windows from the display.

Conclusion

In this chapter, we created **MOVIE**'s interface. In the next chapter, we will create the menu options that we will connect later to member functions.

Architecting the Control

in this chapter, we're going to examine how to connect menu options to the functions that perform some action. While we do this, we'll examine the architecture needed to generate messages that cause things to happen in the MOVIE program.

Key Concepts

responding to events

thinking about architecture

Working with MOVIE3

Components Of MOVIE3

In Figure **1** on page 58, we discussed the components of the **MOVIE** application we've been working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE3,** the third component of **MOVIE.** The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
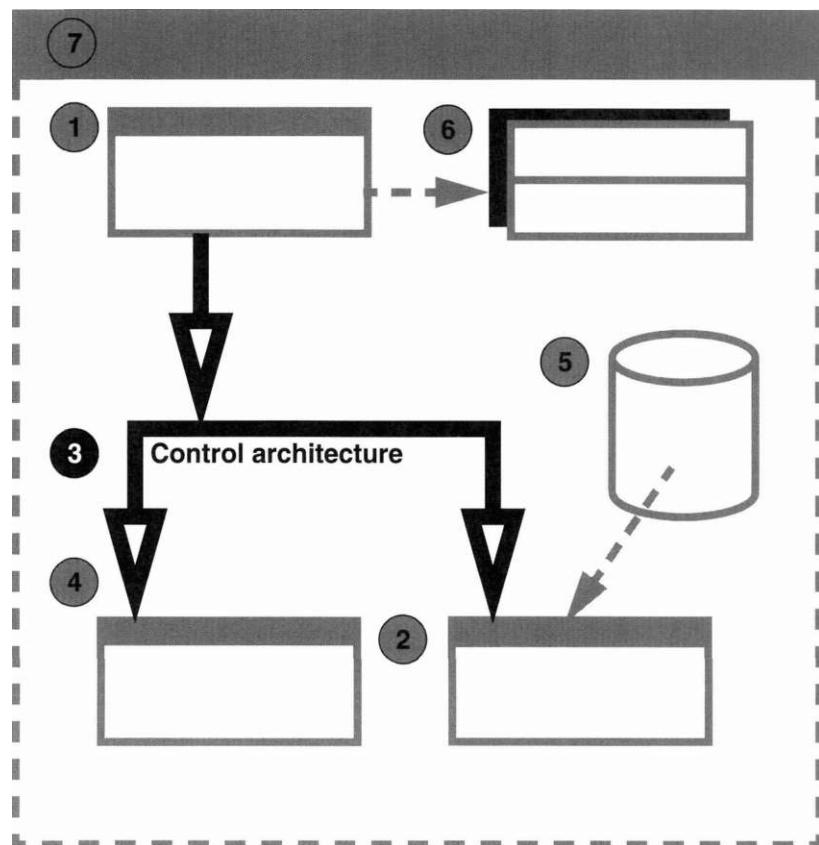
Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 6. Components of MOVIE3

Type of file	Name of file	Description of file
User-pro-vided files	**MOVIE3.CPP**	The main program
	MOVIE3.HPP	Class definitions, identi-fications, and messages
Designer-generated files	**P_MOVIE3.CPP**	Code for tying Designer objects to our program
	P_MOVIE3.HPP	Identifications and help contexts
	P_MOVIE3.DAT	Persistent object storage

The Movie Control window

The first thing we'll look at in this tutorial is the Movie Control window. Its definition in **MOVIE3.HPP** contains several interesting things.

```
class MOVIE_CONTROL : public UIW_WINDOW
{
public:
  MOVIE_CONTROL(void);
  virtual EVENT_TYPE Event(const UI_EVENT &event);
private:
  EVENT_TYPE MovieCreate(const UI_EVENT Sevent);
  EVENT_TYPE MovieDelete(const UI_EVENT Sevent);
  EVENT_TYPE MovieLoad(const UI_EVENT Sevent);
  EVENT_TYPE MovieStore(const UI_EVENT &event);
>;
```

First, **MOVIE_CONTROL** derives from **UIW_WINDOW,** allowing **MOVIE_CONTROL** to inherit all the properties of the **UIW_WINDOW** class. Also, the Movie Control window's overloaded constructor will call the base class **UIW_WINDOW** constructor by default.

In addition to an overloaded constructor, **MOVIE_CONTROL** also has an **Event()** function. By defining **Event()** for **MOVIE_CONTROL,** it will receive messages before **UIW_WINDOW MOVIE_CONTROL** can process the events itself, or dispatch the events to a child process, such as the Movie Information window or the control selection. Or it can pass the information to **UIW_WINDOW.**

MOVIE_CONTROL has four private member functions: **MovieCreate(), MovieDelete(), MovieLoad(**), and **MovieStore(**). We'll discuss these later in the tutorial.

The constructor

Let's examine the **MOVIE_CONTROL** constructor more closely.

```
MOVIE_CONTROL::M0VIE_C0NTR0L(void) :
  UIW_WINDOW("MOVIE_CONTROL", defaultStorage)
{
  // Center the window at the top of the screen.
  windowManager->Center(this);
  relative.bottom = relative.Height() - 1;
  relative.top = 0;
}
```

It takes no arguments, but it loads the window from a **.DAT** file by calling the base **UIW_WINDOW** class with the window name *MOVIEjCONTROL*, and a pointer to the default persistent storage, which we will examine in a moment. Next, the constructor centers the window in the display by calling *windowManager->***Center()**, and moves it to the top of the screen by resetting its *relative.bottom* and *relative.top* values.

Let's examine **Main()** to see what the definition of default storage is.

```
int UI_APPLICATION::Main(void)
{
  // Fix linkers that don't look for main in the .LIBs.
  UI_APPLICATION::LinkMain();
  // Provide a general storage module.
  UI_WINDOW_OBJECT::defaultStorage =
    new ZIL_STORAGE_READ_ONLY("p_movie3.dat");
```

The **UI_WINDOW_OBJECT** base class has a member variable called *defaultStorage.* This variable is a global storage object that contains our **.DAT** file, which is **P_MOVIE3.DAT** The default storage is initialized by calling **new ZIL_STORAGE_READ_ONLY** with the argument **P_MOVIE3.DAT.** This allows us not only to retrieve the Movie Control window from the **.DAT** file, but the Movie Information and Movie Selection

windows as well. Next, notice that the application creates a new **MOVIE_CONTROL** window instead of a new **UIW_WINDOW** window. Then it attaches the Movie Control window to the Window Manager.

Finally, the application sets the Window Manager *screenID* to the Movie Control window's *screenID.* This tells the Window Manager that even if there are several windows on the display, the application should exit when the Movie Control window is removed from the display.

In the previous tutorial, we looked at all three windows on the display at once; to leave the application, we had to remove each window from the display until no OpenZinc windows remained. With this code, removal of just the Movie Control window closes the application—even if we have several windows on the screen.

Event handling

The next important aspect of the Movie Control window is event handling. But to cause the Movie Control window to respond to events, we must decide what it must do when it receives those events. Here's what the Movie Control window must do:

1. Create, delete, load, or store movie records. (We've seen how this will work with the pull-down menu and the bitmapped buttons.)

2. Exit when finished receiving user input.

The most efficient way to cause the Movie Control window to do these things is to introduce six new messages into our system:

- *OPT_HELP*
- *OPT_MO VIE_CREA TE*
- *OPT_MO   VIE_DELETE*
- *OPT_MOVIE_LOAD*
- *OPT_MOV1E__ST0RE,*   and
- a system message, *S_CLOSE.*

OpenZinc reserves the values 10,000 and above and -10,000 and below for user events. In our case we will assign values of 10,000 and above to create, delete, load, and store messages. And we will also provide a special message for help. The values for messages are :

```
const ZIL_USER_EVENT OPT_HELP     = 10000;
const ZIL_USER_EVENT OPT_MOVIE_CREATE= 10001;
const ZIL_USER_EVENT OPT_MOVIE_DELETE= 10002;
const ZIL_USER_EVENT OPT_MOVIE_LOAD  = 10003;
const ZIL USER EVENT OPT MOVIE STORE = 10004;
```

When our Movie Control window receives these messages, it will call differ-
ent member functions. As we already discussed, **MOVIE3.HPP** defines four
private member functions for **MOVIE_CONTROL,** which are
**MovieCreate(), MovieDelete(), MovieLoad(**), and **MovieStore(**). To
intercept these messages, we overload **MOVIE_CONTROL**'s **Event()**
function for **MOVIE_CONTROL,** and then check for one of these mes-
sages. (We'll discuss help messages in the next chapter.)

```
EVENT_TYPE MOVIE_CONTROL::Event(const UI_EVENT &event)
{
  // Check for special requests.
  EVENT_TYPE ccode = event.type;
  if (ccode == OPT_MOVIE_CREATE)
    ccode = MovieCreate(event);
  else if (ccode == OPT_MOVIE_DELETE)
    ccode = MovieDelete(event);
  else if (ccode == OPT_MOVIE_LOAD)
    ccode = MovieLoad(event);
  else if (ccode == OPT_MOVIE_STORE)
    ccode = MovieStore(event);
  else
    ccode = UIW_WINDOW::Event(event);
  return (ccode);
}
```

If the Movie Control window receives one of these messages, it calls the
member function for that message.

In this tutorial we won't worry about the controlling movie information and
movie selection. So in the **MovieCreate(), MovieDelete(**), and
MovieLoad(**) members, all that we will do is add a new **UIW_WINDOW.

```
EVENT_TYPE MOVIE_CONTROL::MovieCreate(const UI_EVENT &event)
{
  *windowManager + new UIW_WINDOW("MOVIE_INFORMATION",
    defaultStorage);
  return (event.type);
}
```

In the **MovieStore(**) member, we will simply return without performing any
action.

```
EVENT_TYPE MOVIE_CONTROL::MovieStore(const UI_EVENT &event)
{
  return (event.type);
}
```

We'll discuss deriving movie information and movie selection in the next
MOVIE tutorial.

Now that our source file has an underlying architecture with messages, we'll connect the Movie Control window to these messages.

Connecting messages

In this section, we'll connect the messages to the Movie Control window and menu options. Invoke the Designer and load the Movie Control window.
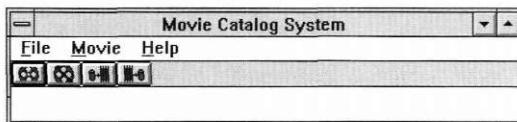
Connecting messages to the pull-down menu

Connect the messages to the pull-down menu, so that the program knows to send the messages when the user selects those menu options. Invoke the window information notebook by double-clicking on the window. Then, in the **Subobjects** page, double-click on the pull-down menu in the **Directories** list. Then double-click on the MOVIE_OPTION object in the **Directories** list. Now the screen should look like this:
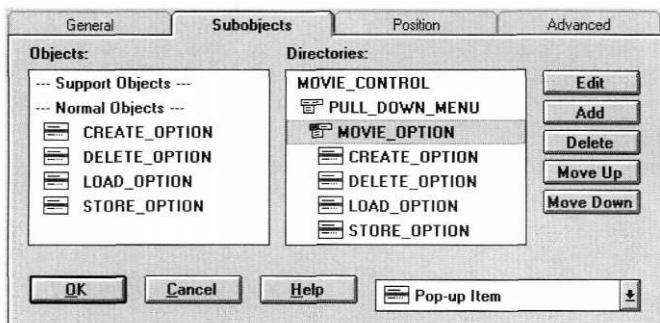
Connecting messages to the Movie Control window

Now connect messages to the Movie Control window. Editing each of the movie options, enter a value, and then set the option so the value passes through the system when the option is selected. Change the *CREATE_OPTION* object first. Invoke the create option information notebook and change the value on the general page to

```
10001
```

and set the *Send user message* flag from **Item features.**
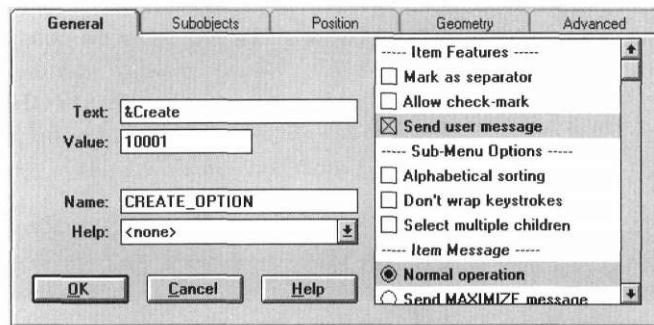
Do the same thing with *DELETE_OPTION* by entering the value

```
10002
```

and setting the *Send user message* flag.
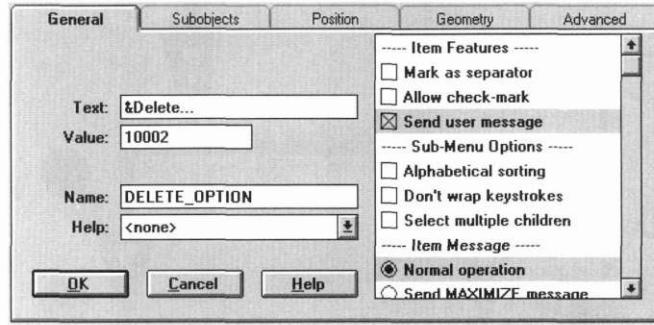
Enter the value

```
10003
```

for *LOAD_OPTION,* then set the *Send user message* flag.
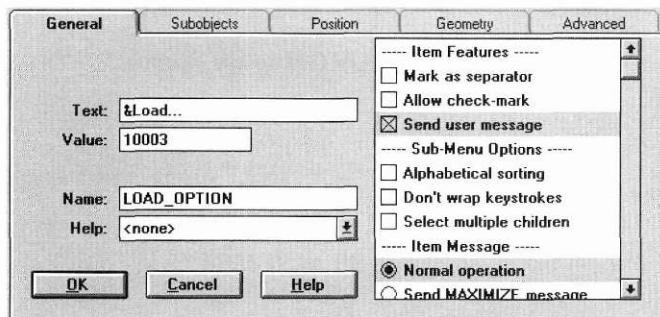
Finally, set the value

```
10004
```

for *STORE_OPTION* and set the *Send user message* flag.
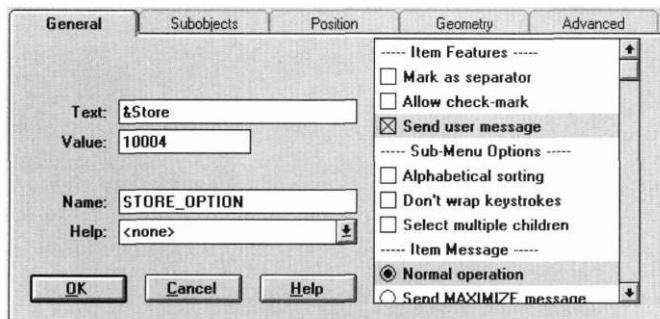
Save the changes by pressing **OK.**

Closing a window   OpenZinc automatically includes *S_CLOSE.* When we want to close a window or terminate an application, we can send the *S_CLOSE* message through the system. If the current front window receives *S_CLOSE,* OpenZinc will remove the window from the display or terminate the application.

Let's associate S_CLOSE with **File I Exit** in our pull-down menu. Invoke the information notebook and move to **File I Edit.** In the information notebook, enter the value

```
-11
```

and set *Send user message* from the item features.
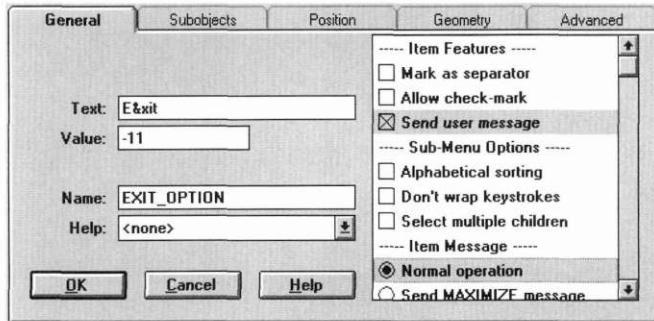
The value -11 corresponds to the S_CLOSE message. To view a list of logical and system events that we can connect to our user values, simply move to the Message Editor and invoke the **Help I Logical events** or **Help I System events** menu item from the pull-down menu.
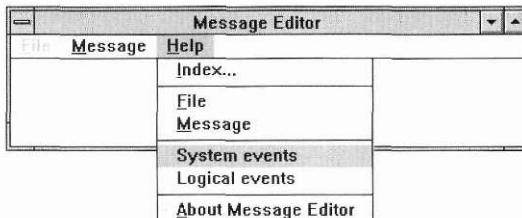
These help options list all supported OpenZinc messages and their values. When we set the **File I Exit** value to **-11**, we are telling the item to send a **-11** value into the system, which it interprets as a message to close the window. Then as the window closes, the Window Manager will close our application.

Connecting menu items to functions

Now that we've connected the messages to the Movie Control window and to the pull-down menu, we need to connect the Movie Control window's pull-down menu items to the member functions. Each of the pull-down menu options was marked as a "send message" item, which means any time the user selects an option like **Movie I Create,** the option will create a message and put it on the event queue. For example, when the user selects **Movie I Create,** the option creates the value **10,001** and sends it to the Event Manager's event queue. Then our application will retrieve that event.

Finishing the tool bar buttons

Before we go back into the code and look at exactly how our application will retrieve values, finish setting up the message system with tool bar buttons. These buttons need the same information as the pull-down items.

Use the **Create** button as an example. Invoke the button information notebook, then enter

```
10,001
```

into the value field. Then set the *Send user message* flag from the **Settings** list.
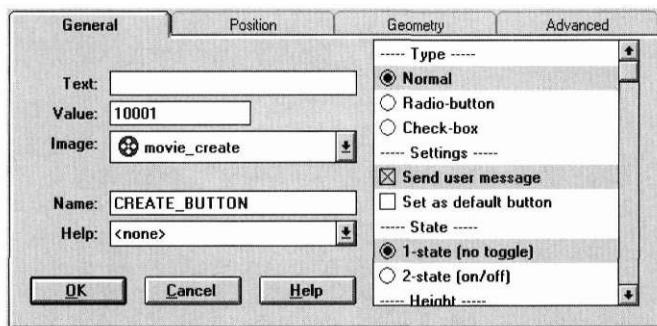
Do the same for the **Delete, Load,** and **Store** buttons by entering 10,002 for delete, 10,003 for load, and 10,004 for store, and by setting *Send user message.* Finally, connect the help contexts by entering the value 10,000 and setting the *Send user message* for each of the help pull-down items. Save the changes to the movie catalog system.

Processing messages

Let's review the code that processes these messages once again. In the function **UI_APPLICATION::Main()**, one line sends control to OpenZinc.

```
// Process user responses.
  UI_APPLICATION::Control();
```

At run time, **UI_APPLICATION::Control()** passes messages to **MOVIE_CONTROL::Event()**—system events, logical events, operating system-specific events, or user-defined events. **Movie I Create, Moviel Delete, Movie I Load,** and **Movie I Store** generate user-defined messages that **MOVIE_CONTROL::Event()** will interpret.

What the Movie Control window's Event() does

Let's look at what **MOVIE_CONTROL::Event()** does.

```
EVENT_TYPE MOVIE_CONTROL::Event(const UI_EVENT Sevent)
{

    // Check for special requests.
```

```
EVENT__TYPE ccode = event.type;
if (ccode == OPT_MOVIE_CREATE)
  ccode = MovieCreate(event);
else if (ccode == OPT_MOVIE_DELETE)
  ccode = MovieDelete(event);
else if (ccode == OPT_MOVIE_LOAD)
  ccode = MovieLoad(event);
else if (ccode == OPT_MOVIE_STORE)
  ccode = MovieStore(event);
else
  ccode = UIW_WINDOW::Event(event);
return (ccode);
}
```

First, it sets the control code, recognized as *ccode* in the file, which represents the message sent to our Movie Control window. When **Event()** receives a message to which it doesn't respond, it sends the information up to **UIW_WINDOW::Event()**. But when the user selects **Create, Delete, Load,** and **Store,** the program intercepts and processes the messages. To see how this happens, let's first look at **Movie I Create.** When the user selects it, it sends the *OPT_MOVIE_CREATE* message to the Movie Control window, which calls the **MovieCreate()** member function. In turn, this function adds a new Movie Information window to the Window Manager.

The same things happen when the user selects other options. When the user selects the **Movie I Delete** option, it sends a message to the Movie Control window, which calls **MovieDelete()**. In turn, this function adds a new Movie Selection window to the Window Manager. When the user selects **Movie I Load,** it also sends a message to the Movie Control window, which calls the **MovieLoad()** function. Then this function adds a new Movie Selection window. And when the user selects **Movie I Store,** it sends a message to the Movie Control window, which calls **MovieStore()**. But unlike the other member functions, **MovieStore()** doesn't do anything right now— we have not yet assigned it information or processing tasks. We will do so later.

Notice that when the program called **MovieDelete()** and **MovieLoad(),** both constructed a Movie Selection window. When the user selects **Movie I Delete,** we will tell the Movie Selection window to delete a movie. When the user selects **Movie I Load,** we will tell the Movie Selection window that we wish to load a movie. In both cases, we need the Movie Selection window to select the movie to delete or load.

We will discuss **MovieDelete()** and **MovieLoad()** in greater detail in the next tutorial. But the important thing to learn in this tutorial is **MOVIE**'s architecture, or specifically, that main control first goes to the Movie Control window, which invokes the movie selection process.

Viewing the application

With the changes we just made, we may now compile and run the **MOVIE** application.
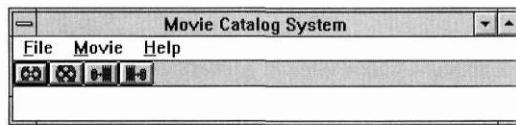
The main difference between the state of **MOVIE** in this tutorial and in the previous one is that in this state, it has all the movie options. If we select **Movie I Create, Movie I Delete,** or **Movie I Load,** we get additional information. For example, if we select **Movie I Create** from the pull-down menu, the Movie Information window appears on the screen.
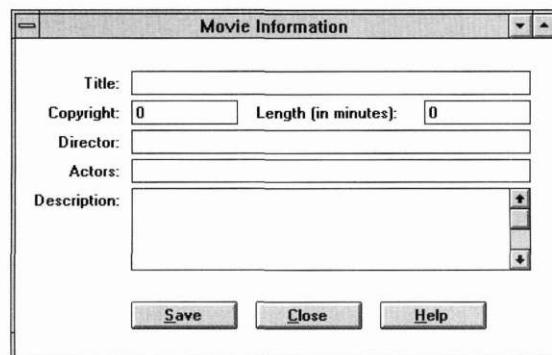
If we select **Movie | Delete** or **Movie | Load,** the Movie Selection window appears on the screen.
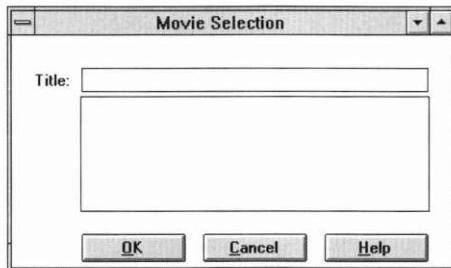
If we select **File I Exit,** the application exits.

Message flow

Now that we've compiled and run **MOVIE,** let's examine the message system to find out what's going on under the hood. First, the movie catalog window appears on the screen because we have added a new Movie Control window to the Window Manager.

```
// Add the Movie Control window.
*windowManager + new M0VIE_C0NTR0L;
```

Second, when we select **Movie I Create,** the create object sends the value 10,001 through OpenZinc. The Movie Control window intercepts that message and calls the **MovieCreate()** member function.

```
EVENTJTYPE MOVIE_CONTROL::Event(const UI_EVENT &event)
{
  // Check for special requests.
  EVENTJTYPE ccode = event.type;
  if (ccode == OPT_MOVIE_CREATE)
    ccode = MovieCreate(event);
```

MovieCreate() then constructs a new movie information window and adds it to the screen.

```
EVENT_TYPE MOVIE_CONTROL::MovieCreate(const UI_EVENT &event)
{
```

```
      *windowManager + new UIW_WINDOW("MOVIE_INFORMATION",
        defaultStorage);
      return (event.type);
   }
```

The same process happens when we select **Movie** I **Delete** and **Movie I Load.** For example, if we select **Movie I Delete,** the Movie Control window intercepts the value **10,002.** Then it processes this event and calls the **MovieDelete()** member function.

Finally, when we select **File I Exit,** the option sends the value **-11,** OpenZinc's *S_CLOSE* message, through the system. The Window Manager, rather than the Movie Control window, intercepts the message. In response, the Window Manager deletes the movie catalog system window from the display. Since this window is the Window Manager's main window, the application terminates.

Conclusion

in this tutorial, we learned how to assign values to messages. We also learned how to use messages to connect menu options to functions that create, delete, load, and store movie records. In the next tutorial, we're going to define the architecture of the Movie Selection and Movie Information windows.

Deriving Support Modules

in previous tutorials, we used the Designer to create windows and their window objects, and looked at how to create source code that uses these windows. In this tutorial, we'll round out the **MOVIE** application's architecture. We'll do this by creating two classes, **MOVIE_SELECTION** and **MOVIE_INFORMATION,** and by connecting their messages to the Movie Control Window and source code. We will also finish connecting the help system to our application.

Working with MOVIE4

Components Of MOVIE4

In Figure 1 on page 58, we discussed the components of the **MOVIE** application we've been working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE4,** the fourth component of **MOVIE.** The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
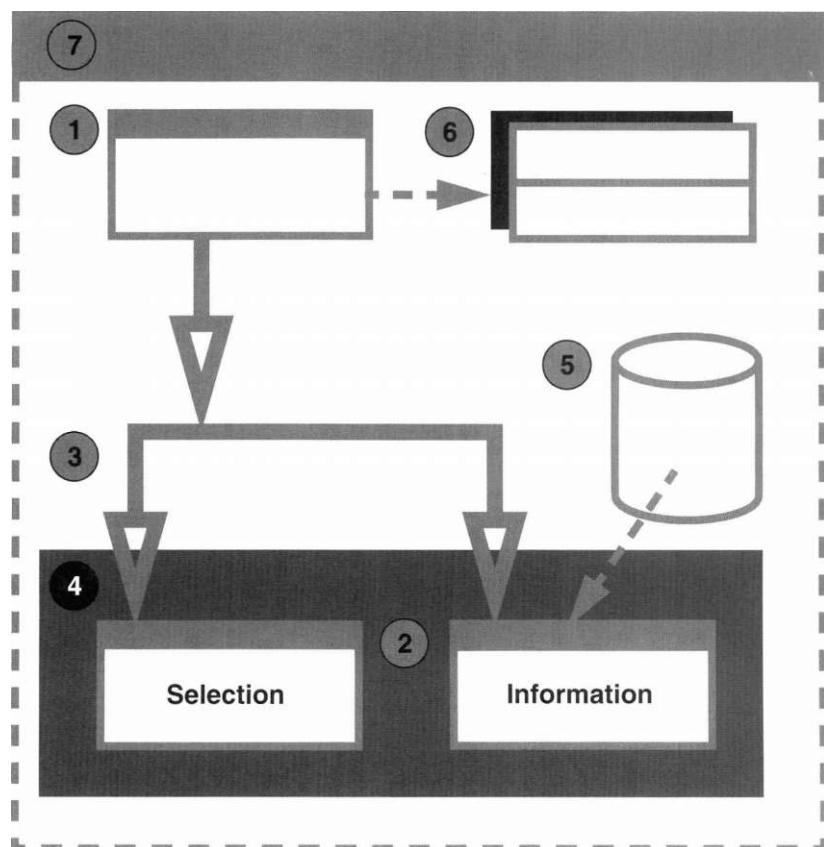
Source files   Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 7. Components of MOVIE4

Type of file	Name of file	Description of file
User-pro-vided files	**MOVIE4.CPP**	The main program
	MOVIE4.HPP	Class definitions, identi-fications, and messages
Designer-generated files	**P_MOVIE4.CPP**	Code for tying Designer objects to our program
	P_MOVIE4.HPP	Identifications and help contexts
	P_MOVIE4.DAT	Persistent object storage

The Movie Selection window

Let's begin rounding out the **MOVIE** application's architecture by working on the Movie Selection window. To prepare it, we'll go in to the Designer and change a few of the options. By using the Designer to modify windows, we'll be making changes to the **P_MOVIE4.DAT** file, the container for the Movie Selection and the Movie Information windows.

Changing its information	We must change two things with the Movie Selection window, so open it for editing. First, in the information notebook for the main window, set the object to a dialog window. Do this by selecting the dialog object option from the general list in its information notebook.
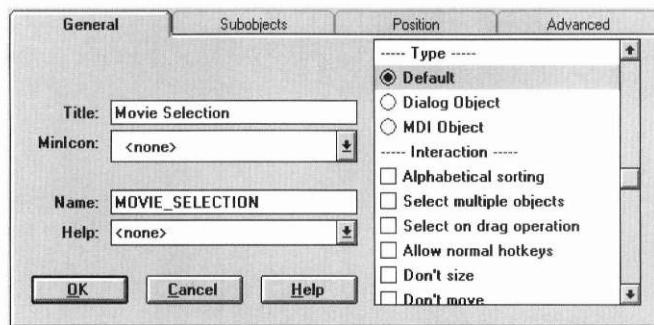
The main difference between a dialog object and a normal window is how they look. In most systems, the border of a dialog window is flat, whereas the border for a normal window is three dimensional. In addition, most dialog objects are marked as nonsizeable modal objects, which means a user cannot continue an application until he or she has entered a title name into the Movie Selection and pressed OK or Cancel.

Assigning messages to buttons	Let's make the OK and Cancel buttons functional by entering a value and selecting a user message for each of these buttons.

First, let's change the information for the **OK** button by invoking the button information notebook, then by changing the value of the button to 10,005. (This is a new value that will correspond to an *OPT_MOVIE_OK* message, defined in **MOVIE4.HPP.)** Then set the *Send user message* flag from the options in the general list.
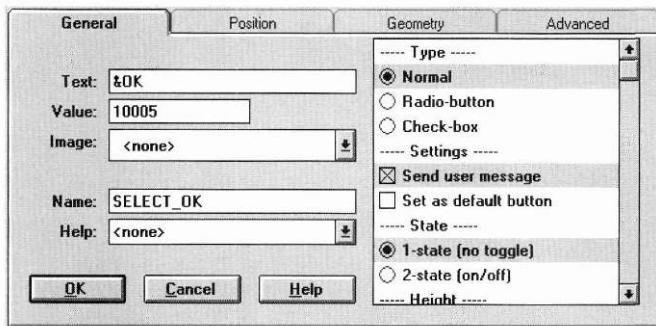
Change information for the **Cancel** button by invoking the button information notebook, by setting the *Send user message* flag, then by specifying the value -11 in the value field in the notebook. (The value -11 corresponds to *S_ CLOSE.)*

Last, change information for the **Help** button using its information notebook. Enter the value 10,000, which corresponds to the *OPT_HELP* value, and set the *Send user message* flag in the settings list.

Now we have connected the buttons to messages. The **OK** button sends an *OPT_MOVIE_OK* message, the **Cancel** button sends the *S_CLOSE* message, and the **Help** button sends the *OPT_HELP* message.

We implement this, first by defining the option messages, then by deriving **MOVIE_SELECTION** from **UIW_WINDOW,** and overloading the constructor and the **Event()** function.

```
const ZIL_USER_EVENT OPT_HELP       = 10000;
const ZIL_USER_EVENT OPT_MOVIE_CREATE= 10001;
const ZIL_USER_EVENT OPT_MOVIE_DELETE= 10002;
const ZIL_USER_EVENT OPT_MOVIE_LOAD  = 10003;
const ZIL_USER_EVENT OPT_MOVIE_STORE = 10004;
const ZIL_USER_EVENT OPT_MOVIE_OK    = 10005;
class MOVIE_SELECTION : public UIW_WINDOW
{

public:
  MOVIE_SELECTION(void);
  virtual EVENT_TYPE Event(const UI_EVENT &event);
```

```
    };
```

The constructor   Now that we've assigned messages to the buttons, let's look at the construc-
tor code for the Movie Selection window. The constructor is similar to the
Movie Control Window's constructor—we call the base **UIW_WINDOW**
class and pass **MOVIE_SELECTION** and the default storage as our param-
eters. In addition, just as in the Movie Control Window, we center the win-
dow on the display.

```
MOVIE_SELECTION::MOVIE_SELECTION(void) :
  UIW_WINDOW("MOVIE_SELECTION", defaultStorage)
{
  // Center the window on the screen.
  windowManager->Center(this);
}
```

In order to allow the Movie Selection window to respond to messages, we
must overload **MOVIE_SELECTION::Event()**. Once overloaded, the
Event() function will intercept messages that the base **UIW_WINDOW**
would otherwise receive. **Event()** will respond when the user presses the
Help, OK, and **Cancel** buttons. The following code does this:

```
EVENT_TYPE MOVIE_SELECTION::Event(const UI_EVENT &event)
{
  // Check for special requests.
  EVENT_TYPE ccode = event.type;
  if (ccode == OPT_HELP)
    helpSystem->DisplayHelp(windowManager, helpContext);
  else if (ccode == OPT_MOVIE_OK)
    eventManager->Put(S_CLOSE); // Close the window,
  else
    ccode = UIW_WINDOW::Event(event);
  return (ccode);
}
```

When the user pushes the **Help** button, **Event()** intercepts *OPT_HELP*.
Then the help system calls **DisplayHelp()**, which brings up a help window.

When the user pushes the **OK** button, **Event()** intercepts *OPT_MOVIE_OK,*
and a movie will be loaded or deleted. Since we're implementing architec-
ture and not functionality, for now, we'll just pass the *S_CLOSE* message to
the Event Manager. The Window Manager will process this message and
will remove the Movie Selection window from the display.

Earlier, we assigned the *S_CLOSE* message to the **Cancel** button. When the
user pushes the **Cancel** button, the Window Manager intercepts the message
and removes the window from the display.

Here's the old code that invoked the Movie Selection window—this came from our tutorial that discussed the **MovieDelete()** and **MovieLoad()** member functions.

```
EVENT_TYPE MOVIE_CONTROL::MovieDelete(const UI_EVENT &event)
{

  *windowManager + new UIW_WINDOW("MOVIE_SELECTION",
    defaultStorage);
  return (event.type);

}

EVENT_TYPE MOVIE_CONTROL::MovieLoad(const UI_EVENT &event)
{

  *windowManager + new UIW_WINDOW("MOVIE_SELECTION",
    defaultStorage);
  return (event.type);

}
```

The new **MOVIE_SELECTION** adds a Movie Selection window to the Window Manager in **MovieDelete()** and **MovieLoad()**, instead of in **UIW_WINDOW'S** constructor. This allows **MOVIE_SELECTION,** not **MOVIE_CONTROL,** to process the information. We will see the benefits of this in the next chapter, when we begin implementing storage.

```
EVENT_TYPE M0VIE_C0NTR0L::MovieDelete(const UI_EVENT &event)
{

  *windowManager + new MOVIE_SELECTION;
  return (event.type);

}

EVENT_TYPE MOVIE_CONTROL::MovieLoad(const UI_EVENT &event)
{

  *windowManager + new MOVIE_SELECTION;
  return (event.type);

}
```

Movie   Information

Now that we've modified the Movie Selection window, we can modify the Movie Information window—and we'll do it like we did Movie Selection. We'll derive a new window class called MOVIE_INFORMATION Then we'll overload its Event( ) function, then intercept messages sent by the Movie Information dialog window.

Changing its information

Launch the Designer and open the Movie Information window to change its information.

We'll change several of its flags, as well as the values of its buttons. Let's get started by invoking the general window information notebook. Change the window to be a dialog object by selecting the Dialog Object flag from the general window object features list.
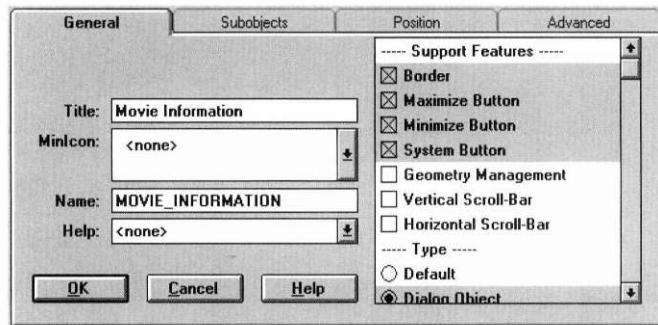
Next, change the Save button information by invoking the information note-book for the button. Once invoked, enter the value

```
10,005
```

which corresponds to the *OPT_MOVIE_OK* option. Then set its flag to *Send user message* by selecting that option from the options list.
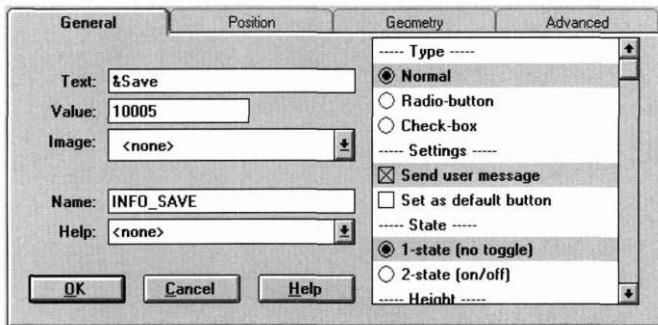
Next, change the value of the Close button to

```
-11
```

which corresponds to S_CLOSE. Then set its flag to *Send user message* by selecting that option from the options list.
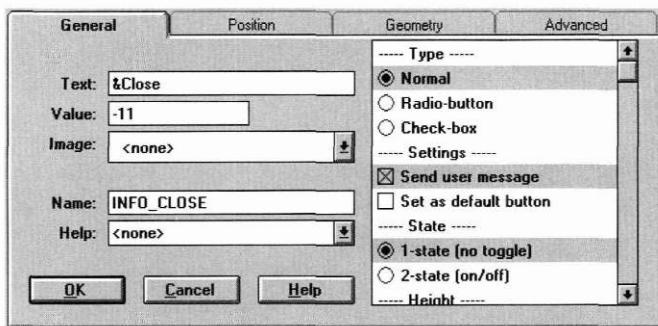
Finally, change the Help button's value to

```
10000
```

Then set its flag to *Send user message* by selecting that option from the options list.
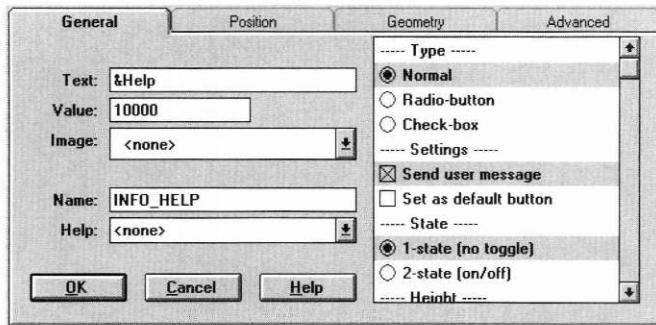
Movie Information definitions

In **MOVIE4.HPP, MOVIE_INFORMATION** derives from **UIW_WIN-DOW.**

```
class MOVIE_INFORMATION : public UIW_WINDOW
{
public:
  MOVIE_INFORMATION(void);
  virtual EVENT_TYPE Event(const UIEVENT &event);
};
```

The Event( ) function

MOVIE4.CPP contains Movie Information's constructor code and its **Event()** function. The constructor is nearly the same as Movie Selection's constructor.

```
MOVIE_INFORMATION::MOVIE_INFORMATION(void) :
  UIW_WINDOW("MOVIE_INFORMATION", defaultStorage)
{
  // Center the window on the screen.
  windowManager->Center(this);
}
```

MOVIE_INFORMATION::Event() differs slightly from **MOVIE_SELECTION::Event()**. Just like Movie Selection, it intercepts the *OPT_HELP* message and calls the help system. But it also processes the *OPT_MOVIE_STORE* message, which has a value 10,005, and which tells the **Event()** function to store the record. We'll implement how it stores the record in the next tutorial, though we'll take care of some other functionality in this tutorial.

```
EVENT_TYPE MOVIE_INFORMATION::Event(const UI_EVENT &event)
{
```

```
                // Check for special requests.
                EVENT_TYPE ccode = event.type;
                if (ccode == OPT_HELP)
                  helpSystem->DisplayHelp(windowManager, helpContext);
                else if (ccode == OPT_MOVIE_STORE)
                {
                  eventManager->DeviceState(E_MOUSE, DM_WAIT);
                  // save storage till later.
                  eventManager->DeviceState(E_MOUSE, DM_VIEW);
                }
                else
                  ccode = UIW_WINDOW::Event(event);
                return (ccode);
            }
```

When the **Event()** function receives the *OPT_MOVIE_STORE* message, it stores the information to disk. But since a storage operation may take some time, we should give the user some information about the operation's status. To give the user this information, we call the Event Manager's **DeviceState()** function to tell the mouse to display a "wait" cursor while the **Event()** function will store a record in the data file. After the record is stored, we tell the mouse cursor to display a "view" cursor, which informs the user that he may continue using the application.

The **Event()** function need not handle the *S_CLOSE* message; it can pass it to the base **UIW_WINDOW::Event()** function, which processes the message directly.

Help system, persistence architecture

Now that we've added architecture for passing a storage message, we need to add one additional piece of code to **Main().**

```
        // Provide a general storage module,
        static ZIL_ICHAR _fileName[] = "p_movie4.dat";
        UI_WINDOW_OBJECT:rhelpSystem = new UI_HELP_SYSTEM(_fileName);
        UI_WINDOW_OBJECT::defaultStorage = new
          ZIL_STORAGE_READ_ONLY(_fileName);
```

This code generalizes Movie Information's data file, assigning its name to a parameter called *JileName.* Also, it creates a new **UI_HELP_SYSTEM,** the default system that displays a window whenever the user requests help. Finally, it creates read-only storage for the window by calling the constructor with the same filename.

Testing OUT handiwork

Save these changes to the Movie Information and Movie Selection windows, and add the changes to create Movie Information and Movie Selection classes. Go ahead and compile **MOVIE4.** Then run the executable to examine the features.
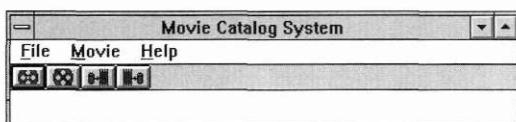
During execution, most features of the movie catalog system window are the same as in previous tutorials. The main addition is that the window's help buttons now bring up a general help window. (We'll further refine the help system in "Refining the help system" on page 165.)

The options work like they did in our last tutorial—**Create** brings up the Movie Information window, and **Delete** and **Load** both bring up the Movie Selection window. We'll implement the **Store** option in the next tutorial.

After creating a new Movie Information record, selecting **Help** brings up the help system. Selecting **Close** or **Save,** however, removes the Movie Information window from the display. The **Save** button doesn't save anything yet. As we discussed earlier, we'll connect the **Save** button to the storage module in the next tutorial.

If we select **Movie** I **Delete,** the Movie Selection window appears on the display.
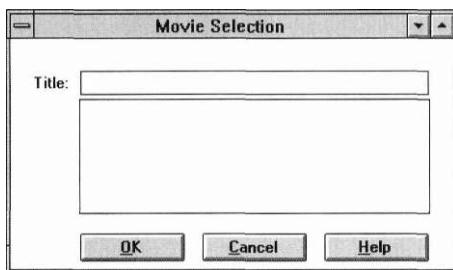
Selecting the **Help** button brings up the help window. And since we have not implemented all of the features of the **OK** button, pressing **OK** or **Cancel** doesn't yet close the Movie Selection window.

Conclusion

In this tutorial, we've created **MOVIE_SELECTION** and **MOVIE_IN-FORMATION** classes that encapsulate and localize the Movie Selection and Movie Information windows. Though the Movie Control Window architecture still controls the overall application, these windows now respond to events.

In the next tutorial, we'll begin implementing the next major component of MOVIE—reading and writing records.

Loading and Storing Data

We've reached a point in the tutorials where we can fill in the pieces of

our architecture. Up to now, we've used the Designer to create windows that present information to the screen; we've also discussed the architecture of the window objects. Now our program has many holes; but with our architecture finished, we can fill in those holes. This tutorial shows that we can design the architecture of our programs and defer filling in the holes until later in the programming process.

| Key Concepts | storing data using a **.DAT** file |
| | completing the programming process |

Here, we'll examine the code for reading and storing the movie data, as well as for managing movie data. We've already written a large portion of **MOVIE,** so as we read this tutorial, we'll see the benefits of the architecture we set up (and deferred) in previous tutorials, which now load and store, and manage information efficiently and understandably.

First, we'll look at the groundwork for implementing storage, the definitions of **MOVIE_SELECTION** and **MOVIE_INFORMATION.** Then we'll disassemble the implementations piece by piece to learn how they work. In these implementations, we'll pay close attention to the communication between the Movie Control Window, the Movie Selection window, and the Movie Information window. We'll also examine loading and storing records using a **.DAT** file.

Working with MOVIE5

Components Of MOVIE5

In Figure 1 on page 58, we discussed the components of the **MOVIE** application we've been working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE5,** the fifth component of **MOVIE.**

The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
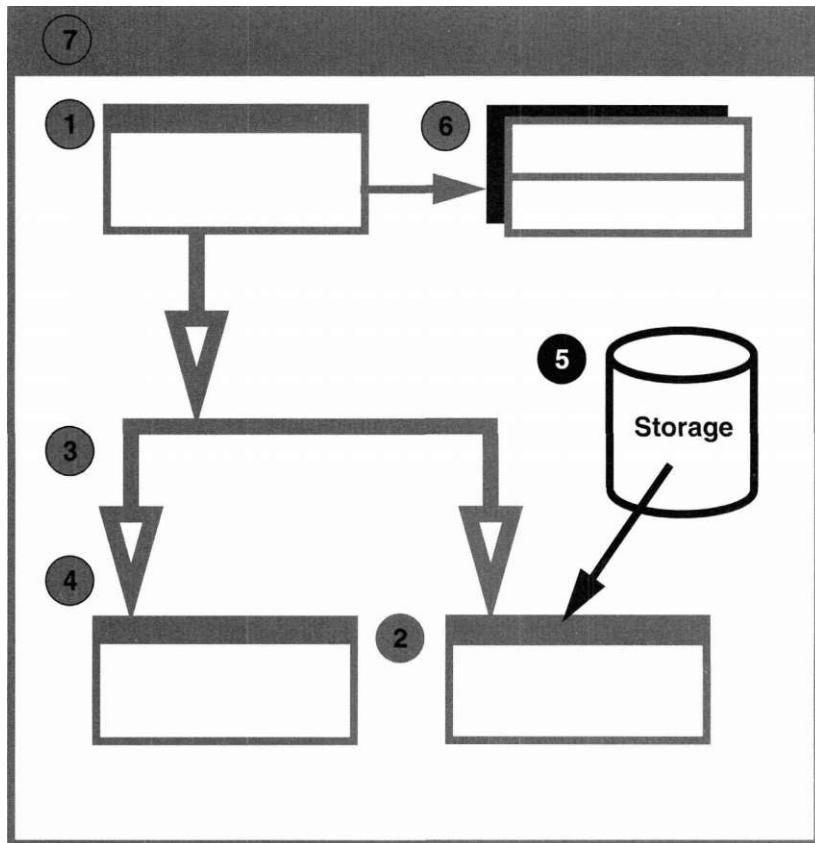
Source files
Below is a table of source files we**'ll** be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 8. Components of MOVIE5

Type of file	Name of file	Description of file
User-pro-vided files	**MOYIE5.CPP**	The main program
	MOVIE5.HPP	Class definitions, identifications, and messages
	MOVIE. DAT	User data storage
Designer-generated files	**P_MOVIE5.CPP**	Code for connecting Designer objects to our program
	P_MOVIE5.HPP	Identifications and help contexts

Laying the ground work for storage

Let's start filling in the holes by looking at **MOVIE5.HPP** At the top is the definition for **MOVIE_CONTROL.** Its public section contains the constructor and the **Event()** function we've discussed at in previous tutorials. Its private section contains functions **MovieCreate(), MovieDelete(), MovieLoad(**), and **MovieStore().**

```
class MOVIE_CONTROL : public UIW_WINDOW
{
public:
    static ZIL_STORAGE *_dataFile;
    static ZIL_ICHAR _movieName[64];
    MOVIE_CONTROL(void);
    ~MOVIE_CONTROL(void);
    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
private:
    EVENT_TYPE MovieCreate(const UI_EVENT &event);_
    EVENT_TYPE MovieDelete(const UI_EVENT &event);
    EVENT_TYPE MovieLoad(const UI_EVENT Sevent);
    EVENT_TYPE MovieStore(const UI EVENT &event);
};
```

This tutorial has a new member function, the destructor. It also contains two static members, **ZIL_STORAGE** *_dataFile* and **ZIL_ICHAR** *_movieName[64].* The *_dataFile* member stores movie record information, and the *_movieName* member stores the name of the current movie record.

NOTE: The **ZIL_STORAGE** data file is *not* a true database. We are using it in this tutorial for simplicity. OpenZinc data files are built for persistence, not data storage; the methods in this tutorial work for simple data storage and retrieval, but not for advanced database operations. But later in the tutorial, we**'ll** show where we could bolt up a third-party database to **MOVIE.**

The Movie Control window

The next piece of code defines a new message called *OPT_RESET_SELECTION,* which will allow individual Movie Selection items to communicate with the Movie Selection window.

MOVIE_CONTROL will work a little differently from objects in previous tutorials; it will do more than mere file management. We also add more code to all options so that they do specific things.

The Movie Selection window

MOVIEjSELECTION is much the same as in the last tutorial, except for a new private member called *request,* which will identify the type of request **MOVIE_SELECTION** received. The request can be either *OPT_MOVIE_LOAD,* which causes the class to load a new record, or *OPT_MOVIE_DELETE,* which causes it to delete a record.

```
Class MOVIE_SELECTION : public UIW_WINDOW
{
public:
  MOVIE_SELECTION(ZIL_STORAGE_READ_ONLY * file,
    ZIL_USER_EVENT request);
  virtual EVENTTYPE Event (const UI_EVENT Seventh-
private :
  ZIL_USER_EVENT request;
};
```

We will also pass to our constructor the data file and type of request that we want the **MOVIE_SELECTION** to perform. The file will be a pointer to the library catalog, and the request will be either *OPT_MOVIE_LOAD* or *OPT_MOVIE_DELETE.* To expand the definition from previous definitions of **MOVIE_SELECTION,** this window will display all of the movies currently available in the database, then respond to the Movie Control Window, telling it what type of movie needs to be loaded or deleted.

The Movie Information window

MOVIE_INFORMATION is also similar, except for two new member functions, **Load()** and **Store(). Load()** gets a specific record title from the database. The first argument of **Load()** is the name of the record that we will be loading, and the second is the data file.

```
class MOVIE_INFORMATION : public UIW_WINDOW
{
public:
  MOVIE_INFORMATION(ZIL_ICHAR *name = ZIL_NULLP(ZIL_ICHAR));
  virtual EVENT_TYPE Event(const UI_EVENT Sevent);
  virtual void Load(const ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *file,
    ZIL_ST0RAGE_0BJECT_READ_0NLY *object =
      ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY),
    UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),
    UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
  virtual void Store(const ZIL_ICHAR *name,
    ZIL_ST0RAGE *file = ZIL_NULLP(ZIL_STORAGE),
    ZIL_STORAGE_OBJECT *object = ZIL_NULLP(ZIL_ST0RAGE_0BJECT),
    UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),
    UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
};
```

Store() works much the same way, except that it stores movie information to our data file. Again, the first parameter is the name of the record that we want to store, and the second parameter is a pointer to the data file.

Writing   the   load   and   store functionality

Opening and closing the data file in the constructor

We've already entered all of the information in the **.DAT** file, so now we need only work with the source files. Let's start by implementing the part of the Movie Control Window's constructor and destructor that will open and close the data file.

```
MOVIE_CONTROL::MOVIE_CONTROL(void) :
  UIW_WINDOW("MOVIE CONTROL", defaultStorage)
{
  // Give the window a unique searchID.
  searchID = ID_MOVIE^CONTROL;
  // Center the window at the top of the screen.
  windowManager->Center(this);
  relative.bottom = relative.Height() - 1;
  relative.top = 0;
```

```
    // Initialize the data file.
    _dataFile = new ZIL_STORAGE("movie.dat",
        UIS_OPENCREATE | UIS_READWRITE);
}
```

The constructor opens the data file **MOVIE.DAT** with read and write access. The **MOVIE_CONTROL** destructor saves the data file, then deletes the data file object.

```
MOVIE_CONTROL::~MOVIE_CONTROL(void)
{
    // Save the data file.
    _dataFile->Save();
    delete _dataFile;
}
```

Deleting the data file object closes the file and preserves all the information that we've saved during the **MOVIE**'s execution.

The Event( ) function

The **Event()** function works exactly as in the previous tutorial—it intercepts messages and dispatches them to the proper member function. As a brief reminder,

1. *OPT_HELP* is handled by the help system when we call *helpSystem->DisplayHe\p().*

2. *OPT_MOVIE_CREATE* is dispatched to **MovieCreate().**

3. *OPT_MOVIE_DELETE* is dispatched to **MovieDelete().**

4. *OPT_MOVIE_LOAD* is dispatched to **MovieLoad().**

5. *OPT_MOVIE_STORE* is dispatched to **MovieStore().**

```
EVENTTYPE MOVIE^CONTROL::Event(const UI_EVENT &event)
{
    // Check for special requests.
    EVENT_TYPE ccode = event.type;
    if (event.type == OPT_HELP)
        helpSystem->DisplayHelp(windowManager, helpContext);
    else if (ccode == OPT_MOVIE_CREATE)
        ccode = MovieCreate(event);
    else if (ccode == OPT_MOVIE_DELETE)
        ccode = MovieDelete(event);
    else if (ccode == OPT_MOVIE_LOAD)
        ccode = MovieLoad(event);
    else if (ccode == OPT_MOVIE_STORE)
        ccode = MovieStore(event);
    else
        ccode = UIW_WINDOW::Event(event);
    return (ccode);
```

```
}
```

MovieCreate( )

Now that we remember how **Event()** dispatches messages to member functions, let's examine each of the member functions, starting with **MovieCreate()**. We won't change **MovieCreate()** much from the previous tutorial—here, too, we create a new movie information window and attach it to the Window Manager.

```
EVENT_TYPE MOVIE_CONTROL::MovieCreate(const UI_EVENT Sevent)
{
   *windowManager + new MOVIE_INFORMATION;
   return (event.type);
}
```

Adding a new Movie Information window displays the window on the screen with all information blank. Also, **MOVIE_INFORMATION** will handle its own storage.

MovieDelete( )

The **MovieDelete()** code is similar to that in the previous tutorial, except that we have two new components. The first component checks to see if a movie record is active. If so, it deletes it from the data file by calling _dflrflFi7e**->DestroyObject()**, with the movie name as the parameter. Then it resets the movie name for the next time it's called.

```
EVENT_TYPE MOVIE_CONTROL::MovieDelete(const UI_EVENT Sevent)
{
   if (_movieName[0])
   {
      _dataFile->DestroyObject(movieName);
      _movieName[0] = '\0';
   }
   else
      *windowManager + new MOVIE_SELECTION(_dataFile,
         OPT_MOVIE_DELETE);
   return (event.type);
}
```

MOVIE_SELECTION will set _movieName_ when the user presses **OK** after choosing a movie from the list. We will look at that a little later in the tutorial.

For now, let's work on the second piece of **MovieDelete()**. This code is similar to that in the previous tutorial, except that it passes in a pointer to the data file in addition to the type of request.

```
EVENT_TYPE MOVIE_CONTROL::MovieDelete(const UI_EVENT &event)
{
```

```
if (_movieName[0])
{

  _dataFile->DestroyObject(_movieName);
  _movieName[0] = '\0';
}
else
  *windowManager + new MOVIE_SELECTION(_dataFile,
    OPT_MOVIE_DELETE);
return (event.type);
}
```

In this function, the request will be *OPT_MOV1E_DELETE.* Recall that in the Movie Control Window, **Event()** had a **case** for *OPT_MOVIE_DELETE,* which called **MovieDelete()** with the original event, *event.type = OPT_MO VIE_DELETE.*

```
else if (ccode == OPT_MOVIE_DELETE)
  ccode = MovieDelete(event);
```

Here, we call **MOVIE_SELECTION** and request that we want to use the **Delete** operation instead of the **Load** operation.

MovieLoad( ) Let's now look at the **MovieLoad()** member function. It works like **MovieDelete()**, only it makes a different request.

First, **MovieLoad()** checks the movie name. If it exists, it loads it from the data file by creating a new movie information record, by assigning it a valid movie name, then by attaching it to the Window Manager. Then it resets the movie name, so that the next time **MovieLoad()** is called, it has a fresh name field.

```
EVENT_TYPE MOVIE_CONTROL::MovieLoad(const UI_EVENT &event)
{
  if (_movieName[0])
  {

    *windowManager + new MOVIE_INFORMATION (_movieName);
    _movieName[0] = '\0';
  }
  else
    *windowManager + new MOVIE_SELECTION(_dataFile,
      OPT_MOVIE_LOAD);
  return (event.type);
}
```

But if no movie name exists, then **MovieLoad()** opens the Movie Selection window using the *_dataFile* pointer and the request *OPT_MOVIE_LOAD.*

```
EVENTJTYPE MOVIE_CONTROL::MovieLoad(const UI_EVENT Sevent)
```

```
{
  if (_movieName[0])
  {

    *windowManager + new MOVIE_INFORMATION(_movieName);
    _movieName[0] = '\0';
  }
  else
    *windowManager + new MOVIE_SELECTION(_dataFile,
      OPT_MOVIE_LOAD);
  return (event.type);
}
```

This tells **MOVIE_SELECTION** that we requested a movie for loading,
rather than for deleting.

The type of request **MOVIE_SELECTION** receives is important. Earlier,
we said we can call **MOVIE_SELECTION**'s **Delete()** or **Load()** opera-
tion. Upon completion, **MOVIE_SELECTION** will tell the Movie Control
Window whether to delete or to load a new movie. (We'll examine this in
detail when we examine the **MOVIE_SELECTION** class.)

MovieStore( )   At any given time, the Window Manager will have at least the Movie Con-
trol Window on the screen. It may also have one or more Movie Selection
windows. **MovieStore()** finds the first window that matches the movie
name, and then stores the information out to the data file.

In the previous tutorial, **MovieStore()** was stubbed out. Now, in keeping
with the mission of this tutorial, we'll fill in the holes and write code to store
the movie record. This code will look at all the windows on the screen to see
if any are of type **MOVIE_INFORMATION.** If so, they are stored to disk.

```
EVENT_TYPE MOVIE_CONTROL::MovieStore(const UI_EVENT &event)
{

  UI_WINDOW_OBJECT *window =
    windowManager->Get("MOVIE_INFORMATION");
  if (window)
    window->Store(2IL_NULLP(ZIL_ICHAR), _dataFile);
  return (event.type);
}
```

**The Movie
Control window**

Now that we've discussed what happens in the header file, let's review what the Movie Control Window is doing.

- First, the constructor creates or opens a new or existing data file.

- Then, when it receives create, delete, load, or store requests, it calls the appropriate member function.

- The member function **MovieCreate()** creates a new Movie Information record. However, the member functions **MovieDelete()** and **MovieLoad()** are more complex. We can call them in one of two different circumstances—when a movie name is present, or when one isn't.

- Finally, **MovieStore()** stores out the first window that it finds that matches the string id *MOVIE_INFORMATION*. These members form the major components of the control of the **MOVIE** application.

**The Movie
Selection class**

Let's now examine the functionality of **MOVIE_SELECTION.** Here's an overview of what it does.

1. It creates and displays a Movie Selection window.

2. Then it lists all movies in the catalog data file.

3. When created, the Movie Control Window specifies an operation, either delete or load.

4. Once the user presses **OK, MOVIE_SELECTION** sends a message that tells the Movie Control Window whether the *OPT_MOVIE_DELETE* or *OPT_MOVIE_LOAD* operation is requested, and also sets the movie name.

```
else if  (ccode == OPT_MOVIE_OK)
{
  Get(FLD_SELECT_TITLE)->Information(I_COPY_TEXT,
    MOVIE_CONTROL::_movieName);
  eventManager->Put(S_CLOSE);// Close the window.
  eventManager->Put(request);// Send response to the main
                             // control.
}
```

Now **MOVIEjSELECTION**'s constructor has a new section that loads movie information. One of the first lines in the section is a constructor of an event.

```
UI_EVENT addEvent(S_ADD_OBJECT);
```

The event, *S_ADD_OBJECT,* communicates with the vertical list inside the Movie Selection window. It tells it to add to itself the individual movie names, which are its items.

The next line calls the member function **Get()** with the field identifier
FLD_SELECT_SELECT10N.

```
UI_WINDOW_OBJECT *list = Get(FLD_SELECT_SELECTION);
```

With the Designer, we assigned *FLD_SELECT_SELECTION* to the vertical
list.

Get() calls **MOVIE_SELECTION** and asks for a field that matches this
identification. This will be the vertical list where we present the movie
records. Next, we use **FindFirstObject()** to find every record inside the
data file.

```
ZIL_ICHAR *entry = dataFile->FindFirstObject(_allObjects);
```

The next several lines continue to find data records until we've run out of
information in the file.

```
for (; entry; entry = dataFile->FindNextObject())
    if (strcmp(entry, _currentDirectory) &&
      strcmp(entry, _parentDirectory))
    {
      addEvent.windowObject = new UIW_BUTTON(0, 0, 30, entry,
        BTF_NO_TOGGLE | BTF_NO_3D | BTF_SEND_MESSAGE,
        WOF_NO_FLAGS, ZIL_NULLF(ZIL_USER_FUNCTION),
        OPT_RESET_SELECTION);
      list->Event(addEvent);

    }
```

We create a new list entry for each movie record by calling the
UIW_BUTTON constructor and by passing the name of the movie. Then
we set up the button to send a user message by setting the *BTF_SEND-
_MESSAGE* request and by setting our request as *OPT_RESET-
_SELECTION.*

```
addEvent.windowObject = new UIW_BUTTON(0, 0, 30, entry,
  BTF_NO_TOGGLE | BTF_NO_3D | BTF_SEND_MESSAGE,
  WOF_NO_FLAGS, ZILJNULLF(ZIL_USER_FUNCTTON),
  OPT_RESET SELECTION);
```

Finally, we add the button to the list by calling //s**?->Event()** with the mes-
sage *S_ADD_OBJECT.*

```
list->Event(addEvent);
```

The Event() function. Now that we've introduced **MOVIE_SELECTION**'s
constructor, let's look at its **Event()** function. When the user presses **OK,**
MOVIE_SELECTION receives the message *OPT_MOVIE_OK.* Handling

that request, **Event()** gets the information from the window's title field, sends a delete or load request to the Movie Control Window, and then closes the Movie Selection window.

```
else if  (ccode == OPT_MOVIE_OK)
{
  Get(FLD_SELECT_TITLE)->Information(I_COPY_TEXT,
    MOVIE_CONTROL::_movieName);
  eventManager->Put(S_CLOSE);// Close the window.
  eventManager->Put(request);// Send response to the
                             //main control.
}
```

These three lines of code expose much of OpenZinc's architecture. In the first line, we get information from our **MOVIE_SELECTION** window by calling the **Get()** function with our field identifier of *FLD_SELECT_TITLE*, and by making the request of *I_COPY_TEXT*. Then we pass in the Movie Control Window movie name.

```
Get(FLD_SELECT_TITLE)->Information(I_COPY_TEXT,
  MOVIE_CONTROL::_movieName);
```

This tells the selected title field that we want to copy the text currently in its field to *_movieNcime*. Then we put an *S_CLOSE* message into the Event Manager. As discussed in a previous tutorial, this message is sent through the Event Manager, picked up by the system. And since the selection window is the top window on the display, Window Manager will close the *selection* window.

This is why we copy the movie information to *jnovieName* before we put the *S_CLOSE* in the Event Manager. Once our selection window is closed, it will be destroyed. So if we passed messages in a different order, we would delete the information before we copied it.

The last thing we do is put the load or delete request on the event queue. Now the Movie Control Window will pick it up, since it's the first window on the screen after our window is removed.

```
eventManager->Put(request); // Send response to the main
                            // control.
```

The selection reset command. The final piece of code that we need to examine is the *OPT_RESET_SELECTION* command. Each button sends this command whenever the user selects a movie from the list.

```
else if  (ccode == OPT_RESET_SELECTION)
{
  ZIL_ICHAR *title;
```

```
          event.windowObject->Information(I_GET_TEXT, Stitle);
          Get(FLD_SELECT__TITLE)->Information(I_SET_TEXT, title);
    }
```

When the program receives the *OPT_RESET_SELECTION* command, it gets the text from the current item and inserts it into the title field. When a button sends the *OPT_RESET_SELECTION* message, it also includes a pointer to itself inside of the *event.windowObject* variable. This is where we get the text for the title field, using the *I_GET_TEXT* request.

```
          event.windowObject->Information(I_GET_TEXT, &title);
```

Finally, setting the title selection with the text, we get the title field from our window, and copy the movie title with the command *I_SET_TEXT*.

```
          Get(FLD_SELECT_TITLE)->Information(I_SET_TEXT, title);
```

This code updates the window, displaying the new title in the title field.

Summarizing Movie Selection. **MOVIE_SELECTION** might seem complicated, so let's review what it does so we can keep a clear picture in our minds. In the constructor, we pass the data file and the request, either *OPT_MOVIE_LOAD* or *OPT_MOVIE_DELETE*. Then we traverse the data file, looking for all available movie records inside the vertical list.

In the **Event()** function we look for two messages, *OPT_MOVIE_OK* and *OPT_RESET_SELECTION*. When it receives *OPT_MOVIE_OK*, it sets the *jnovieNcune* variable name. Then it sends a response to the Movie Control Window, telling it that we've completed the request. Finally, we reset a movie selection inside the movie list by sending the *OPT_RESET_SELECTION* message, and getting the information text from the current window object. Then it sets the selected title in the title field.

The Movie Information class

Now that we've reviewed **MOVIE_SELECTION,** the last code we'll need to write in this tutorial is **MOVIE_INFORMATION.** Its definition is simple, but there are many new pieces in the **Load** and **Store** operations.

The constructor. Let's start by looking at the constructor. When we call the constructor we pass in a name argument. The name is the actual name of the movie record that we want to load or create. Then if we have a name, we load the record information. The Movie Control Window will either call the constructor with **NULL** to create a new movie record, or it will call the constructor with the valid movie name.

```
    MOVIE^INFORMATION::MOVIE_INFORMATION(ZIL_ICHAR *name)  :
      UIW_WINDOW("MOVIE_INFORMATION", defaultStorage)
```

```
{
  // Center the window on the screen.
  windowManager->Center(this);
  // Load the record information,
  if (name)
    Load(name, MOVIECONTROL:: dataFile);
}
```

In **Event()**, we only look for the additional request of *OPT_MOVIE_STORE*. In this case, we call the **Store()** member function to pass a NULL name and a pointer to the actual data file.

```
EVENT_TYPE M0VIE_INF0RMATI0N::Event(const UI_EVENT &event)
{
  // Check for special requests.
  EVENT_TYPE ccode = event.type;
  if (ccode == 0PT_HELP)
    helpSystem->DisplayHelp(windowManager);
  else if  (ccode = OPT_MOVIE_STORE)
    Store(ZIL_NULLP(ZIL_ICHAR), MOVIE_CONTROL:: dataFile);
  else
    ccode = UIW_WINDOW::Event(event);
  return (ccode);
}
```

We need not associate the actual name with **Store()**, because the window's title field contains the name of the movie.

The Load() function. Now that we're done with the constructor, let's look at the **Load()** function. **Load()** creates the storage record and loads the data in two steps. First, it creates the record by constructing a load object called **ZIL_STORAGE_OBJECT_READ_ONLY,** and by passing the data file handle and the load name.

```
Z IL_STORAGE_OB JECT__READ_0NLY lObject (*file, name, 0);
```

This code opens a record in the data file. Then it loads its information with the overloaded **Load()** function.

```
lObject.Load(&sData, MAXLENGTH); // title.
Get(FLD_INFO_TITLE)->Information(I_SET_TEXT,
  sData);
10bject.Load(&iData); // copyright,
int copyright = iData;
Get(FLD_INFO_COPYRIGHT)->Information(I_SET_VALUE,
  &copyright);
10bject.Load(&iData); // length,
int length = iData;
Get(FLD_INFO_LENGTH)->Information(I_SET_VALUE,
```

```
  &length);
1Object.Load(&sData, MAX_LENGTH); // director.
Get(FLD_INF0_DIRECTOR)->Information(I_SET_TEXT,
  sData);
1Object.Load(&sData, MAX_LENGTH); // leads.
Get(FLD_INF0_LEADS)->Information(I_SET_TEXT,
  sData);
lObject.Load(&sData, MAX_LENGTH); // description.
Get(FLD_INF0_DESCRIPTI0N)->Information(I_SET_TEXT,
  sData);
```

In this code, we use a 16-bit integer and character array overloads for the **Load()** function. OpenZinc automatically defines many overloads for the **Load()** function. Here is a list of some of them:

```
virtual int Load(ZIL_INT16 *value);
virtual int Load(ZIL_UINT16 *value);
virtual int Load(ZIL_INT32 *value);
virtual int Load(ZIL_UINT32 *value);
virtual int Load(ZIL_UINT8 *value);
virtual int Load(ZIL_INT8 *value);
virtual int Load(void *buff, int size, int length);
virtual int Load(ZIL_ICHAR *string, int length);
virtual int Load(ZIL_ICHAR **string);
```

Once we've loaded the record's information, we set the appropriate field inside our movie information record. First, we load the information for the movie's title by calling *object.***Load()** and by passing the pointer to *sData,* with the maximum length of the read buffer.

```
1Object.Load(&sData, MAX_LENGTH);
```

Then we set the *FLD_INFO_TITLE* information to the loaded title.

```
Get(FLD_INFO_TITLE)->Information(I_SET_TEXT, sData);
```

We do similar type calls for the copyright, length, director, actors, and description, though the copyright information is an integer.

```
lObject.Load(&iData);
```

For the length, we load an integer value.

```
lObject.Load(&iData);
```

For the director field, we load a string.

```
lObject.Load(&sData);
```

For the actors and actresses, we also load a string.

```
lObject.Load(&sData);
```

Finally, for the description field, we again load a string.

```
10bject.Load(&sData);
```

Above, we load information from a **.DAT** file used as a flat-file database, but we could load it from a third-party database instead. However, the focus of this tutorial is not how to use a third-party database, but how to build an application by architecting it first and implementing it later. The reason we even mention third-party databases? This is where OpenZinc and third-party databases intersect. If you were to exceed the scope of this tutorial and store movie information in a third-party database, this is where you would load that information.

The Store() function. The **Store()** operation mirrors the **Load()** operation, but we create and store movie information rather than load it. This code gets the record name from the window title.

```
if (!name || !name[0])
  Get(FLD_INFO_TITLE)->Information(I_GET_TEXT, &name);
```

The next line creates a new object and gives us read and write privileges.

```
ZIL_STORAGE_OBJECT sObject(*file, name, 0,
  UIS_CREATE | UIS_READWRITE);
```

Next, we store the record's information, first getting the data from the field, then storing the information. For example, the title field gets the information from *FLD_INFO_TITLE*, and calls *sObject.*Store(), with *sData* as its parameter. Subsequently, we do the same for the copyright, length, director, actors and actresses, and description fields.

```
Get(FLD_INFO_TITLE)->Information(I_GET_TEXT,
  SsData);
sObject.Store(sData); // title,
int copyright;
Get(FLD_INFO_COPYRIGHT)->Information(I_GET_VALUE,
  Scopyright);
iData = (ZIL_INT16)copyright; // copyright.
sObject.Store(iData);
int length;
Get(FLD_INFO_LENGTH)->Information(I_GET_VALUE,
  &length);
iData = (ZIL_INT16)length; // length.
sObject.Store(iData);
Get(FLD_INFO_DIRECTOR)->Information(I_GET_TEXT,
  &sData);
sObject.Store(sData); // director.
Get(FLD_INFO_LEADS)->Information(I_GET_TEXT,
  &sData);
```

```
sObject.Store(sData); // leads.
Get(FLD_INF0_DESCRIPTI0N)->Information(I_GET_TEXT,
  SsData);
sObject.Store(sData); // description.
```

Below is a representation of the composition of files in the movie catalog library:
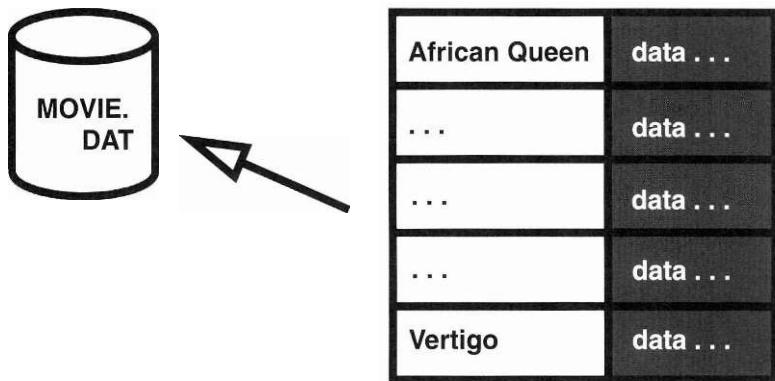
Summarizing the Movie Information c ﹐ a s s

MOVIE_INFORMATION might seem complicated, so let's summarize what it does so we can keep a clear picture in our minds. If we pass a name to MOVIE_INFORMATION's constructor, it loads the information from the data file; but if we pass it a NULL, it creates a new movie record. **Event()** only checks for the *OPT_MOVIE_STORE* message in **Event()**. If it receives the message, **Event()** calls **Store()**, which stores the movie record.

Admiring our handiwork

We have now filled in MOVIE's holes. Having architected it in the earlier tutorial, we were able to implement storage and management in this tutorial without any hassles. This hassle-free implementation shows that we can write a properly designed program—that is, a program that functions by passing messages between objects—by creating the architecture first, and implementing the main functionality later.

Go ahead and compile and run it.
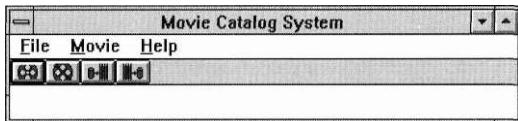
Once it's running, select one of the movie operations.

Movie I Create works as in previous tutorials, but now we can enter information into the title, copyright, length, director, actors, and description fields, and save the information by pressing **Save.**

Movie I Delete brings up the Movie Selection window. Clicking on an item in the list updates the window's title field—our list item tells the window to update the title field. If we press **OK,** the button sends the *OPT_MOVIE_OK* message to the Movie Selection window. The window then resets the movie catalog system's movie name, and sends the message for the movie catalog to delete or load a movie.

Movie I Load also brings up the Movie Selection window. Selecting *Bridge on the River Kwai, The,* from the list brings up the record for *Bridge on the River Kwai, The* in a Movie Information window.
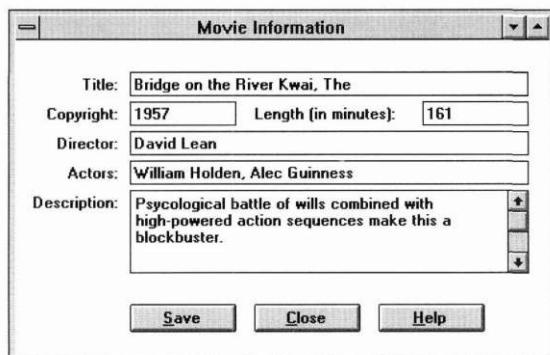
We can view or change the information inside this record and then save the changes by pressing **Save.** We can cancel and close the window by selecting **Close.**

Our movie catalog system is nearly complete. Take a few minutes to play with it, then exit the application by selecting **File I Exit.**

Conclusion

T

.**A.** his tutorial showed how we could write a program by architecting it first and implementing the main functionality later. Understanding the connections between the .DAT file, the windows, the data file, and the source code sends us well on our way to understanding complex applications.

Chapter 8

Making Movie Robust

in the last tutorial, we described the construction of the movie application. In this tutorial we will make the movie application more robust by including an exit function, adding geometry management, including an error system with error checking, and by beefing up the help system. The addition of these features will combine creating new windows in OpenZinc Designer with adding functionality in our code.

Key Concepts

exit functions

geometry management

the error system and error checking

Working with MOVIE6

Components of MOVIE6

In Figure 1 on page 58, we discussed the components of the **MOVIE** application we've been working with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE6,** the sixth component of **MOVIE.** The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
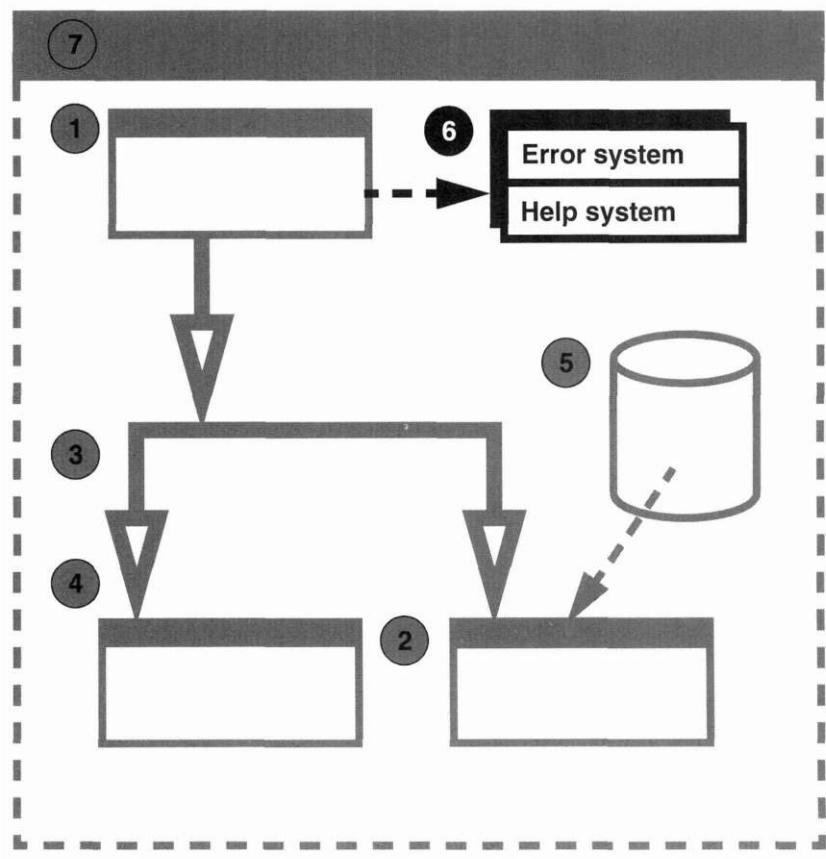
Source files

Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in **/OpenZinc/TUTOR/MOVIE.**

TABLE 9. Components of MOVIE6

Type of file	Name of file	Description of file
User-provided files	**MOVIE6.CPP**	The main program
	MOVIE6.HPP	Class definitions, identifications, and messages
	MOVIE. DAT	User data storage
Designer-generated files	**P MOVIE6.CPP**	Code for tying Designer objects to our program
	P MOVIE6.HPP	Identifications and help contexts
	P MOVIE6.DAT	Persistent object storage

Adding features to the Movie Control window

Let's start by adding four features to the Movie Control window—a minimize icon, an exit window, a status bar, and a help option.

Minimize icon

To add a minimize icon to the Movie Control window, enter OpenZinc Designer, open **P_MOVIE6.DAT,** and then invoke the Image Editor.
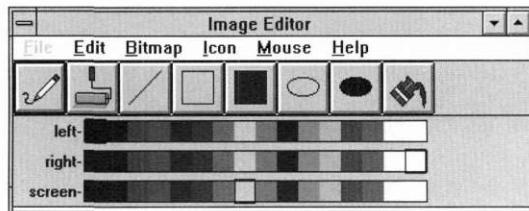
Import the icon from the **P_MOVIE.DAT** file by

1. selecting **Icon I Import,** then by

2. selecting **P_MOVIE.DAT,** then by

3. selecting the icon entitled **movie_icon.**
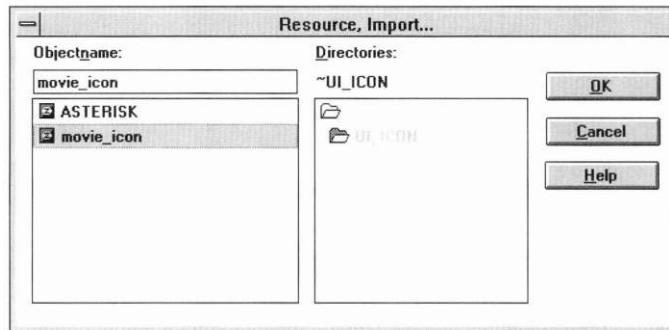
Once the icon is imported into the **P_MOVIE6.DAT** file, we can load the Movie Control window. Associate the Movie Icon with the Movie Catalog System window by invoking the window's information notebook and selecting the **movie_icon** from the **MinIcon** list.
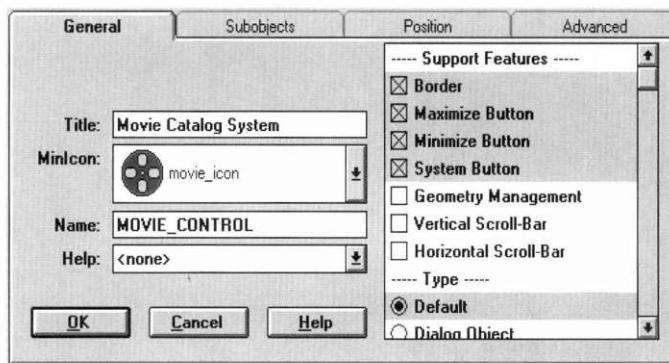
Save the change by pressing OK, then save the window by selecting Window I Store. The minimize icon is now associated with the movie catalog system and will appear when the Movie Control window is minimized. To view this feature select Window I Test then select the minimize button from the catalog system window.
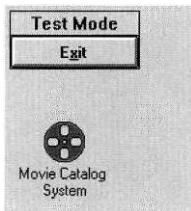
Exit window

Now let's associate an exit window with our application. The exit window will contain text that states that pressing OK will close the application. We will also attach an icon and two buttons, OK and Cancel, and modify the title to read *Exit Application.*
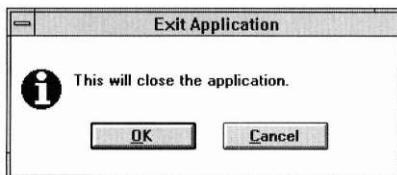
Create a window and modify the following:

1. Enter the new title

    `Exit Application`

2. Enter the name

    EXIT_APPLICATION

    into the Name field, then

3. Deselect the maximize and minimize options from the support features.

4. Change the window's style by selecting **Dialog Object, Don't size,** and **Modal** from the options list.
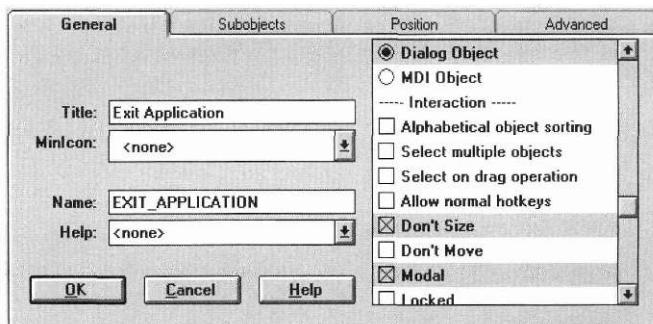
Save the changes, then create an icon field and place it at position (1,1). Enter the information notebook for the new icon field. When OpenZinc creates a new **.DAT** file, it automatically inserts five default icons into the file. These include the application, asterisk, exclamation, hand, and question icons. To use the asterisk icon, select it from the **Image** combo box.
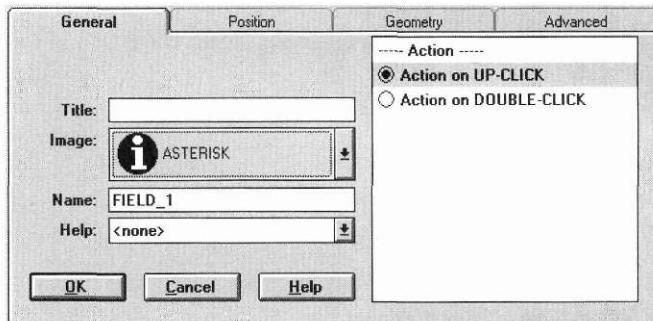
By default, the icon image does not have a title and is not selectable by the user. We will keep the default information associated with this icon.

Now create a string field at position (7,1) and give it a size of (35,1). Edit the information by bringing up the string information notebook and typing in the text

```
This will close the application.
```

Change the default length to 64 characters, then change the mode of interaction on the advanced page to be **View only** and **Noncurrent.**
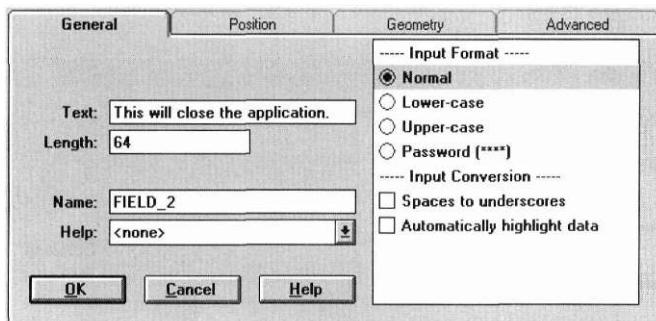
Now create the two buttons that will have the **OK** and **Cancel** options. Place the **OK** button at position (9,2) with size (12,1), and place the **Cancel** button at position (24,2) with size (12,1). Change the information associated with the **OK** button by invoking the information notebook, entering the text

```
&OK
```

in the text field, changing the value to

1,000

for the value field, and by setting the flags *Send user message* and *Set as default button.* Save the changes, then edit the **Cancel** button. Change the text by entering the string

```
&Cancel
```

and then the value

-ii.

Then set the *Send user message* flag. (*S_CLOSE* has the value -11.) Save the information for the exit window by selecting **Window** I **Store,** then **File** I **Save.** Then exit the application.

We must do three things to connect an exit function to the source code. First, we need to define the exit function in one of our classes. Since this is a control operation, we will define **Exit** in the class **MOVIE_CONTROL.**

```
class MOVIE_CONTROL : public UIW_WINDOW
{

public:

private:
```

```
static EVENT_TYPE Exit(UI_DISPLAY *, UI_EVENT_MANAGER *,
  UI_WINDOW_MANAGER *windowManager);
```

Next, we need to set the Window Manager's exit function to point to our exit function.

```
windowManager->exitFunction = MOVIE_CONTROL::Exit;
```

This tells the Window Manager that it must call our exit function before it actually exits the application. This gives our application final control to tell whether we should continue or exit. We will display one additional window to confirm the exit process. Finally, we define the actual MOVIE_CONTROL::Exit() function.

```
EVENT_TYPE MOVIE_CONTROL::Exit(UIJDISPLAY *,
  UI_EVENT_MANAGER *, UI_WINDOW_MANAGER *windowManager)
{
  // Read the exit window.
  UI_ERROR_STUB::Beep();
  UIW_WINDOW *windcw = new UIW_WIISlDCW( "EXIT_APPLICATION", def aultStorage);
  windowManager->Center(window);
  *windowManager + window;
  return (S_CONTINUE);
}
```

The steps in this process are:

1. Sound the bell on the computer,

2. Load in the exit application window,

3. Center the window on the display,

4. Attach the window to the display, and

5. Return the message *S_CONT1NUE*. This message tells the Window Manager we want to continue our application.

When we run the application with these changes, and select File I Exit, the exit application window appears on the screen.
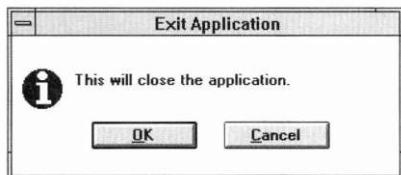
We can continue with our program by selecting Cancel or we can quit by selecting OK. When we select OK, the final exit message is sent to the Window Manager, our application windows are removed from the screen, and control returns to the operating system.

Status bar

Now, let's return to the Designer to add a status bar to the Movie Control window. In the Designer, load in MOVIE_CONTROL from the resource file. Select the status bar object from the button bar, then place the object in the movie catalog system window.
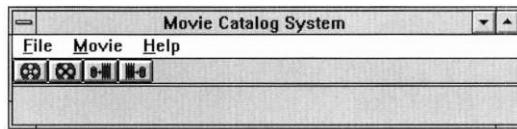
Size the window to be one cell taller— to (60,5)—so that all the information fits inside the window. Place a prompt inside the status bar at position (1,1). Then create and place a string object and stretch the object so that it fits across the status bar.
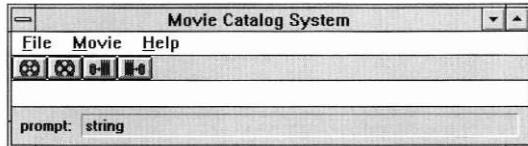
Change the name and text associated with the movie prompt by invoking the prompt information notebook and entering the text

```
Movie:
```

with name

STATUS_PMOVIE_TITLE.

General	Position	Geometry	Advanced

Text: Movie:

Name: STATUS_PMOVIE_TITLE

OK   Cancel   Help
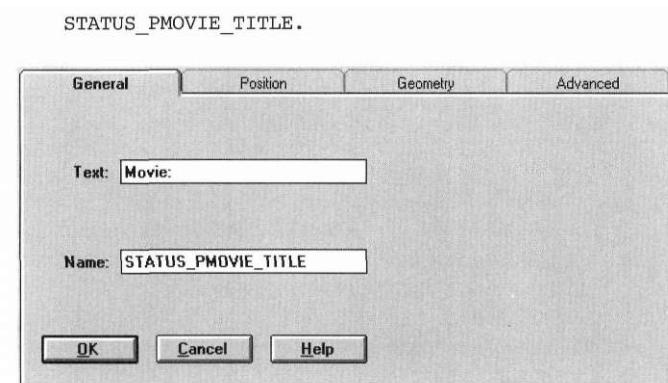
Change the name of the string object by invoking the string information notebook and changing the Name field to

```
STATUS_MOVIE_TITLE
```

giving it no default text. Then change the length to 128. When we run our application, this field will be changed any time we load in a new movie record, or if we have multiple records on the screen and change the focus from one movie record to another.

Let's see how this is done by saving the window, exiting the Designer, and viewing the new code in P_MOVIE6.CPP.

Here's the design of the status bar:

1. The Movie Control window actually controls the presentation of the status bar.

2. Each of the movie information windows will send a message to the Movie Control window to update the status bar information.

The Movie Control window updates the status bar from Information( ) when it receives the information request *I_UPDATE_STATUS*.

```
void *CONTROL_WINDOW::Information(ZIL_INFO_REQUEST request,
  void *data, ZIL_OBJECTID objectID)
{
  if (request == I_UPDATE_STATUS)
  {
    Get(STATUS_MOVIE_TITLE)->Information(I_SET_TEXT, _movieName);
    _movieName[0] = '\0';
  }
  else
    data = UIW_WINDOW::Information(request, data, objectID);
  return (data);
```

```
    }
```

This request is sent to the Movie Control window any time we receive an S_CURRENT or *S_NON_CURRENT* message in the MOVIEINFORMA-TION::Event() function.

```
else if (ccode == S_CURRENT || ccode == S_NON_CURRENT)
{
  ccode = UIW_WINDOW::Event(event);
  UI_WINDOW_OBJECT *window =
    windowManager->Get("MOVIE_CONTROL");
  if (window && ccode == S_CURRENT)
  {
    Get(FLD_INFO_TITLE)->Information(I_COPY_TEXT,
        MOVIE_CONTROL::_movieName);
    window->Event(I_UPDATE_STATUS, ZIL_NULLP(void));
  }
}
```

Compile and run the application again to see the status bar change.

Error handling   Now let's look at how we can beef up the movie catalog's record information. Launch the Designer and load the movie information window. The main things that we want to change in this record are the copyright date, length, and code to ensure that we have a valid title.

The copyright date is changed by entering the copyright field information window and specifying a valid range for the date. In our case let's enter the range in years from 1900 to 2020. This is done by moving to the range field, and entering

```
1900..2020
```
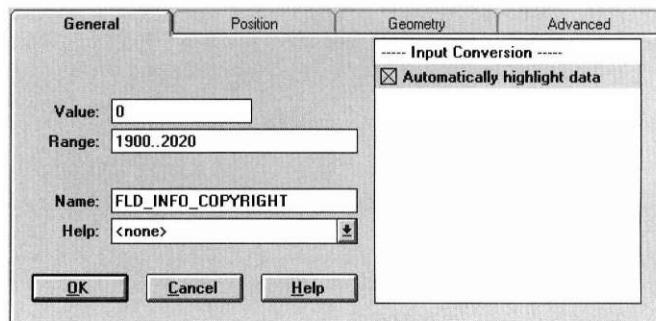
Now when the user enters a date outside of this range, an error window will appear on the screen, indicating that they have entered a copyright date that is outside of the years 1900 to 2020.

Next, let's change the length of the movie to be a minimum of 5 minutes and a maximum of 5 hours. This is done by invoking the length field information notebook and by entering the range

```
5..300
```

where 300 is (5 x 60) minutes.
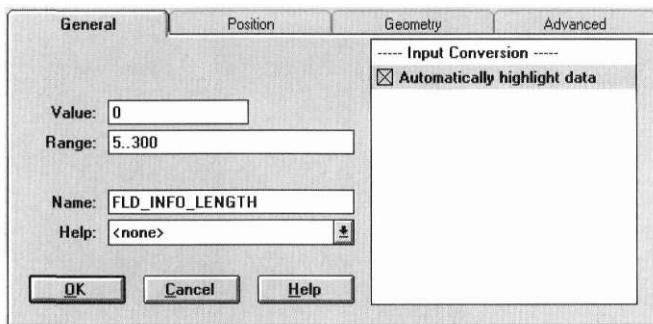
These two changes add validation for the copyright and length.

To insure a valid title, we must go to the source code. Save the changes to the movie information window, then exit the Designer. We will display an error any time the Store operation is called and we do not have a name for the title. This is done by checking the name argument for NULL.

```
if (Iname || !name[0])
  errorSystem->ReportError(windowManager, WOS_NO_STATUS,
    "Sorry, cannot save the movie without a name.");
```

If an invalid name is entered, we call the error system with the message

```
Big Time Movie Error
```

The final thing that we need to do is to actually invoke OpenZinc Application Framework's error system. This is done by attaching a new error system to the base UI_WINDOW_OBJECT::trrar.S>v/ew member.

```
UI_WINDOW OBJECT:rerrorSystem = new UI_ERROR_SYSTEM;
```

If we recompiled the application and launched MOVIE6, we could test the title, copyright, and length error handling by typing invalid data into any of these fields.

Adding geometry management

Movie Selection

Let's now beef up the movie selection by adding geometry management. Geometry management is a programming option where we can tie window objects to a specific position of its parent window or to a side of the window, so the object moves or stretches when we size the window. Let's first look to see how we can size a window and have the vertical list and buttons move appropriately.

After you load the movie selection window, bring up its information note-book. To turn on geometry management, select the **Geometry management** option from the support features. At this time, we will also turn off the **Dialog Object** option, which we turned on in an earlier tutorial—doing so will allow us to resize the window. (Dialog windows cannot be sized in some environments.) Turn this option off by selecting the **Default** option in the **Type** section of the options list.
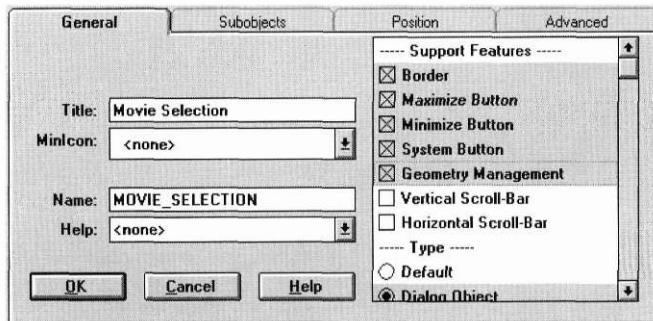
Individual Objects

Once we have turned on the window's geometry management, we can turn on the geometry management for individual objects. We want to turn on geometry management for the **OK, Cancel,** and **Help** buttons, as well as for the vertical list and string that contains the name of our movie title.

The goal behind geometry management is to allow us to size the window to a larger size and have the buttons follow the window border at the bottom, and to have the list and title fields follow the window and grow when the window grows larger on the bottom right side.
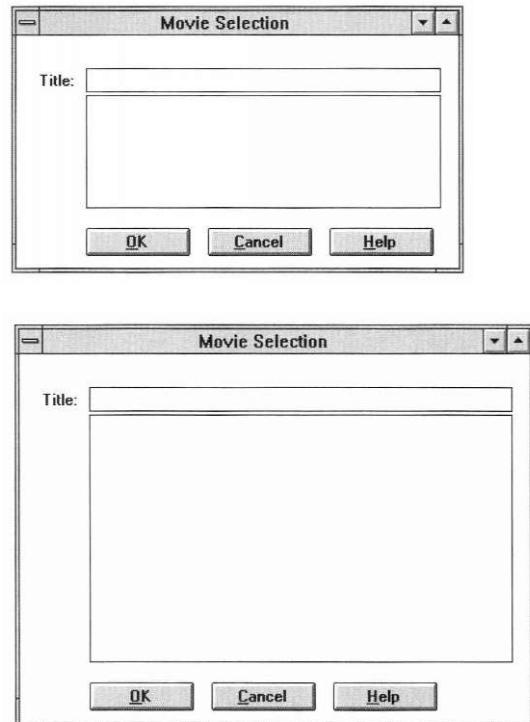
Do this by attaching items to the vertical list, the title field, and to each of the buttons. Let's start by attaching geometry to each of the buttons. First, choose the OK button. To attach geometry management, we will need to do change the coordinates for our button to be mini-cells, and create the attachments on the Geometry page of our information notebook.

The minicell coordinates are contained in the second column of numbers on the **Position/Size** group. Choose this coordinate scale by selecting the minicell bitmap button.
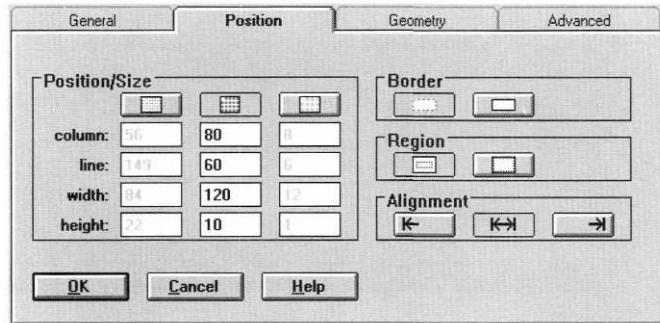
The reason that we want to change the type of coordinate is so that it will align more closely with the bottom of the window. If we were to attach on cell boundaries, the buttons and the vertical list would collide with the bor-

Now, move to the **Geometry** page. Attach the button to the bottom of the window by selecting the bottom attachment located on the bottom-left portion of the geometry page.
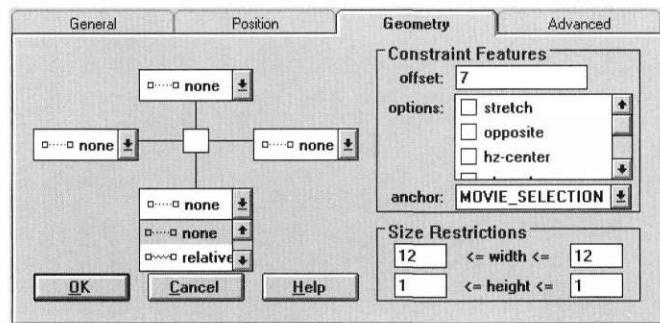
Choose an **absolute** coordinate for attachment by selecting **absolute** from the combo box.
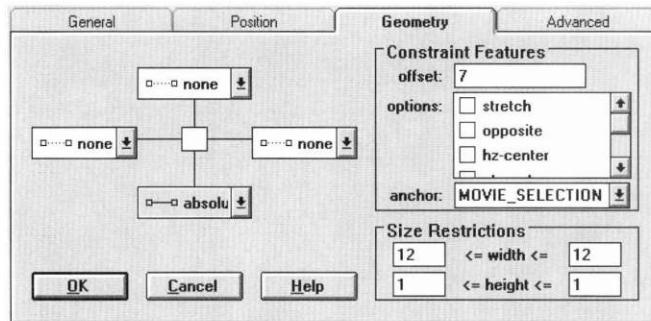
An absolute attachment is like connecting a physical wire from our object to the parent window. And attaching a relative constraint is like attaching a spring from our object to the window.

We are now working in mini-cell coordinates, so in the constraint features, under offset, we want to enter a mini-cell value that represents appropriate space between the bottom of the window and our button. Here, let's set the mini-cell coordinate to 4.
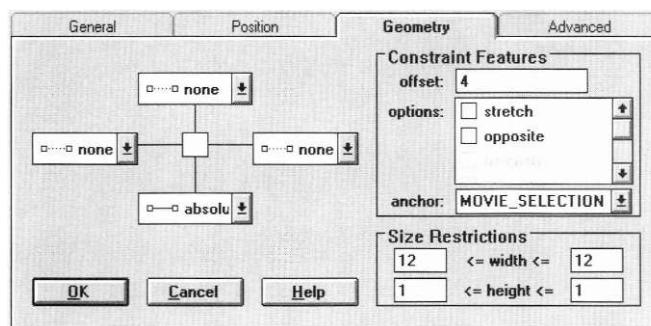
Save these changes by pressing **OK.** We have now attached the constraint to our **OK** button. Follow the same process to attach the **Cancel** and **Help** buttons to the bottom of the window. Remember, first change the coordinates of each object to *mini-cell*, then change the geometry management to *absolute* bottom attachment, with a mini-cell offset of 4.

Test these geometry management constraints by sizing the edit window on the screen. Notice that when enlarging the window, the buttons move down with the window.
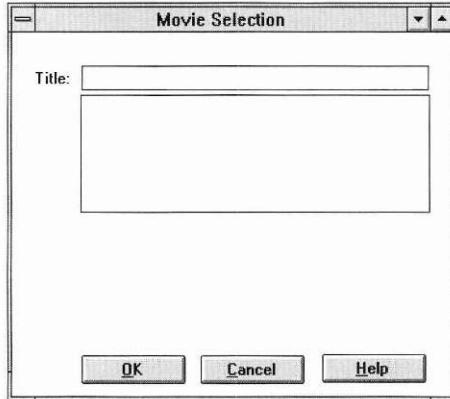
Change the geometry for **vertical list** by entering the vertical list information notebook and moving to the **Geometry** page. With this object we can keep cell coordinates. Attach an *absolute* bottom constraint to the vertical list and enter an offset of 2. This means that the vertical list will move within two cells of the bottom of the window. Next, set the **stretch** feature in the **Constraint Features** options list so that it will grow when the window grows, and shrink when the window shrinks.
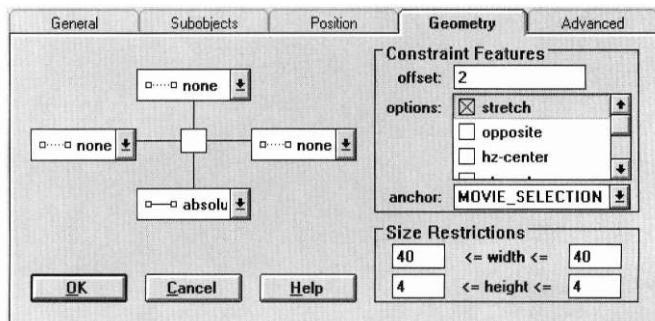
We should also add a right constraint to our vertical list so that it will grow on the right side if we size the window. This is done by selecting an absolute constraint on the right side and by once again entering the offset value of 2. Allow stretching of the width by selecting the **stretch** option from the options list.

We have now attached geometry with the vertical list. When the window grows from either the right or the bottom the vertical list will stretch with the window.

Let's complete the geometry management by selecting the title field, moving to the geometry page and adding an absolute constraint on the right side. Enter an offset of 2 and select the stretch option.
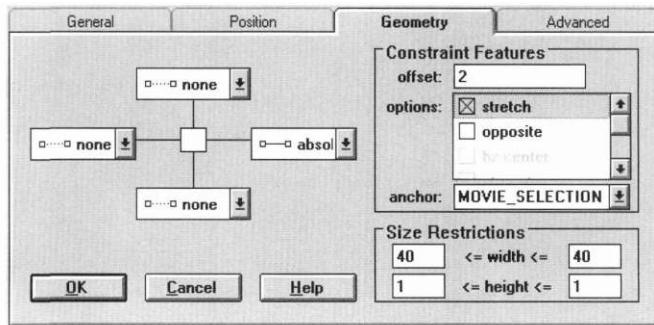
We have now tied constraints to the title field, the vertical list, and all of our buttons. Save the changes and exit the application. You can now test this feature by recompiling and running the application. As you invoke the movie application and load the movie selection window, notice how you can size the window and see the buttons, vertical list, and title stretch with the size of the window.

Refining the help system

This tutorial's final area of emphasis is to include context-specific help. Recall that we connected general help to our application by introducing the message *OPT_HELP,* and then connecting it with each of the pull-down help items and the Help buttons located at the bottom of the information and selection windows.
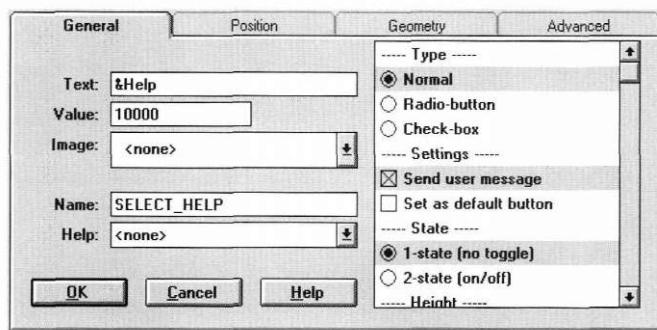
To specify context-sensitive help, we associate help contexts with pull-down items and dialog windows, then call the help system with the appropriate help context in our code. The Designer allows us to connect help contexts to pull-down items and dialog windows.

Let's look at how to import the help contexts. Launch the Designer and open the P_MOVIE6.DAT file.

Import all the help contexts from P_MOYIE.DAT by calling the Help Editor, then by selecting the Context I Import option from the pull-down menu.
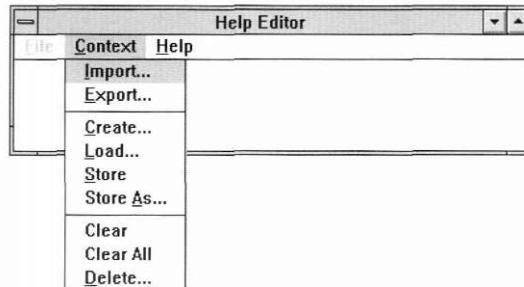
Select the file **P_MOVIE.DAT** for importing, then select all of the help contexts from the resource list.
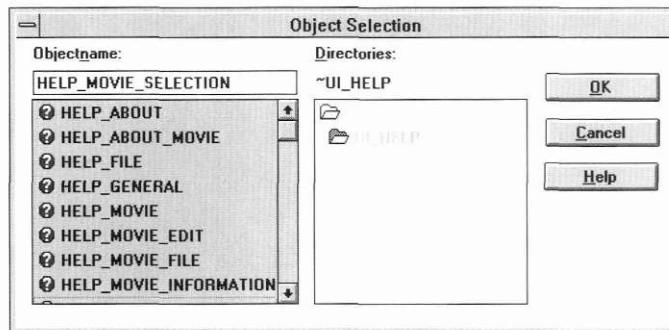
We now have a full set of help contexts from which to choose. The names of the imported help contexts are associated directly with the type of help we will be identifying, except that each help context has the prefix *HELP_*.

Item help

Let's start the association by:

1. minimizing the Help Editor,

2. loading the Control Window,

3. bringing up the control window's information notebook,

4. bring up the window's **Subobjects** page,

5. moving to the pull-down menu's directory,

6. moving to the **FILE_OPTION** directory,

7. moving down to the first help item, **ABOUT_OPTION,** located under the **File** option.
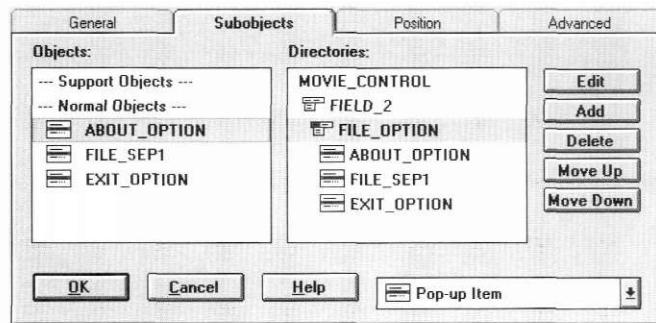
Open the item's information notebook, then choose the *HELP_ABOUT* option from the help field.
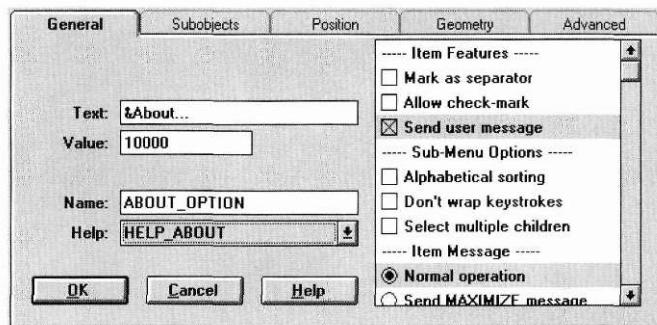
Press **OK** to save the changes.

Now associate the remaining menu items with these help contexts:

- *HELP_MOVIE_FILE* with the **File** menu item
- *HELP_MOVIE_EDIT* with the **Edit** menu item
- *HELP_ABOUT_MOVIE* with the **About Movie Catalog...** menu item

Dialog help

Now let's connect help for the selection and information dialog windows. We want to associate help with the general dialog window, then make sure the system calls it when the user presses the system help key while a field in the dialog window is current, or when the user selects the **Help** button in a dialog window.

We specify the help context by bringing up the window's information notebook, then by selecting the appropriate help context. The following help contexts correspond with our dialog windows:

- *HELP_MOVIE_SELECTION* with the selection dialog.
- *HELP_MOVIE_INFORMATION* with the information dialog.

The source code does not require any modification. Remember, we intercept the *OPT_HELP* message and call the help system with a variable *helpContext*.

```
EVENT_TYPE MOVIE_SELECTION::Event(const UI_EVENT Sevent)
{
    // Check for special requests.
    EVENT_TYPE ccode = event.type;
    if (ccode == OPT HELP)
      helpSystem->DisplayHelp(windowManager,
        event.windowObject->helpContext);
```

The variable *helpContext* is the identification number of the help we want to display. By default, *helpContext* is defined to be *HELP_CONTEXT_NONE.* We now have changed the values for the information and selection windows to *HELP_MOVIE_INFORMATION* and *HELP_MOVIE_SELECTION.*

General application help

We must make a final help connection in the source code; we must specify *HELP_GENERAL* in the second argument of the help system's constructor.

```
UI_WINDOW_OBJECT::helpSystem = new UI_HELP_SYSTEM(_fileName,
    HELP_GENERAL);
```

Specifying this argument causes invokes *HELP_GENERAL* any time we call the help system and where no other help is available. For example, if we had just launched **MOVIE** and pressed the help key, the application would display the general help window.

Run time

We have now connected help to various parts of our application. Take a few minutes to recompile and view the application with the added help information.

Conclusion

In this tutorial we have looked at several ways in which we can beef up our application. There are many things that we can do to the application that involve: modifying the windows in OpenZinc Designer, creating new windows, and modifying the source code.

Through all of our modifications, we have worked the architecture of our program and the overall design of our code. We have simply added new feature points where we can either bring up a new window or add options using the Designer that help our windows look better.

Since we're finished with this section, you may want to think of additional ways to strengthen **MOVIE** to enhance or improve its strength.

Chapter 9

Generating an Internationalized Application

We have now covered a lot of territory with the movie application. We

need to address one final component—globalizing MOVIE. Here, we will examine the Message Editor, discuss importing locale and language information, introduce delta storage, and finally, learn to use Unicode with our application.

Key Concepts

globalizing an application

working with Unicode

Working with MOVIE7

Components of MOVIE7

In Figure **1** on page 58, we discussed the components of the **MOVIE** application we've worked with in this series of tutorials. In this part of the tutorial, we'll be working with **MOVIE7,** the seventh and last component of **MOVIE. (MOVIE7** is exactly equivalent to **MOVIE,** which we compiled and ran in the first **MOVIE** tutorial.) The diagram below shows the components we're working on in relationship to the other components of the tutorial. (The components we're not working with are grayed out.)
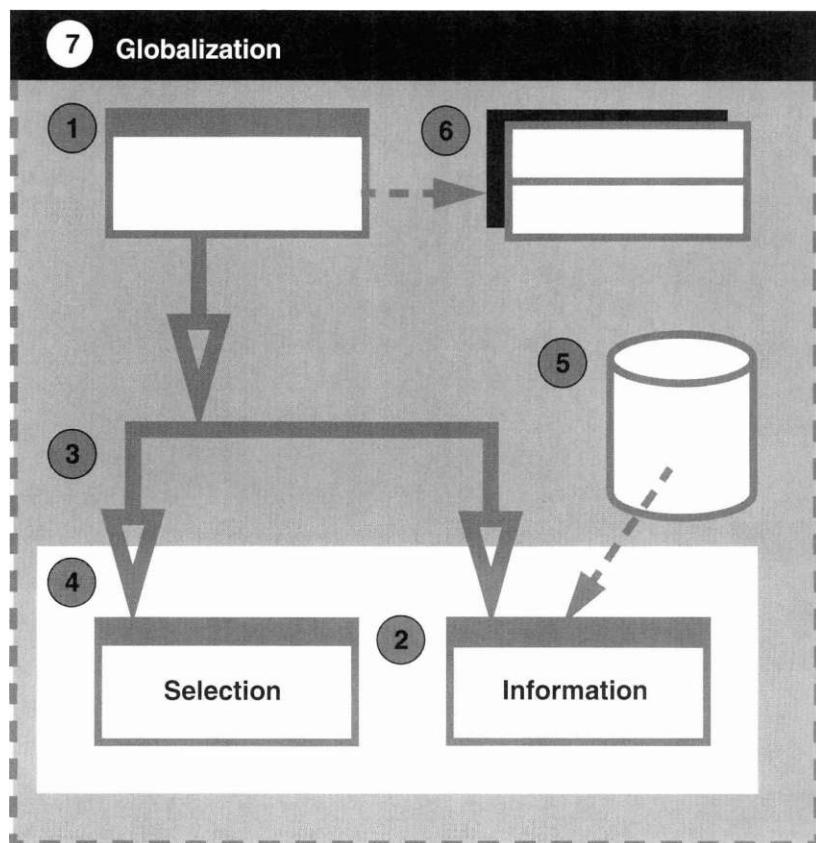
Source files   Below is a table of source files we'll be working with in this part of the tutorial. We can find these files in /OpenZinc/TUTOR/MOVIE.

TABLE 10. Components of MOVIE7

Type of file	Name of file	Description of file
User-pro-vided files	**MOVIE7.CPP**	The main program
	MOVIE7.HPP	Class definitions, identi-fications, and messages
	MOVIE. DAT	User data storage
Designer-generated files	**P MOVIE7.CPP**	Code for tying Designer objects to our program
	P MOVIE7.HPP	Identifications and help contexts
	P MOVIE7.DAT	Persistent object storage

Message Editor

Let's start by looking at the conversion of internal strings in our application. Recall the error message that indicated that the application could not save a movie record without a movie title. When we write applications for multiple languages and locales, we cannot use embedded strings written in English. In addition to the store error message already in the application, we will implement three more error messages: one for movie delete, one for movie load, and one for movie selection.

To internationalize this portion of our program we need to create these strings using abstract names. The way this is done in OpenZinc is by using the Designer to construct a message table that has a number identifier and a string that can be stored and loaded to and from disk.

Construction of a message table is accomplished through the Message Editor module of the Designer. In our application we will create a message table and then associate four strings with four unique identifiers. We'll create the

identifiers   *ZMSG_STORE_ERROR,*   *ZMSG_LOAD_ERR OR,*   *ZMSG_ DELETE_ERROR,* and *ZMSG_NAME   ERROR.* Let's add these messages to our data file. Start the Designer and invoke the Message Editor.
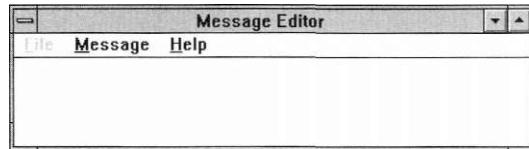
Create a new message table by selecting Message I Create.
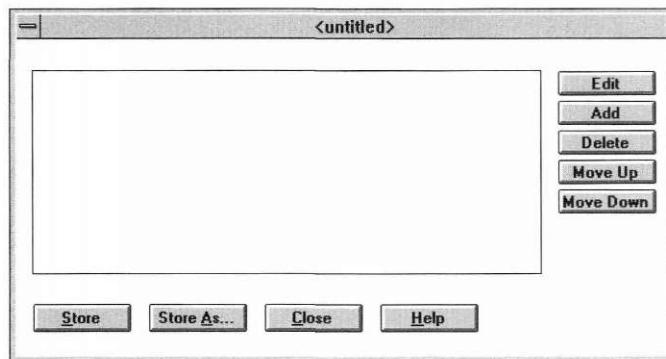
Add four messages to the table by selecting Add four times.
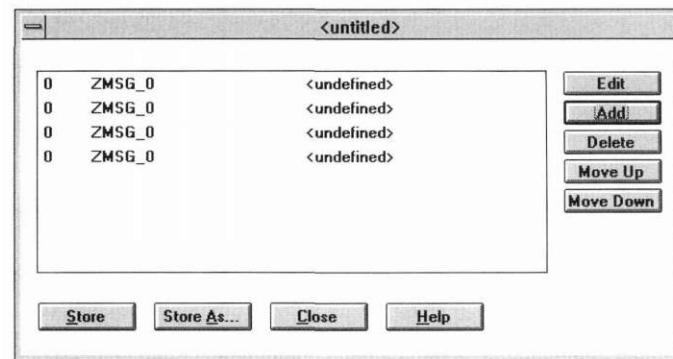
To edit any item, click on it and press **Edit,** or double-click it.

```
┌─────────────────────────────────────────────┐
│ ▭│          Message Information              │
├─────────────────────────────────────────────┤
│                                             │
│ Message:    │<undefined>                   │ │
│ NumberID:   │0                    │         │
│ StringID:   │ZMSG_0                        │ │
│                                             │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐   │
│  │    OK    │  │  Cancel  │  │   Help   │   │
│  └──────────┘  └──────────┘  └──────────┘   │
└─────────────────────────────────────────────┘
```
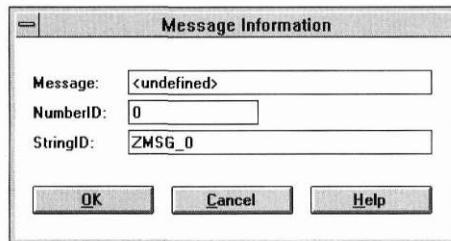
The message information window has three components: *message, numberID,* and *stringID.* The *message* field is the message we want to display. The *numberID* field contains a programming number that will be associated with the message. The *string ID* is a **#define** variable that identifies in code what our *numberID* will be.

Edit the four embedded movie string messages to see exactly how the message information is correlated. Do the following to create the information for a **Store()** error:

1. Select the top message item and enter:

   ```
   Store error: cannot save the movie without a name into the
   field.
   ```

2. Enter the value
   ```
   1
   ```

   into the *numberID* field.

3. Enter the string

   ```
   ZMSG_STORE_ERROR
   ```
   into the *stringID* field.
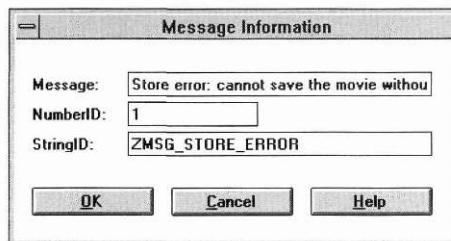
```
┌─────────────────────────────────────────────┐
│ ▭│          Message Information              │
├─────────────────────────────────────────────┤
│                                             │
│ Message:   │Store error: cannot save the movie withou│ │
│ NumberID:  │1                    │          │
│ StringID:  │ZMSG_STORE_ERROR              │ │
│                                             │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐   │
│  │    OK    │  │  Cancel  │  │   Help   │   │
│  └──────────┘  └──────────┘  └──────────┘   │
└─────────────────────────────────────────────┘
```

Press OK to save the changes.

Now create a message for the **Load()** error. Select the second item, then in the message field, enter the text

```
Load error: cannot find the specified movie "%s"."
```

In the *numberlD* field enter the value

```
2
```

and in the *stringlD* field enter the string

```
ZMSG_LOAD_ERROR.
```

Message Information
Message:	Load error: cannot find the specified movi
NumberID:	2
StringID:	ZMSG__LOAD_ERROR
OK	Cancel	Help
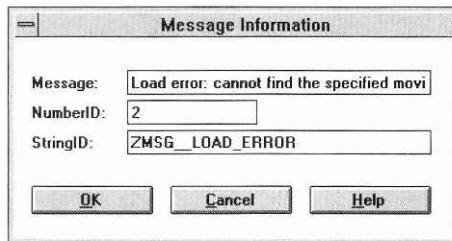
For the **Delete()** error, enter the message,

```
Delete error: cannot find the specified movie "%s"."
```

In the *numberlD* field, enter the value

```
3
```

In the *stringlD* field enter the string

```
ZMSG_DELETE_ERROR
```

Message Information
Message:	Delete error: Cannot find the specified mo
NumberID:	3
StringID:	ZMSG_DELETE_ERROR
OK	Cancel	Help

The final message occurs when no name has been entered into the selection window. Edit this message and enter the message text

```
Name error: a name must be specified to complete the operation.
```

In the *numberlD* field, enter the value

```
4
```

and in the *stringlD* field, enter the string

```
ZMSG NAME ERROR
```
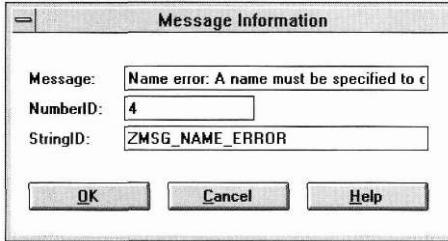

We now have four messages in the message table. Store them by selecting **Store As...** and entering the name *MSG_TABLE*. Press **OK** to save the changes.

When we save the **P_MOVIE7.DAT** file, the Designer saves four **const** values in the **P_MOVIE7.HPP** file.

```
#ifdef USE_MSG_TABLE
const ZIL_NUMBERID ZMSG_STORE_ERROR                = 0x0001;
const ZIL_NUMBERID ZMSG__LOAD_ERROR                = 0x0002;
const ZIL_NUMBERID         ZMSG_DELETE_ERROR       = 0x0003;
const ZIL_NUMBERID ZMSG_NAME_ERROR                 = 0x0004;
#endif
```

The **#ifdef USE_MSG_TABLE** entry allows us to choose whether we want to include the message table information. If we define *USE_MSG_TABLE,* then the four messages will be included in our application. The four messages contain the values 1,2,3, and 4. The name associated with each number is the text we entered when editing each message item. For example, the store error has the line *ZMSG_STORE_ERROR = 0X0001.*

When we replace strings in an application, we need to specify the constant identifier that corresponds to the message we want to display. So we replace the embedded string with a constant value. Let's see how we replace the strings in **MOVIE.**

We define an error message table by declaring *_errorMsgTable* in the Movie Control Window as a static member.

```
class MOVIE_CONTROL : public UIW_WINDOW
{
public:
    static ZIL_ICHAR *_pathName;
    static ZIL_ICHAR *_exitName;
    static ZIL_STORAGE *_dataFile;
```

```
static ZILJ3TORAGE_READ_0NLY *_intlStorage;
static ZIL_ICHAR _movieName[64];
static ZIL_LANGUAGE *_errorMsgTable;
```

Now let's look at **MOVIE7.CPP.** The first thing that we need to define when we use the message table and its string identifiers is the variable *USE_MSG_TABLE.*

```
// Include the appropriate directives.
#include <ui_win.hpp>
#include "movie7.hpp"
#define USE_MOVIE_CONTROL
#define USE_MOVIE_SELECTION
#define USE_MOVIE_INFORMATION
#define USE_MSG_TABLE
#include "p_movie7.hpp"
```

Next, initialize the message table by creating a new object of class **ZIL_LANGUAGE,** and by passing in the table name and the storage file where the message table is located.

```
_errorMsgTable = new ZIL_LANGUAGE(_tableName, _intlStorage);
```

We are now ready to use the message table in our application. The first occurrence will be in the **MovieDelete()** function. We make a call to *_errorMsgTable*->**GetMessage()** and pass the identifier *ZMSG_DELETE_ERROR.* This call returns a pointer to the requested string.

```
if (!_dataFile->FindFirstObject(__movieName))
{

  ZILICHAR *message = MOVIE_CONTROL::_errorMsgTable->
    GetMessage(ZMSG_DELETE_ERROR);
  errorSystem->ReportError(windowManager, WOS_NO_STATUS,
    message, _movieName);
}
```

The call to **GetMessage()** abstracts our string, so that we can identify a number instead of a string with the actual message. The message is then passed to the error system.

Now let's look at the message associated with **MovieLoad().**

```
if (!_dataFile->FindFirstObject(_movieName))
{

  ZIL_ICHAR *message = MOVIECONTROL::_errorMsgTable->
    GetMessage(ZMSG_LOAD_ERROR);
  errorSystem->ReportError(windowManager, WOS_NO_STATUS,
    message, _movieName);
}
```

The **GetMessage()** call is just like **MovieDelete()**, except that the message identifier is *ZMSG__LOAD_ERROR*. This causes the program to read the *Load error:* message instead of the *Delete error:* message.

The **MOVIE_SELECTION::Event()** and **MOVIE_SELECTION::Store()** error message retrieval is just like **MovieDelete()** and **MovieLoad()**, except that the message identifier will correspond to the appropriate error message.

```
EVENT_TYPE MOVIE_SELECTION::Event(const UI_EVENT &event)
{

    if(!MOVIE_CONTROL::_movieName[0])
    {
      ZILICHAR *message = MOVIE_CONTROL::_errorMsgTable->
        GetMessage(ZMSG_NAME_ERROR);
      errorSystem->ReportError(windowManager, WOS_NO_STATUS,
        message);
    }


void MOVIE_INFORMATION::Store(const ZIL_ICHAR *name,
  ZIL_STORAGE *file, ZIL_STORAGE_OBJECT *, UI_ITEM *, UI_ITEM *)
{

    if (!name || !name[0])
    {
      ZIL_ICHAR *message = MOVIECONTROL::_errorMsgTable->
        GetMessage(ZMSGNAMEERROR);
      errorSystem->ReportError(windowManager, WOS_NO_STATUS,
        message);
    }
```

In addition to placing the error messages in a message table, we also placed the names of the windows we created in the Designer in a message table. We won't need to change these names if we change languages, but placing them in a message table ensures Unicode compatibility.

We'll begin discussing Unicode in a moment. Right now, all we need remember is that we need to remap hard-coded strings to define the message in the Message Editor, then to replace the code with the embedded string with the appropriate message identifier. In a moment we will look at how we can change this message table to different languages without changing the executable.

Using multiple languages

Now let's look at the method used to associate multiple languages with an application. We'll first look at the so-called ISO languages, deferring Unicode until later in the chapter.

Replacing language strings

To replace OpenZinc library language strings with ISO language strings in an application, import the language strings from a preexisting language file. Now, we'll import the French and German languages into **P_MOVIE7.-DAT.** Open the language component by opening the Defaults Editor.
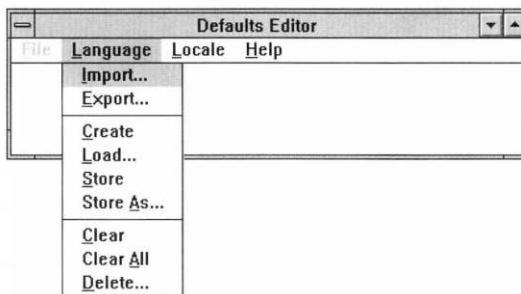
Take a moment to browse through the **P_MOVIE.DAT** file to see the various languages that we have imported. To do this, open the **P_MOVIE.DAT** file using **File I Open,** and select **Language I Load** in the Defaults Editor.
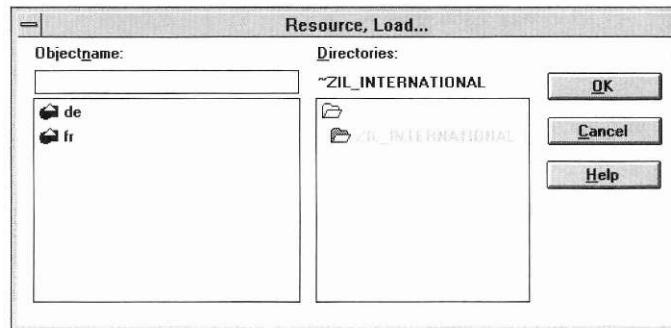
To select French, select the object item marked *fr* from the object list.
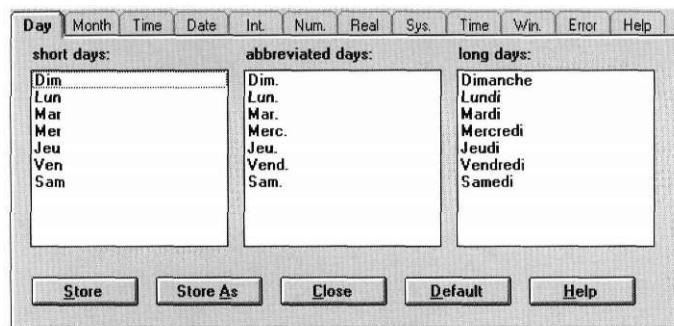
The information notebook of the *fr* language object shows OpenZinc's default string information. By importing the object, we can retrieve all the strings that appear in this notebook translated into French, including the date, month, am/pm specifiers, date, integer strings, number, real, system values, time, window messages, error and help messages.

By browsing through the available languages, we will find several Unicode languages, including *ja* (Japanese), and *ko* (Korean). These will display correctly only in Unicode.

Changing locale information

In addition to language, we also find support for locale information. Locale information stores data for specific geographic locations. To view locale information, select Locale I Load. To view the information for France, select the *FR* option.
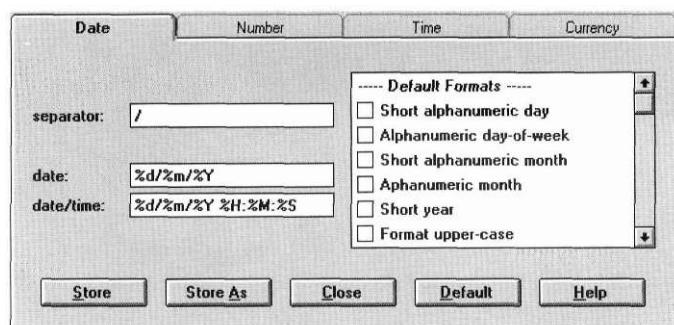
Locale information includes items such as date, number, and time formats, as well as currency symbols. Like languages, certain locales like *JP* and *KR* will work only in Unicode.

Importing language and locale

Now import languages and locales to **MOVIE.** Select **File** I **Open,** and choose **P_MOVIE7.DAT.** Then select **Language I Import** from the Defaults Editor. Again select **P_MOVIE.DAT** file, then mark *the fr* and *de* options, the French and German languages, from the object list. Then select **OK**
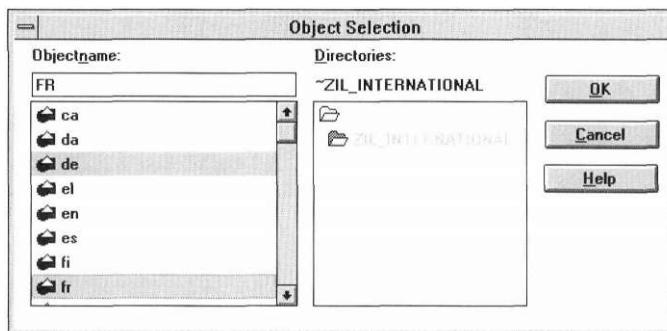
Now import the French and German locales. Select **Locale I Import,** the choose **P_MOVIE.DAT.** Then select the French and German locales *FR* and *DE,* and press **OK.**
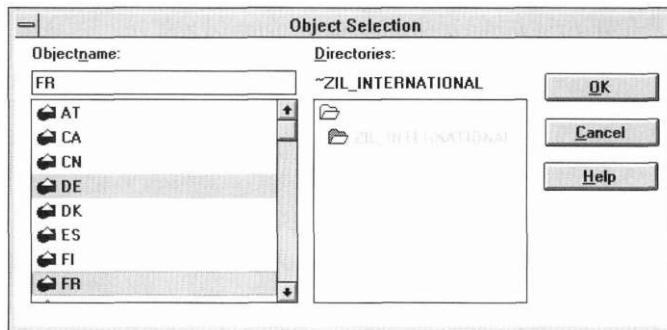
We now have imported the language and locale information for France and Germany. View this information in the **P_MOVIE7.DAT** file by selecting **Language I Load** or **Locale I Load** and by selecting the appropriate language or locale extension. Save the changes by selecting **File I Save** and then exit the Designer.

Setting language and locale at run time

Importing library language and locale information does not require recompiling source. To change the default language and locale in command-line environments, simply type in

```
set ZINC_LANG=fr_FR
```

The **set** argument tells the operating system to create a new environment variable called *ZINC_LANG*. Then when we assign it the value *fr_FR,* we associates with it the French language and locale—the first two letters are the language *(fr),* and the last two letters the locale *(FR).*

When we initialize OpenZinc, the language and locale libraries look for the environment variable ZINC_*LANG*. If they find that variable, they locate the language and locale information, if any, from the appropriate **.DAT** file. Here, we have associated the French language and French locale with our **P_ MOVIE7.DAT** file. Setting the ZINC_*LANG* environment variable tells OpenZinc to use the French language and locale.

If we now run the application, the system button, the error messages, and any default OpenZinc information will appear in the French language. For example, launch **MOVIE,** then generate an error by selecting **Movie I Create** and entering in the copyright field the year 3000. An error message appears on the screen because 3000 is outside the valid range. The error message, however, appears in French.
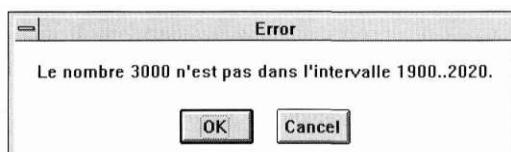
The system button also supports multiple languages. If you select the system button, the French words for restore, maximize, minimize, move, size, and close appear on the pull-down menu. (This only happens in environments that support multiple languages concurrently).
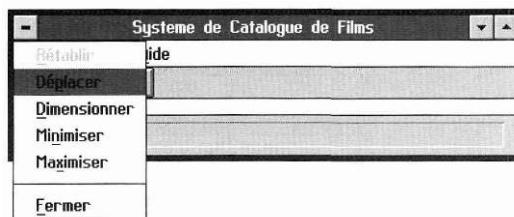
We can do the same thing with German by entering

```
set OpenZinc LANG=de DE.
```

Delta   storage

Now that we've imported languages and locales, change the movie windows and their message strings.

To restore the US language and locale to the system, enter the line

```
set OpenZinc_LANG=en_US
```

We'll change and save the window and message information with a technique called *delta storage*. Delta storage allows us work with a core set of data, retrieve it, enter any differences of various languages or locales, and save only the changes—not a new copy—to a different file. To see how, enter the Designer.

Enabling delta storage in the Designer

Select the **File I Preferences** option in the Window Editor.
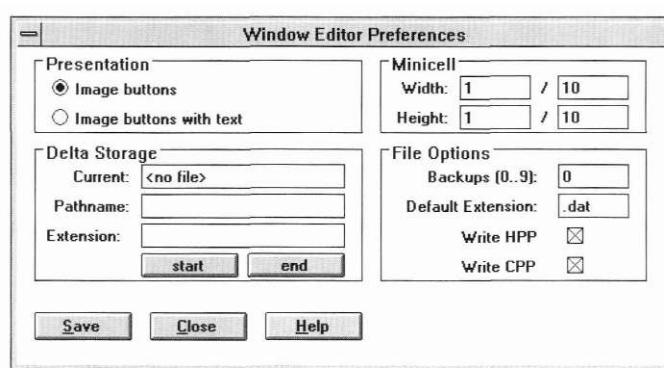
In the *Pathname* field, enter the string

```
P_M0VIE7.fr
```

In the *Extension* field enter

```
fr
```

and select Start.
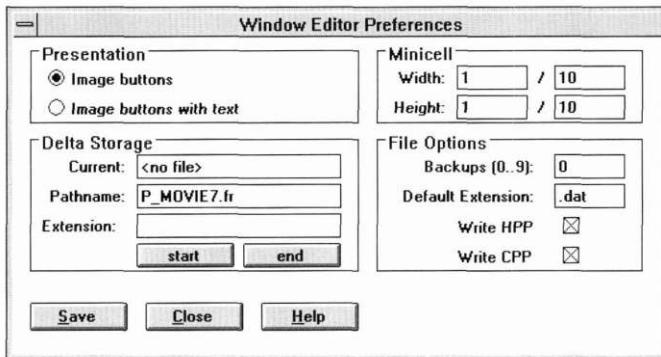
We have now opened the delta storage component of P_MOVIE7.DAT. Now we will save any changes we make to existing objects in P_MOVIE7.FR instead of the main file, P_M0VIE7.DAT.

To change the strings of the movie catalog system, open the Window Editor and read the following table to substitute the French equivalents of each string for the English string. For instance, change the string *Movie Catalog System* to *Systeme de Catalogue de Films.*

TABLE 11. String equivalents

English	*French*	*German*
Movie Catalog System	Systeme de Catalogue de Films	Filmverwaltungsprogramm
File	Fichier	Datei
Movie	Film	Film
Help	Aide	Hilfe
About	Concernant	Info
Exit	Sortie	Beenden
About Movie Catalog	Concernant le Catalogue de Films	Produktinformation
Create	Creer	Anlegen
Delete	Supprimer	Loschen
Load	Charger	Laden
Store	Archiver	Speichern

TABLE 11. **String equivalents**

English	French	German
Movie Selection	Selection d'un Film	Film Auswahl
Title	Titre	Titel
OK	OK	OK
Cancel	Annuler	Abbruch
Movie Information	Informations sur le Film	Film Information
Copyright	Droits d'Auteur	Copyright
Length (in minutes)	Duree (en minutes)	Dauer (in Minuten)
Director	Directeur	Regisseur
Actors	Acteurs	Schauspieler
Description	Description	Beschreibung
Save	Sauvegarder	Speichern
Close	Fermer	SchlieBen

Once the strings have been changed, we can save them to the delta file by selecting **Window I Store**—now the delta storage contains the French equivalents of the original English strings.

We can do the same for the Movie Selection and Movie Information windows. Their French equivalents are shown below:
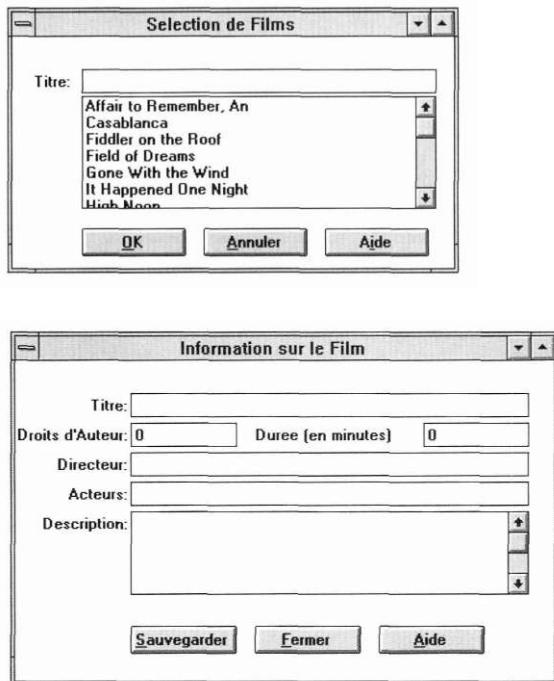
Once we have saved the all the delta changes, finish by entering the Preferences window and selecting Close from the Delta Storage group.

Now we've finalized the changes for these windows. We can do the same thing to the German windows, but with the delta file P_MOVIE7.DE. Here's what the Movie Catalog, Movie Selection, and Movie Information windows look like in German.
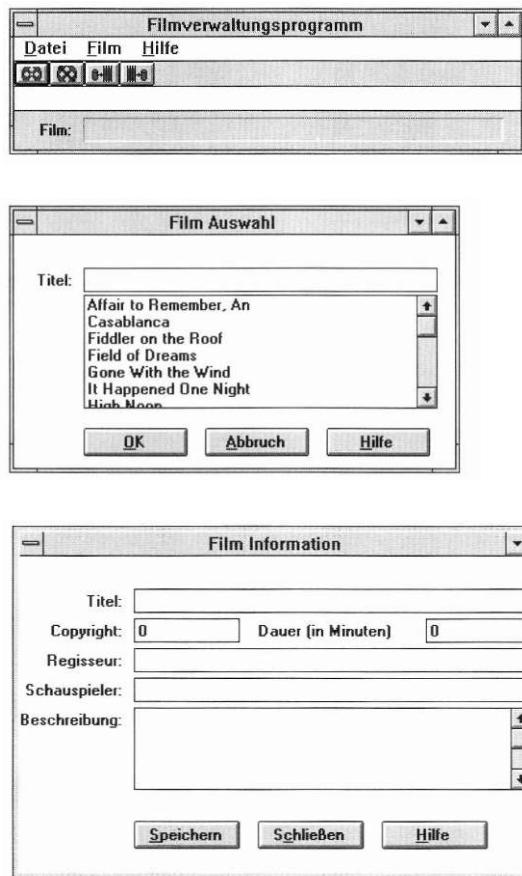
The Message Editor works like the Window Editor, only without delta storage. Open either the **P_MOYIE7.FR** or **P_MOVIE7.DE** delta file, and import the original message table from **P_M0VIE7.DAT.** Load the message table by selecting **Message I Load.** Change the information associated with each message by invoking the item's editor and changing the message field.

```
┌──────────────────────────────────────────────┐
│ ─│            Message Information              │
├──────────────────────────────────────────────┤
│                                                │
│  Message:     │Erreur d'Enregistrement: Ne peut pas sauv│
│  NumberID:    │1                      │         │
│  StringID:    │ZMSG_STORE_ERROR       │         │
│                                                │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐      │
│  │   OK     │  │  Cancel  │  │   Help   │      │
│  └──────────┘  └──────────┘  └──────────┘      │
└──────────────────────────────────────────────┘
```
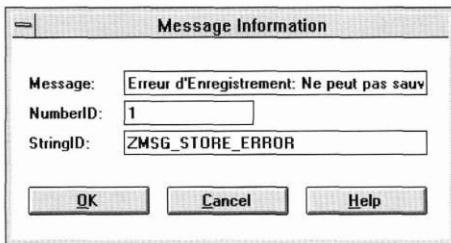
Once all the changes have been made to the message table, store the information to the delta file by selecting **Message I Store.** Finally, we can change help context information the same way, by opening the delta file, invoking the Help Editor, importing the original help contexts from **P_MOVIE7.DAT,** selecting **Context I Load,** changing the information, then by selecting **Context I Store.**

We have now changed the language information associated with our Message Catalog System window, the Movie Selection window, the Movie Information window, the message table, and the help contexts. Save all these changes by selecting **File I Save.**

We switch languages at run time exactly as we did earlier, by setting the ZINC_*LANG* environment variable.

```
set ZINC_LANG=fr_FR
```

Enabling delta storage in the source code

Now that we've saved our changes in the Designer, we need to enable delta storage in the source code. Doing so requires five changes.

First, for each window constructor, we supply an extra parameter that gives the delta storage object to the appropriate window. For example, the Movie Control window originally took a pathname and supplied the default storage. Now the Movie Control window calls the **UIW_WINDOW** constructor with the pathname supplied twice, once for the original window name, and once for the delta window name, the language extension, a pointer to the default storage, and a pointer to the delta storage.

```
M0VIE_C0NTR0L::M0VIE_C0NTR0L(void) :
```

```
UIW_WINDOW(_pathName, _pathName, ZIL_NULLP(ZIL_ICHAR),
  defaultStorage, _intlStorage)
```

We follow the same process for the Movie Selection and Movie Information constructors, and the constructor for the exit window.

```
MOVIE_SELECTION::MOVIE _SELECTION(ZIL_STORAGE_READ_ONLY
    *dataFile, ZIL_USER_EVENT _request) :
  UIW_WINDOW(_pathName, _pathName, ZIL_NULLP(ZIL_ICHAR),
    defaultStorage, MOVIE_CONTROL::_intlStorage)

MOVIE_INFORMATION::MOVIE_INFORMATION(ZIL_ICHAR *name)  :
  UIW_WINDOW(_pathName, _pathName, ZILNULLP(ZILICHAR),
    defaultStorage, MOVIE_CONTROL::_intlStorage)

EVENT_TYPE MOVIE_CONTROL::Exit(UI_DISPLAY *,
  UI_EVENT_MANAGER *, UI_WINDOW_MANAGER *windowManager)
{
  // Read the exit window.
  UI_ERROR_STUB::Beep();
  UIWWINDOW *window = new UIW_WINDOW(_exitName,  exitName,
    ZIL_NULLP(ZIL_ICHAR), MOVIE_CONTROL::defaultStorage,
    _intlStorage);
```

The original windows are stored in the .DAT file, and a directory identified by language contains the delta storage. When we run the application, our *defaultStorage* pointer will point to the original window in the data file, and the *_intlStorage* pointer will point to the differences contained in the language directory.

```
// Set up strings used to open general and delta storage
// files.
static ZIL_ICHAR _fileName[] =
  { 'p','_','m','o','v','i','e','7','.',0 };
static ZIL_ICHAR _extension[ ] = { 'd','a','t', 0};
ZIL_ICHAR _intlFileName[32], _baseFileName[32];
strcpy(_intlFileName, _fileName);
strcat(_intlFileName, languageManager.defaultName);
strcpy(_baseFileName, _fileName);
strcat(_baseFileName, _extension);
// Create error and help systems.
UI_WINDOW_OBJECT::errorSystem = new UI_ERROR_SYSTEM;
UI_WINDOW_OBJECT::helpSystem =
  new UI_HELP_SYSTEM(_baseFileName);
// Open general storage file which contains original windows.
UI_WINDOW_OBJECT::defaultStorage =
  new ZIL_STORAGE_READ_ONLY(_baseFileName);
// Open delta storage file.
MOVIE_CONTROL::_intlStorage =
  new ZIL_STORAGE_READ_ONLY(_intlFileName);
```

The only code change required for the help context or error message strings involves opening the delta storage file.

```
UI_WINDOW_OBJECT::helpSystem =
    new UI_HELP_SYSTEM(_intlFileName);

_errorMsgTable = new ZIL_LANGUAGE(_tableName, _intlStorage);
```

Remember, we associated constant values with the *help* and the *messages,* so the identifying number is still the same. Only the string in the delta file has changed.

Unicode

As the coup de grace to this series of tutorials, we**'ll** make **MOVIE** Unicode compatible.

In a Unicode-compatible application, each letter gets two bytes of information instead of one.

Almost all of our string information is contained in **.DAT** files, which are Unicode compatible by design; OpenZinc will automatically convert 8-bit values into 16-bit values if the application is running in Unicode mode. The only things that we need to replace to make them Unicode compatible are the definitions for filenames and class names. These will always remain the same, no matter what language we are using.

We can create Unicode-compatible strings one of two ways. The first way is to use a message table as discussed in this chapter. We can put most strings in a message table, but we still need to create a few in our source code that will specify the name of the **.DAT** file containing the message table and the name of the message table itself. We can't put these in a **.DAT** file. Instead, we need to make them 16-bit Unicode values instead of 8-bit values.

Naming Unicode strings
Unfortunately, at present, most compilers do not support the naming of Unicode strings easily. We get around this is by comma delimiting each character in the name. Here, we need to replace the base filename, the extension for the base file, and the name of the message table.

The way to internationalize these is to pull out the embedded string and make it either a static variable or a static member of a class. Here, in **MOVIE7.CPP,** we make it a static variable. For instance, we now have a static, global variable called *jnsgTableName* that contains the name of the message table.

```
static ZIL_ICHAR __msgTableName [ ] =
  { 'M','S','G','_','T','A','B','L','E',0 };
```

UI_APPLICATION::Main() contains two more static variables, one for the **.DAT** file name and one for the **.DAT** file extension.

```
static ZIL_ICHAR _fileName[] =
  { 'p','_','m','o','v','i','e','7','.',0 };
static ZIL_ICHAR _extension[] = { 'd','a','t', 0};
```

The other strings we need to make into static variables are the names of the window resources in the .DAT file, which are loaded from the message table and stored in static member variables.

```
// Initialize static name strings.
MOVIE_CONTROL::_pathName =
  _nameMsgTable->GetMessage(ZMSG_CTRL_PATH_NAME);
MOVIE_CONTROL::_exitName =
  _nameMs gTable->GetMe s s age(ZMSG_CTRL_EXIT_NAME);
MOVIE_SELECTION: :_pathName =
  _nameMsgTable->GetMessage(ZMSG_SEL_PATH_NAME);
MOVIE_SELECTION::_allObjects =
  _nameMsgTable->GetMessage(ZMSG_SEL_ALL_NAME);
MOVIE_INFORMATION::_pathName =
  _nameMsgTable->GetMessage(ZMSG_INFO_PATH_NAME);
```

Changing these strings in the source code or loading them from a message table allows us to compile our application for Unicode. We can see the advantage of running in Unicode by running the DOS executable—with Unicode, we can retrieve the Korean and Japanese language information in our DOS application. In addition, we can run the application under environments other than DOS with multiple languages running concurrently, as long as the operating system supports that language in its font set.

Conclusion

We made **MOVIE** portable and internationalized following several steps. First, we removed all language-dependent, embedded strings. For instance, in our application, we replaced the load, delete, movie selection, and store error strings with identifiers. Then we included the language and locale information for French and German languages. We converted our screen information by moving to each window, translating the information and saving it out to a new a file for the desired language or locale. Next, we changed the message tables. Finally, we changed the help contexts.

If the program is architected properly, globalizing an application is easy. Once the program is globalized, we can translate all the data without touching the executable. This means that to port our program to additional languages and locales, we need only send the interface strings to a translator and include the results with the program—without touching the executable.

This is the end of the **MOVIE** tutorial, as well as the end of our tutorial section. The next section is a reference concerning the different functionality of OpenZinc Designer.

Enjoy OpenZinc!

section three

OpenZinc Designer

reference

File Options

The File category options control the general operations of OpenZinc Designer files. Selecting File causes the following menu to appear:
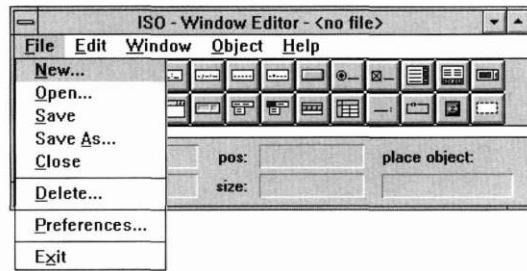
New

The **New...** option allows you to create a new file. Selecting it causes a window similar to the following to appear:
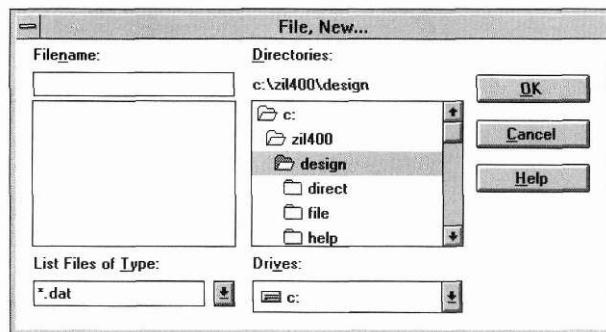
Filename

If you want to create a new file for an application, enter the name for the new file here. If you do not include an extension yourself, a **.DAT** extension will be automatically attached to the name when the file is actually created.

Other files that belong to the current directory—all of the type designated by the **List Files of Type** field—are shown in the scrollable list below the **Filename** field. If one of these files is selected and the **OK** button is pressed, you will be asked if you want to overwrite the existing file. (For information on opening a previously created file, see the description of the **Open** option below.)

List Files Of Type

This field determines the type of files, based on the file extension, to list in the **Filename** field. Clicking on the arrow button reveals the other types of files available. Selecting one of these extensions causes the files of that type to be displayed in the **Filename** list. Selecting the *.* extension causes all files in the current directory to be displayed.

Directories

The current directory is shown below the **Directories** prompt. Your file will be saved to this directory. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current drive, the current directory being highlighted and child directories shown below the current one and parent directories above.

Drives   This field displays the current drive. Clicking on the arrow button reveals the other drives that are available on your system. Selecting a drive causes the files and directories on that drive to be displayed in their respective fields.

OK   Selecting this button causes a file to be created which will be given the name entered at the **Filename** prompt. If creation of the file is successful, the **New** window will close and the title bar of the control window will be updated to include the name of the current file. If no information has been entered within the **New** window and the **OK** button is selected, you will receive an error message.

Cancel   Selecting this button causes the window to close without executing any changes.

Help   Additional information about creating new files appears when this button is selected.

Open

The **Open...** option allows you to open a previously created file. Selecting it causes a window to appear that is similar to the **New** window:

Filename	To open an existing file, either enter the name at the **Filename** prompt, or select it from the list below, and the name of the file will automatically appear at the prompt.
	Other files that belong to the current directory—all of the type designated by the **List Files of Type** field—are listed in the scrollable list below the **Filename** field. As mentioned above, selecting one of these files causes the name to appear in the **Filename** field. Double clicking on a name listed in the files list will cause that file to be opened immediately.
List Files of Type	This field determines the type of files, based on the file extension, to list in the **Filename** list. Clicking on the arrow button reveals the other types of files available. Selecting one of these extensions causes the files of that type to be displayed in the **Filename** list. Selecting the *.* extension causes all files of the current directory to be displayed.
Directories	The current directory is shown below the **Directories** prompt. Your file will be saved to this directory. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current drive, the current directory being highlighted and child directories shown below the current one and parent directories above.
Drives	This field displays the current drive. Clicking on the arrow button reveals the other drives that are available on your system. Selecting a drive causes the files and directories on that drive to be displayed in their respective fields.
OK	Selecting this button causes the file specified at the **Filename** prompt to be opened. If the open procedure is successful, the window will close and the title bar of the control window will be updated to include the name of the current file. If the file entered at the **Filename** prompt does not exist, or if no information has been entered, you will receive an error message at this time.
Cancel	Selecting this button causes the window to close without executing any changes.
Help	Additional information about opening existing files appears when this button is selected.

Save

Selecting the **Save** option causes the current file to be saved in its present condition. If the file has not been named, the **Save As** window will appear so that you can give the file a name. (See the **Save As** section for details on how to save a file for the first time.)

Upon every save operation, OpenZinc Designer automatically creates a **.DAT** file, which contains the binary information associated with the objects saved in the application. In addition, the following files are created by default but can be bypassed by changing the information contained in **File I Preferences:**

- a **.CPP** file, which contains the definition for *_objectTable,* an array that provides the functions needed to load objects saved to disk, as well as the definition for *_userTable*, an array of function access points for user call-back and compare functions.

- an **.HPP** file, which contains the numeric identifications (identified using the text entered for the object's name) unique to each field or help context.

- one or more **.BK#** (backup) files, specified in **File I Preferences.** **(NOTE:** Only one backup file is created per Designer session and only if a previous **.DAT** file existed.)

Save As

Save As... is usually used to either save a file that has not been previously named or to save the current file under another name. Selecting it causes a window to appear that is similar to the **New** and **Open** windows:
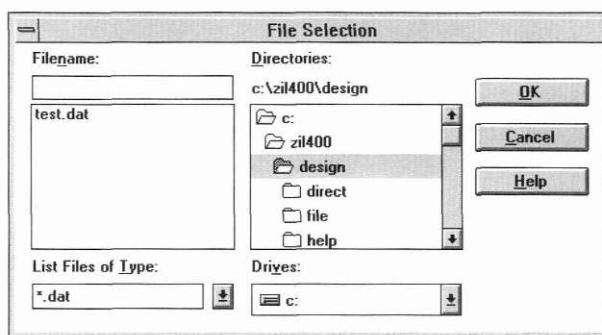
Filename
Enter a name for the file at the **Filename** prompt, or select it from the list below, and the name of the file will automatically appear at the prompt. If you do not include an extension yourself, a **.DAT** extension will be automatically attached to the name when the file is actually created. A new file will be created under that name with the current modifications, if any.

Other files that belong to the current directory—all of the type designated by the **List Files of Type** field—are listed in the scrollable field below the **Filename** field. As mentioned above, selecting one of these files causes the name to appear in the **Filename** field. Double clicking on a name listed in the files list will cause a window to appear that asks if you want to overwrite the existing file.

List Files Of Type
This field determines the type of file, based on the file extension, to list in the **Filename** field. Clicking on the arrow button reveals the other types of files available. Selecting one of these extensions causes the files of that type to be displayed in the **Filename** list. Selecting the *.* extension causes all files of the current directory to be displayed.

Directories
The current directory is shown below the **Directories** prompt. Your file will be saved to this directory. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays

other available directories of the current drive, the current directory being highlighted and child directories shown below the current one and parent directories above.

Drives

This field displays the current drive. Clicking on the arrow button reveals the other drives that are available on your system. Selecting a drive causes the files and directories on that drive to be displayed in their respective fields.

OK

Selecting this button causes the file to be saved under the name entered at the **Filename** prompt. If the save operation is successful, the **Save** As window closes.

If you have entered a file name that already exists, a modal window will appear, indicating such. If you select the **Yes** button of this window, the current information replaces the previous information of that file, and both the modal window and the **Save As** windows close. Selecting the **No** button simply closes the modal window and allows you to enter information again in the **Save** As window.

If no information has been entered within the **Save As** window and you select the OK button, the window will close and no other action will take place.

Cancel

Selecting this button causes the window to close without executing any changes.

Help

Additional information about saving files appears when this button is selected.

Close

Selecting the **Close** option causes the screen to be cleared and the current file to close. The title bar of the control window will be updated immediately to read **P_TEMP.DAT,** which is the default Designer file.

Delete

The **Delete...** option allows you to delete a file. Selecting it causes a window similar to the following to appear:
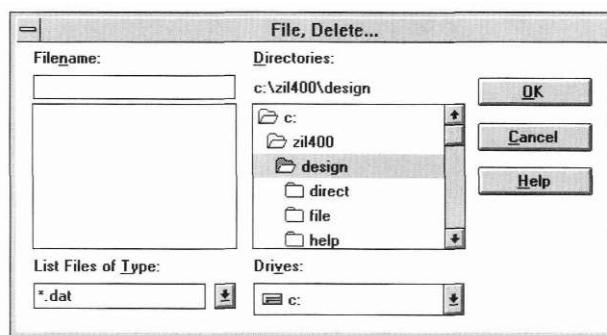
Filename
To delete a file, either enter the name at the **Filename** prompt, or select it from the list below, and the name of the file will automatically appear at the prompt.

Other files that belong to the current directory—all of the type designated by the **List Files of Type** field—are listed in the scrollable field below the **Filename** field. As mentioned above, selecting one of these files causes the name to appear in the **Filename** field. Double clicking on a name listed in the files list will cause that file to be deleted immediately.

List Files of Type
This field determines the type of file, based on the file extension, to list in the **Filename** field. Clicking on the arrow button reveals the other types of files available. Selecting one of these extensions causes the files of that type to be displayed in the **Filename** list. Selecting the *.* extension causes all files of the current directory to be displayed.

Directories
The current directory is shown below the **Directories** prompt. Your file will be saved to this directory. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current drive, the current directory being highlighted and child directories shown below the current one and parent directories above.

Drives   This field displays the current drive. Clicking on the arrow button reveals the other drives that are available on your system. Selecting a drive causes the files and directories on that drive to be displayed in their respective fields.

OK   Selecting this button causes a modal window to appear which is similar to the following:

The purpose of this window is to make sure that you want to delete the file. If you select the **OK** button, the file indicated at the **Filename** prompt is deleted, and both the confirmation window and the **Delete** window close. If you choose the **Cancel** button, the file is not deleted and just the modal window closes.

If the name of the current file is entered, or if the file entered does not exist, you will receive an error message when the **OK** button is selected.

If no information has been entered within the window, selecting **OK** causes an error message to appear.

Cancel   Selecting this button causes the window to close without executing any changes.

Help   Additional information about deleting files appears when this button is selected.

Preferences

The **Preferences...** option allows you to change the default settings of OpenZinc Designer. These settings are saved in the **OpenZinc.CFG** file. Selecting **Preferences** causes a window similar to the one below to appear:

```
┌─────────────────────────────────────────────────────┐
│ ─ │          Window Editor Preferences              │
├─────────────────────────────────────────────────────┤
│ ┌─Presentation──────────┐  ┌─Minicell──────────────┐│
│ │ ⦿ Image buttons       │  │ Width:  [1]   /  [10] ││
│ │ ○ Image buttons with text│ Height: [1]   /  [10] ││
│ └───────────────────────┘  └───────────────────────┘│
│ ┌─Delta Storage─────────┐  ┌─File Options──────────┐│
│ │ Current:  [<no file>] │  │ Backups (0..9): [0]   ││
│ │ Pathname: [         ] │  │ Default Extension:[.dat]││
│ │ Extension:[         ] │  │ Write HPP    ⊠        ││
│ │  [ start ]  [  end  ] │  │ Write CPP    ⊠        ││
│ └───────────────────────┘  └───────────────────────┘│
│   [ Save ]   [ Close ]   [ Help ]                    │
└─────────────────────────────────────────────────────┘
```
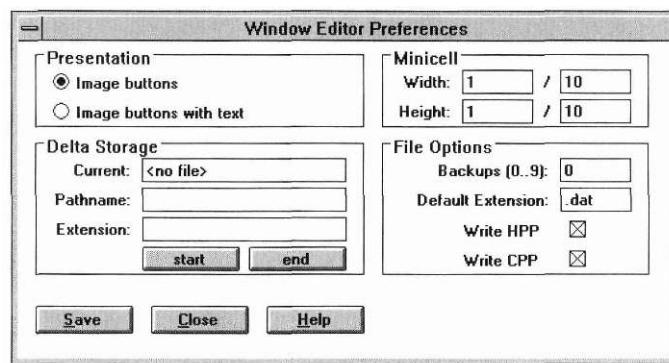
Presentation

This field contains the options for the presentation of objects on the button bar.

Image buttons. Allows only images to be displayed on an object button.

Image buttons with text. Allows both images and text to be displayed on an object button.

Delta Storage

This group gives you access to delta storage, which allows you to save only the changes of the current file to another file. This feature is very useful in some situations, such as when translating an application to another language, where the fields do not change but the text must be replaced. Instead of copying the original file and having two complete copies, delta storage allows you to keep the original as the master file and another file that only contains the alterations (e.g., the translated text) made to that master file. The resulting saved memory can be significant when compared to maintaining two complete files.

The **Delta Storage** group contains the following fields:

Current. Displays the path name of the delta file. This field is not editable. When no delta file is open, it displays **<no file>.**

Pathname. Designates the name for the delta file. When no delta file is open, this field is blank.

Extension. Designates the subdirectory where the delta object will be stored. For example, entering the extension **me** causes the object to be stored in the **\delta\me** directory of the current file.

Selecting the **start** button causes the delta file designated at the **Pathname** prompt to be opened. Any modifications made to the current file thereafter will be saved to the delta file only—the current, or original, file will remain exactly as it was when it was last saved before the delta file was opened.

Selecting the **end** button causes the delta file designated at the **Pathname** prompt to be closed. Thereafter, any changes to the current file will actually be saved to that current, or original, file.

The **Start** and **End** buttons work independently of the three main buttons of the Window Editor **Preferences** window. For example, when you select the delta storage **Start** button, you are immediately in delta storage mode, even if you subsequently select the main **End** button of the preferences window.

While you have opened a delta file and not yet closed it, a **(D)** appears on the control window's title bar to the right of the file name. This serves as a reminder that all changes are being saved to the delta file instead of to the actual original file.

Minicell

This field allows you to set the default minicell ratios. The default minicell width and height are 1/10 of a cell.

File options

This group contains options for file backups, file extensions, and files to save. The options are:

Backups. Enter in this field the number of backups that you would like the designer to maintain. Each backup file will be saved under the same name as the main file but with an extension that indicates the backup number of the copy. For example, a file with the name of **TEST.DAT** will have a backup copy called **TEST.BK1** if only **1** is entered at the prompt.

If any number greater than 1 is entered at the prompt, each time a save operation occurs another backup file will be created, up to the maximum specified. For example, a **3** at the prompt will cause the creation of a **TEST.BK1** file at the first save operation, a **TEST.BK2** file at the second save, and a **TEST.BK3** at the third save. Thereafter, these three backup files would be

updated on subsequent saves, with the most recent information being saved in **TEST.BK1** and the oldest information in **TEST.BK3.** The Designer will create up to nine backups.

Default Extension. This field designates the default extension for files saved in the designer. OpenZinc's default is **.DAT,** but you can enter a different default for your applications, if desired.

Write HPP. Selecting this option causes an **.HPP** file to be generated with each save operation. When the option is not in effect, save operations will be quicker. While it is not necessary to write an **.HPP** file while working in the designer, it will be necessary to write one before actually compiling your application.

Write CPP. Selecting this option causes a **.CPP** file to be generated with each save operation. When the option is not in effect, save operations will be quicker. While it is not necessary to write a **.CPP** file while working in the designer, it will be necessary to write one before actually compiling your application.

OK   Selecting this button closes the **Window Editor Preferences** window and causes the information selected to take effect. If no information has been entered within the window, it will close and no other action will take place.

Cancel   Selecting this button causes the window to close without executing any changes.

Help   Additional information about default settings appears when this button is selected.

Exit

Selecting the **Exit** option allows you to exit OpenZinc Designer. If you have not saved the current file, a modal window will appear that asks whether or not you want to save it before exiting. Selecting the **Yes** button causes the file to be saved and then exits out of the program. Selecting **No** causes the program

to exit without saving the current file—any changes made since the last save operation will be lost. Selecting the **Cancel** button simply closes the modal window.

If you have not made any changes within OpenZinc Designer, selecting **Exit** causes a modal window to appear which is similar to the following:
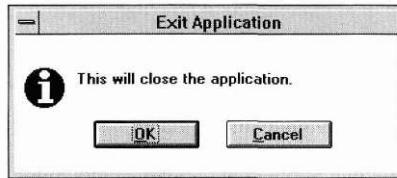
The purpose of this window is to make sure that you want to exit OpenZinc Designer. If you select the OK button, the program exits. If you choose the **Cancel** button, the program does not exit and the modal window closes.

File Options

Edit Options

Edit category options are used to edit the appearance and performance of objects within the current file. Selecting **Edit** causes the following menu to appear:
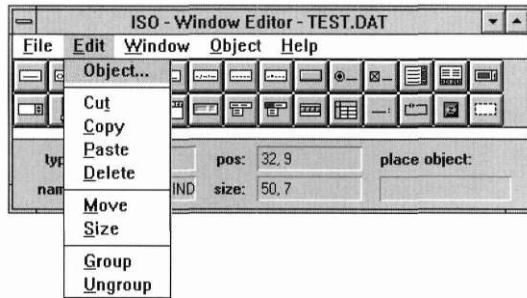
Object

Each object created with OpenZinc Designer can be modified through interaction with its object information notebook. Selecting **Object...** causes the information notebook for the current object to appear, which is similar to the following:
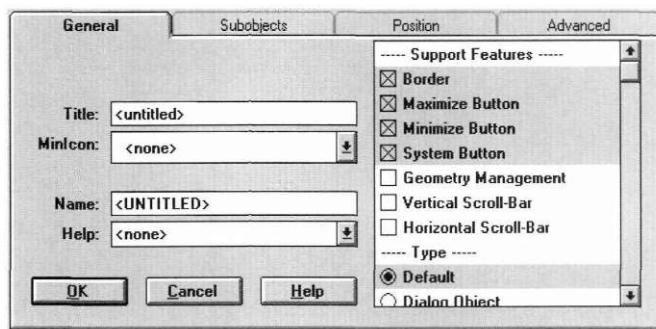
The object information notebook controls how the object looks. Each information notebook is divided into notebook pages that contain related information. To view the information contained in a page, click on its title tab with the mouse. Since each object has its own specific requirements, each information notebook's pages and their related fields will vary, but all contain one or more of those described in the next several sections.

(See Chapter 14 through Chapter 17 for more specific information regarding individual field objects.

NOTE: The remainder of this section is identical to the information given in Chapter 13.)

Each notebook page within the information notebook includes three buttons which operate in the following manner:

OK. Selecting this button saves the edit information and closes the object information notebook. The current object will reflect the editing changes immediately. If no information has been entered within the object information notebook, its window will close with no other action taking place.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about the current object appears when this button is selected. Help information is unique for each page. The contents depend on the type of object and the current page.

General page

The **General** notebook page, shown above, contains information related to the specific object being edited. It is always the first page visible upon entering the information notebook. Since each object is different, the editable properties will vary, but the following fields are common to most objects:

Text or Value. This field allows you to enter information to be displayed within the object exactly as you want it to appear in your application. Objects that use the **Text** field are:

- string
- formatted string
- text
- button
- radio button
- check box
- pull-down item
- pop-up item
- prompt, and
- group.

Some objects have a field similar to **Text,** but the name **Value** is used in place of **Text.** Objects that use the **Value** field are:

- date
- time
- bignum
- integer, and
- real.

Compare. Some objects allow a compare function, which is used to sort information. If you want to have a compare function associated with the object, you can enter the function's name in this field.

When using a compare function, the function itself must be defined somewhere in your code under the same name that is entered in the information notebook. This is necessary so that OpenZinc Designer can find it and execute the

designated action. (For more information on creating compare functions, refer to the description of the object's constructor in the *Programmer's Reference.)*

Name. This field contains the name, sometimes called a string identification, for the object and is present in every object information notebook. The default name for an unnamed object attached to another object is **FIELD** plus a unique number corresponding with the order in which it was attached to the parent window. For example, the default name for the second object created within a resource window would be **FIELD_2.**

Because these objects appear in various lists throughout the program, it is recommended that you override the default name and enter a string that more specifically identifies the object. The name will appear in all locations exactly as you have entered it in the object's information notebook.

Help. This field designates the help context to be associated with the object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the object and requests help. (See Chapter 20 for information on creating help contexts.)

options list. This field is located on the right side of the General page. It displays options which control the general presentation and operation of the current object. All of these items are listed with either check boxes or radio buttons. To toggle an option from noncurrent to current or vice versa, select it by either clicking on it with the mouse or by scrolling to it and pressing <space>. There is no limit to the number of check box options that can be in effect at a given time. However, only one radio button option per designated group can be in effect at one time.

Position page

The **Position** notebook page contains information related to an object's position, border, region, and alignment. Selecting the **Position** tab causes a page similar to the following to appear:

The fields of the **Position** page are the same for every object. They are as follows:

Position/Size. This field allows you to alter the positioning and size of the current object. The first row within the field contains the following three buttons that determine the coordinate scale to be used:

- *pixel.* Depressing the left button causes screen coordinates to be based on a pixel scale. This option allows for the greatest precision in positioning and sizing. It is only applicable in graphics mode.

- *mini-cell.* Depressing the middle button causes screen coordinates to be based on a mini-cell scale, which is 1/10 of a cell coordinate by default. This option allows for more precision than the cell option, but less than the pixel option. It is applicable in graphics mode only.

- *cell.* Depressing the right button causes screen coordinates to be based on a whole cell scale. This is the default setting.

Underneath each button in the **Position/Size** field is a column of fields that display the precise location and size of the object. These are based upon the coordinate option selected, such as pixel, mini-cell, or cell. Only the column underneath the currently selected option is selectable and editable.

The top two fields of each column, **column** and **line,** determine the object's location. Changing the numbers in either or both of these fields will cause the object to move to the corresponding coordinates on the screen.

The lower two fields, **width** and **height,** determine the object's size. Changing the numbers in either or both of these fields will cause the object to alter in size according to the coordinate scale being used. For example, if the cell option is in effect, entering a width of 20 and a height of 10 in the cell column would produce an object that is 20 cells wide and 10 cells high.

Border. The two buttons contained in this field determine whether or not the object is displayed with a border. Depressing the left button causes the object to be displayed without a border, while depressing the right button causes a border to be displayed.

Region. The two buttons contained in this field determine the region allocated for the object. Depressing the left button causes the object to occupy the region specified by the values displayed in the **Position/Size** field. Depressing the right button causes the object to ignore its position and size parameters and occupy the remaining available space in its parent window.

Alignment. The three buttons contained in this field determine the alignment of the text within the object. Depressing the left button causes the text to be justified to the left margin, depressing the middle button causes the text to be centered, and depressing the right button causes the text to be justified to the right margin. For objects that do not contain text, such as scroll bars, the **Alignment** field is shaded, indicating that it is not selectable.

Geometry page

The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For example, using geometry management, you can determine how an object will react to resizing of its parent object—whether it stays at a fixed location or moves relative to the parent, whether it stays the same size or grows with the parent, etc.

Geometry management also allows you to establish a connection relationship between two child objects and to set size constraints for individual objects.

Selecting the **Geometry** tab causes a page similar to the following to appear:

The following are the fields of the **Geometry** page, which are the same for every object:

attachment group. This area, located on the left side of the page, is designed to visually represent the relationship between the current object and its surroundings. The small square in the center represents the object itself, and the four lines, one radiating from each side of the square, represent possible connections to the surrounding objects. Each line ends at a combo box field which displays the type of connection for that line.

To change the type of connection, select the combo box button and a list of available types will be displayed. The following three types are available:

- *none.* Indicates that no connection is in effect.

- *relative.* Causes the object's border to position itself within its parent relative to the parent's borders. In this case, the value entered into the **offset** field (described below) will represent a percentage ratio for placement. For example, if an object's left connection has an offset value of 10, it will always be placed within the parent's left border at a distance measuring 10 percent of the parent object's size. Therefore, if the parent object measures 20 minicells wide, the child's left border will be two minicells within the parent's left border. If the parent is subsequently enlarged to be 100 minicells wide, the child will reposition at the edge of the tenth minicell.

  NOTE: A relative connection is possible only between a child object and its parent—not between two child window objects.

- *absolute.* Causes the object's border to position itself relative to its parent or sibling object at a fixed distance. In this case, the value entered into the **offset** field (described below) will represent a fixed measurement for

placement. For example, if an object's left connection has an offset value of 10, it will always be placed within the parent's left border at a distance measuring 10 units of whatever coordinate scale is in effect (i.e., pixel, mini-cell or cell). Therefore, if the parent object measures 20 minicells wide, the child's left border will be ten minicells within the parent's left border. If the parent is subsequently enlarged to be 100 minicells wide, the child will remain at the edge of the tenth minicell.

As connections increase, so do potential conflicts. For example, if an object has a left absolute connection offset at 10 cells and a right absolute connection offset at 20 cells, both constraints cannot always be accommodated. When conflict arise, the objects follow an established priority schedule:

- 1st priority—left
- 2nd priority—top
- 3rd priority—right
- 4th priority—bottom

Therefore, in our example above, the left connection would consistently offset the designated 10 cells, while the right connection would be overridden to accommodate the left.

Constraint features. This group determines the constraint features of the current connection. It contains the following fields:

- *offset.* Determines the offset of the current connection, or the amount of space between its inside and outside anchors. The actual measurement of the value depends upon whether the connection is relative, in which case the value represents a percentage ratio, or absolute, in which case the value represents a fixed measurement. (See the **connections plane** section above for further information on connection types.) The value of this number also depends upon the coordinate scale in effect (either pixel, minicell or cell) for the current object. For example, for an object using pixel units, an absolute connection's offset of 20 would translate into a blank space measuring 20 pixels. (For further information on coordinate scales, see the **Position** page section, page 213 of this chapter.)

To change the value of **offset,** either enter a new number, or increment/decrement the number by clicking on the appropriate spin control button.

- *options.* Presents several options that alter the typical behavior of the connections. The following options are available:

- *stretch.* Causes the actual object to stretch or shrink as the anchor border is moved. For example, an object with a bottom connection to its parent window's bottom border will stretch vertically as the window's bottom border is pulled downward.

- *opposite.* Causes the connection to be anchored to the opposite border of the object designated in **connect.** For example, if object A is located to the left of object B and its left border connection is set to run to object B, then the connection will actually be anchored to object B's *right* border.

- *hz-center.* Causes the connection to be anchored from the horizontal center of the current object. This allows for more consistent placement, especially when centering the object. This option is selectable for a relative connection only.

- *vt-center.* Causes the connection to be anchored from the vertical center of the current object. This allows for more consistent placement, especially when centering the object. This option is selectable for a relative connection only.

- *anchor.* This field determines what the current connection is anchored to. As in the above examples, it is common for an object to run connections to the border of its parent object, such as a parent resource window. However, it is also possible to run absolute connections to other objects.

  The **anchor** field applies to each connection individually, not to all four collectively. Therefore, it is possible to have each border of the object connected to a different object or parent border. The **anchor** field displays the anchor for the connection currently being edited.

  To choose an anchor for the current connection, first select a relative or absolute connection, described above, then select the **anchor** field's combo box button. A list that includes the surrounding objects and the parent object will appear. Selecting one of these displays the object's name in the **anchor** field.

NOTE: Since a relative connection can only be anchored to its parent object, the **anchor** field is not selectable when a the current connection is relative.

- *Size Restrictions.* This field determines the minimum and maximum size for the object. The first row determines the range of width values, while the second row determines the range of height values. The first value of

each row represents the minimum size possible, and the second the maximum possible. All of these values are measured in cell units. For example, an object with a minimum width value of 5 and a maximum value of 20 is constrained to be no less than 5 cells wide but no greater than 20 cells.

Advanced page

The **Advanced** notebook page contains information about an object's advanced properties. It is designed for use by the experienced user only. The following fields are common to most objects:
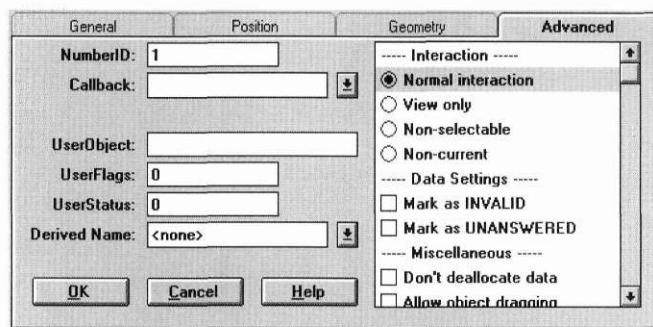
NumberID. This field assigns a unique programmer number to the object. By default this number corresponds to the order in which the object was created in the program.

Callback. To associate a callback function or user function with the object, enter the function's name in this field. A callback function causes a designated action to occur when the user interacts with the object.

When using a callback function, the function itself must be defined somewhere in your code under the same name that is entered in the information notebook. This is necessary so that OpenZinc Designer can find it and execute the designated action. (For more information on creating callback and compare functions, refer to the description of the object's constructor in the *Programmer's Reference.)*

User Object. This field designates a void pointer to a user-specified object.

UserFlags. This field contains any flags that are set and maintained by the programmer.

UserStatus. This field contains any status flags that are set and maintained by the programmer.

Derived Name. This field contains the name of the derived class to inherit the properties of the current object. If a name is entered, the object table in the **.CPP** file, generated by the Designer, will contain an entry for the derived object's **New()** function. As part of the code for the derived class, the programmer must create a static **New()** function that is able to call the constructor for the derived class. The **.HPP** file will contain a definition of the derived class' identification.

options list. This field is located on the right side of the **Advanced** page, and it displays options which control the advanced properties and operations of the current object. All of these items are listed with either check boxes or radio buttons. To toggle an option from noncurrent to current or vice versa, select it by either clicking on it with the mouse or by scrolling to it and pressing <space>. There is no limit to the number of check box options that can be in effect at a given time. However, only one radio button option per designated group can be in effect at one time.

Options that are not applicable to the current object are grayed out, meaning that they cannot be selected.

The first section of the list presents options that determine the extent to which the end user can interact with or edit the object. Only one of these can be current at a time. The options are:

- *Normal interaction.* Allows the end user to have normal interaction with the object, including being able to select, position on and edit it (if it is an editable object).

- *View only.* Prevents the object from being edited. If this flag is set, the end user will not be able to edit an object's information but will be able to browse through the information.

- *Nonselectable.* Prevents the object from being selected. If this flag is set, the user will not be able to edit or position on the object's information.

- *Noncurrent.* The object cannot be made current. If this flag is set, users will not be able to select the object from the keyboard nor with the mouse.

The second section of the list presents options that determine the initial settings of the object. The options are:

•   *Mark as INVALID.* Sets the initial status of the object to **invalid.** This forces the user to enter an acceptable value before moving to another field.

•   *Mark as UNANSWERED.* Sets the initial status of the field to be **unanswered.** An unanswered field is displayed as blank space on the screen.

The third section of the list presents options that determine other default settings for the object. The options are:

•   *Don't deallocate data.* Causes the object to not allocate its own buffer for data. If this option is set, you must allocate a buffer that will be used to store the object's data.

•   *Allow object dragging.* Allows the object to be dragged.

•   *Allow object dropping.* Allows objects to be dropped onto the current object.

•   *Support object.* Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window, such as a window's border and title.

Subobjects page   Objects which can host other objects also contain a **Subobjects** notebook page within their information notebooks. This page allows you to modify the subobjects. It contains the following fields:

Objects. This field displays the objects, listed in the order in which they were created, that are attached to the current object. The list is divided into two sections, the first for support objects, or objects that are not displayed as part of the user region, and the second for normal objects. The information note-

book of any one of these objects can be accessed by one of the following methods: double click on the desired object with a mouse, scroll to it and press <Space>, or click on the **Edit** button while it is highlighted in the list.

To move an object in the list up one line, either select the **Move Up** button, or hit <Ctrl up-arrow>. To move an object down in the list one line, either select the **Move Down** button, or hit <Ctrl down-arrow>. Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it also changes the order that the items appear within the parent lists. To delete an object from the list, either select the **Delete** button, or hit <Ctrl+Delete>.

Directories. This field displays the parent objects and subobjects related to the current object. Double-clicking on one of the names (or pressing <Space> while the name is current) causes the **Objects** field to display the subobjects of the selected object. This feature allows you to move within an extensive hierarchy—such as a pull-down menu—without having to interact with every level.

Edit. Selecting this button causes the information notebook of the object highlighted in the **Objects** list to appear.

Add. Selecting this button causes the object appearing in the combo box (located in the lower right corner of the page) to be added to the current object. The new subobject's name will immediately appear at the end of the **Objects** list.

Delete. Selecting this button causes the highlighted object in the **Objects** list to be deleted. (<Ctrl Delete> also performs this action.)

Move Up. Selecting this button causes the highlighted object in the **Objects** list to move up one line in the list. (<Ctrl up-arrow> also performs this action.) Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it also changes the order that the items appear within the parent lists.

Move Down. Selecting this button causes the highlighted object in the **Objects** list to move down one line in the list. (<Ctrl down-arrow> also performs this action.) Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it also changes the order that the items appear within the parent lists.

Add object combo box. This field designates the type of object to be added when the **Add** button (described above) is selected. Select the combo box button to view a list of the available objects. If you select one of the objects listed, it will be added to the current parent object whenever the **Add** button is selected.

Cut

Selecting the **Cut** option removes the current object from the screen and places it in a global paste buffer.

Copy

Selecting the **Copy** option copies the current object and places the copy in a global paste buffer.

Paste

Selecting **Paste** allows you to recall and position on the screen the contents of the global paste buffer (placed there by a **Cut** or **Copy** procedure). After selecting **Paste,** position the mouse cursor where you would like the paste to occur and press the left mouse button.

Only one edit group is permitted per window at one time. Upon grouping a second region, the first group in the same window will be ungrouped.

A group can be edited collectively by double-clicking on any part of the gray area. The following information notebook will appear:
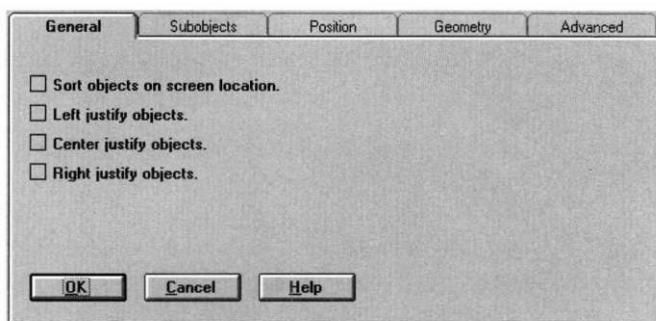
Edit Group page

The **Edit Group** notebook page contains the following options:

Sort objects on screen location. Causes the objects within the group to be sorted in top-left to bottom-right priority. This change in priority can be seen in the **Objects** field of the **Subobjects** page, where the group's objects will be reordered in the list according to their screen position, instead of according to their order of creation.

Left justify objects. Causes all of the objects within the group to be displayed next to the left border of the gray area.

Center justify objects. Causes all of the objects within the group to be displayed in the center of the gray area.

Right justify objects. Causes all of the objects within the group to be displayed next to the right border of the gray area.

Subobjects page

The objects contained within the group can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for an edit group is a string.

Delete

Selecting **Delete** removes the current object from the screen and deletes it from the file.

Move

Selecting **Move** allows you to move the current object by using the arrow keys.

Size

Selecting **Size** allows you to size the selected region relative to the top left corner using the arrow keys.

Group

Selecting **Group** allows you to select a region of objects to be combined into one unit, referred to as an edit group. After selecting **Group** place the mouse cursor at a corner of the region to be grouped and drag the mouse to the opposite corner. Any object overlapped by the drag process will be included in the group. Upon releasing the mouse, the background of the entire region will be marked by a light-gray shadow. This designates the area encompassing the group.

NOTE: Grouping objects can also be accomplished by pressing <Ctrl> and then dragging the mouse to mark the desired region.

Position page	The **Position** page contains the information related to an edit group's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.
Geometry page	The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.
Advanced page	The **Advanced** page contains information relating to the advanced properties of an edit group. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Ungroup

Selecting **Ungroup** causes the group within the current window to be dissolved. In other words, the grey shadow surrounding the grouped objects disappears, and the objects behave once again as separate entities. If the current window does not contain an edit group, **Ungroup** has no effect.

NOTE: Creating another group will also dissolve an edit group.

Edit Options

Window Options

T
A. he Window category options allow you to create, modify and retrieve window resources in the current file. Selecting **Window** causes the following menu to appear:
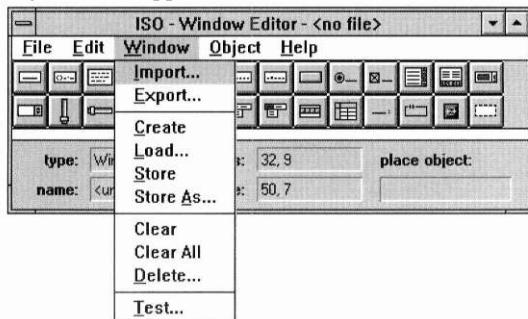
Import

Import allows you to import a resource window from another **.DAT** file or Windows **.RC** file. This process involves two simple steps, the first of which is selecting the file containing the desired resource. Consequently, upon selecting **Import,** a window similar to the **File I Open** window appears:
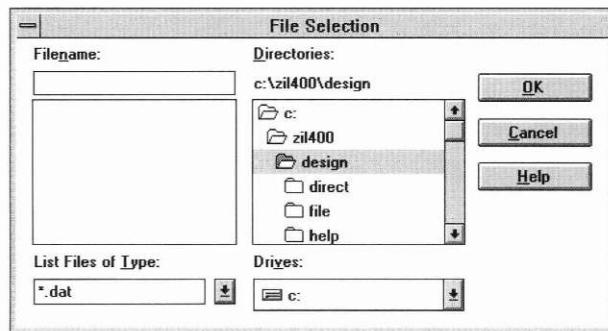
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File** I **Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:
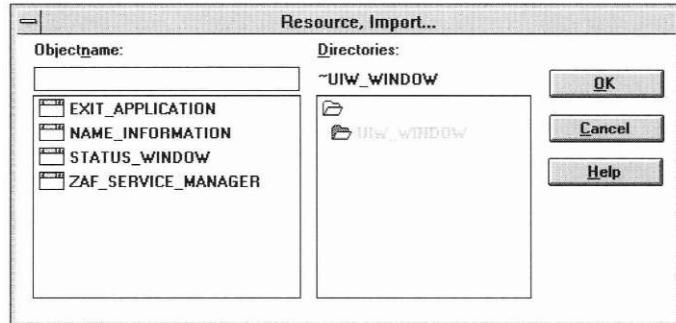
This window requests the actual resource to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a resource window, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the resource will automatically appear at the prompt.

Other resources that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these resources causes the name to appear in the **Objectname** field. Double clicking on a name listed in the resource list will cause that resource to be imported immediately.

Directories. The current directory is shown below the **Directories** prompt. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current **.DAT** file. The .. characters represent the parent directory, and, if selected, will display the other sub-directories of the current path, all of which are also selectable.

OK. Selecting this button causes the resource specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the resource entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.
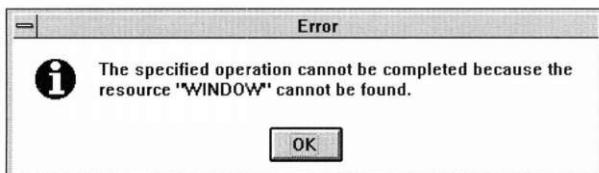
Once the resource has been imported, it can be accessed through **Window I Load...** (See page 233 of this chapter for more information on loading resources.).

Cancel. Selecting this button causes the window to close without executing any changes.

*Help.* Additional information about importing resources appears when this button is selected.

Export

Export allows you to export a resource window to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file to which you would like to export the resource window. Consequently, upon selecting **Export,** a window similar to the **File I Open** window appears:
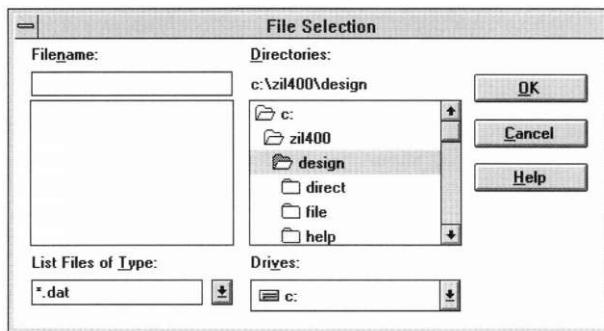
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:
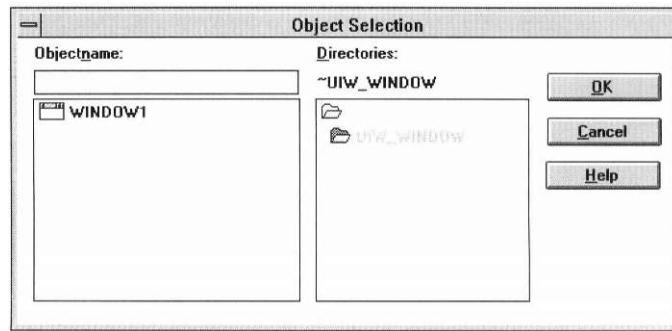
This window requests the actual resource window to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a resource window, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the resource window will automatically appear at the prompt.

Other resource windows that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these resource windows causes the name to appear in the **Objectname** field. Double clicking *on* a name listed in the resource window list will cause that resource window to be exported immediately.

Directories. The current directory is shown below the **Directories** prompt. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current drive, the current directory being highlighted and child directories shown below the current one and parent directories above.

OK. Selecting this button causes the resource window specified at the **Objectname** prompt to be exported. If the export procedure is successful, the window will close. If the resource window entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

*Cancel.* Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting resource windows appears when this button is selected.

Create

Selecting **Create** automatically places the following window on the screen, complete with a title bar, a system button, and minimize and maximize buttons.
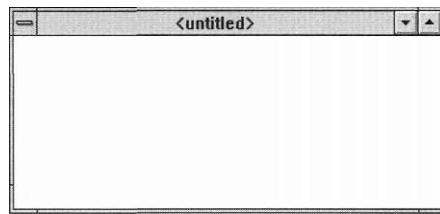
To modify the properties of the window resource, call its information notebook by double clicking with the mouse anywhere within the border. The following window appears:
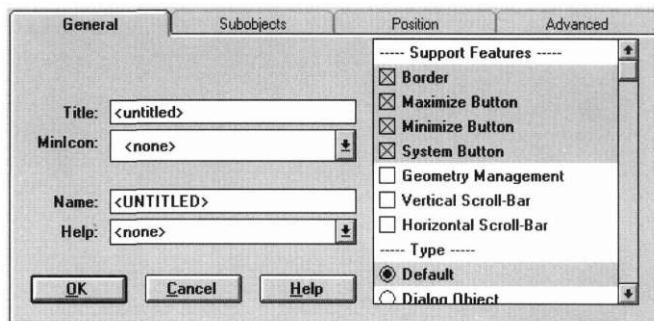
For information on interacting with the resource window's information notebook, refer to the Subwindow section of Chapter 17 on page 329.

Any object can be attached to the resource window by selecting it from the button bar, or from the Object menu, and positioning it in the window. (See Chapter 13 for more information on creating window objects.)

Load

Load... is used to recall a previously created resource window from the current file. Selecting it causes a window similar to the following to appear:
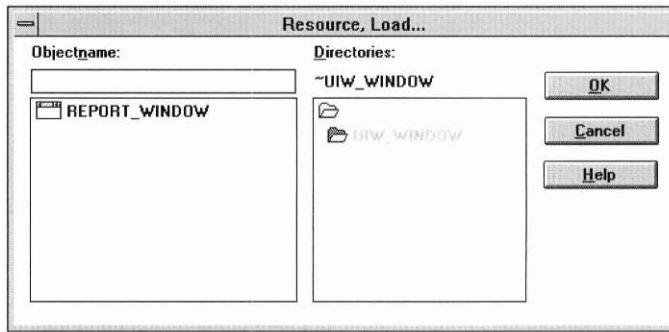
Objectname. To load a resource window, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the resource will automatically appear at the prompt.

Other resources that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these resources causes the name to appear in the **Objectname** field. Double clicking on a name listed in the files list will cause that resource to be loaded immediately.

Directories. The current window directory is shown below the **Directories** prompt. Other window directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

*OK.* Selecting this button causes the resource specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the **Resource I Load** window will close and the resource window, containing its child objects (if any), appears on the screen in the exact location and condition it was last stored.

If the resource entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

*Cancel.* Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading resources appears when this button is selected.

Once the resource has been loaded and appears on the screen, it is the current object and can be modified in any way. When the **Resource I Store** option is subsequently selected, the resource will be saved in its present condition, replacing the original version. (See the **Store** and **Store As** sections of this chapter for more information on storing resources.)

Store

Selecting the **Store** option causes the current resource window to be saved in its present condition to the current file. The name given the resource will be the name which appears at the **Name** prompt of the resource window's information notebook. If you have not entered a name for the resource in its information notebook or through a **Store** As operation, you will be queried for a name before you can store the resource.

NOTE: Each time a store operation is performed, the previous contents of the resource are completely replaced by the current information.

Store As

Store As... is generally used to store the current resource under another name. Selecting it causes a window to appear that is similar to the following:
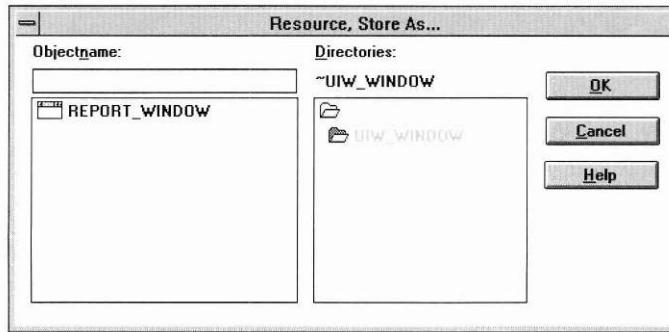
Objectname. Enter a name for the resource at the **Objectname** prompt, or, if you want to replace a previously created resource with the current information, select one from the field below, and the name for that resource will automatically appear at the prompt.

Other resources that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these resources causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that resource to be stored immediately.

Directories. The current window directory is shown below the **Directories** prompt. Other window directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the resource to be stored under the name entered at the **"Objectname"** prompt. If the store operation is successful, the **"Resource I Store As..."** window closes.

If you have entered a window name that already exists, a modal window will appear, indicating such. If you select the **"Yes"** button of this window, the current information replaces the original information of that file, and both

the modal window and the **"Store As"** window close. Selecting the **"No"** button simply closes the modal window and allows you to enter information again in the **"Store As"** window.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing resources appears when this button is selected.

Clear

Selecting **Clear** causes the current resource window to be removed from the screen. It does not, however, delete the resource from the file. If you have not stored the current resource immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the resource is neither stored nor cleared.

NOTE: In order to avoid unintentional clearing, **Clear** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and pressing <Enter>.

Clear All

Selecting **Clear All** causes all resource windows currently displayed to be removed from the screen. It does not, however, delete any of those resources from the file. If you have not stored any of the resources immediately before, selecting **Clear All** causes a modal window to appear that asks if you want to store them before clearing them from the screen. Selecting **Yes** causes the

window resources to be stored and then cleared, selecting **No** causes them to be cleared without storing them first, and selecting **Cancel** simply closes the modal window and the resources are neither stored nor cleared.

NOTE: In order to avoid unintentional clearing, **Clear All** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and pressing <Enter>.

Delete

The **Delete...** option allows you to delete a resource window from the current file. Selecting it causes a window similar to the following to appear:
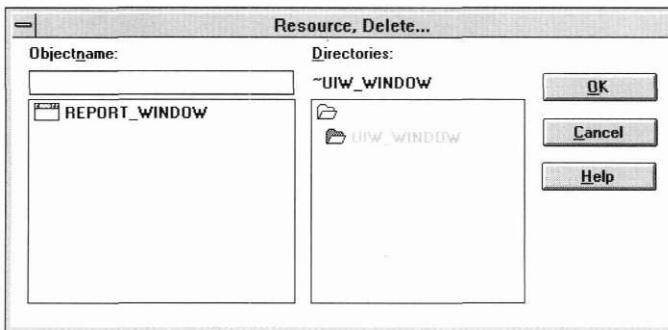
Objectname. Enter the name for the resource to be deleted at the **Object-name** prompt, or select one from the field below, and the name for that resource will automatically appear at the prompt.

Other resources that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these resources causes the name to appear in the **Objectname** field.

Directories. The current window directory is shown below the **Directories** prompt. Other window directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

The puipose of this window is to make sure that you want to delete the resource. If you select the **OK** button, the resource indicated at the **Name** prompt is deleted from the current file, and both the confirmation window and the **Delete Resource** window close. If you choose the **Cancel** button, the resource is not deleted and just the confirmation window closes.

If the resource entered does not exist, you will receive an error message when the OK button is selected.

If no information has been entered within the window, selecting OK causes an error message to appear.

If the delete operation is successful, the **Delete Resource** window closes, and the resource window, including its child objects (if any), is removed from the screen and is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting resources appears when this button is selected.

Test

The **Test** option allows you to test the objects of your current application resource so that you can see how they will function for an end user. Selecting **Test** causes the control window to be cleared from the screen and moves your application into test mode, which looks similar to the following:
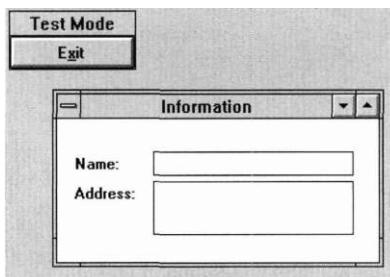
In test mode the objects of your application will look and act as they will for an end user. For example, check boxes and radio buttons will actually toggle and scroll bars will actually scroll information. No objects can be created or modified while in test mode.

When you have finished testing the resource, select the **Exit Test** button and the screen will return to normal mode. The control window will be displayed again, and you will be able to modify your application in any manner.

Window Options

Chapter 13

Object Options

The **Object** category provides options that allow you to actually create objects. Selecting **Object** causes the following menu to appear:
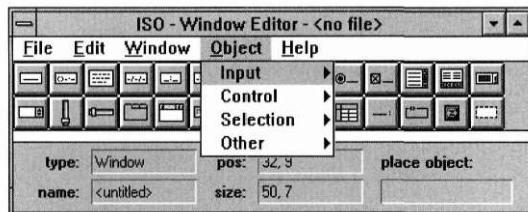
Each of the options on this menu is a category under which several window objects are classified. Selecting one of the options causes another associated menu to appear, which lists the actual window objects of that category.

To create an object, select it from the associated menu. Position the mouse cursor where you want the object to appear on the resource window and press the left mouse button. Subsequently pressing the right mouse button creates another instance of the most recently placed object which can then be placed on the window.

**NOTE:** All objects must be attached to a resource parent window; they cannot be attached directly to the screen. (See Chapter 12 for more information on creating resource windows.)

The information notebook of each of these objects can be accessed by either of the following methods:

- Select **Edit I Object** while the object is current
- Double-click on the object with the mouse

Each object created with OpenZinc Designer can be modified through interaction with its object information notebook. Selecting **Edit** I **Object** causes the information notebook for the current object to appear, which is similar to the following:
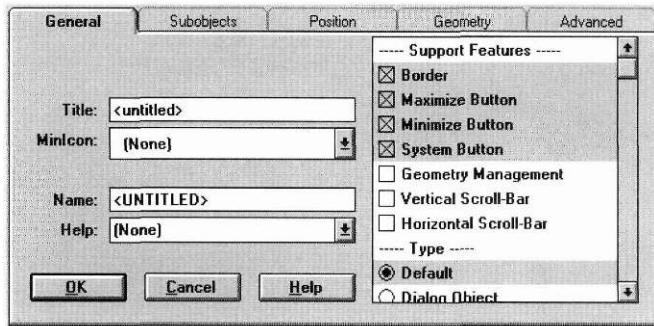
The object information notebook controls the general presentation of the object. Each information notebook is divided into notebook pages that contain related information. To summon the information contained in a page, simply click on its title tab with the mouse. Since each object has its own specific requirements, each information notebook's pages and their related fields will vary, but all contain one or more of the pages described in the next several sections.

(See Chapter 14 through Chapter 17 for more specific information regarding individual field objects.)

(NOTE: The remainder of this chapter is identical to the information given in the **Object** section of the **Edit** chapter.)

Each notebook page within the information notebook includes three buttons which operate in the following manner:

OK. Selecting this button saves the edit information and closes the object information notebook. The current object will reflect the editing changes immediately. If no information has been entered within the object information notebook, its window will close with no other action taking place.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about the current object appears when this button is selected. Help information is unique for each page. The contents depend on the type of object and the current page.

General page

The **General** notebook page (shown above) contains information related to the specific object being edited. It is always the first page visible upon entering the information notebook. Since each object is different, the editable properties will vary, but the following fields are common to most objects:

Text or Value. This field allows you to enter information to be displayed within the object exactly as you want it to appear in your application. Objects that use the **Text** field are:

- string
- formatted string
- text
- button
- radio button
- check box
- pull-down item
- pop-up item
- prompt, and
- group.

Some objects have a field similar to **Text,** but the name **Value** is used in place of **Text.** Objects that use the **Value** field are:

- date
- time
- bignum
- integer, and
- real.

Compare. Some objects allow a compare function, which is typically used to sort information. If you want to have a compare function associated with the object, you can enter the function's name in this field.

When using a compare function, the function itself must be defined somewhere in your code with the same name that is entered in the information notebook. This is necessary so that OpenZinc can find it and execute the designated action. (For more information on creating compare functions, refer to the description of the object's constructor in the *Programmer's Reference.)*

Name. This field contains the name (sometimes called a string identification) for the object and is present in every object information notebook. The default name for an unnamed object attached to another object is **FIELD** plus a unique number corresponding with the order in which it was attached to the parent window. For example, the default name for the second object created within a resource window would be **FIELD**_2.

Because these objects appear in various lists throughout the program, it is recommended that you override the default name and enter a string that more specifically identifies the object. The name will appear in all locations exactly as you have entered it in the object's information notebook.

Help. This field designates the help context to be associated with the object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the object and requests help. (See the Help Editor section of Chapter 18 for information on creating help contexts.)

options list. This field is located on the right side of the General page. It displays options which control the general presentation and operation of the object. All of these items are listed with either check boxes or radio buttons. To toggle an option from selected to unselected or vice versa, select it by either clicking on it with the mouse or by scrolling to it and pressing <space>. There is no limit to the number of check box options that can be in effect at a given time. However, only one radio button option per designated group can be in effect at one time.

Position page

The **Position** notebook page contains information related to an object's position, border, region, and alignment. Selecting the **Position** tab causes a page similar to the following to appear:

The fields of the Position page are the same for every object. They are as follows:

Position/Size. This field allows you to alter the positioning and size of the object. The first row within the field contains the following three buttons that determine the coordinate scale to be used:

- *pixel.* Depressing the left button causes screen coordinates to be based on a pixel scale. This option allows for the greatest precision in positioning and sizing. It is only applicable in graphics mode.

- *mini-cell.* Depressing the middle button causes screen coordinates to be based on a mini-cell scale, which is 1/10 of a cell coordinate by default. This option allows for more precision than the cell option, but less than the pixel option. It is applicable in graphics mode only.

- *cell.* Depressing the right button causes screen coordinates to be based on a whole cell scale. This is the default setting.

Underneath each button in the **Position/Size** field is a column of fields that display the precise location and size of the object. These are based upon the coordinate option selected, such as pixel, minicell, or cell. Only the column underneath the currently selected option is selectable and editable.

The top two fields of each column, **column** and **line,** determine the object's location. Changing the numbers in either or both of these fields will cause the object to move to the corresponding coordinates on the screen.

The lower two fields, **width** and **height,** determine the object's size. Changing the numbers in either or both of these fields will cause the object to alter in size according to the coordinate scale being used. For example, if the cell option is in effect, entering a width of 20 and a height of 10 in the cell column would produce an object that is 20 cells wide and 10 cells high.

Border. The two buttons contained in this field determine whether or not the object is displayed with a border. Depressing the left button causes the object to be displayed without a border, while depressing the right button causes a border to be displayed.

Region. The two buttons contained in this field determine the region allocated for the object. Depressing the left button causes the object to occupy the region specified by the values displayed in the **Position/Size** field. Depressing the right button causes the object to ignore its position and size parameters and occupy the remaining available space in its parent window.

Alignment. The three buttons contained in this field determine the alignment of the text within the object. Depressing the left button causes the text to be justified to the left margin, depressing the middle button causes the text to be centered, and depressing the right button causes the text to be justified to the right margin. For objects that do not contain text, such as scroll bars, the **Alignment** field is shaded, indicating that it is not selectable.

Geometry page

The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For example, using geometry management, you can determine how an object will react to re-sizing of its parent object—whether it stays at a fixed location or moves relative to the parent, whether it stays the same size or grows with the parent, etc.

Geometry management also allows you to establish a connection relationship between two child objects and to set size constraints for individual objects.

Selecting the **Geometry** tab causes a page similar to the following to appear:

The fields of the **Geometry** page are the same for every object. They are as follows:

attachment group. This area, located on the left side of the page, is designed to visually represent the relationship between the current object and its surroundings. The small square in the center represents the object itself, and the four lines, one radiating from each side of the square, represent possible connections to the surrounding objects. Each line ends at a combo box field which displays the type of connection for that line.

To change the type of connection, select the combo box button and a list of available types will be displayed. The following three types are available:

- *none.* Indicates that no connection is in effect.

- *relative.* Causes the object's border to position itself within its parent relative to the parent's borders. In this case, the value entered into the **offset** field (described below) will represent a percentage ratio for placement. For example, if an object's left connection has an offset value of 10, it will always be placed within the parent's left border at a distance measuring 10 percent of the parent object's size. Therefore, if the parent object measures 20 minicells wide, the child's left border will be two minicells within the parent's left border. If the parent is subsequently enlarged to be 100 minicells wide, the child will reposition at the edge of the tenth minicell.

  NOTE: A relative connection is possible only between a child object and its parent—not between two child window objects.

- *absolute.* Causes the object's border to position itself relative to its parent or sibling object at a fixed distance. In this case, the value entered into the **offset** field (described below) will represent a fixed measurement for

placement. For example, if an object's left connection has an offset value of 10, it will always be placed within the parent's left border at a distance measuring 10 units of whatever coordinate scale is in effect (i.e., pixel, mini-cell, or cell). Therefore, if the parent object measures 20 minicells wide, the child's left border will be ten minicells within the parent's left border. If the parent is subsequently enlarged to be 100 minicells wide, the child will remain at the edge of the tenth minicell.

It should be noted that the more connections are made on an object the more likely it is that conflicts will arise. For example, if an object has a left absolute connection offset at 10 cells and a right absolute connection offset at 20 cells, it is reasonable to assume that both constraints cannot always be accommodated. In such cases, an established priority schedule is put into effect that settles the conflict. The priority schedule is as follows:

- 1st priority—left

- 2nd priority—top

- 3rd priority—right

- 4th priority—bottom

Therefore, in our example above, the left connection would consistently offset the designated 10 cells, while the right connection would be overridden to accommodate the left.

Constraint features. This group determines the constraint features of the connection in the **attachment group** currently being edited. It contains the following fields:

- *offset.* Determines the offset of the current connection, or the amount of space between its inside and outside anchors. The actual measurement of the value depends upon whether the connection is relative, in which case the value represents a percentage ratio, or absolute, in which case the value represents a fixed measurement. (See the **attachment group** section above for further information on connection types.) The value of this number also depends upon the coordinate scale in effect (either pixel, minicell, or cell) for the current object. For example, for an object using pixel units, an absolute connection's offset of 20 would translate into a blank space measuring 20 pixels. (For further information on coordinate scales, see the **Position** page section, page 245, of this chapter.)

- *options.* Presents several options that alter the typical behavior of the connections. The following options are available:

stretch. Causes the actual object to stretch or shrink as the anchor border is moved. For example, an object with a bottom connection to its parent window's bottom border will stretch vertically as the window's bottom border is pulled downward.

opposite. Causes the connection to be anchored to the opposite border of the object designated in **connect.** For example, if object A is located to the left of object B and its left border connection is set to run to object B, then the connection will actually be anchored to object B's right border.

hz-center. Causes the connection to be anchored from the horizontal center of the current object. This allows for more consistent placement, especially when centering the object. This option is selectable for a relative connection only.

vt-center. Causes the connection to be anchored from the vertical center of the current object. This allows for more consistent placement, especially when centering the object. This option is selectable for a relative connection only.

- *anchor.* This field determines what the current connection is anchored to. As in the above examples, it is common for an object to run connections to the border of its parent object, such as a parent resource window. However, it is also possible to run absolute connections to other objects. The **anchor** field applies to each connection individually, not to all four collectively. Therefore, it is possible to have each border of the object connected to a different object or parent border. The **anchor** field displays the anchor for the connection currently being edited.

To choose an anchor for the current connection, first select a relative or absolute connection (described above), then select the **anchor** field's combo box button. A list that includes the surrounding objects and the parent object will appear. Selecting one of these displays the object's name in the **anchor** field.

NOTE: Since a relative connection can only be anchored to its parent object, the **anchor** field is not selectable when the current connection is relative.

Size Restrictions. This field determines the minimum and maximum size for the object. The first row determines the range of width values, while the second row determines the range of height values. The first value of each row represents the minimum size possible, and the second the maximum possi-

ble. All of these values are measured in cell units. For example, an object with a minimum width value of 5 and a maximum value of 20 is constrained to be no less than 5 cells wide but no greater than 20 cells.

Advanced page   The **Advanced** notebook page contains information relating to the advanced properties of an object. It is designed for use by the experienced user only.
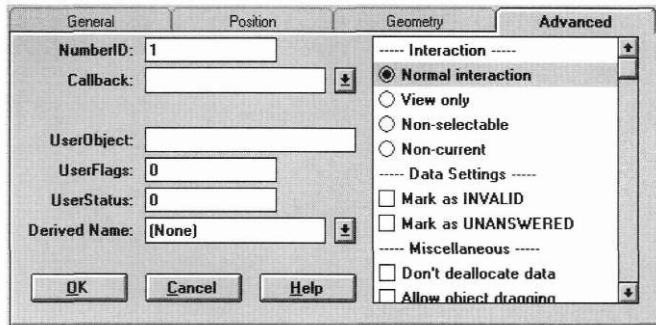
The following fields are common to most objects:

NumberlD. This field assigns a unique programmer number to the object. By default this number corresponds to the order in which the object was created in the program.

Callback. If you want to have a callback function, or user function, associated with the object, you can enter the function's name in this field. A callback function causes a designated action to occur when the user interacts with the object.

When using a callback function, the function itself must be defined somewhere in your code with the same name that is entered in the information notebook. This is necessary so that OpenZinc Designer can find it and execute the designated action. (For more information on creating callback and compare functions, refer to the description of the object's constructor in the *Programmer's Reference.)*

UserObject. This field designates a void pointer to a user-specified object.

UserFlags. This field contains any flags that are set and maintained by the programmer.

User Status. This field contains any status flags that are set and maintained by the programmer.

Derived Name. This field contains the name of the derived class to inherit the properties of the current object. If a name is entered, the object table in the **.CPP** file, generated by the Designer, will contain an entry for the derived object's **New()** function. As part of the code for the derived class, the programmer must create a static **New()** function that is able to call the constructor for the derived class. The **.HPP** file will contain a definition of the derived class' identification.

options list. This field is located on the right side of the Advanced page, and it displays options which control the advanced properties and operations of the current object. All of these items are listed with either check boxes or radio buttons. To toggle an option from selected to unselected or vice versa, select it by either clicking on it with the mouse or by scrolling to it and pressing <space>. There is no limit to the number of check box options that can be in effect at a given time. However, only one radio button option per designated group can be in effect at one time.

Options that are not applicable to the current object are grayed out, meaning that they cannot be selected.

The first section of the list presents options that determine the extent to which the end user can interact with or edit the object. Only one of these can be current at a time. The options are:

- *Normal interaction.* Allows the end user to have normal interaction with the object, including being able to select, position on and edit it (if it is an editable object).

- *View only.* Prevents the object from being edited. If this flag is set, the end user will not be able to edit an object's information but will be able to browse through the information.

- *Nonselectable.* Prevents the object from being selected. If this flag is set, the user will not be able to edit or position on the object's information.

- *Noncurrent.* The object cannot be made current. If this flag is set, users will not be able to select the object from the keyboard nor with the mouse.

The second section of the list presents options that determine the initial settings of the object. The options are:

- *Mark as INVALID.* Sets the initial status of the object to be **invalid.** This forces the user to enter an acceptable value before moving to another field.

- *Mark as UNANSWERED.* Sets the initial status of the field to be **unanswered.** An unanswered field is displayed as blank space on the screen.

The third section of the list presents options that determine other default settings for the object. The options are:

- *Don't allocate data.* Causes the object to not allocate its own buffer for data. If this option is set, you must allocate a buffer that will be used to store the object's data.

- *Allow object dragging.* Allows the object to be dragged in a drag and drop operation.

- *Allow object dropping.* Allows objects to be dropped onto the current object in a drag and drop operation.

- *Support object.* Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window, such as a window's border and title.

Subobjects page   Objects which can host other objects also contain a **Subobjects** notebook page within their information notebooks. This page allows you to modify the subobjects.

It contains the following fields:

Objects. This field displays the objects, listed in the order in which they were created, that are attached to the current object. The list is divided into two sections, the first for support objects, or objects that are not displayed as part of the user region, and the second for normal objects. The information notebook of any one of these objects can be accessed by one of the following methods: double click on the desired object with a mouse, scroll to it and press <Space>, or click on the **Edit** button while it is highlighted in the list.

To move an object in the list up one line, either select the **Move Up** button, or hit <Ctrl up-arrow>. To move an object down in the list one line, either select the **Move Down** button, or hit <Ctrl down-arrow>. Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it changes the order that the items appear within the parent lists. To delete an object from the list, either select the **Delete** button, or hit <Ctrl Deletex

Directories. This field displays the other objects—both parent and subobjects—that are related to the current object. Double clicking on one of the names (or hitting <Enter> while the name is current) causes the **Objects** field to display the subobjects of the selected object. This feature allows you to move within an extensive hierarchy—such as a pull-down menu—without having to interact with every level of it.

Edit. Selecting this button causes the information notebook of the object highlighted in the **Objects** list to appear.

Add. Selecting this button causes the object appearing in the combo box (located in the lower right corner of the page) to be added to the object being edited. The new subobject's name will immediately appear at the end of the **Objects** list.

Delete. Selecting this button causes the highlighted object in the **Objects** list to be deleted. (<Ctrl Delete> also performs this action.)

Move Up. Selecting this button causes the highlighted object in the **Objects** list to move up one line in the list. (<Ctrl up-arrow> also performs this action.) Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it changes the order that the items appear within the parent lists.

Move Down. Selecting this button causes the highlighted object in the **Objects** list to move down one line in the list. (<Ctrl down-arrow> also performs this action.) Moving an object changes the tab order of objects on the screen. For list item objects, such as pop-up items, it changes the order that the items appear within the parent lists.

add object combo box. This field designates the type of object to be added when the **Add** button (described above) is selected. Select the combo box button to view a list of the available objects. If you select one of the objects listed, it will be added to the current parent object whenever the **Add** button is selected.

To test how an object will actually appear and function for the end user, try it in test mode, which is accessed by selecting **Window I Test** while the parent resource window is active. (See Chapter 12 for more information on testing objects.)

A description of each window object, grouped according to its category type, is documented in the following four chapters. For more specific information on how these objects are created, refer to the *Programmer's Reference*.

Chapter 14   # Input Objects

The input category includes objects that are used specifically for data input. Selecting the **Input** option causes the following associated menu to appear:
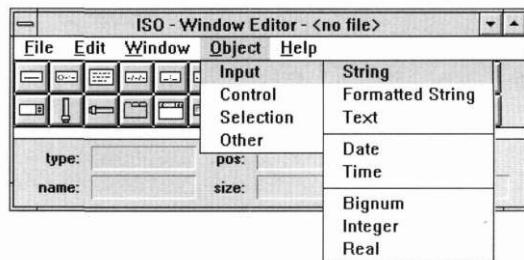
String

A string object is used to present and collect alphanumeric string information. Selecting **String** causes the following object to appear:

```
string
```

The string object may be placed by clicking on the window with the left mouse button. To modify the string object, call its information notebook by double clicking the mouse on the object. The following window will appear:
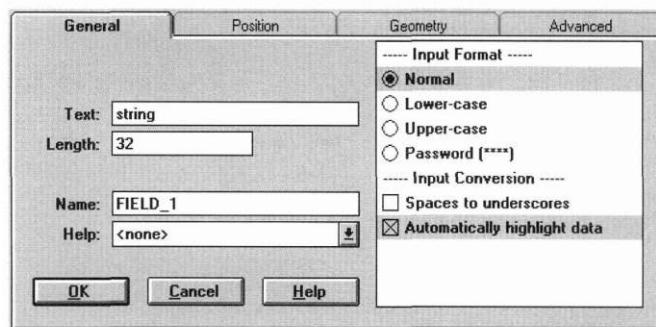
General

The **General** page contains information related specifically to the string object being edited. It contains the following fields:

Text. This field allows you to enter information to be displayed within the object exactly as you want it to appear in your application. If it contains more characters than the **Length** limitation allows, only the number of characters that fall within the limit will be displayed. If the string object is not long enough to display all of the entered text, it can be sized using the mouse or the arrow keys.

Length. The number in this field determines the number of characters that the string object will display. The default length is 32. The maximum length is 32,767.

Name. Enter in this field a name that will distinguish the string object from other objects on the window.

Help. This field designates the help context to be associated with the string. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the string and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the string object are listed in the field on the right half of the window. The first section presents options for formatting the string's input. Only one of these can be selected at a time. The options are:

- *Normal.* Causes the character input to be displayed exactly as it is entered.
- *Lowercase.* Converts all character input to lowercase values.
- *Uppercase.* Converts all character input to uppercase values.
- *Password.* Causes the characters entered into the string field to not be echoed to the screen; rather, the default character that is environment specific is printed for each character typed.

The second section of the list presents options for converting character input. The options are:

- *Spaces to underscores.* Converts the space character to an underscore value.
- *Automatically highlight data.* Causes the string to be highlighted when the user tabs to the string field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the string buffer will be cleared immediately.

Position
The **Position** page contains information related to the string's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry
The **Geometry** notebook page allows you to place constraints on a string object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced
The **Advanced** page contains information related to the advanced properties of the string. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Formatted string

A formatted string object is used to display and collect information that requires a specific format. For example, telephone numbers and zip codes are best presented as formatted strings. Selecting **Formatted String** causes the following object to appear:

[...] ...-....

To modify the formatted string object, call its information notebook. The following window appears:
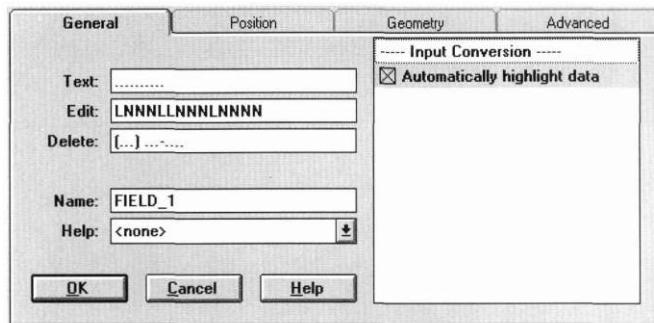
General　　　　The **General** page contains information related specifically to the formatted string object being edited. It contains the following fields:

Text. Enter text in this field as you want it to initially appear in the formatted string object. It must conform to the specifications set by the **Edit** and **Delete** fields. For example, a string *8017858900* would be appropriate for a U.S. formatted telephone number.

Edit. This field determines the type of characters that the formatted string will accept. The following characters can be used to define the edit mask:

- *a.* Allows the end user to enter a space (' ') or any letter (i.e., 'a' through 'z' or 'A' through 'Z').

- *A.* Same as the 'a' character option except that a lowercase letter is automatically converted to an uppercase letter.

- *c.* Allows the end user to enter a space (' '), a number (i.e., '0' through '9'), or any alphabetic character (i.e., 'a' through 'z' or 'A' through 'Z').

- *C.* Same as the 'c' character option except that a lowercase character is automatically converted to uppercase.

- L. Uses this position as a literal place holder. Using this character causes the formatted string to get the character to be read and displayed from the literal mask. The end user cannot edit this character.

- *N.* Allows the end user to enter any digit.

- x Allows the end user to enter any printable character (i.e., ' ' through

- *X.* Same as the 'x' character option except that a lowercase letter is automatically converted to an uppercase alphanumeric letter.

Enter in the **Edit** field a string of characters that will define the acceptable format for the string. For example, an edit mask of *LNNNLLNNNNLNNNN* would be appropriate for a U.S. formatted telephone number.

Delete. Enter into this field a string of literal characters that will be used whenever a character is deleted from a particular position in the formatted string. For example, a string of (...) .... would be appropriate for a U.S. formatted telephone number.

Name. Enter in this field a name that will distinguish the formatted string object from other objects on the window.

Help. This field designates the help context to be associated with the formatted string. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the formatted string and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the string object are listed in the field on the right half of the window. The following input conversion option is available:

- *Automatically highlight data.* Causes the current formatted string value to be highlighted when the user tabs to the field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the formatted string buffer will be cleared immediately.

Position

The **Position** page contains information related to the string's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry

The **Geometry** notebook page allows you to place constraints on a string object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced

The **Advanced** page contains information related to the advanced properties of the string. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Text

A text object is used to present and collect alphanumeric textual information in a multi-line format. Selecting **Text** causes the following box to appear:
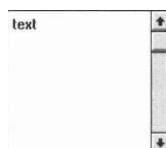
To modify the text object, call its information notebook. The following window will appear:
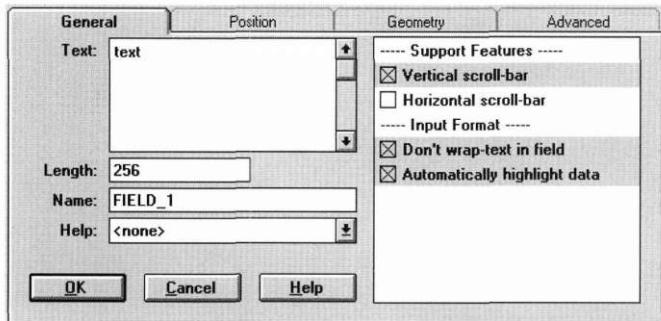
General

The **General** page contains information related specifically to the text object being edited. It contains the following fields:

Text. This field allows you to enter information to be displayed within the object exactly as you want it to appear in your application. If it contains more characters than the **Length** limitation allows, only the number of characters that fall within the limit will be displayed. If the text object is not large enough to display all of the entered text, it can be sized using the mouse or the arrow keys.

Length. The number in this field determines the number of characters that the text object will display. The default length is 256. The maximum length is 32,767.

Name. Enter in this field a name that will distinguish the text object from other objects.

Help. This field designates the help context to be associated with the text object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the text object and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the text object are listed in the field on the right half of the window. The first section presents options for support features. The options are:

- *Vertical scroll bar.* Adds a vertical scroll bar inside the right border of the text field.

- *Horizontal scroll bar.* Adds a horizontal scroll bar inside the bottom border of the text field.

The second section of the list presents options for formatting input. The options are:

- *Don't wrap text infield.* Disables the default word wrap in the text field.

- *Automatically highlight data.* Causes the current text value to be highlighted when the user tabs to the field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the text buffer will be cleared immediately.

Position The **Position** page contains information related to the text object's position, border and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry The **Geometry** notebook page allows you to place constraints on a text object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced The **Advanced** page contains information related to the advanced properties of the text object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Date

A date field displays and collects date information. Selecting **Date** causes a date field to appear that contains the current date, similar to the figure below:

8/26/1994
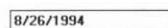
To modify the date, call its information notebook. The following window will appear:
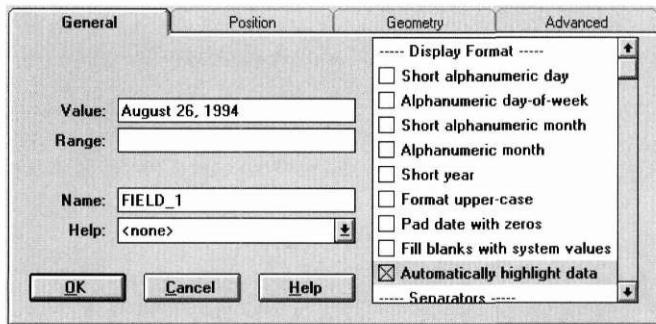
General

The **General** page contains information related specifically to the date object being edited. It contains the following fields:

Value. Enter in this field the date that you want to appear in the date object. In the U.S., the default format to which this date will be automatically converted is *month, day, year*, with the month spelled out.

Range. If you want to specify a certain range of acceptable dates, enter in this field the valid date ranges. This range is in a universal year-month-day format. For example, if you want to accept only those dates within the 1995 calendar year, enter the range of *1995-1-1..1995-12-31*. If no range is entered, any date will be accepted.

Name. Enter in this field a name that will distinguish the date object from other objects in the window.

Help. This field designates the help context to be associated with the date object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the date object and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the date object are listed in the field on the right half of the window. The first section presents options for formatting input. The options are:

• *Short alphanumeric day.* Adds a shortened day-of-week text to the date.

- *Alphanumeric day-of-week.* Adds an ASCII day-of-week string to the date.

- *Short alphanumeric month.* Uses a shortened alphanumeric month in the date.

- *Alphanumeric month.* Formats the month to be displayed as an ASCII string value.

- *Short year.* Forces the year to be displayed as a two-digit value.

- *Format uppercase.* Converts the alphanumeric date to uppercase.

- *Pad date with zeros.* Forces the year, month and day values to be zero filled when their values are less than 10.

- *Fill blanks with system value.* Fills a blank date with the system date. For example, if a blank date were entered by the end user and this option were set, the date would be set to the system date.

- *Automatically highlight data.* Causes the current date value to be highlighted when the user tabs to the date field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the date buffer will be cleared immediately.

The second section presents options for separating date values. The options are:

- *System defaults.* Separates each date value according to the default settings for the current system, which is typically using hyphens as separators for U.S. formats.

- *Dash separators.* Separates each date value with a dash, regardless of the default country date separator.

- *Slash separators.* Separates each date value with a slash, regardless of the default country date separator.

The third section presents options for formatting according to country and military standards. The options are:

- *System default format.* Formats the date according to the default settings for the current system (e.g., *month/day/year* for U.S. formats).

- *European format.* Forces the date to be displayed and interpreted in the European format (i.e., *day/month/year),* regardless of the default country information.

- *Asian format.* Forces the date to be displayed and interpreted in the Far East Asian format (i.e., *year/month/day),* regardless of the default country information.

- *Military format.* Forces the date to be displayed and interpreted in the U.S. Military format (i.e., *day month year* where *month* is a 3 letter abbreviated word), regardless of the default country information.

- *U.S. format.* Forces the date to be displayed and interpreted in the U.S. format (i.e., *month/day/year),* regardless of the default country information.

Position

The **Position** page contains information related to the date object's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry

The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced

The **Advanced** page contains information related to the advanced properties of the date object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Time

A time field displays and collects time information. Selecting **Time** causes a time field to appear that contains the current time, similar to the figure below:

10:50 a.m.

To modify the time object, call its information notebook. The following window will appear:
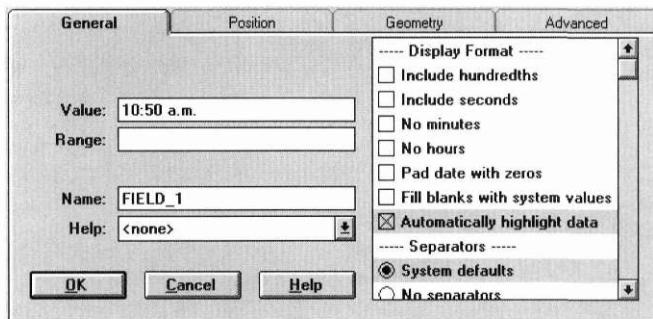
General

The **General** page contains information related specifically to the time object being edited. It contains the following fields:

Value. Enter in this field the time that you want to appear in the time object. This time will be automatically converted to the appropriate country format. For example, the format in the U.S. is *hour:minutes a.m.* or *hour:minutes p.m.* A space between numbers will be interpreted as a colon, and necessary periods (for *a.m.* and *p.m.)* are automatically inserted. Since any hour value under 12 is interpreted as morning, it is necessary to enter *p.m.* if the hour value is meant to be in post-meridian time and you are using a 12-hour clock. If you enter the time value according to a 24-hour clock, there is no need to enter *a.m.* or *p.m.*—the object will interpret and convert the value into the default format. Again, the time value's default formatting is country dependant.

Range. If you want to specify a certain range of acceptable time values, enter in this field the valid time ranges. For example, if you want to accept only those times whose values fall in post-meridian time, enter the range of *12:00..23:59:59.* If no range is entered, any time value will be accepted.

Name. Enter in this field a name that will distinguish the time object from other objects on the window.

Help. This field designates the help context to be associated with the time object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the time object and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the time object are listed in the field on the right half of the window. The first section presents options for display format. The options are:

- *Include hundredths.* Includes the hundredths value in the time. (By default the hundredths value is not included.)

- *Include seconds.* Includes the seconds value in the time. (By default the seconds value is not included.)

- *No minutes.* Does not display nor interpret a minute value for the time object.

- *No hours.* Does not display nor interpret an hour value for the time object.

- *Pad date with zeros.* Forces the hour, minute and second values to be zero filled when their values are less than 10.

- *Fill blanks with system values.* Fills a blank time with the system time. For example, if a blank ASCII time value were entered by the end user and the this option were set, the time would be set to the current system time.

- *Automatically highlight data.* Causes the current time value to be highlighted when the user tabs to the time field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the time buffer will be cleared immediately.

The second section presents options for separating time values. The options are:

- *System defaults.* Separates each time value according to the default settings for the current system. For example the U.S. format separator is a colon.

- *No separators.* Does not use any separator characters to delimit the time values.

- *Colon separators.* Separates each time value with a colon.

The third section presents options for formatting time input. The options are:

- *Normal.* Does not convert the input into lower nor uppercase, but displays it as it is entered by the end-user.
- *Format lowercase.* Converts the time to lowercase.
- *Format uppercase.* Converts the time to uppercase.

The fourth section presents options for formatting according to country standards. The options are:

- *System default format.* Formats the date according to the default settings for the current system. For example, the U.S. format is based on a 12-hour clock.
- *24 hour.* Forces the time to be displayed and interpreted using a 24-hour clock, regardless of the default country information.
- *12 hour.* Forces the time to be displayed and interpreted using a 12-hour clock, regardless of the default country information.

Position

The **Position** page contains information related to the time object's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry

The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced

The **Advanced** page contains information related to the advanced properties of the time object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Bignum

A bignum object is used to display and collect numeric information. It can be formatted in various ways, such as for numbers presented as percentages, currency and credit. Selecting **Bignum** causes the following object to appear:

```
0.00000000
```

To modify the bignum object, call its information notebook. The following window will appear:
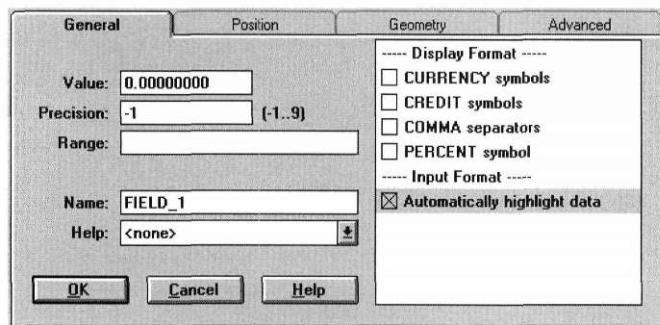
General

The **General** page contains information related specifically to the bignum object being edited. It contains the following fields:

Value. Enter in this field the number that you want to appear in the bignum field. The number will be displayed with the number of decimal places designated by the **Precision** field. A bignum object can have up to thirty digits to the left of the decimal place and up to eight digits to the right of the decimal place.

Precision. Enter in this field the number of decimal places to be displayed. Any whole number from -1 to 9 is acceptable.

Range. If you want to specify a certain range of acceptable bignum values, enter in this field the valid bignum range. For example, if you want to accept only numbers between 100 and 100,000, enter the range of *100..100000.* If no range is entered, any numeric value will be accepted.

NOTE: When specifying the range, OpenZinc requires that decimal places be separated by periods, regardless of the country's standard format.

Name. Enter in this field a name that will distinguish the bignum object from other objects on the window.

Help. This field designates the help context to be associated with the bignum object. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the bignum object and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation of the bignum object are listed in the field on the right half of the window. The first section presents options for formatting how the bignum is displayed. The options are:

- *CURRENCY symbols.* Displays the number with the country-specific currency symbol.
- *CREDIT symbols.* Displays the number with the country-specific credit symbols whenever the number is negative (e.g., '(' and ')' in U.S format).
- *COMMA separators.* Displays the number with commas.
- *PERCENT symbol.* Displays the number with a percentage symbol.

The second section presents the following option for formatting input:

- *Automatically highlight data.* Causes the current bignum value to be highlighted when the user tabs to the field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the bignum buffer will be cleared immediately.

Position   The **Position** page contains information related to the bignum object's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry   The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced The **Advanced** page contains information related to the advanced properties of the bignum object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Integer

An integer object is used to present and collect numeric information for integers. It cannot be formatted. (The bignum object must be used for numbers requiring special formatting capabilities.) Selecting **Integer** causes the following object to appear:

fo

To modify the integer object, call its information notebook. The following window appears:
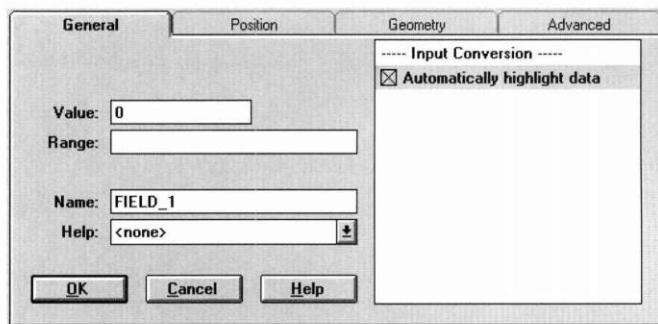
General The **General** page contains information related specifically to the integer object being edited. It contains the following fields:

Value. Enter in this field the integer that you want to appear in the integer field.

Range. If you want to specify a certain range of acceptable integer values, enter in this field the valid integer ranges. For example, if you want to accept only numbers between 100 and 10,000, enter the range of *100.. 10000.* If no range is entered, any integer value will be accepted.

Name. Enter in this field a name that will distinguish the integer object from other objects on the window.

Help. This field designates the help context to be associated with the integer field. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the integer field and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the input conversion of the integer object are listed in the field on the right half of the window. The following option is available:

- *Automatically highlight data.* Causes the current integer value to be highlighted when the user tabs to the integer field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the integer buffer will be cleared immediately.

Position The **Position** page contains information related to the integer object's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced The **Advanced** page contains information related to the advanced properties of the integer object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Real

A real number object is used to present and collect floating-point numeric information. Decimal numbers will be displayed using decimal notation. When the decimal strings are too large for the input field, they are automatically converted to scientific notation. Selecting **Real** causes the following object to appear:

| 0 |

To modify the real number object, call its information notebook. The following window appears:
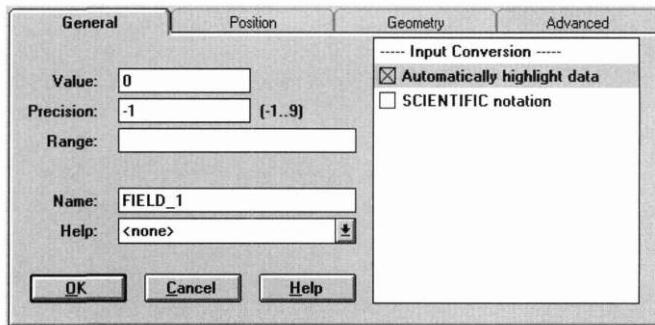
General   The **General** page contains information related specifically to the real number object being edited. It contains the following fields:

Value. Enter in this field the number that you want to appear in the real number field.

Precision. Enter in this field the number of decimal places to be displayed. Any whole number from -1 to 9 is acceptable.

Range. If you want to specify a certain range of acceptable real number values, enter in this field the valid real number range. For example, if you want to accept only numbers between 10.0 and 1000.0, enter the range of *10.0..1000.0.* If no range is entered, any real number value will be accepted.

Name. Enter in this field a name that will distinguish the real number object from other objects on the window.

Help. This field designates the help context to be associated with the real number field. Select the combo box button to view a list of the available help contexts. If you select one of the help contexts listed, the help message of that context will be displayed whenever the user positions on the real number field and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the input conversion of the real number object are listed in the field on the right half of the window. These options are:

- *Automatically highlight data.* Causes the current real value to be highlighted when the user tabs to the real number field (from another window field). If the user presses a key, without first having pressed any movement or editing keys, the real number buffer will be cleared immediately.

- *Scientific notation.* Displays the number using scientific notation.

Position

The **Position** page contains information related to the real number object's position, border, region and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry

The **Geometry** notebook page allows you to place constraints on an object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced

The **Advanced** page contains information related to the advanced properties of the real number object. It is designed for interaction by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Chapter 15   # Control Objects

The control category includes objects that are used to control the various operations of an application, its windows and window objects. Selecting the **Control** option causes the following associated menu to appear:
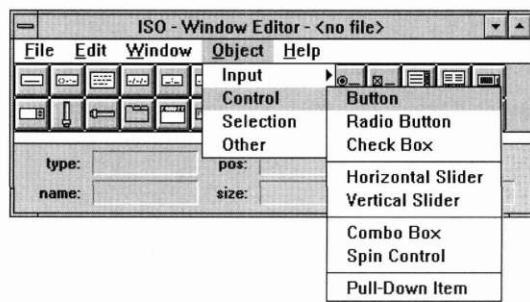
Button

A button is used to provide a selectable object that performs an action when selected. Selecting **Button** causes the following object to appear:

To modify the button, call its information notebook. The following window will appear:
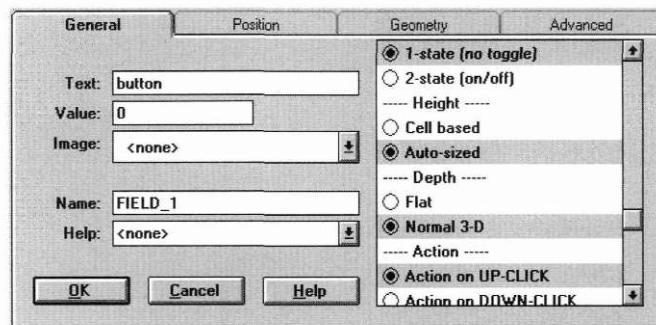
General page

The **General** page contains information related specifically to the button object being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear on the button. It will be centered vertically automatically. If the text string is longer than the length of the button, the button must be sized in order to display the entire text.

Value. This field allows you to enter a value that serves as a unique identification for a button. For example, you could associate the value 0 with an **OK** button, and a value of 1 with a **Cancel** button. This allows you to define one callback function that looks at the button values, instead of several functions that are tied to each button object. If the *Send user message* option is set, the value must be an event type, and a callback function should be assigned in the Advanced page.

Image. This field designates the bitmap image to be associated with the button. Select the combo box button to view a list of the available bitmaps. If you select one of the bitmaps listed, it will be displayed on the button. (See Chapter 19 for information on creating bitmap images.)

Name. Enter in this field a name that will distinguish the button object from other objects on the window.

Help. This field designates the help context to be associated with the button. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the button and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the button are listed in the field on the right half of the window. The first section presents options that determine the type of button. These options are:

- *Normal.* Causes the button to be presented as a normal three-dimensional button.

- *Radio-button.* Causes the button to appear and function as a radio button. All of the radio buttons in a group, list box, or window are considered to be members of the same group. Only one radio button in a group may be selected at any one time.

  (NOTE: A radio button can also be created by selecting **Object I Control I Radio Button** or by selecting it from the object bar. For more information on radio buttons, see the Radio Button section in this chapter.)

- *Check box.* Creates a check box that can be toggled when selected. More than one check box in a group may be selected at any one time. **(NOTE:** A check box can also be created by selecting Object I Control I Check Box or by selecting it from the button bar. For more information on check boxes, see the Check Box section in this chapter.)

The second section presents options that determine the button's settings. The options are:

- *Send user message.* Causes an event to be created from the button's value and put on the event queue when the button is selected. Any temporary windows are removed from the display when this message is sent. No callback function should be assigned in the Advanced page.

- *Set as default button.* Causes the button to be the default button on the window. A default button is selected when the user hits <Enter> from anywhere on the window. Only one button per window should be marked as a default button.

The third section presents options that determine the state of the button. The options are:

- *one-state (no toggle).* Does not toggle the button's state. A single-state button simply performs its action when selected. It does not remain in a selected state.

- *two-state (on/off).* Toggles the button's state. A two-state button performs its action when selected, but remains in a selected state until it is un-selected. The button will be drawn differently to signify that it is in a selected state. (The button will also have its WOS_SELECTED status flag set when it is in a selected state.)

The fourth section presents options that determine the sizing of the button. The options are:

- *Cell based.* The button's height is set to one cell height, or about the same height as a string object.

- *Auto-sized.* Automatically computes the run-time height of the button. If the application is running in text mode, the height is set to 1. If the application is running in graphics mode, the button is approximately 120% of the default cell height.

The fifth section presents options that determine the appearance of depth for the button. The options are:

- *Flat.* Causes the button to be displayed without a three dimensional appearance.

- *Normal 3-D.* Causes the button to be displayed with a three dimensional appearance.

The sixth section presents the options that determine when the button's action is performed. The options are:

- *Action on UP-CLICK.* Completes the button action on a down-click and release action.

- *Action on DOWN-CLICK.* Completes the button action on a button down-click, rather than on a down-click and release action.

- *Action on DOUBLE-CLICK.* Completes the button action when the button has been selected twice in rapid succession.

- *Action on TIME INTERVALS.* Continuously repeats the action if the user continues to hold the mouse button down on the button.

Position page   The **Position** page contains information related to the buttons's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a button object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information related to the advanced properties of the button. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Radio   button

A radio button is a type of button that displays not only text, but also an indicator that toggles. All of the radio buttons in a group, list box, or window are considered to be members of the same group. Only one radio button in a group may be selected at any one time. **(NOTE:** The radio button's parent

must not have the *Select multiple children* option set, or multiple radio buttons would be able to be selected.) Selecting **Radio Button** causes the following object to appear:

○ radio-button

NOTE: To use multiple radio button groups on the same window, use the group object. (See "Group" on page 317 for information on creating groups.)

To modify the radio button object, call its information notebook. The following window will appear:
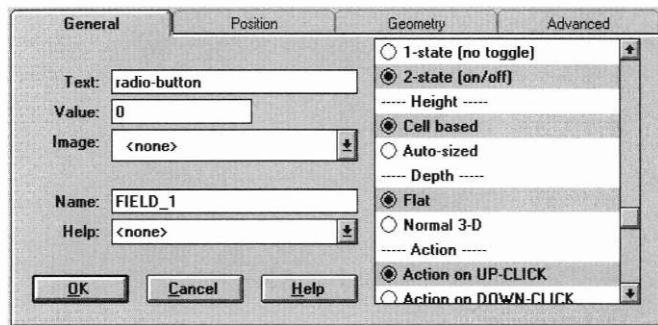
NOTE: The information notebook for the radio button is actually the information notebook for the standard button object but with the **Radio button** option set. If the **Normal** or **Check box** options are selected, the button will no longer be displayed as a radio button.

General page

The **General** page contains information related specifically to the radio button object being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear on the radio button. It will be centered vertically automatically. If the text string is longer than the length of the radio button, the button must be sized in order to display the entire text.

Value. This field allows you to enter a value that serves as a unique identification for a radio button. For example, you could associate the value 0 with an **OK** button and a value of 1 with a **Cancel** button. This allows you to define one callback function that looks at the radio button values, instead of

several functions that are tied to each button object. If the *Send user message* option is set, the value must be an event type, and a callback function should be assigned in the Advanced page.

Image. This field designates the bitmap image to be associated with the radio button. **(NOTE:** Do *not* attach a bitmap to a radio button since radio buttons by nature do not have bitmaps.)

Name. Enter in this field a name that will distinguish the radio button object from other objects on the window.

Help. This field designates the help context to be associated with the radio button. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the radio button and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the radio button are listed in the field on the right half of the window. The first section presents options that determine the type of button. These options are:

- *Normal.* Causes the button to be presented as a normal three-dimensional button.

- *Radio-button.* Causes the button to appear and function as a radio button. All of the radio buttons in a group, list box, or window are considered to be members of the same group. Only one radio button in a group may be selected at any one time. **(NOTE:** A radio button can also be created by selecting Object I Control I Radio Button or by selecting it from the button bar.)

- *Check-box.* Creates a check box that can be toggled when selected. More than one check box in a group may be selected at any one time.

  (NOTE: A check box can also be created by selecting **Object** I **Control** I **Check Box** or by selecting it from the button bar. For more information on check boxes, see the Check Box section in this chapter.)

The second section presents options that determine the radio button's settings. The options are:

- *Send user message.* Causes an event to be created from the radio button's value and put on the event queue when the radio button is selected. Any temporary windows are removed from the display when this message is sent. No callback function should be assigned in the Advanced page.

- *Set as default button.* Causes the radio button to be the default button on the window. A default button is selected when the user hits <Enter> from anywhere on the window. Only one button per window should be marked as a default button. Generally, a radio button should not be designated to be a default button.

The third section presents options that determine the state of the radio button. The options are:

- *one-state (no toggle).* Does not toggle the radio button's state. A single-state button simply performs its action when selected. It does not remain in a selected state. A radio button is a two-state button by nature.

- *two-state (on/off).* Toggles the radio button's state. A two-state button performs its action when selected, but remains in a selected state until it is un-selected. The radio button will be drawn differently to signify that it is in a selected state. (The radio button will also have its *WOS_SELECTED* status flag set when it is in a selected state.)

The fourth section presents options that determine the sizing of the radio button. The options are:

- *Cell based.* The radio button's height is set to one cell height, or about the same height as a string object. A radio button is always cell based.

- *Auto-sized.* Automatically computes the run-time height of the radio button. If the application is running in text mode, the height is set to 1.

The fifth section presents options that determine the appearance of depth for the radio button. The options are:

- *Flat.* Causes the radio button to be displayed without a three dimensional appearance. A radio button is always flat.

- *Normal 3-D.* Causes the radio button to be displayed with a three dimensional appearance.

The sixth section presents the options that determine when the radio button's action is performed. The options are:

- *Action on UP-CLICK.* Completes the radio button action on a down-click and release action.

- *Action on DOWN-CLICK.* Completes the radio button action on a radio button down-click, rather than on a down-click and release action.

- *Action on DOUBLE-CLICK.* Completes the radio button action when the radio button has been selected twice in rapid succession.

- *Action on TIME INTERVALS.* Continuously repeats the action if the user continues to hold the mouse radio button down on the radio button.

Position page
The Position page contains information related to the radio buttons's position, border, region, and alignment. For detailed information on the Position page, refer to the general description on page 213.

Geometry page
The Geometry notebook page allows you to place constraints on a radio button that specify how the object should be sized and positioned under specific conditions. For detailed information on the Geometry page, refer to the general description on page 214.

Advanced page
The Advanced page contains information related to the advanced properties of the radio button. It is designed for use by the experienced user only. For detailed information on the Advanced page, refer to the general description on page 218.

Check box

A check box is a type of button that displays not only text, but also an indicator that toggles. Any number of check boxes in a group may be selected at one time (the check box's parent should have the *Select multiple children* option set). Selecting **Check box** causes the following object to appear:

☐ check-box

To modify the check box object, call its information notebook. The following window will appear:
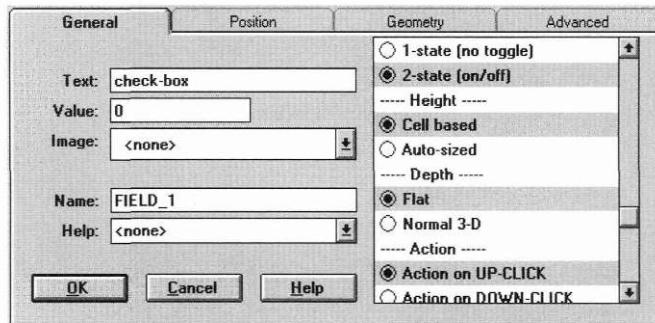
NOTE: The information notebook for the check box is actually the information notebook for the standard button object but with the **Check-box** option set. If the **Normal** or **Radio-button** options are selected, the button will no longer be displayed as a radio button.

General page

The General page contains information related specifically to the check box object being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear on the check box. It will be centered vertically automatically. If the text string is longer than the length of the check box, the button must be sized in order to display the entire text.

Value. This field allows you to enter a value that serves as a unique identification for a check box. For example, you could associate the value 0 with an **OK** button and a value of 1 with a **Cancel** button. This allows you to define one callback function that looks at the check box values, instead of several functions that are tied to each button object. If the *Send user message* option is set, the value must be an event type. No callback function should be assigned in the Advanced page.

Image. This field designates the bitmap image to be associated with the check box. **(NOTE:** Do *not* attach a bitmap to a check box, since check boxes do not have bitmaps.)

Name. Enter in this field a name that will distinguish the check box object from other objects on the window.

Help. This field designates the help context to be associated with the check box. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the check box and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the check box are listed in the field on the right half of the window. The first section presents options that determine the type of button. These options are:

* *Normal.* Causes the button to be presented as a normal three-dimensional button.

* *Radio-button.* Causes the button to appear and function as a radio button. All of the radio buttons in a group, list box, or window are considered to be members of the same group. Only one radio button in a group may be selected at any one time. **(NOTE:** A radio button can also be created by selecting **Object** I **Control I Radio Button,** or by selecting it from the button bar.)

* *Check-box.* Creates a check box that can be toggled when selected. More than one check box in a group may be selected at any one time. **(NOTE:** A check box can also be created by selecting **Object I Control I Check Box,** or by selecting it from the button bar. For more information on check boxes, see the Check Box section in this chapter.)

The second section presents options that determine the check box's settings. The options are:

* *Send user message.* Causes an event to be created from the check box's value and put on the event queue when the check box is selected. Any temporary windows are removed from the display when this message is sent. No callback function should be assigned in the Advanced page.

* *Set as default button.* Causes the check box to be the default button on the window. A default button is selected when the user hits <Enter> from anywhere on the window. Only one button per window should be marked as a default button. Generally, a check box should not be designated to be a default button.

The third section presents options that determine the state of the check box. The options are:

* *one-state (no toggle).* Does not toggle the check box's state. A single-state button simply performs its action when selected. It does not remain in a selected state, A check box is a two-state button by nature.

* *two-state (on/off).* Toggles the check box's state. A two-state button performs its action when selected, but remains in a selected state until it is un-selected. The check box will be drawn differently to signify that it is in a selected state. (The check box will also have its *WOS_SELECTED* status flag set when it is in a selected state.)

The fourth section presents options that determine the sizing of the check box. The options are:

* *Cell based.* The check box's height is set to one cell height, or about the same height as a string object. A check box is always cell based.

* *Auto-sized.* Automatically computes the run-time height of the check box. If the application is running in text mode, the height is set to 1.

The fifth section presents options that determine the appearance of depth for the check box. The options are:

* *Flat.* Causes the check box to be displayed without a three dimensional appearance. A check box is always flat.

* *Normal 3-D.* Causes the check box to be displayed with a three dimensional appearance.

The sixth section presents the options that determine when the check box's action is performed. The options are:

* *Action on UP-CLICK.* Completes the check box action on a down-click and release action.

* *Action on DOWN-CLICK.* Completes the check box action on a check box down-click, rather than on a down-click and release action.

* *Action on DOUBLE-CLICK.* Completes the check box action when the check box has been selected twice in rapid succession.

* *Action on TIME INTERVALS.* Continuously repeats the action if the user continues to hold the mouse check box down on the check box.

Position page

The **Position** page contains information related to the check box's position, border, region and, alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a check box that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information related to the advanced properties of the check box. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Horizontal   slider

A horizontal slider is typically used to visually indicate the current value relative to the range of possible values and to allow the setting of a value. For example, a slider could be used as a volume control for an application. It is different from a horizontal scroll bar, which is designed to scroll another object, such as a horizontal list. A slider is typically added directly to the current resource window, independent of any other object. In most environments a slider and a scroll bar look different, in that the slider track is narrower than that of a scroll bar. Also, a vertical slider's thumb button moves *up* as the slider's value increases, whereas a scroll bar's thumb button moves *down* as the scroll bar's value increases. Selecting **Horizontal Slider** causes the following object to appear:

To modify the horizontal slider, call its information notebook. The following window will appear:
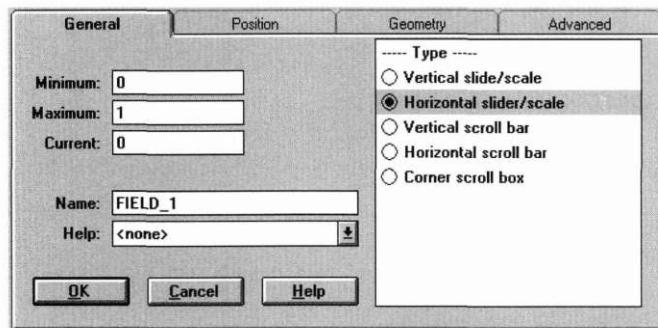
NOTE: The information notebook for the horizontal slider is actually an information notebook for a generic scroll bar object but with the **Horizontal slider/scale** option set. If this option is toggled off, or if another type is set, the object will no longer be a horizontal slider.

General page

The **General** page contains information related specifically to the horizontal slider object being edited. It contains the following fields:

Minimum. This field specifies the minimum value of the slider range. The actual number entered will not have meaning to the Designer, except that it is used, in conjunction with a **maximum** value, to determine the slider's incremental scale. For example, if the slider has a minimum value of 2 and a maximum value of 10, the slider thumb will be able to scroll eight increments from the leftmost to the rightmost position.

Maximum. This field specifies the maximum value of the slider range. The actual number entered will not have meaning to the Designer, except that it is used, in conjunction with a **minimum** value, to determine the slider's incremental scale.

Current. This field specifies the initial value of the slider, or the initial position of the slider thumb. The value entered must be within the range set by the minimum and maximum values. For example, if the slider has a minimum value of 2, a maximum value of 10, and a current value of 4, the slider thumb will initially be positioned two increments to the right of the leftmost position.

Name. Enter in this field a name that will distinguish the horizontal slider from other objects on the window.

Help. This field designates the help context to be associated with the horizontal slider. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the slider and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that determine the type of slider are listed in the field on the right half of the window. These options are:

- *Vertical slider/scale.* Defines the object to be a vertical slider.
- *Horizontal slider/scale.* Defines the object to be a horizontal slider.
- *Vertical scroll bar.* Defines the object to be a vertical scroll bar.
- *Horizontal scroll bar.* Defines the object to be a horizontal scroll bar.
- *Corner scroll box.* Defines the object to be a corner scroll box.

Position page The **Position** page contains information related to the horizontal slider's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page The **Geometry** notebook page allows you to place constraints on a horizontal slider that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page The **Advanced** page contains information related to the advanced properties of the horizontal slider. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Vertical slider

A vertical slider is typically used to visually indicate the current value relative to a range of possible values and to allow the setting of a value. For example, a slider could be used as a volume control for an application. It is different from a vertical scroll bar, which is designed to scroll another object, such as a vertical list. A slider is typically added directly to the current resource window, independent of any other object. In most environments a slider and a scroll bar look different, in that the slider track is narrower than that of a scroll bar. Also, a vertical slider's thumb button moves *up* as the slider's value increases, whereas a scroll bar's thumb button moves *down* as the scroll bar's value increases. Selecting **Vertical Slider** causes the following object to appear:

To modify the vertical slider, call its information notebook. The following window will appear:
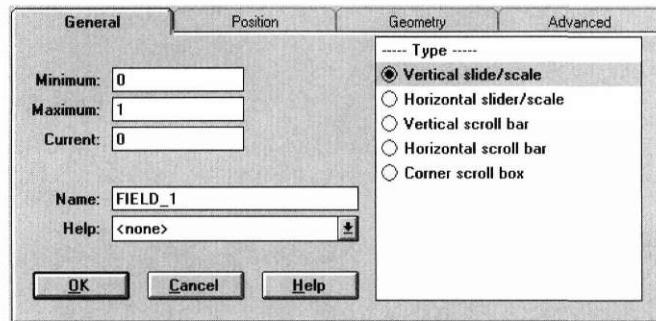
NOTE: The information notebook for the vertical slider is actually an information notebook for a generic scroll bar object but with the **Vertical slider/scale** option set. If this option is toggled off, or if another type is set, the object will no longer be a vertical slider.

General page

The **General** page contains information related specifically to the vertical slider object being edited. It contains the following fields:

Minimum. This field specifies the minimum value of the slider range. The actual number entered will not have meaning to the Designer, except that it is used, in conjunction with a **maximum** value, to determine the slider's incremental scale. For example, if the slider has a minimum value of 2 and a maximum value of 10, the slider thumb will be able to scroll eight increments from the leftmost to the rightmost position.

Maximum. This field specifies the maximum value of the slider range. The actual number entered will not have meaning to the Designer, except that it is used, in conjunction with a **minimum** value, to determine the slider's incremental scale.

Current. This field specifies the initial value of the slider, or the initial position of the slider thumb. The value entered must be within the range set by the minimum and maximum values. For example, if the slider has a minimum value of 2, a maximum value of 10, and a current value of 4, the slider thumb will initially be positioned two increments above the bottommost position.

Name. Enter in this field a name that will distinguish the vertical slider from other objects on the window.

Help. This field designates the help context to be associated with the vertical slider. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the slider and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that determine the type of slider are listed in the field on the right half of the window. These options are:

- *Vertical slider/scale.* Defines the object to be a vertical slider.
- *Horizontal slider/scale.* Defines the object to be a horizontal slider.
- *Vertical scroll bar.* Defines the object to be a vertical scroll bar.
- *Horizontal scroll bar.* Defines the object to be a horizontal scroll bar.
- *Corner scroll box.* Defines the object to be a corner scroll box.

Position page The **Position** page contains information related to the vertical slider's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a vertical slider that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information related to the advanced properties of the vertical slider. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Combo box

A combo box is a combination of a string field and a scrollable list box. It is used to display a list of selectable items without requiring a lot of screen space. Selecting **Combo Box** causes the following object to appear:
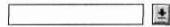
The scrollable list is displayed when the button to the right of the string field is selected. If the end user types in a string of characters, the item on the list that most closely matches the character string will be highlighted. When one of the items of the list is selected, it is copied into the string field and the list box disappears.

Objects are added to the combo box's list by selecting them from the menu or button bar and placing them on the combo box object. By default, they will be automatically aligned in a single column in the order in which they were created.

To modify the combo box object, call its information notebook. The following window will appear:
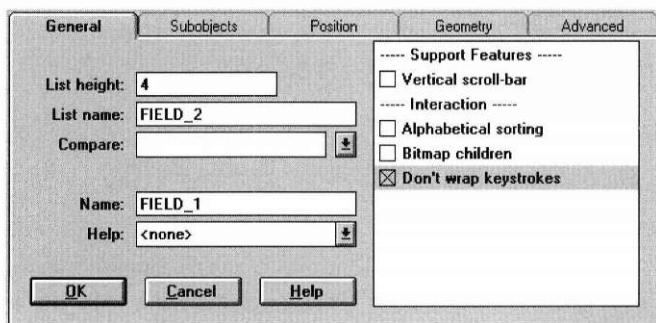
General page

The General page contains information related specifically to the combo box object being edited. It contains the following fields:

List Height. This field determines the height of the combo box's drop-down list. By default it is measured in cell coordinates, unless pixel or minicell coordinate scales have been selected on the Position page.

List Name. Enter in this field a name for the drop-down list portion of the combo box.

Name. Enter in this field a name that will distinguish the combo box from other objects on the window.

Help. This field designates the help context to be associated with the combo box. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the combo box and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the combo box are listed in the field on the right half of the window. The first section presents the following option:

• *Vertical scroll-bar.* Adds a vertical scroll bar to the drop-down list of the combo box.

The second section presents options that alter interaction with the combo box's drop-down list. The options are:

- *Alphabetic sort.* Causes the list options to be sorted and displayed in alphabetical order.

- *Bitmap children.* Allows objects that contain bitmaps to be added to the combo box.

- *Don't wrap keystrokes.* Prevents arrowing up or down to wrap from the end of the list to the beginning or vice versa.

Subobjects page

The objects contained within the combo box's drop-down list can be modified through interaction with the Subobjects page. For detailed information on the Subobjects page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a combo box is a button.

Position page

The **Position** page contains information related to the combo box's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page

The **Geometry** notebook page allows you to place constraints on a combo box that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page

The **Advanced** page contains information related to the advanced properties of the combo box. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Spin control

A spin control object is used to allow the user to select a value from a finite range of values. The spin control can be used to set integer, real and bignum values, as well as times and dates. The user can type a value into the field, or he can spin through the values by clicking on the spinner's up or down arrows until the desired value is displayed. Selecting **Spin Control** causes the following object to appear:
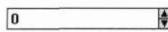
To modify the spin control object, call its information notebook. The following window appears:
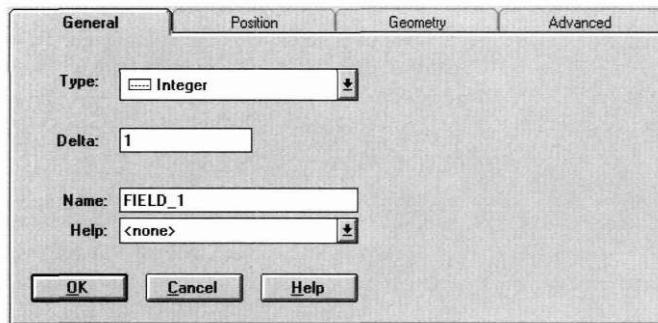
General page

The General page contains information related specifically to the spin control object being edited. It contains the following fields:

Type. This field determines the type of object to be associated with the spin control. Select the combo box button to view a list of the available object types. Selecting one of these causes the spin control to appear as that object but with the associated spinner arrow buttons.

Edit... Selecting this button brings up the information notebook for the type of object associated with the spin control, since that object is what determines the spin control's specifications. For example, if time is the object type, selecting **Edit...** causes the time object information notebook to appear. Any changes made in this information notebook will be reflected in the spin control's time field.

Delta. This field assigns the value by which the object value will be adjusted when the object is spun. It should be specified in units appropriate to the object type. For example, if the object type is a time object, the delta value must be given in hundredths of seconds.

Name. Enter in this field a name that will distinguish the spin control from other objects on the window.

Help. This field designates the help context to be associated with the spin control. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the spin control and requests help. (See Chapter 20 for information on creating help contexts.)

Pull-down item

A pull-down item serves as the first level of selection in a pull-down menu. The figure below shows a pull-down menu with one pull-down item attached to it:
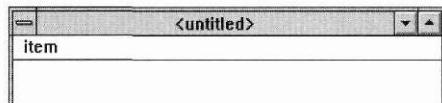
The multi-level effect of a pull-down menu is achieved by adding pop-up items to the pull-down item.

To modify the pull-down item, the information notebook must be called. This can only be done by selecting the item from the **Subobjects** page of the pull-down menu's information notebook. Upon doing so, the following window appears:
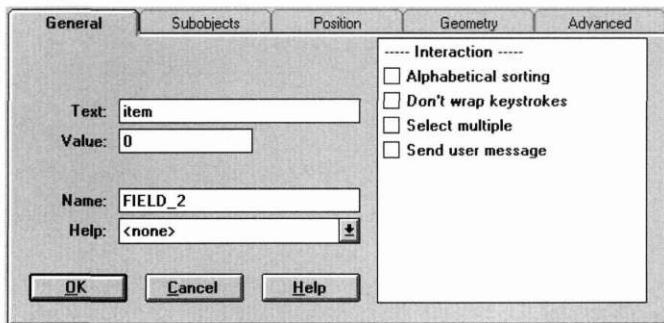
General page

The **General** page contains information related specifically to the pull-down item being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear on the pull-down item. It will be centered vertically automatically.

Value. This field allows you to enter in a value that serves as a unique identification for a pull-down item. This allows you to define one callback function that looks at the pull-down item values, instead of several functions that are tied to each pull-down item object. If the *Send user message* option is set, the value must be an event type. And no callback function should be assigned in the Advanced page.

Name. Enter in this field a name that will distinguish the pull-down item from other objects on the window.

Help. This field designates the help context to be associated with the pull-down item. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the pull-down item and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control interaction with the pull-down item are listed in the field on the right half of the window. The options are:

- *Alphabetic sort.* Causes the menu items to be sorted in alphabetical order.

- *Don't wrap keystrokes.* Prevents the current item in the pop-up menu from wrapping between the top and bottom items when arrowing through the list.

- *Select multiple.* Allows more than one item in the pop-up menu to become selected at the same time. If this flag is set, the pop-up menu will still close when a selection is made, but selecting another option later will not cause the previously selected item to be un-selected.

- *Send user message.* Causes the event associated with the pull-down item's value to be created and put on the event queue when the pull-down item is selected. Any temporary windows are removed from the display when this message is sent. No callback function should be assigned in the **Advanced** page.

Subobjects page   The objects contained within the pull-down item's pop-up menu can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a pull-down item is a pop-up item.

Position page   The **Position** page contains information related to the pull-down item's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a pull-down item that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information related to the advanced properties of the pull-down item. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Pop-up item

A pop-up item is used to display and select options associated with a list of menu items. It can be attached to a pull-down item (as the second level of selection within a pull-down menu), or to another pop-up item. A pop-up item can only be added by selecting it from within its parent information notebook's add-item field (located on the **Subobjects** notebook page).

The multi-level effect of a pop-up menu or a pull-down menu is further achieved by adding sub-pop-up items to the parent pop-up item. OpenZinc Designer will allow you to continue adding additional levels as long as there is available memory for them.

To modify the pop-up item, the information notebook must be called. This can only be done by selecting the item from the Subobjects page of either the parent pop-up item's information notebook or the pull-down item's information notebook. Upon doing so, the following window appears:
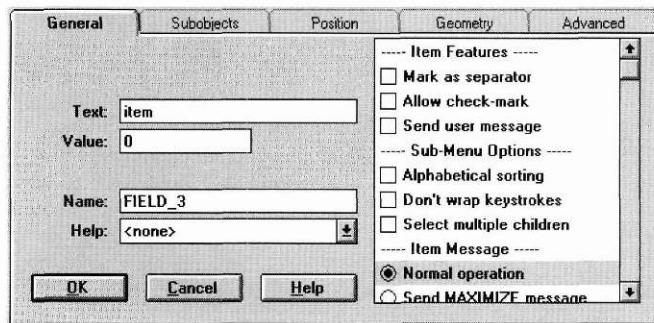
General page

The General page contains information related specifically to the pop-up item being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear on the pop-up item. It will be centered vertically automatically.

Value. This field allows you to enter in a value that serves as a unique identification for a pop-up item. This allows you to define one callback function that looks at the pop-up item values, instead of several functions that are tied to each pop-up item object. If the *Send user message* option is set, the value must be an event type.

Name. Enter in this field a name that will distinguish the pop-up item from other objects on the window.

Help. This field designates the help context to be associated with the pop-up item. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the pop-up item and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that determine the item's features are listed in the field on the right half of the window. The options are:

- *Mark as separator.* Causes the pop-up item to be a separator that will appear as a horizontal line when placed in a pop-up menu.

- *Allow check-mark.* Causes the pop-up item to display a check mark at the front of the text when the pop-up item is selected.

- *Send user message.* Causes the event associated with the pop-up item's value to be created and put on the event queue when the pop-up item is selected. Any temporary windows are removed from the display when this message is sent. And no callback function should be assigned in the **Advanced** page.

The second section presents options that apply to any pop-up menu lists associated with the pop-up item. The options are:

- *Alphabetic sort.* Causes the menu items to be sorted in alphabetical order.

- *Don 't wrap keystrokes.* Prevents the current item in the pop-up menu from wrapping between the top and bottom items when arrowing through the list.

- *Select multiple.* Allows more than one item in the pop-up menu to become selected at the same time. If this flag is set, the pop-up menu will still close when a selection is made, but selecting another option later will not cause the previously selected item to be un-selected.

The third section presents options for associating messages with the item. The options are:

- *Normal operation.* Does not cause any message to be sent.
- *Send MAXIMIZE message.* Causes the window to be maximized when the pop-up item is selected.
- *Send MINIMIZE message.* Causes the window to be minimized when the pop-up item is selected.
- *Send MOVE message.* Causes the window to enter a mode that allows it to be moved.
- *Send SIZE message.* Causes the window to enter a mode that allows it to be sized.
- *Send SWITCH message.* Causes the operating system's task list or window list to display. This option only works in Windows and OS/2.
- *Send RESTORE message.* Causes the window to be restored to its original size if it is in a maximized or minimized state.
- *Send CLOSE message.* Causes the window to be closed when the pop-up item is selected.

Subobjects page
The objects contained within the pop-up item's pop-up menu can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a pop-up item is a pop-up item.

Position page
The **Position** page contains information related to the pop-up item's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page
The **Geometry** notebook page allows you to place constraints on a pop-up item that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page
The **Advanced** page contains information related to the advanced properties of the pop-up item. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Selection Objects

The selection category includes objects that are used to display select-able objects. Selecting the **Selection** option causes the following associated menu to appear:
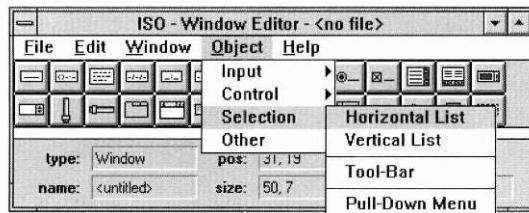
Horizontal list

A horizontal list is used to display related information in a multiple-column fashion within a window. The list is only scrollable horizontally. Selecting **Horizontal List** causes the following object to appear:
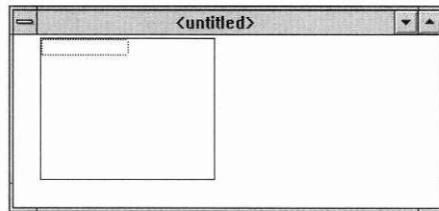
Notice that the list is initially empty. A horizontal list is actually a framework to which other objects can be attached. For example, a list of strings could be added to a horizontal list by repeatedly selecting the string object and placing it within the list. The items will be aligned automatically in rows and columns.

To modify the horizontal list object, call its information notebook. The following window will appear:
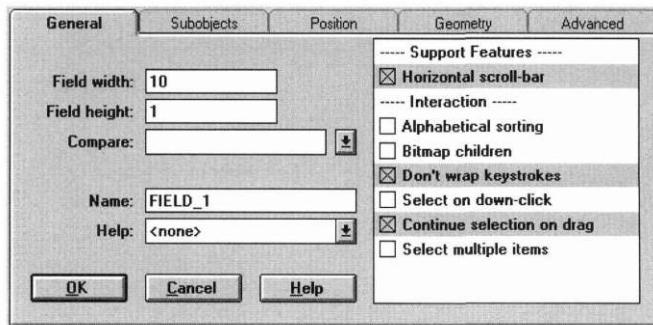
General page   The General page contains information relating specifically to the horizontal list being edited. It contains the following fields:

Field width. Enter in this field a number to specify the maximum cell width of a single list item. If the list is wider than the specified width, it will be displayed with multiple columns. The default width is 10.

Field height. Enter in this field a number to specify the maximum cell height of a single list item. If the list is taller than the specified height, it will be displayed with multiple rows. The default height is 1.

Compare. If you want to have a compare function associated with the list, you can enter the name of it in this field. The function must be defined somewhere in your code with the same name that is entered so that OpenZinc Designer can find it and execute the designated action. (For more information on creating compare functions, refer to the description of the object's constructor in the *Programmer's Reference.)*

Name. Enter in this field a name that will distinguish the horizontal list object from other objects on the window.

Help. This field designates the help context to be associated with the horizontal list. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the horizontal list and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the horizontal list are listed in the field on the right half of the window. The first section presents the following support feature option:

- *Horizontal scroll-bar.* Causes a horizontal scroll bar to be added to the list.

The second section presents options that determine interaction with the list. The options are:

- *Automatic sorting.* Causes the options to be sorted in alphabetical order.
- *Bitmap children.* Indicates that some of the objects contain bitmaps. Setting this flag will affect the spacing of objects in the list. Normally, objects are spaced according to a pre-determined cell height value. If this

flag is set, however, the objects will be spaced according to the actual height of the objects. This flag should be set when adding check boxes or radio buttons to the horizontal list.

Don't wrap keystrokes. Will not allow arrowing up, down, left or right to wrap from the end of the list to the beginning or vice versa.

Select on down-click. Selects the item on a button down-click, rather than on a down-click and release action.

Continue selection on drag. Allows the end-user to drag through the list options with the mouse button pressed. If this flag is not set, the highlight on the list items will not follow the dragging mouse.

Select multiple items. Allows more than one object to be selected at a time.

Subobjects page   The objects contained within the horizontal list can be modified through interaction with the Subobjects page. For detailed information on the Subobjects page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a horizontal list is a button.

Position page   The **Position** page contains information related to the horizontal list's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a horizontal list that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the horizontal list. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Vertical list

A vertical list is used to display items in a single-column fashion. The list is only scrollable vertically. Selecting **Vertical List** causes the following object to appear:
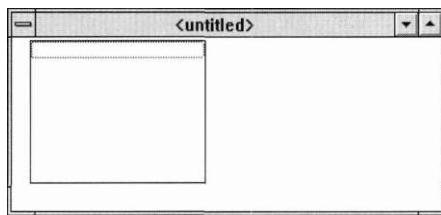
Notice that the list is initially empty. A vertical list is actually a framework to which other objects can be attached. For example, a list of strings could be added to a vertical list by repeatedly selecting the string object and placing the string within the list. These objects will be aligned in a single-column fashion automatically.

To modify the vertical list object, call its information notebook. The following window will appear:
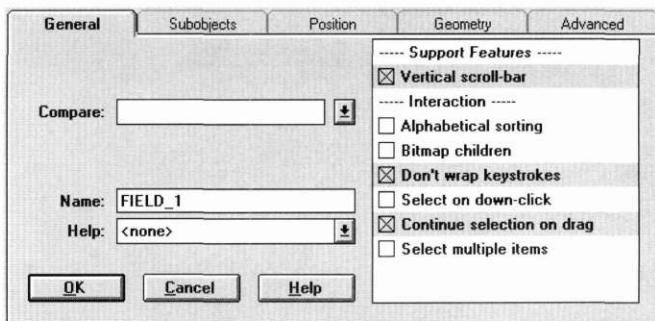
General page   The General page contains information relating specifically to the vertical list being edited. It contains the following fields:

Compare. To have a compare function associated with the list, enter the name of it in this field. The function must be defined somewhere in your code with the same name that is entered so that OpenZinc Designer can find it and

execute the designated action. (For more information on creating compare functions, refer to the description of the object's constructor in the *Programmer's   Reference.)*

Name. Enter in this field a name that will distinguish the vertical list object from other objects on the window.

Help. This field designates the help context to be associated with the vertical list. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the vertical list and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the vertical list are listed in the field on the right half of the window. The first section presents the following support feature option:

- *Vertical scroll-bar.* Causes a vertical scroll bar to be added to the list.

The second section presents options that determine interaction with the list. The options are:

- *Alphabetic sort.* Causes the options to be sorted in alphabetical order.
- *Bitmap children.* Indicates that some of the objects contain bitmaps. Setting this flag will affect the spacing of objects in the list. Normally, objects are spaced according to a pre-determined cell height value. If this flag is set, however, the objects will be spaced according to the actual height of the objects. This flag should be set when adding check boxes or radio buttons to the vertical list.
- *Don't wrap keystrokes.* Will not allow arrowing up, down, left or right to wrap from the end of the list to the beginning or vice versa.
- *Select on down-click.* Selects the item on a button down-click, rather than on a down-click and release action.
- *Continue selection on drag.* Allows the end-user to drag through the list options with the mouse button pressed. If this flag is not set, the highlight on the list items will not follow the dragging mouse.
- *Select multiple items.* Allows more than one object to be selected at a time.

Subobjects page   The objects contained within the vertical list can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a vertical list is a button.

Position page
The **Position** page contains information related to the vertical list's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page
The **Geometry** notebook page allows you to place constraints on a vertical list that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page
The **Advanced** page contains information relating to the advanced properties of the vertical list. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Tool bar

A tool bar is used as a controlling structure for a set of selectable window objects. It differs from the pull-down menu in that a variety of objects can be added to it—not just textual items. The tool bar will automatically occupy the upper-most area available in a window, positioning itself directly below the pull-down menu, if one exists. Multiple tool bars may be added to a window. Selecting **Tool Bar** and attaching it to a window causes the following object to appear:
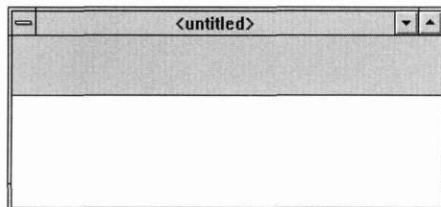
An object can be added to the tool bar by selecting the desired object from the control window's menu or button bar and placing it on the resource window's tool bar. The control window's button bar itself is an example of a group of bitmapped buttons that have been attached to a tool bar.

To modify the tool bar, call its information notebook. The following window will appear:
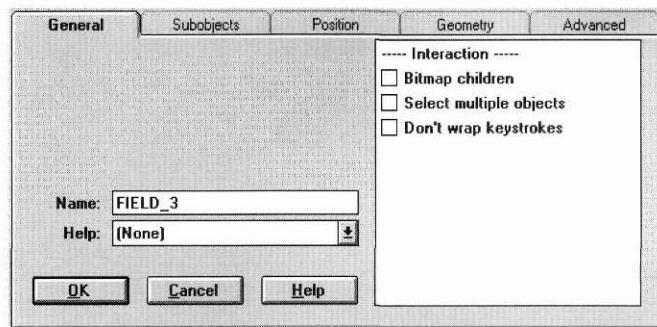
General page   The General page contains information relating specifically to the tool bar object being edited. It contains the following fields:

Name. Enter in this field a name that will distinguish the tool bar from other objects on the window.

Help. This field designates the help context to be associated with the tool bar. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the tool bar and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control interaction with the tool bar are listed in the field on the right half of the window. The options are:

- *Bitmap children.* Indicates that some of the **objects contain bitmaps. Set-**ting this flag will affect the spacing of objects in **the list. Normally,** objects are spaced according to a pre-determined cell **height value. If this** flag is set, however, the objects will be spaced according **to the actual** height of the objects.

- *Select multiple objects.* Allows more than one object to be selected at a time. This option must be set if check boxes are added to the tool bar.

- *Don't wrap keystrokes.* Causes objects placed in the tool bar to be positioned according to their specified coordinates. By default, objects within a tool bar are automatically positioned so that they are edge-to-edge from left-to-right, in the order in which they were created. If more objects are added than can fit on a single line of the tool bar, the tool bar will wrap and place the remaining objects on the next line. If this option is set, however, objects on the tool bar will not be automatically positioned, but will be positioned to the location at which they were placed.

Subobjects page The objects contained within the tool bar can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page page 220.

The default add object (shown in the lower right combo box field) for a **tool** bar is a button.

Position page The **Position** page contains information related to the tool bar's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page The **Geometry** notebook page allows you to place constraints on a tool bar that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page The **Advanced** page contains information relating to the advanced properties of the tool bar. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Pull-down menu

A pull-down menu acts as a structure for selectable menu items that appear in a single horizontal line. It automatically occupies the length of the top portion of the window to which it is attached. Selecting the **Pull-Down Menu** option and attaching it to a window causes the following object to appear:
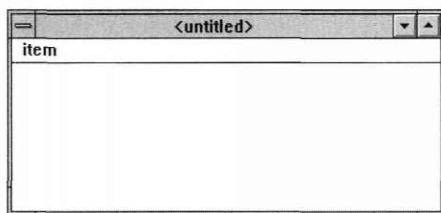
A multi-level selectable menu is created by adding pull-down items and pop-up items to the pull-down menu. The pull-down menu object is created with one pull-down item automatically attached to it.

To modify the pull-down menu, call its information notebook from its parent window's **Subobjects** page. The following window will appear:
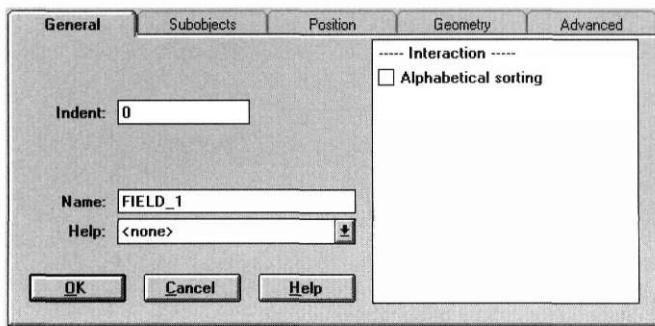
General page   The General page contains information relating specifically to the pull-down menu being edited. It contains the following fields:

Indent. This field determines the blank space, measured in cell units, between the first menu item and the left edge of the menu. (This applies only to the first item. Subsequent menu items will be automatically positioned next to the previous menu item.)

Name. Enter in this field a name that will distinguish the pull-down menu object from other objects on the window.

Help. This field designates the help context to be associated with the pull-down menu. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the pull-down menu and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control interaction with the pull-down menu are listed in the field on the right half of the window. The following option is available:

- *Alphabetic sort.* Causes the menu items to be sorted and displayed in alphabetical order.

Subobjects page   The objects contained within the pull-down menu can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default add object (shown in the lower right combo box field) for a pull-down menu is a pull-down item.

Position page   The **Position** page contains information related to the pull-down menu's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** notebook page allows you to place constraints on a pull-down menu that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the pull-down menu. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Other Objects

The final object category includes window objects that do not fit into the other general object categories. Selecting the **Other** option causes the following associated menu to appear:
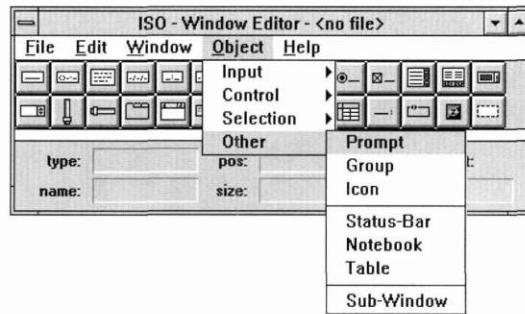
Prompt

A prompt object is used to provide lead information for another window object. Selecting **Prompt** causes the following object to appear:

prompt:

To modify the prompt object, call its information notebook. The following window will appear:
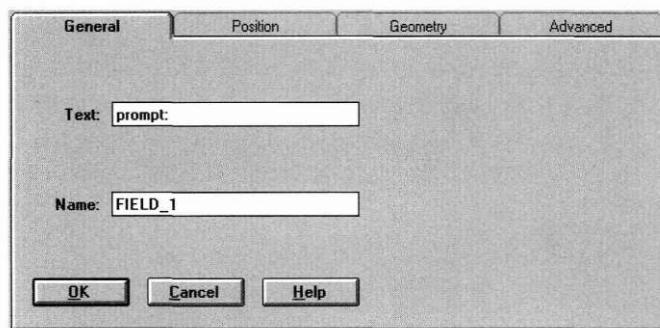
General page

The **General** page contains information relating specifically to the prompt object being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear in the prompt. It will be automatically centered vertically. If the text string is longer than the length of the prompt field, the field must be sized in order to display the entire text.

*Name.* Enter in this field a name that will distinguish the prompt object from other objects on the window.

Position page

The **Position** page contains information related to the prompts's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page

The **Geometry** page allows you to place constraints on a prompt object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced proper-
ties of the prompt. It is designed for use by the experienced user only. For
detailed information on the **Advanced** page, refer to the general description
on page 218.

Group

The group object is used to allow physical grouping of window objects. For
example, a series of radio buttons can be grouped together by first creating a
group object and then adding the radio buttons. Selecting **Group** causes the
following object to appear:
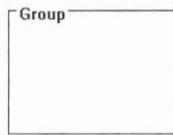
To modify the group object, call its information notebook. The following
window will appear:
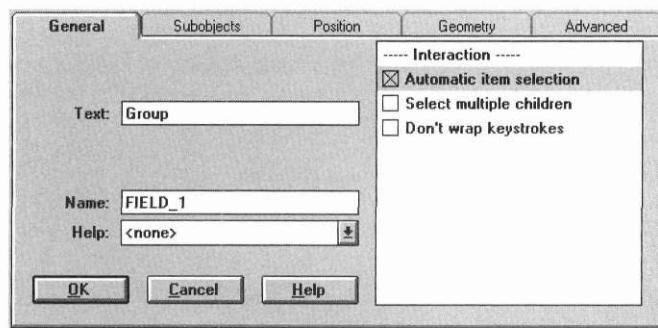
General page   The General page contains information relating specifically to the group
object being edited. It contains the following fields:

Text. Enter in this field text exactly as you want it to appear in the upper left corner of the group object's border. If the text string is longer than the width of the group box, only the portion that fits will be displayed.

Name. Enter in this field a name that will distinguish the group object from other objects on the window.

Help. This field designates the help context to be associated with the group box. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the group box and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control interaction with the group object are listed in the field on the right half of the window. These options are:

- *Automatic item selection.* Causes each object in the group to be automatically selected when it becomes current. Typically this option is used when the group contains radio buttons. If the user arrows through the radio buttons, the current button will always be the selected button.

- *Select multiple children.* Allows more than one item in the group to become selected at the same time. This option is typically used if the group contains check boxes.

- *Don't wrap keystrokes.* Prevents arrowing up or down to wrap from the end of the group's items to the beginning or vice versa.

Subobjects page   The objects contained within the group object can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page page 220.

The default **add** object (shown in the lower right combo box field) for a group box is a button.

Position page   The **Position** page contains information related to the group's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** page allows you to place constraints on a group object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

The **Advanced** page contains information relating to the advanced properties of the group box. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Icon

An icon is used to display a 32x32 pixel image on the screen. It is often present in an application as an indicator of some sort that cannot be interacted with; however, an icon can also be created for interaction purposes, such as a question mark icon that displays help when selected. Selecting **Icon** causes the following object to appear:

To modify the icon, call its information notebook. The following window will appear:
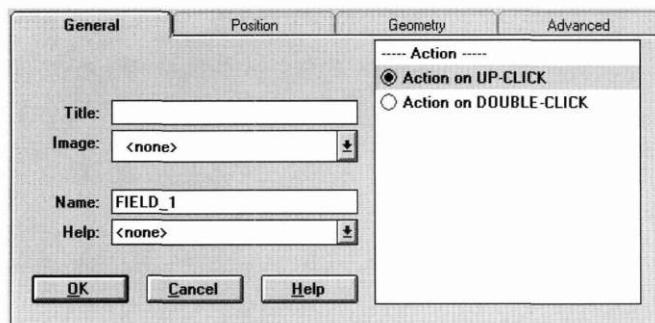
General page   The **General** page contains information relating specifically to the icon object being edited. It contains the following fields:

Title. If you want a title to appear in the rectangular region below the icon, enter in this field the exact text for the title.

Image. This field designates the image to be associated with the icon. Select the combo box button to view a list of the available images. If you select one of the images listed, it will be displayed on the icon. (See Chapter 19 for information on creating icon images.)

Name. Enter in this field a name that will distinguish the icon object from other objects on the window.

Help. This field designates the help context to be associated with the icon. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the icon and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that determine how to activate an action associated with the icon are listed in the field on the right half of the window. These options are:

- *Action on UP-CLICK.* Completes the action on a down-click and release action.
- *Action on DOUBLE-CLICK.* Completes the action when the icon has been selected twice in rapid succession.

Position page   The **Position** page contains information related to the icons's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** page allows you to place constraints on an icon object that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the icon. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Status   bar

A status bar presents status information to the user. It is not interactive, but is a means of displaying information. A status bar is like a modified window that always occupies the bottom width of the parent window. Selecting **Status Bar** and attaching it to a window causes the following object to appear:

The following objects can be added to a status bar: string, date, time, and number objects; icons; and buttons.

To modify the status bar, call its information notebook. The following window will appear:
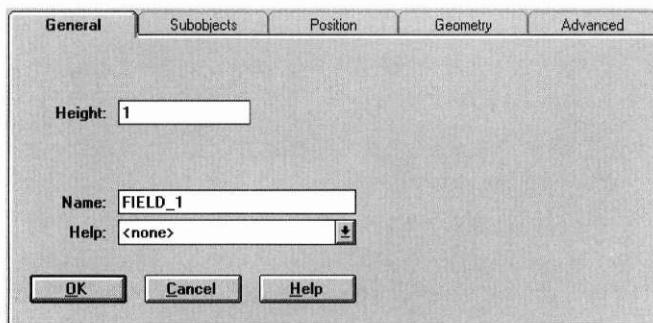
General page

The **General** page contains information relating specifically to the status bar being edited. It contains the following fields:

Height. This field designates the height of the status bar within the window.

Name. Enter in this field a name that will distinguish the status bar from other objects.

Help. This field designates the help context to be associated with the status bar. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the status bar and requests help. (See Chapter 20 for information on creating help contexts.)

Subobjects page   The objects contained within the status bar can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page page 220.

The default **add** object (shown in the lower right combo box field) for a status bar is a string.

Position page   The **Position** page contains information related to the status bar's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** page allows you to place constraints on a status bar that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the status bar. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Notebook

A notebook is used to display multiple pages of related objects. Each of these pages has a tab at the top of it which, when selected by the mouse, causes the complete page to be displayed. Selecting **Notebook** and attaching it to a window causes the following object to appear:
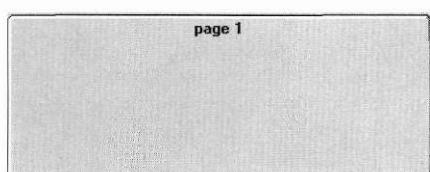
The notebook object itself is actually an invisible framework for notebook pages, which become the notebook's subobjects. When you create a notebook object in the designer, it is automatically created with one page. Since the pages within the notebook are actually modified windows, adding more pages is accomplished by selecting the *window* object and placing it within the notebook structure. All of the information notebooks for window objects in the designer are examples of the notebook object.

To modify the notebook, call its information notebook. The following window will appear:
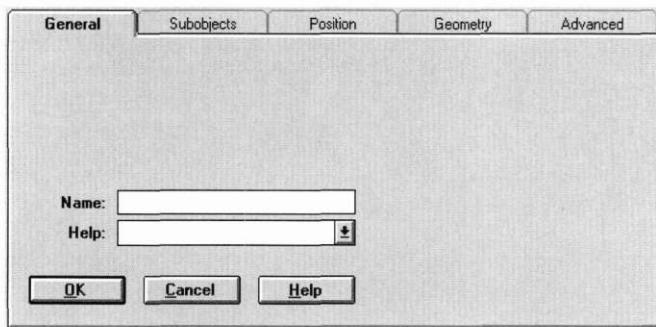
General page

The **General** page contains information relating specifically to the notebook object being edited. It contains the following fields:

Name. Enter in this field a name that will distinguish the notebook object from other objects on the window.

Help. This field designates the help context to be associated with the notebook. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the notebook and requests help. (See Chapter 20 for information on creating help contexts.)

Subobjects page

The objects, including the actual window pages, contained within the notebook structure can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default **add** object (shown in the lower right combo box field) for a notebook is a window.

Position page   The **Position** page contains information related to the notebook's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** page allows you to place constraints on a notebook that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the notebook. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Table

A table object is used to create a table of records, which can be scrolled both vertically and horizontally. Records in the table can have one or more fields which may be of different types. One use of the table object would be a spreadsheet. Selecting **Table** causes the following object to appear:
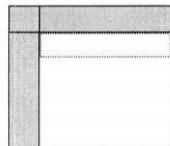
A table object differs in one important way from the other objects of the Designer, in that it is merely a template for a table. The data for the fields in the table cannot be initialized from within the Designer; it must be set at run-time. For example, though we may designate one cell of the table as a string field, we cannot specify the string's initial data from within the Designer. This is different from a horizontal list with a string field, which displays information that we assign to it.

The table object requires four layers to accomplish its purpose. These layers are described below.

1. The base layer of the table object is the table framework itself, which hosts the other components and allows them to function together.

2. The second layer includes the horizontal, vertical, and corner scroll bars, and the column, row, and corner headers, represented by the gray regions on the left side and top of the table.

3. The third layer consists of table records, which are the actual regions to which specific fields are attached, such as string, date, or number fields. Table records are used in the headers to display information—typically labels that identify the contents of the row or column—and in the table directly, to display or collect data. By default, the table object is created with four table records—one each in the column and row headers (where the 'XXX' prompts are displayed), one in the corner header, and one within the actual table (shown with a dotted border surrounding it). Only one table record should be added to the table directly. The table will use a virtual record to draw all the other displayed records, using the data passed to the table at run-time.

4. Finally, the fourth layer of the table object is its object data fields, which are added directly to the table records. As mentioned, the column and row header records each appear by default with one prompt field. The text of these prompt fields can be modified by traversing through the associated headers' information notebooks. The record of the actual table (shown with a dotted border surrounding it) will accept any input or control types of objects. Simply select the desired object and place it within the table border. These fields become subobjects of the table record.

In summary, the four levels of the table object can be visually represented in the following figure:
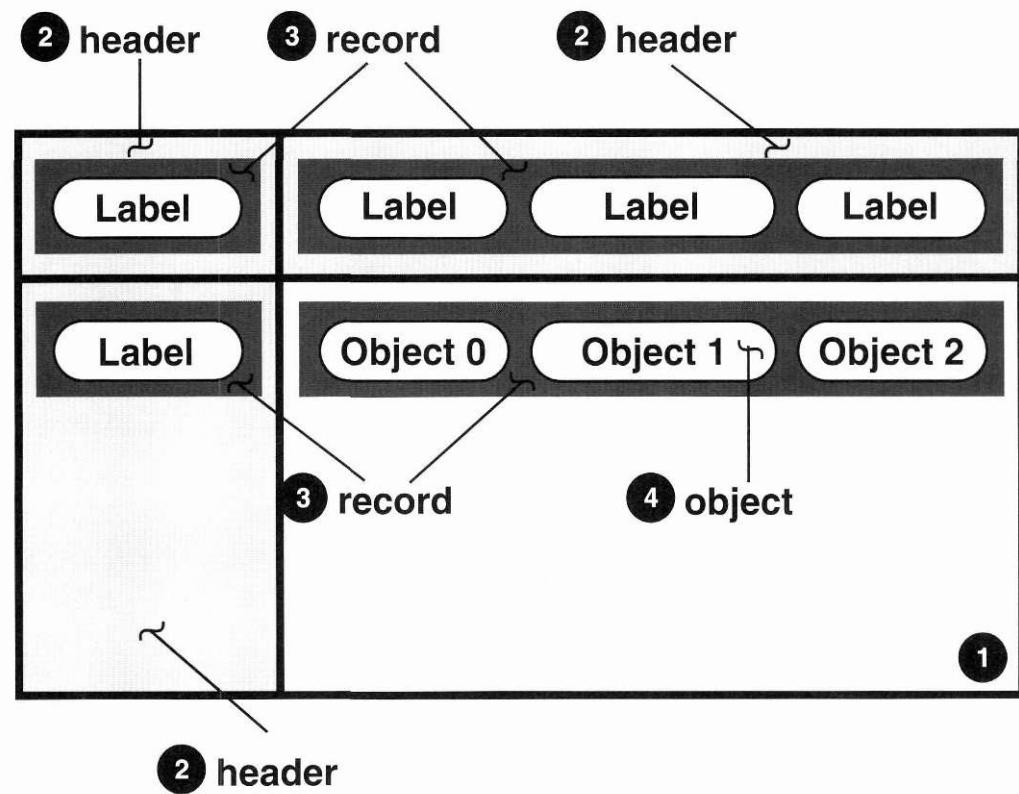
To modify the table object, call its information notebook. The following window will appear:
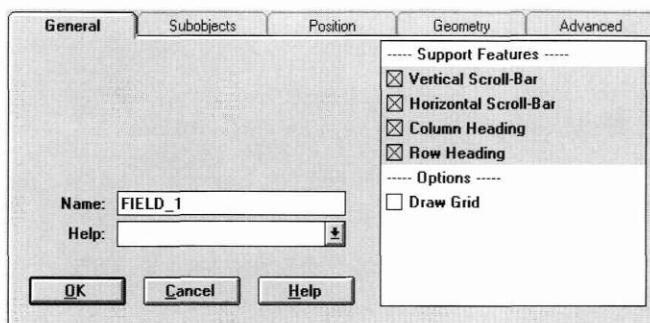
General page

The **General** page contains information relating specifically to the table object being edited. It contains the following fields:

Name. Enter in this field a name that will distinguish the table object from other objects on the window.

Help. This field designates the help context to be associated with the table. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the table and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that determine the presentation of the table are listed in the field on the right half of the window. The first section presents options that determine the support features of the table. These options are:

• *Vertical scroll bar.* Places a vertical scroll bar inside the right border of the table.

• *Horizontal scroll bar.* Places a horizontal scroll bar inside the bottom border of the table.

   NOTE: A corner scroll bar is automatically added when both a vertical and a horizontal scroll bar are selected.

• *Column heading.* Places a column header, with one prompt field attached to it, at the top of the table.

• *Row heading.* Places a row header, with one prompt field attached to it, at the left side of the table.

> **NOTE:** A corner header is automatically added when both a vertical and a horizontal header are selected.

The second section presents the following option:

- *Draw grid.* Displays lines that divide the table's columns and rows.

Subobjects page

The objects, including the header, scroll bars, and records, contained within the table structure can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default **add** object, shown in the lower right combo box field, for a table is a string.

Position page

The **Position** page contains information related to the table's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page

The **Geometry** page allows you to place constraints on a table that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page

The **Advanced** page contains information relating to the advanced properties of the table. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Subwindow

A window is used as a controlling structure for displaying and interacting with other objects. This object is known as a subwindow or child window in order to distinguish it from the main resource window to which it must be attached. Selecting **Subwindow** causes the following object to appear:
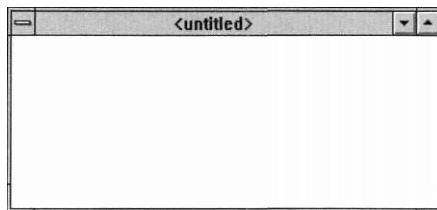
By default, the window is created with a title, a system button, a maximize button, and a minimize button. Other objects can be added by simply selecting them from the menu or button bar and placing them on the window.

To modify the child window object, call its information notebook. The following window will appear:
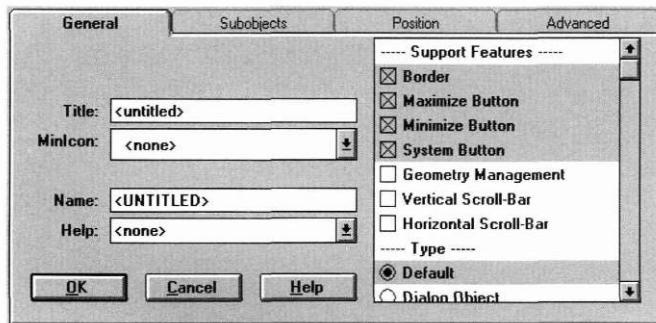
General page   The **General** page contains information relating specifically to the child window object being edited. It contains the following fields:

Title. Enter in this field text exactly as you want it to appear in the window's title. It will be automatically centered vertically.

MinIcon, This field designates the icon to be associated with the window when it is minimized. Select the combo box button to view a list of the available icons. If you select one of the icons listed, it will be used to represent the window when the window is in a minimized state. (See Chapter 19 for information on creating icon images.)

Name. Enter in this field a name that will distinguish the subwindow from other objects on the window.

Help. This field designates the help context to be associated with the child window. Select the combo box button to view a list of the available help contexts. If you select one of the contexts listed, the help message of that context will be displayed whenever the user positions on the child window and requests help. (See Chapter 20 for information on creating help contexts.)

options list. The options that control the presentation and operation of the subwindow are listed in the field on the right half of the window. The first section presents options that determine the support features of the subwindow. These options are:

- *Border.* Draws a thin border around the outer perimeter of the window.
- *Maximize button.* Attaches a maximize button to the window that will enlarge the window to its maximum size when selected.
- *Minimize button.* Attaches a minimize button to the window that will reduce the window to its minimum size when selected.
- *System button.* Attaches a system button to the window. When selected, a system button displays the following selectable options: **Restore, Move, Size, Minimize, Maximize,** and **Close.**
- *Geometry management.* Enables the geometry management feature for the subwindow, meaning that objects which are attached to the window can then be positioned and sized according to their own geometry management specifications. If this option has not been previously set and one of the child objects is using geometry management, it will automatically be turned on. (For more information on geometry management, see Chapter 11 and Chapter 13.)
- *Vertical scroll bar.* Places a vertical scroll bar inside the right border of the child window.
- *Horizontal scroll bar.* Places a horizontal scroll bar inside the bottom border of the child window.

The second section presents options that determine the type of subwindow. The options are:

- *Default.* Creates a normal window.
- *Dialog object.* Creates the window as a dialog box. A dialog box is a temporary window used to display or receive information from the user. Using this flag will cause a dialog style border to be displayed.
- *MDI object.* Creates the window as an MDI window. An MDI parent must have a pull-down menu. In general, other than the standard support objects (i.e., system button, border, title, etc.) and the pull-down menu, MDI parent windows should only contain MDI children.

  If the MDI window is added directly to another MDI window, it will become an MDI child object. MDI child windows can be moved or sized but will remain entirely within the MDI parent window.

The third section presents options that determine interaction with the sub-window. The options are:

- *Alphabetical sort.* Causes the objects to be sorted and displayed in alphabetical order on the window.

- *Select multiple objects.* Allows more than one object to be selected at a time.

- *Select on drag operation.* Causes any objects overlapped by a drag operation to be selected.

- *Allow normal hotkeys.* Allows the user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required. Care should be taken when using this option in a window, as editable objects will no longer work properly.

- *Don't size.* Prevents the user from changing the size of the window at run-time. This option should be set if the window is to be a non-MDI child.

- *Don't move.* Prevents the user from changing the screen location of the window at run-time. This flag should be set if the window is to be a non-MDI child.

- *Modal.* Prevents any other window from receiving event information. A modal window receives all event information until it is removed from the screen.

- *Locked.* Prevents the user from removing the window from the screen.

- *Temporary.* Causes the window to only occupy the screen temporarily. Once another window is selected, the temporary window is removed. Once removed, a temporary window will be destroyed if the *Don't destroy object* option is not set.

- *Don't destroy object.* Prevents the window from being destroyed when it is closed. If this option is set, the window can be removed from the display, but the programmer is responsible for destroying the window.

The fourth section presents the following special option:

- *Renumber subobjects.* Causes the objects within the window to be renumbered sequentially. This is especially useful after several delete, cut, or similar actions that result in gaps in the original numbering of the objects.

Subobjects page   The objects contained within the child window can be modified through interaction with the **Subobjects** page. For detailed information on the **Subobjects** page, refer to the general description on page 220.

The default **add** object (shown in the lower right combo box field) for a child window is a string.

Position page   The **Position** page contains information related to the child window's position, border, region, and alignment. For detailed information on the **Position** page, refer to the general description on page 213.

Geometry page   The **Geometry** page allows you to place constraints on a child window that specify how the object should be sized and positioned under specific conditions. For detailed information on the **Geometry** page, refer to the general description on page 214.

Advanced page   The **Advanced** page contains information relating to the advanced properties of the child window. It is designed for use by the experienced user only. For detailed information on the **Advanced** page, refer to the general description on page 218.

Other Objects

Help Options

T̸he Help category allows us to receive help at any time during a OpenZinc Designer session. This chapter discusses the different help areas in OpenZinc Designer we can use.

Selecting Help causes the following menu to appear:

Index

The **Index...** option allows you to view all help topics created within OpenZinc Designer. Selecting it causes an index list to appear, from which we can select these help topics. When we select a specific help topic from the list, its help window appears.

File

Selecting **File** displays help regarding the Window Editor's **File** options.

Edit

Selecting **Edit** displays help regarding the Window Editor's **Edit** options.

Window

Selecting **Window** displays help regarding the Window Editor's **Window** options.

Object

Selecting **Object** displays help regarding the Window Editor's **Object** options.

About Window Editor

Selecting **About Window Editor** causes information to be displayed regarding OpenZinc Designer's current version number and copyright information.

Help Options

Chapter 19   # Image Editor

The Image Editor provides options that allow you to create icon, bitmap, and mouse cursor images that can be used throughout your application.

Control window layout

Invoking the Image Editor causes the following control window to appear:
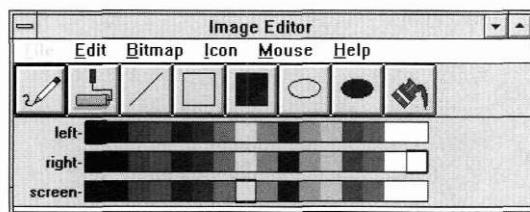
The menu bar   Using the options presented in the menus of the Image Editor control window, bitmaps, icons and mouse cursor images can be created and saved for use with OpenZinc resources. Selecting some menu items causes an action to take place immediately, while selecting others causes a related window to appear, from which more options are available. Menu items that cause another window to appear are distinguished by ellipses (...). A brief explanation of each menu item follows:

File. This menu is nonselectable in the Image Editor, since all file-related operations are handled in the Window Editor.

Edit. This menu consists of options that edit images. The edit options are:

- **Grid**
- **Roller Size**
- **Pattern**
- **Erase**
- **Cut**
- **Copy**
- **Paste**
- **Delete**
- **Group...,** and
- **Ungroup.**

Bitmap. This menu consists of options that control the creation of bitmap images specifically. The selectable items on this menu are:

- **Import...**
- **Export...**
- **Create**
- **Load...**
- **Store**
- **Store As...**
- **Clear**
- **Clear All,** and
- **Delete...**

Icon. This menu consists of options that control the creation of icon images specifically. The selectable items on this menu are:

- **Import...**
- **Export...**
- **Create**
- **Load...**
- **Store**
- **Store As...**
- **Clear**
- **Clear All,** and
- **Delete...**

Mouse. This menu consists of options that control the creation of mouse cursor images specifically. The selectable items on this menu are:

- **Import...**
- **Export...**
- **Create**
- **Load...**
- **Store**
- **Store As...**
- **Clear**
- **Clear All,** and
- **Delete...**

Help. This option provides general help for the Image Editor. The selectable items on this menu are:

- **Index...**
- **File**
- **Edit**
- **Bitmap**
- **Icon**
- **Mouse,** and
- **About Image Editor.**

All of these menu items are discussed in more detail in their respective sections that follow.

The button bar   The button bar presents the pencil, brush, line, rectangle, rectangle solid, ellipse, ellipse solid, and fill options. It is designed to allow you to easily select these options with a mouse. For details on how to use the button bar in creating images, refer to the creation window sections below.

The Color bars   The available colors are displayed in this area. To select a color, click on it with the left or right mouse button in the corresponding palette. For more details on how to use the color bars in creating images, refer to the creation window sections below.

Edit

The Edit category options are used to edit the appearance and performance of images within the current file. Selecting **Edit** causes the following menu to appear:
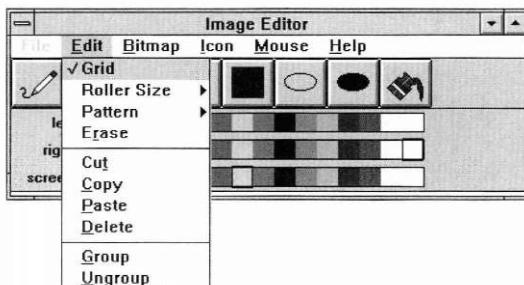
Grid   Selecting the **Grid** option causes a grid coordinate system to be displayed in the image creation field. When this option is in effect, a check mark is displayed next to it.

Roller Size	Selecting the **Roller Size** option causes another menu to appear which lists roller size choices. The size of roller affects how wide the brush stroke is when using the roller brush. The choices range from 1 pixel by 1 pixel through 1 pixel by 5 pixels (1x1 to 1x5).
Pattern	Selecting the **Pattern** option allows you to choose what type of drawing tool or pattern to use. It causes another menu to appear with the following options:

Pencil. This option sets the current drawing pen to the width of a pencil (one pixel wide).

Roller. This option sets the current drawing device to be a roller brush. In other words, it has the capacity to draw with a width greater than one pixel at a time. The actual width is determined by **Edit I Roller Size.**

Line. This option allows you to draw a line. A line is drawn by clicking the left mouse button (specifying the beginning point) and moving the mouse cursor to the ending point and releasing the mouse button.

Rectangle. This option allows you to create un unfilled rectangle. A rectangle is made by clicking the left mouse button (specifying the beginning point) and, while keeping the mouse button depressed, moving the mouse cursor to the ending point and releasing the mouse button.

Fill Rectangle. This option allows you to create a filled rectangle. A rectangle is made by clicking the left mouse button (specifying the beginning point) and, while keeping the mouse button depressed, moving the mouse cursor to the ending point and releasing the mouse button.

Ellipse. This option allows you to create un unfilled ellipse. An ellipse is made by clicking the left mouse button (specifying the beginning point of a rectangle that defines the ellipse) and, while keeping the mouse button depressed, moving the mouse cursor to the ending point of the defining rectangle and releasing the mouse button.

Fill Ellipse. This option allows you to create a filled ellipse. An ellipse is made by clicking the left mouse button (specifying the beginning point of the defining rectangle) and, while keeping the mouse button depressed, moving the mouse cursor to the ending point of the defining rectangle and releasing the mouse button.

Fill. This option allows you to perform a flood fill with the current drawing color. A flood fill is similar to pouring paint on an area. The color will spread and fill every blank pixel until it reaches an enclosing border. If the "paint" is poured in the middle of an ellipse, for example, the color will fill the blank spaces of the ellipse only. If, on the other hand, the ellipse is missing part of its border, the paint will "leak" out and fill the entire image field.

Erase

Selecting **Erase** causes the current image to be erased from the image field. It does not delete the image from the file.

NOTE: In order to avoid unintentional erasing, **Erase** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and pressing <Enter>.

Cut

Selecting the **Cut** option removes the current image group from the screen and places it in a global paste buffer. This option only has effect if a group is designated within the image field. (See the **Group** section below.)

Copy

Selecting the **Copy** option copies the current image group and places the copy in a global paste buffer. This option only has effect if a group is designated within the image field. (See the **Group** section below.)

Paste

Selecting **Paste** allows you to recall and position on the screen the contents of the global paste buffer (placed there by **Cut** or **Copy** procedures). After selecting **Paste,** position the mouse cursor where you would like the paste to occur and press the left mouse button.

Delete

Selecting **Delete** allows you to delete an image from the current file.

Group

Selecting **Group...** allows you to select a region on the image field to become its own unit, or image group. After selecting **Group** place the mouse cursor at a corner of the region to be grouped and drag the mouse to the opposite corner. Any pixel overlapped by the drag process will be included in the group. Upon releasing the mouse, the region will be marked by a colored border. This designates the area encompassing the group. **NOTE:** Grouping a region can also be accomplished by pressing <Ctrl> and then dragging the mouse to mark the desired area.

Only one image group is permitted at a time. Upon grouping a second region, the first group in the image will be dissolved (ungrouped).

This image group can be moved with the cursor, and it can be copied, cut, or pasted as described above.

Ungroup

Selecting **Ungroup** causes the group within the current image field to be dissolved. In other words, the cyan border surrounding the group disappears, and the section once again becomes a part of the original image. If the current image field does not contain an image group, **Ungroup** has no effect.

NOTE: Creating another group will also dissolve an edit group.

Bitmap   menu   options

The Bitmap category options are used to create bitmap images. Selecting **Bitmap** causes the following menu to appear:
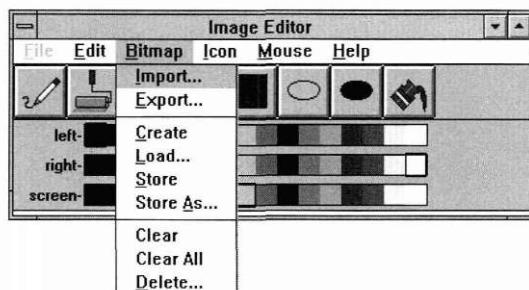
Import

Selecting the **Import** option allows you to import an image from another file. Bitmap images can be imported from **.BMP** (MS Windows and OS/2), **.XPM** (OSF/Motif), and **.DAT** (OpenZinc) files. This process involves two simple

steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Import,** a window similar to the Window Editor's **File I Open** window appears:
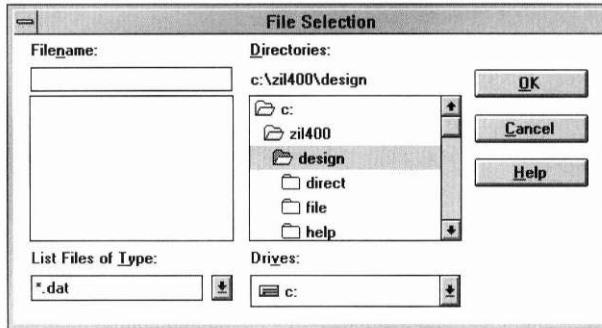
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:
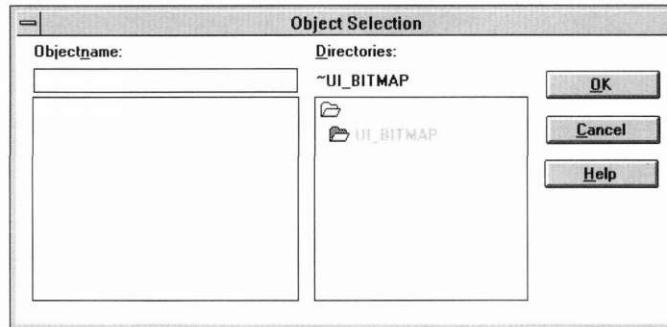
This window requests the actual bitmap image to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a bitmap image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other bitmap images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the bitmap image list will cause that image to be imported immediately.

Directories. This field indicates that you are currently in the bitmap directory. The field is for informational purposes only and cannot be edited.

OK. Selecting this button causes the bitmap image specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

*Help.* Additional information about importing images appears when this button is selected.

Once the image has been imported, it can be accessed through **Bitmap I Load...** (See page 345 of this chapter for more information on loading bitmap images.)

Export
Selecting the **Export** option allows you to export an image to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Export,** a window similar to the Window Editor's **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File** I **Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual image to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a bitmap image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other bitmap images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the bitmap image list will cause that image to be exported immediately.

Directories. This field indicates that you are currently in the bitmap directory. The field is for informational purposes only and cannot be edited.

OK. Selecting this button causes the bitmap image specified at the **Objectname** prompt to be exported. If the export procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting images appears when this button is selected.

Create　　　　　　　**Create** allows you to create a bitmap image. Selecting it automatically places the following window on the screen:
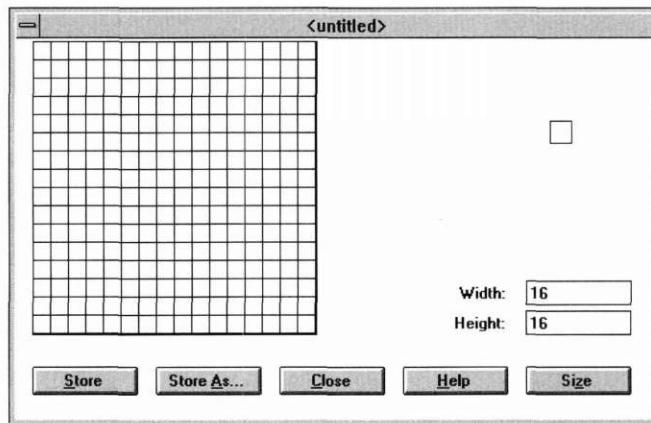
For details on interaction with this window, see the section **Bitmap Creation Window** below.

Load　　　　　　　**Load...** is used to recall a bitmap image from the current file. Selecting it causes a window similar to the following to appear:
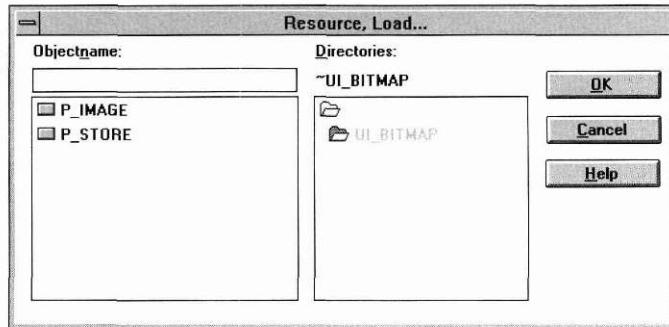
Objectname. To load a bitmap image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other bitmaps that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be loaded immediately.

Directories. The current bitmap directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

*OK.* Selecting this button causes the bitmap image specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the **Resource I Load** window will close and the bitmap image appears on the screen in the exact location and condition it was last stored.

If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading images appears when this button is selected.

Once the bitmap image has been loaded and appears on the screen, it can be modified in any way. When the **Bitmap I Store** option is subsequently selected, the image will be saved in its present condition, replacing the original version. (See the Store and Store As sections of this chapter for more information on storing images.)

Store

Selecting the **Store** option causes the current bitmap image to be saved in its present condition to the current file. If you have not previously named the image through a **Store** As operation, you will be asked for a name before you can store the image.

NOTE: Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store As

Store As... is generally used to store the current bitmap image under another name. Selecting it causes a window to appear that is similar to the following:

Objectname. Enter a name for the image at the **Objectname** prompt, or, if you want to replace a previously created image with the current information, select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current bitmap directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be stored immediately.

Directories. The current bitmap directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the image to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the Resource, Store As... window closes.

If no information has been entered within the Resource, Store As... window and you select the **OK** button, the window will close and no other action will take place.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing images appears when this button is selected.

Clear

Selecting **Clear** causes the current bitmap creation window to be cleared from the screen. It does not, however, delete the image from the file. If you have not stored the current image immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the image is neither stored nor cleared.

NOTE: In order to avoid unintentional clearing, **Clear** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all bitmap creation windows currently displayed to be cleared from the screen. It does not, however, delete any of those images from the file.

NOTE: In order to avoid unintentional clearing, **Clear All** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete

The **Delete...** option allows you to delete a bitmap image from the current file. Selecting it causes a window similar to the following to appear:

Objectname. Enter the name for the image to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current bitmap directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be deleted immediately.

Directories. The current bitmap directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

The purpose of this window is to make sure that you want to delete the image. If you select the **OK** button, the image indicated at the **Objectname** prompt is deleted from the current file, and both the confirmation window and the **Bitmap! Delete** window close. If you choose the **Cancel** button, the image is not deleted and just the confirmation window closes.

If the image entered does not exist, you will receive an error message when the OK button is selected.

If no information has been entered within the window, selecting OK causes an error message to appear.

If the delete operation is successful, the **Bitmapl Delete** window closes, and the image is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting images appears when this button is selected.

Icon menu options

The Icon category options are used to create icon images. Selecting **Icon** causes the following menu to appear:
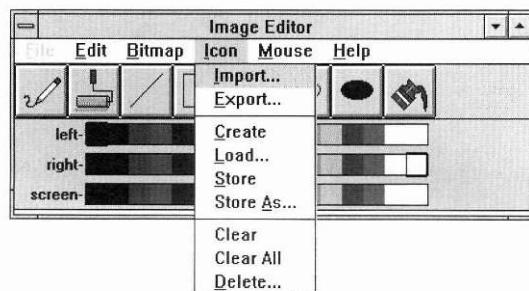
Import

Selecting the **Import** option allows you to import an image from another file. Icon images can be imported from **.ICO** (MS Windows and OS/2), **.XPM** (OSF/Motif), and **.DAT** (OpenZinc) files. This process involves two simple steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Import,** a window similar to the Window Editor's **File I Open** window appears:
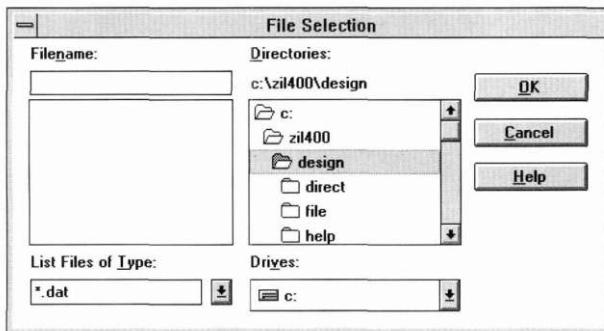
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:
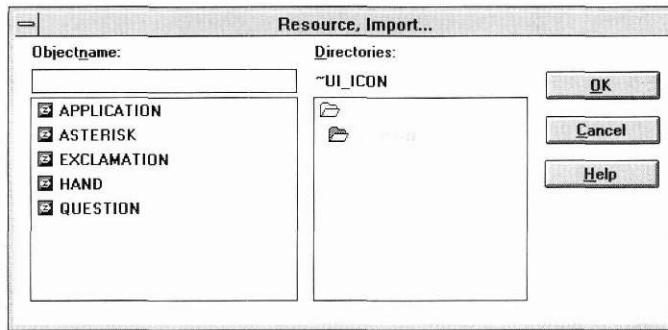
This window requests the actual icon image to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import an icon image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

OpenZinc **Designer**   **355**

Other icon images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the icon image list will cause that image to be imported immediately.

Directories. The current icon directory is shown below the **Directories** prompt. Since this item is not selectable, if you want to make a different directory the current one, it must be done by selecting a new directory from the list below the current directory prompt. This list displays other available directories of the current drive. The characters .. represent the parent directory, and, if selected, will display the other sub-directories of the current path, all of which are also selectable.

OK. Selecting this button causes the icon image specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing images appears when this button is selected.

Once the image has been imported, it can be accessed through **Icon I Load...** (See page 354 of this chapter for more information on loading icon images.)

Export

Selecting the **Export** option allows you to export an image to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Export,** a window similar to the Window Editor's **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual image to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export an icon image, either enter the name at the **Object-name** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other icon images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the icon image list will cause that image to be exported immediately.

Directories. This field indicates that you are currently in the icon directory. The field is for informational purposes only and cannot be edited.

OK. Selecting this button causes the icon image specified at the **Object-name** prompt to be exported. If the export procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

*Cancel.* Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting images appears when this button is selected.

Create

Create allows you to create an icon image. Selecting it automatically places the following window on the screen:
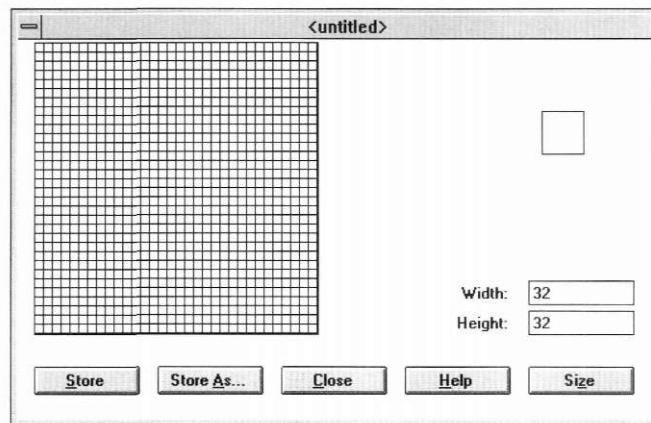
For details on interaction with this window, see the section Icon Creation Window below.

Load

Load... is used to recall a previously created icon image from the current file. Selecting it causes a window similar to the following to appear:
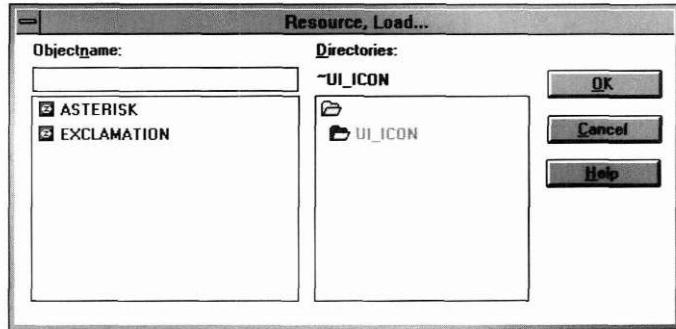
Objectname. To load an icon image, either enter the name **at the Object-name** prompt, or select it from the list below, and the name of the **image will** automatically appear at the prompt.

Other icons that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be loaded immediately.

Directories. The current icon directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

*OK.* Selecting this button causes the icon image specified at the **Object-name** prompt to be loaded. If the load procedure is successful, the **Resource I Load** window will close and the icon image appears on the screen **in** the exact location and condition it was last stored.

If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading images appears when this button is selected.

Once the icon image has been loaded and appears on the screen, it can be modified in any way. When the **Icon I Store** option is subsequently selected, the image will be saved in its present condition, replacing the original version. (See the Store and Store As sections of this chapter for more information on storing images.)

Store

Selecting the **Store** option causes the current icon image to be saved in its present condition to the current file. If you have not previously named the image through a **Store** As operation, you will be asked for a name before you can store the image.

NOTE: Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store   As

Store As... is generally used to store the current icon image under another name. Selecting it causes a window to appear that is similar to the following:

Objectname. Enter a name for the image at the **Objectname** prompt, or, if you want to replace a previously created image with the current information, select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current icon directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be stored immediately.

Directories. The current icon directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the image to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the Resource, Store As... window closes.

If no information has been entered within the Resource, Store As... window and you select the **OK** button, the window will close and no other action will take place.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing images appears when this button is selected.

Clear

Selecting **Clear** causes the current image creation window to be removed from the screen. It does not, however, delete the image from the file. If you have not stored the current image immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the image is neither stored nor cleared.

NOTE: In order to avoid unintentional clearing, **Clear** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all icon image creation windows currently displayed to be cleared from the screen. It does not, however, delete any of those images from the file.

NOTE: In order to avoid unintentional clearing, **Clear All** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete

The **Delete...** option allows you to delete an icon image from the current file. Selecting it causes a window similar to the following to appear:

Objectname. Enter the name for the image to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current icon directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be deleted immediately.

Directories. The current icon directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

The purpose of this window is to make sure that you want to delete the image. If you select the **OK** button, the image indicated at the **Objectname** prompt is deleted from the current file, and both the confirmation window and the **Iconl Delete** window close. If you choose the **Cancel** button, the image is not deleted and just the confirmation window closes.

If the image entered does not exist, you will receive an error message when the **OK** button is selected.

If no information has been entered within the window, selecting **OK** causes an error message to appear.

If the delete operation is successful, the **Icon I Delete** window closes, and the image is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting images appears when this button is selected.

Mouse menu options

The Mouse category options are used to create mouse cursor images. Selecting **Mouse** causes the following menu to appear:
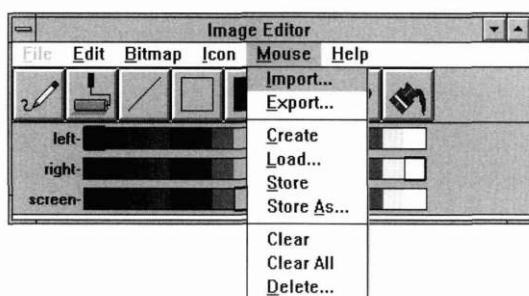
Import

Selecting the **Import** option allows you to import an image from another file. Mouse cursor images can be imported from **.ICO** (MS Windows and OS/2), **.XPM** (OSF/Motif), and **.DAT** (OpenZinc) files, files. This process involves two simple steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Import,** a window similar to the Window Editor's **File I Open** window appears:
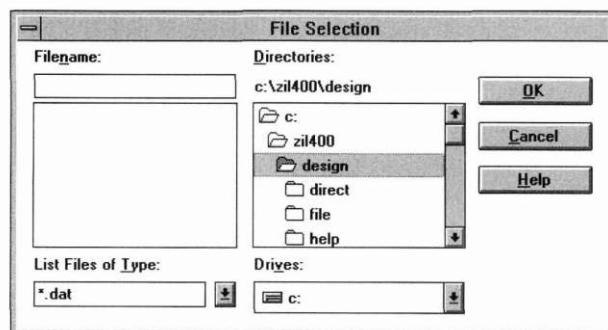
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File** I **Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:
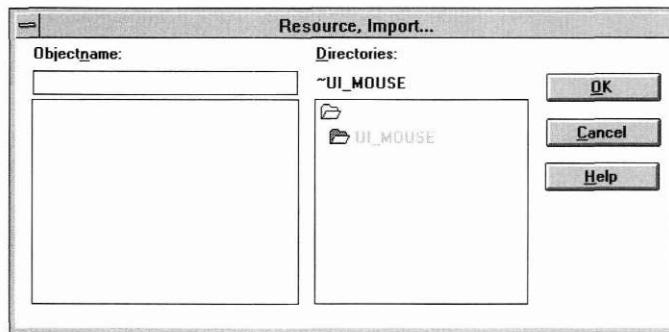
This window requests the actual mouse cursor image to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a mouse cursor image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other mouse cursor images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the mouse cursor image list will cause that image to be imported immediately.

Directories. This field indicates that you are currently in the mouse image directory. The field is for informational purposes only and cannot be edited.

OK. Selecting this button causes the mouse cursor image specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing images appears when this button is selected.

Once the image has been imported, it can be accessed through **Mouse I Load...** (See page 364 of this chapter for more information on loading mouse cursor images.)

Export

Selecting the **Export** option allows you to export an image to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file containing the desired image. Consequently, upon selecting **Export,** a window similar to the Window Editor's **File I Open** window appears:

Enter the name of the desired file at the Filename prompt and select the **OK** button. (Refer to the **File** I **Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual image to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a mouse cursor image, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other mouse cursor images that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the mouse cursor image list will cause that image to be exported immediately.

Directories. This field indicates that you are currently in the mouse image directory. The field is for informational purposes only and cannot be edited.

OK. Selecting this button causes the mouse cursor image specified at the Objectname prompt to be exported. If the export procedure is successful, the window will close. If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting images appears when this button is selected.

Create

Create allows you to create a mouse cursor image. Selecting it automatically places the following window on the screen:
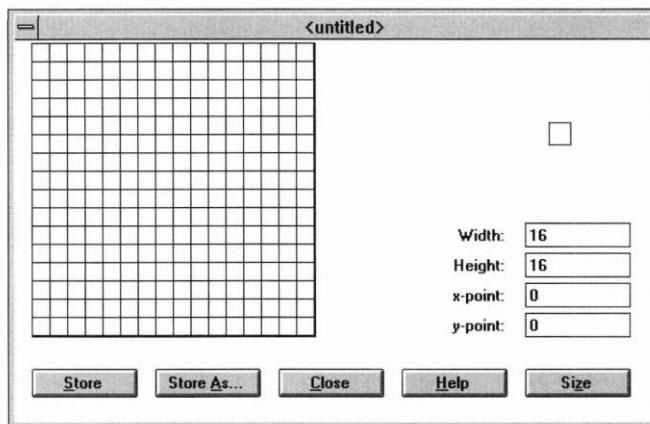
For details on interaction with this window, see the section Mouse cursor Creation Window below.

Load

Load... is used to recall a previously created mouse cursor image from the current file. Selecting it causes a window similar to the following to appear:
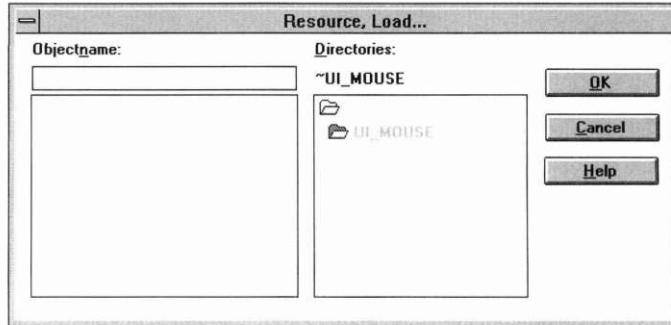
Objectname. To load a mouse cursor image, either enter the name at the Objectname prompt, or select it from the list below, and the name of the image will automatically appear at the prompt.

Other mouse cursors that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of

these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be loaded immediately.

Directories. The current mouse cursor directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the mouse cursor image specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the **Resource I Load** window will close and the mouse cursor image appears on the screen in the exact location and condition it was last stored.

If the image entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading images appears when this button is selected.

Once the mouse cursor image has been loaded and appears on the screen, it can be modified in any way. When the **Mouse I Store** option is subsequently selected, the image will be saved in its present condition, replacing the original version. (See the Store and Store As sections of this chapter for more information on storing images.)

Store

Selecting the **Store** option causes the current mouse cursor image to be saved in its present condition to the current file. If you have not previously named the image through a **Store As** operation, you will be asked for a name before you can store the image.

NOTE: Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store As

Store As... is generally used to store the current mouse cursor image under another name. Selecting it causes a window to appear that is similar to the following:

Objectname. Enter a name for the image at the **Objectname** prompt, or, if you want to replace a previously created image with the current information, select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current mouse cursor directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be stored immediately.

Directories. The current mouse cursor directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the image to be stored under the name entered at the Objectname prompt. If the store operation is successful, the Resource, Store As... window closes.

If no information has been entered within the Resource, Store As... window and you select the **OK** button, the window will close and no other action will take place.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing images appears when this button is selected.

Clear

Selecting **Clear** causes the current image creation window to be removed from the screen. It does not, however, delete the image from the file. If you have not stored the current image immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the image is neither stored nor cleared.

NOTE: In order to avoid unintentional clearing, **Clear** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all mouse cursor image creation windows currently displayed to be cleared from the screen. It does not, however, delete any of those images from the file.

NOTE: In order to avoid unintentional clearing, **Clear All** does not have a hot key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete

The **Delete...** option allows you to delete a mouse cursor image from the current file. Selecting it causes a window similar to the following to appear:

Objectname. Enter the name for the image to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that image will automatically appear at the prompt.

Other images that belong to the current mouse cursor directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these images causes the name to appear in the **Objectname** field. Double-clicking on a name listed in the files list will cause that image to be deleted immediately.

Directories. The current mouse cursor directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

The purpose of this window is to make sure that you want to delete the image. If you select the **OK** button, the image indicated at the **Objectname** prompt is deleted from the current file, and both the confirmation window and the **Resource I Delete** window close. If you choose the **Cancel** button, the image is not deleted and just the confirmation window closes.

If the image entered does not exist, you will receive an error message when the OK button is selected.

If no information has been entered within the window, selecting OK causes an error message to appear.

If the delete operation is successful, the **Resource I Delete** window closes, and the image is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting images appears when this button is selected.

Help menu options

The Help category is available so that you can receive help at any time while working in the Image Editor. The various options represent the different areas within the Image Editor where help information is available.

Selecting **Help** causes the following menu to appear:

INDEX	The **Index...** option allows you to view all help topics created within OpenZinc Designer. Selecting it causes an index list to appear from which these help topics are selectable. When you select a specific help topic from the list, the help window associated with it appears.
FILE	Selecting **File** causes help to be displayed regarding the use of File options in creating an image resource with the Image Editor.
EDIT	Selecting **Edit** causes help to be displayed regarding the use of Edit options in creating an image resource with the Image Editor.
BITMAP	Selecting **Bitmap** causes help to be displayed regarding the use of Bitmap options in creating an image resource with the Image Editor.
ICON	Selecting **Icon** causes help to be displayed regarding the use of Icon options in creating an image resource with the Image Editor.
MOUSE	Selecting **Mouse** causes help to be displayed regarding the use of Mouse options in creating an image resource with the Image Editor.
About Image Editor	Selecting **About Image Editor** causes information to be displayed regarding the general contents and specifics of OpenZinc Designer's Image Editor (e.g., the current version number and copyright information).

Bitmap creation window

The bitmap creation window is where you actually draw and set the size specifications of bitmap images. It is accessed by selecting **Bitmap I Create.**
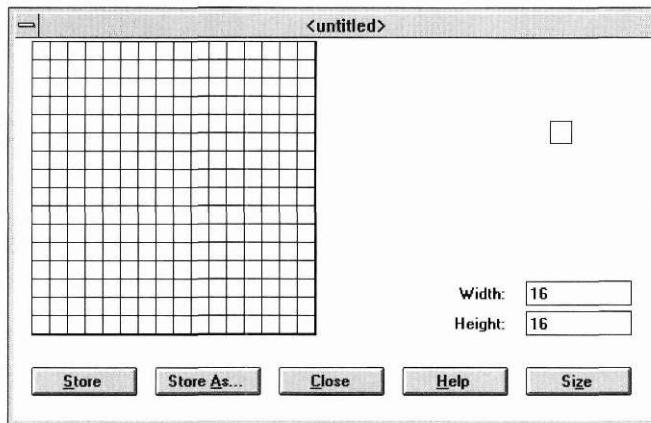
The bitmap image is drawn in the creation window's drawing field, which is the large square region in the upper left corner. This field is made up of individual pixels of the number determined by the **Width** and **Height** fields (described below). You can paint one pixel at a time by positioning on it and pressing a mouse button, or you can paint in continuous motion by holding down a mouse button and dragging the cursor.

Different tools for drawing can be selected from the button bar of the Image Editor control window. Simply select the desired tool. The various options are described in the **Edit I Pattern** section on page 343.

Colors are selected from the color bars of the control window. Notice that there are three separate palettes—one for the left mouse button, one for the right mouse button, and one for screen transparency. The screen transparency palette is used when you want to have part of your image show through to the screen behind it. The color selected is used to represent a transparent color in the image. Typically, a color that does not appear elsewhere in the image should be chosen to avoid confusion over which is transparent and which is part of the image. Whichever mouse button is used to select a color from this palette then has the power to draw a transparent region. Thereafter, whenever a different color is selected from the screen palette, all of the transparent region within the image will change to that color instantly.

As you create your image, it will be displayed in its actual size in the small square region, called the image field, in the upper right corner of the window.

The bitmap creation window also includes the following fields and buttons:

Width. This field determines the pixel width of the bitmap image. The default width for a bitmap is 16 pixels.

Height. This field determines the pixel height of the bitmap image. The default height for a bitmap is 16 pixels.

Store. Selecting this button causes the current bitmap image to be saved in its present condition to the current file. If you have not previously named the image through a **Store** As operation, you will be asked for a name before you can store the image.)

Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store As... This button is generally used to store the current bitmap image under another name. Selecting it causes a window to appear that is similar to the following:

Refer to "Store As" on page 351 for details on interacting with the **Store As...** window.

Close. Selecting this button causes the creation window to close without executing any changes.

*Help.* Additional information about creating images appears when this button is selected.

Size. Selecting this button causes the sizing information entered in the **Width** and **Height** fields to take effect in the drawing field.

Icon creation window

The icon creation window is where you actually draw and set the size specifications of icon images. It is accessed by selecting **Icon I Create.**
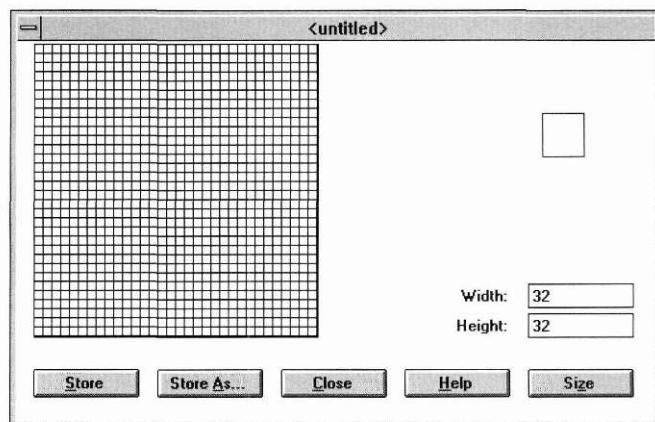
The icon image is drawn in the creation window's drawing field, which is the large square region in the upper left corner. This field is made up of individual pixels of the number determined by the **Width** and **Height** fields (described below). You can paint one pixel at a time by positioning on it and pressing a mouse button, or you can paint in continuous motion by holding down a mouse button and dragging the cursor.

Different tools for drawing can be selected from the button bar of the Image Editor control window. Simply select the desired tool. The various options are described in the **Edit** I **Pattern** section on page 343.

Colors are selected from the color bars of the control window. Notice that there are three separate palettes—one for the left mouse button, one for the right mouse button, and one for screen transparency. The screen transparency palette is used when you want to have part of your image show through to the screen behind it. The color selected is used to represent a transparent color in the image. Typically, a color that does not appear elsewhere in the image should be chosen to avoid confusion over which is transparent and which is part of the image. Whichever mouse button is used to select a color from this palette then has the power to draw a transparent region. Thereafter, whenever a different color is selected from the screen palette, all of the transparent region within the image will change to that color instantly.

As you create your image, it will be displayed in its actual size in the small square region, called the image field, in the upper right corner of the window.

The icon creation window also includes the following fields and buttons:

Width. This field determines the pixel width of the icon image. The only width allowed for an icon is 32 pixels.

Height. This field determines the pixel height of the icon image. The only height allowed for an icon is 32 pixels.

Store. Selecting this button causes the current icon image to be saved in its present condition to the current file. If you have not previously named the image through a **Store** As operation, you will be asked for a name before you can store the image.

Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store As... This button is generally used to store the current icon image under another name. Selecting it causes a window to appear that is similar to the following:

Refer to "Store As" on page 35**1** for details on interacting with the **Store As...** window.

Close. Selecting this button causes the creation window to close without executing any changes.

Help. Additional information about creating images appears when this button is selected.

Size. Selecting this button causes the sizing information entered in the **Width** and **Height** fields to take effect in the drawing field.

Mouse cursor creation window

The mouse cursor creation window is where you actually draw and set the size specifications of mouse cursor images. It is accessed by selecting **Mouse I Create.**
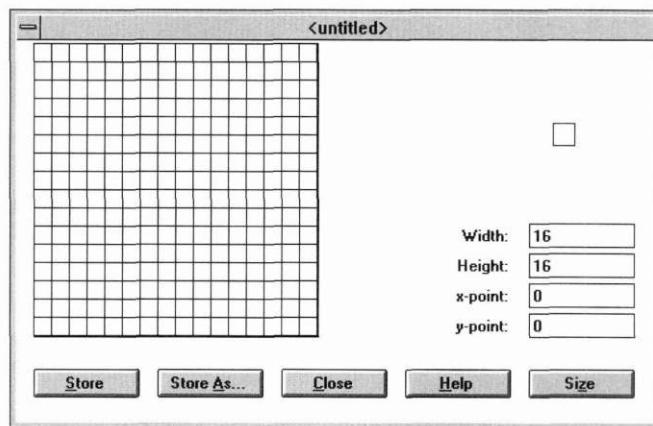
The mouse cursor image is drawn in the creation window's drawing field, which is the large square region in the upper left corner. This field is made up of individual pixels of the number determined by the **Width** and **Height** fields (described below). You can paint one pixel at a time by positioning on it and pressing a mouse button, or you can paint in continuous motion by holding down a mouse button and dragging the cursor.

Different tools for drawing can be selected from the button bar of the Image Editor control window. Simply select the desired tool. The various options are described in the **Edit** I **Pattern** section on page 343.

Colors are selected from the color bars of the control window. Notice that there are three separate palettes—one for the left mouse button, one for the right mouse button, and one for screen transparency. The screen transparency palette is used when you want to have part of your image show through to the screen behind it. The color selected is used to represent a transparent color in the image. Typically, a color that does not appear elsewhere in the image should be chosen to avoid confusion over which is transparent and which is part of the image. Whichever mouse button is used to select a color from this palette then has the power to draw a transparent region. Thereafter, whenever a different color is selected from the screen palette, all of the transparent region within the image will change to that color instantly.

As you create your image, it will be displayed in its actual size in the small square region, called the image field, in the upper right corner of the window.

The mouse cursor creation window also includes the following fields and buttons:

Width. This field determines the pixel width of the mouse cursor image. The default width for a mouse cursor is 16 pixels.

Height. This field determines the pixel height of the mouse cursor image. The default height for a mouse cursor is 16 pixels.

x-point. The value entered into this field determines the horizontal pixel coordinate for the cursor's hot spot.

y-point. The value entered into this field determines the vertical pixel coordinate for the cursor's hot spot.

Store. Selecting this button causes the current mouse cursor image to be saved in its present condition to the current file. If you have not previously named the image through a **Store** As operation, you will be asked for a name before you can store the image.

Each time a store operation is performed, the previous contents of the image are completely replaced by the current information.

Store As... This button is generally used to store the current mouse cursor image under another name. Selecting it causes a window to appear that is similar to the following:

Refer to "Store As" on page 351 for details on interacting with the **Store As...** window.

Close. Selecting this button causes the creation window to close without executing any changes.

Help. Additional information about creating images appears when this button is selected.

Size. Selecting this button causes the sizing information entered in the **Width** and **Height** fields to take effect in the drawing field.

Help Editor

In this chapter, we will discuss the Help Editor. The Help Editor is used to create and modify context-specific help information for an application. The help information is stored in the .DAT file and is automatically loaded at run time by OpenZinc.

Context

The **Context** category options allow you to create, modify and retrieve help contexts in the current file. Selecting **Context** causes the following menu to appear:
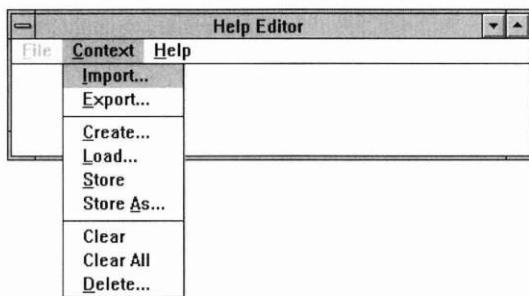
Import

Import allows you to import a help context from another file. This process involves two simple steps, the first of which is selecting the file containing the desired help context. Consequently, upon selecting **Import,** a window similar to the **File I Open** window appears:
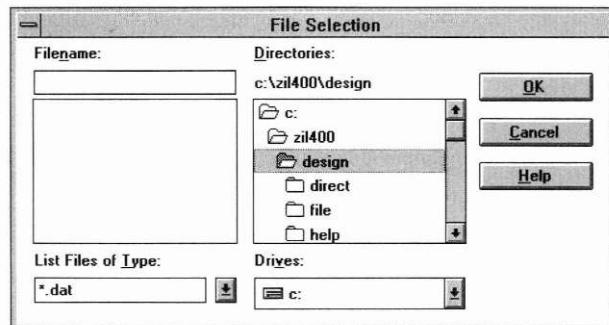
Enter the name of the desired fde at the **Filename** prompt and select the **OK** button. (Refer to "File" on page 393 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:
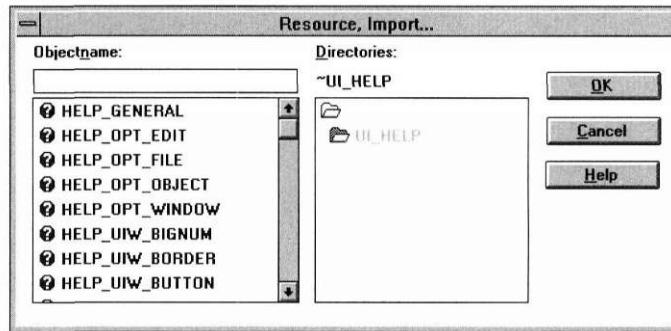
This window requests the actual help context to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a help context, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the help context will automatically appear at the prompt.

Other help contexts that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these help contexts causes the name to appear in the **Objectname** field. Double clicking on a name listed in the help context list will cause that help context to be imported immediately.

Directories. The current help directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the help context specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the help context entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing help contexts appears when this button is selected.

Once the help context has been imported, it can be accessed through **Context I Load...** (Refer to "Context" on page 393 for more information on loading help contexts.)

Export

Export allows you to export a help context to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file to which you would like to export the help context. Consequently, upon selecting **Export,** a window similar to the **File I Open** window appears:
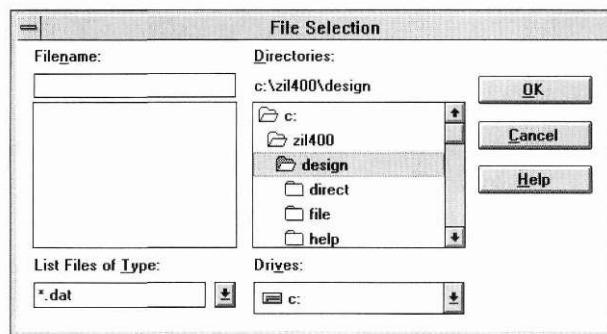
Enter the name of the desired file at the **"Filename"** prompt and select the **"OK"** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **"File I Export"** window.) After the **"File I Export"** window closes, another window, similar to the following, immediately opens:
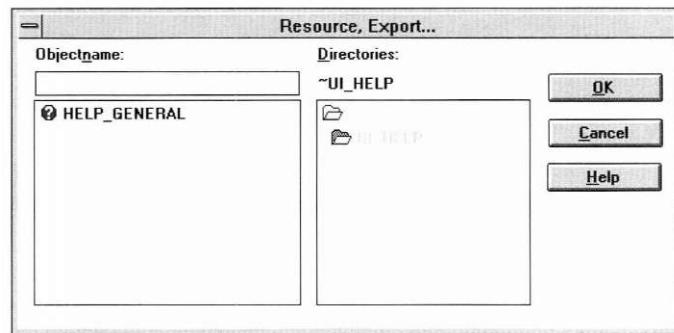
This window requests the actual help context to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a help context, either enter the name at the **"Objectname"** prompt, or select it from the list below, and the name of the help context will automatically appear at the prompt.

Other help contexts that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **"Objectname"** field. As mentioned above, selecting one of these help contexts causes the name to appear in the **"Objectname"** field. Double clicking on a name listed in the help context list will cause that help context to be exported immediately.

Directories. The current help directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the help context specified at the **"Objectname"** prompt to be exported. If the export procedure is successful, the window will close. If the help context entered at the **"Objectname"** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting help contexts appears when this button is selected.

Create

Selecting **Create** places the following window on the screen.
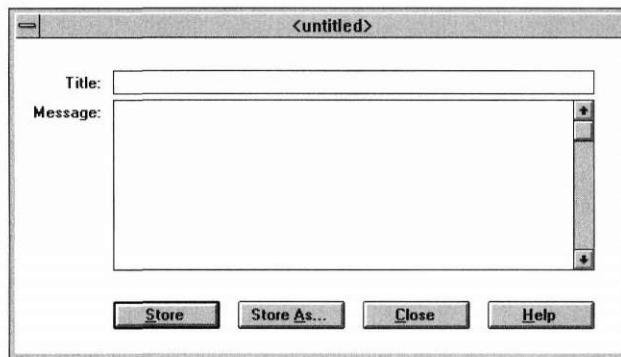
This window allows you to create a new help context. Interaction with its fields is described below.

Title. Enter the text that you would like to appear in the help window's title bar.

Message. Enter the information that you would like to appear in the help window.

Store. Selecting the **Store** option causes the current help context to be saved to the current file. If you have not entered a name for the help context through a **Store As.,** operation, you will be asked for a name before you can store the help context.

Note that each time a store operation is performed, the previous contents of the help context are completely replaced by the current information.

Store As. **Store As.,** is generally used to store the current help context under another name. (Refer to "Store" on page 390, or "Store As" on page 390, for further instructions on performing a **Store As.,** operation.)

Close. Selecting this button causes the window to close without executing any changes.

Help. Additional information about creating help contexts appears when this button is selected.

Load

Load... is used to recall a previously created help context from the current file. Selecting it causes a window similar to the following to appear:
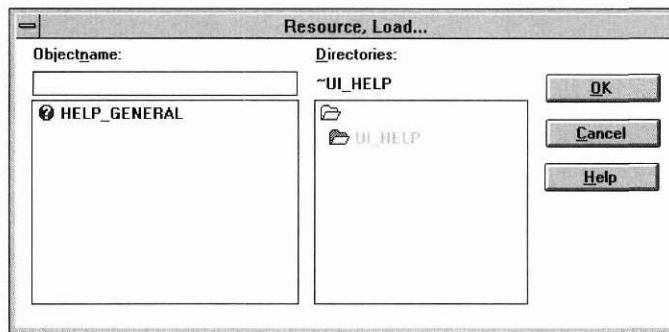
Objectname. To load a help context, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the help context will automatically appear at the prompt.

Other help contexts that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these help contexts causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that help context to be loaded immediately.

Directories. The current help directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the help context specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the *Context, Load* window will close and the help context appears on the screen.

If the help context entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

*Help.* Additional information about loading help contexts appears when this button is selected.

Once the help context has been loaded and appears on the screen, it can be modified. When the **Context I Store** option is subsequently selected, the help context will be saved in its present condition, replacing the original version. (See the **Store** and **Store As** sections of this chapter for more information on storing help contexts.)

Store

Selecting the **Store** option causes the current help context to be saved to the current file. If you have not entered a name for the help context through a **Store As.,** operation, you will be asked for a name before you can store the help context.

NOTE: Each time a store operation is performed, the previous contents of the help context are completely replaced by the current information.

Store As

Store As., is generally used to store the current help context under another name. Selecting it causes a window to appear that is similar to the following:
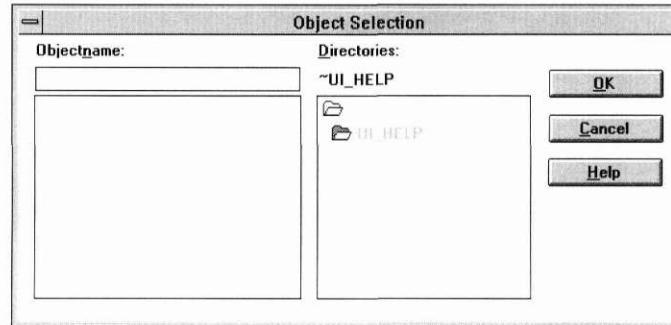
Objectname. Enter a name for the help context at the **Objectname** prompt, or, if you want to replace a previously created help context with the current information, select one from the field below, and the name for that help context will automatically appear at the prompt.

Other help contexts that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these help contexts causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that help context to be stored immediately.

Directories. The current help directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the help context to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the **Context I Store As...** window closes.

If no information has been entered within the **Store As.,** window and you select the OK button, the window will close and no other action will take place.

If you have entered a help context name that already exists, a modal window will appear, indicating such. If you select the **Yes** button of this window, the current information replaces the previous information for that help context, and both the modal window and the **Store As.,** window close. Selecting the **No** button simply closes the modal window and allows you to enter information again in the **Store As.,** window.

*Cancel.* Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing help contexts appears when this button is selected.

Clear

Selecting **Clear** causes the current help context window to be removed from the screen. It does not, however, delete the help context from the file. If you have not stored the current help context immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the help context is neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear** does not have a hot-key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all help context windows currently displayed to be removed from the screen. It does not, however, delete any of those help contexts from the file. If there are any help contexts that you have not stored immediately before selecting **Clear All,** a modal window will appear that asks if you want to store them before clearing them from the screen. Selecting **Yes** causes the help contexts to be stored and then cleared, selecting **No** causes them to be cleared without storing them first, and selecting **Cancel** simply closes the modal window and the help contexts are neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear All** does not have a hotkey assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete

The **Delete...** option allows you to delete a help context from the current file. Selecting it causes a window similar to the following to appear:
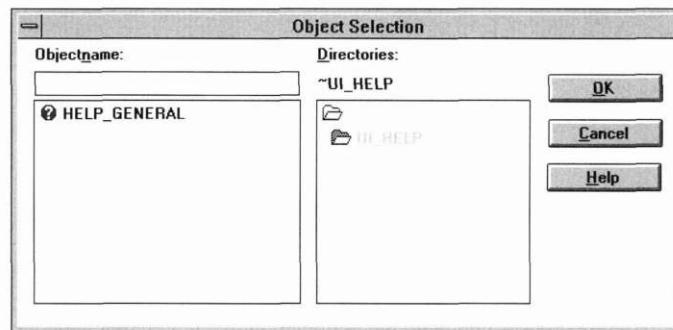
Objectname. Enter the name for the help context to be deleted at the **Object-name** prompt, or select one from the field below, and the name for that help context will automatically appear at the prompt.

Other help contexts that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these help contexts causes the name to appear in the **Objectname** field.

Directories. The current help directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear. The purpose of this window is to make sure that you want to delete the help context. If you select the **OK** button, the help context indicated at the **Objectname** prompt is deleted from the current file, and both the confirmation window and the *Context, Delete* window close. If you choose the **Cancel** button, the help context is not deleted and just the confirmation window closes.

If the help context entered does not exist, you will receive an error message when the **OK** button is selected.

If no information has been entered within the window, selecting **OK** causes an error message to appear.

If the delete operation is successful, the **Context I Delete** window closes, and the help context is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting help contexts appears when this button is selected.

Help

The **Help** category options allow you to get help about various topics in the Help Editor.

Index... Displays a list of all help available in the Help Editor. Selecting a topic from this list will display the help for that topic.

File Displays help for the **File** options.

Context Displays help for the **Context** options.

About Help Editor Provides an overview of the Help Editor.

Help Editor

Message Editor

In this chapter, we will discuss the Message Editor. The Message Editor is used to create and modify tables of messages which can be loaded by your application at run time and used wherever text strings are required. Using the Message Editor has several advantages over placing strings in your source code. The first advantage is that the strings can be changed without requiring the application to be rebuilt, since the strings are loaded from the **.DAT** file at run time. The second advantage is that different message tables can be created for different languages. Your application can load the necessary table at run time.

A message table is created as a **ZIL_LANGUAGE** object. After creating a message table in your application using the name that you used to store the message table in the Designer, you can call the **ZIL_LANGUAGE::Gret-Message()** function to access a particular string. The **GetMessage()** function identifies the string using the *numberlD* you assigned to the message in the Message Editor.

Message

The **Message** category options allow you to create, modify and retrieve messages in the current file. Selecting **Message** causes the following menu to appear:
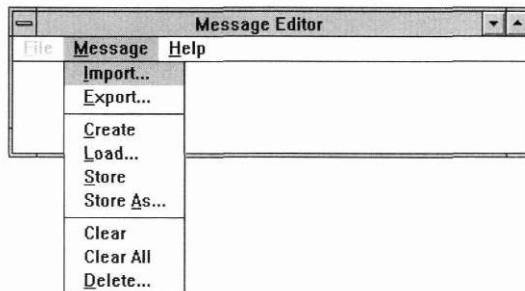
Import

Import allows you to import a message table from another file. This process involves two simple steps, the first of which is selecting the file containing the desired message. Consequently, upon selecting **Import,** a window similar to the **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:

This window requests the actual message table to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a message table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the message table will automatically appear at the prompt.

Other message tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these message tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the message table list will cause that message table to be imported immediately.

Directories. The current message directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the message table specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the message table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing message tables appears when this button is selected.

Once the message table has been imported, it can be accessed through **Message** I **Load...** (Refer to "Message" on page 406 for more information on loading messages.)

Export

Export allows you to export a message table to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file to which you would like to export the message table. Consequently, upon selecting **Export,** a window similar to the **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual message table to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a message table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the message table will automatically appear at the prompt.

Other message tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these message tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the message table list will cause that message table to be exported immediately.

Directories. The current message directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the message table specified at the **Objectname** prompt to be exported. If the export procedure is successful, the window will close. If the message table entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting message tables appears when this button is selected.

Create Selecting **Create** places the following window on the screen.
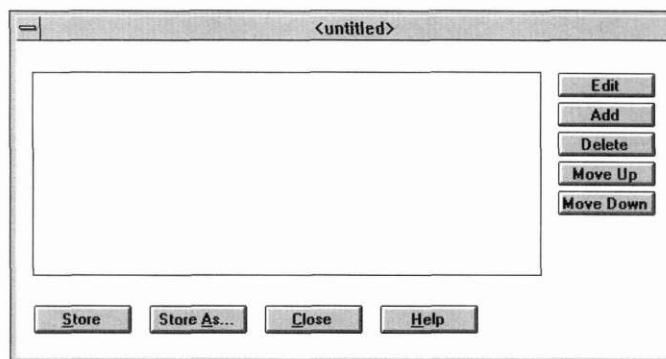
This window allows you to create a new message table. Interaction with its fields is described below.

Message list. This list, initially empty, contains all the messages in the message table. Double-clicking on an entry in the list brings up the message edit window. Interaction with this window is described in the **Edit** section for the message table window, below.

Edit. Selecting the **Edit** option brings up the edit window for the current message in the table. (See "The message edit window" on page 407. for details on interaction with the fields of this window.)

Add. Selecting the **Add** option adds a new default message entry to the end of the message table.

Delete. Selecting the **Delete** option removes the current message from the message table.

Move Up. Selecting the **Move Up** option moves the current message up in the message table one position.

Move Down. Selecting the **Move Down** option moves the current message down in the message table one position.

Store. Selecting the **Store** option causes the current message table to be saved to the current file. If you have not entered a name for the message table through a **Store** As., operation, you will be asked for a name before you can store the message table.

Note that each time a store operation is performed, the previous contents of the message table are completely replaced by the current information.

Store As. **Store As.,** is generally used to store the current message table under another name. (Refer to "Store" on page 402, or "Store As" on page 403, for further instructions on performing a **Store As.,** operation.)

Close. Selecting this button causes the window to close without executing any changes.

Help. Additional information about creating message tables appears when this button is selected.

Load

Load... is used to recall a previously created message table from the current file. Selecting it causes a window similar to the following to appear:
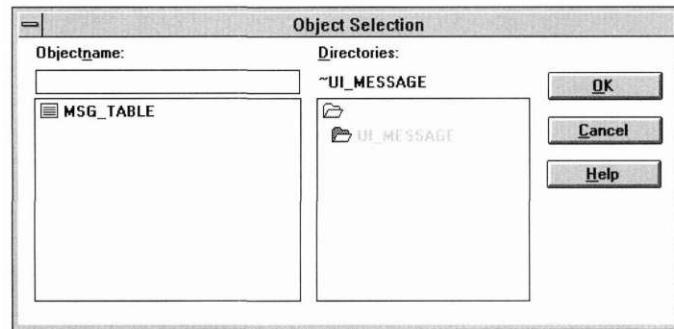

Objectname. To load a message table, either enter the name at the **Object-name** prompt, or select it from the list below, and the name of the message table will automatically appear at the prompt.

Other message tables that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of

these message tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that message table to be loaded immediately.

Directories. The current message table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the message table specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the *Message, Load* window will close and the message table appears on the screen.

If the message table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

*Help.* Additional information about loading message tables appears when this button is selected.

Once the message table has been loaded and appears on the screen, it can be modified. When the **Message I Store** option is subsequently selected, the message table will be saved in its present condition, replacing the original version. (See the **Store** and **Store As** sections of this chapter for more information on storing message tables.)

Store

Selecting the **Store** option saves the current message table to the current file. If you have not entered a name for the message table through a **Store** As., operation, you will be asked for a name before you can store the message table.

NOTE: Each time a store operation is performed, the previous contents of the message table are completely replaced by the current information.

Store As., is generally used to store the current message table under another name. Selecting it causes a window to appear that is similar to the following:
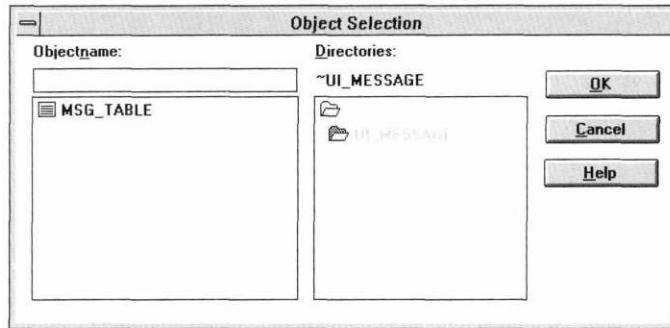
Objectname. Enter a name for the message table at the **Objectname** prompt, or, if you want to replace a previously created message table with the current information, select one from the field below, and the name for that message table will automatically appear at the prompt.

Other message tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these message tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that message table to be stored immediately.

Directories. The current message table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the message table to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the **Message I Store As...** window closes.

If no information has been entered within the **Store As...** window and you select the **OK** button, the window will close and no other action will take place.

If you have entered a message table name that already exists, a modal window will appear, indicating such. If you select the **Yes** button of this window, the current information replaces the previous information for that message

table, and both the modal window and the **Store As...** window close. Selecting the **No** button simply closes the modal window and allows you to enter information again in the **Store As...** window.

Cancel. Selecting this button causes the window to close without executing any changes.

*Help.* Additional information about storing message tables appears when this button is selected.

Clear

Selecting **Clear** causes the current message table window to be removed from the screen. It does not, however, delete the message table from the file. If you have not stored the current message table immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the message table is neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear** does not have a hot-key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all message table windows currently displayed to be removed from the screen. It does not, however, delete any of those message tables from the file. If there are any message tables that were not stored immediately before selecting **Clear,** a modal window will appear that asks if you want to store them before clearing them from the screen. Selecting **Yes** causes the message tables to be stored and then cleared, selecting **No** causes them to be cleared without storing them first, and selecting **Cancel** simply closes the modal window and the message tables are neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear All** does not have a hotkey assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete   The **Delete...** option allows you to delete a message table from the cuiTent file. Selecting it causes a window similar to the following to appear:

```
┌─────────────────────────────────────────────────────────────┐
│ ▭          Object Selection                                  │
│ Objectname:              Directories:                        │
│ ┌─────────────────┐      ~UI_MESSAGE           ┌──────────┐  │
│ │                 │                            │    OK    │  │
│ └─────────────────┘      ┌──────────────────┐  └──────────┘  │
│ ┌─────────────────┐      │ 📂                │  ┌──────────┐  │
│ │ ▤ MSG_TABLE     │      │  📁 UI_MESSAGE    │  │  Cancel  │  │
│ │                 │      │                   │  └──────────┘  │
│ │                 │      │                   │  ┌──────────┐  │
│ │                 │      │                   │  │   Help   │  │
│ │                 │      │                   │  └──────────┘  │
│ │                 │      │                   │                │
│ └─────────────────┘      └──────────────────┘                │
└─────────────────────────────────────────────────────────────┘
```
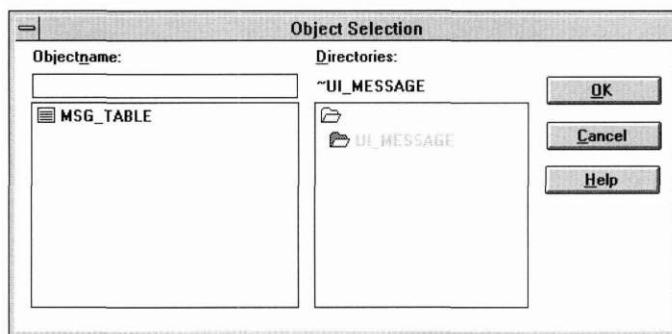
Objectname. Enter the name for the message table to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that message table will automatically appear at the prompt.

Other message tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these message tables causes the name to appear in the **Objectname** field.

Directories. The current message table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear. The purpose of this window is to make sure that you want to delete the message table. If you select the **OK** button, the message table indicated at the **Objectname** prompt is deleted from the current file, and both the confirmation window and the *Message, Delete* window close. If you choose the **Cancel** button, the message table is not deleted and just the confirmation window closes.

If the message table entered does not exist, you will receive an error message when the OK button is selected.

If no information has been entered within the window, selecting **OK** causes an error message to appear.

OpenZinc *Designer*   **405**

If the delete operation is successful, the **Message I Delete** window closes, and the message table is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting message tables appears when this button is selected.

Help

The **Help** category options allow you to get help about various topics in the Message Editor.

Index...	Displays a list of all help available in the Message Editor. Selecting a topic from this list will display the help for that topic.
File	Displays help for the **File** options.
Message	Displays help for the **Message** options.
System events	Displays a list of OpenZinc system events and their values.
Logical events	Displays a list of OpenZinc logical events and their values.
About Message Editor	Provides an overview of the Message Editor.

The message edit window

The message edit window appears when a message in the message list is double-clicked or the Edit button on the window is selected. The edit window is shown below.
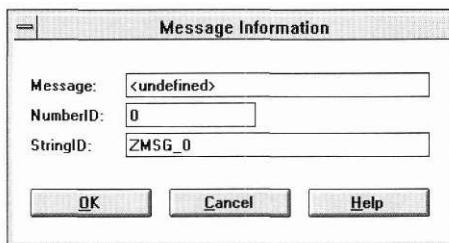
This window allows you to edit the values of the message. Interaction with its fields is described below.

Message   Enter the text for the message string.

NumberID   Enter the *numberID* that will be used to identify the message string within the table. This is the *numberID* that is used by the **GetMessage(**) function to access a specific string from your application.

StringID   Enter the *stringID* that will be used to identify the message within the table. The *stringID* text will be used to create a constant by the same name in the **.DAT** file's associated header file. The constant's value will be the *numberID* created in the *NumberID* field. This constant value can be used to specify the desired string when calling **ZIL_LANGUAGE::GetMessage(**) from your application.

OK   Selecting this button causes the message values to be stored in the table. The *Edit* window will close.

Cancel   Selecting this button causes the window to close without executing any changes.

Help   Additional information about editing messages appears when this button is selected.

Chapter 22 # Defaults Editor

In this chapter, we will discuss the Defaults Editor, used to create and modify tables of strings and locale information used by OpenZinc.

Language

Language options allow you to create, modify and retrieve language tables in the current file. Selecting **Language** causes the following menu to appear:
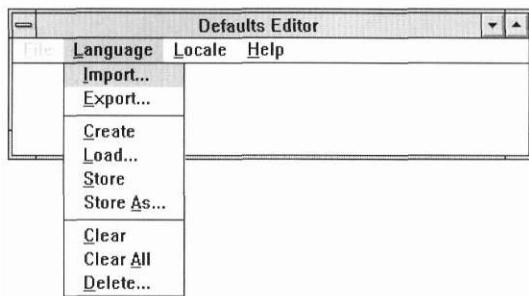

Import

Import allows you to import a language table from another file. This process involves two simple steps, the first of which is selecting the file containing the desired language. Consequently, upon selecting **Import,** a window similar to the **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:


This window requests the actual language table to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a language table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the language table will automatically appear at the prompt.

Other language tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these language tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the language table list will cause that language table to be imported immediately.

Directories. The current language directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the language table specified at the **Object-name** prompt to be imported. If the import procedure is successful, the window will close. If the language table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.


Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing language tables appears when this button is selected.

Once the language table has been imported, it can be accessed through **Language I Load...** (Refer to page 414 for more information on loading language tables.)

Export

Export allows you to export a language table to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file to which you would like to export the language table. Consequently, upon selecting **Export,** a window similar to the **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual language table to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a language table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the language table will automatically appear at the prompt.

Other language tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these language tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the language table list will cause that language table to be exported immediately.

Directories. The current locale directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the language table specified at the **Objectname** prompt to be exported. If the export procedure is successful, the window will close. If the language table entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting language tables appears when this button is selected.

Create

Selecting **Create** places the following window on the screen.
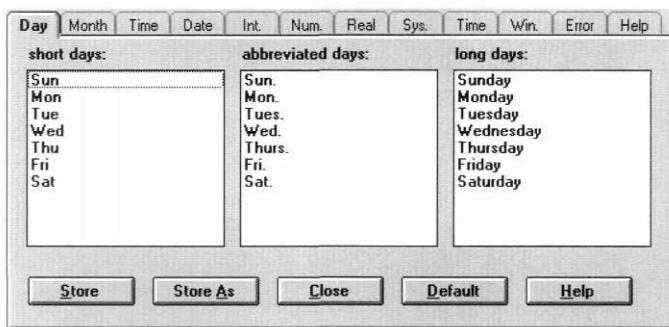
This window allows you to create a new language table. For details on interaction with its fields, see "The language window" on page 429.

Load

Load... is used to recall a previously created language table from the current file. Selecting it causes a window similar to the following to appear:
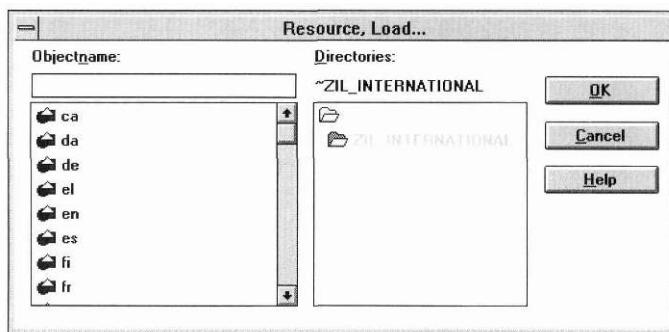
Objectname. To load a language table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the language table will automatically appear at the prompt.

Other language tables that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of

these language tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that language table to be loaded immediately.

Directories. The current language table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the language table specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the **Language I Load** window will close and the language table appears on the screen.

If the language table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading language tables appears when this button is selected.

Once the language table has been loaded and appears on the screen, it can be modified. When the **Language I Store** option is subsequently selected, the language table will be saved in its present condition, replacing the original version. (See the **Store** and **Store As** descriptions in this section for more information on storing language tables.)

Store

Selecting the Store option causes the current language table to be saved to the current file. If you have not entered a name for the language table through a Store As... operation, you will be asked for a name before you can store the language table.

NOTE: Each time a store operation is performed, the previous contents of the language table are completely replaced by the current information.

Store As

Store As... is generally used to store the current language table under another name. Selecting it causes a window to appear that is similar to the following:
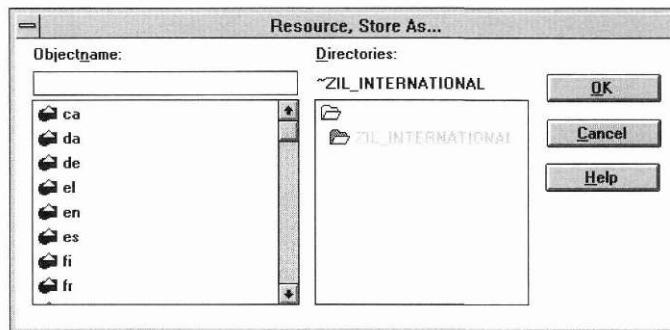
Objectname. Enter a name for the language table at the **Objectname** prompt, or, if you want to replace a previously created language table with the current information, select one from the field below, and the name for that language table will automatically appear at the prompt.

Other language tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these language tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that language table to be stored immediately.

Directories. The current language table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the language table to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the *Language, Store As...* window closes.

If no information has been entered within the **Store As...** window and you select the **OK** button, the window will close and no other action will take place.

If you have entered a language table name that already exists, a modal window will appear, indicating such. If you select the **Yes** button of this window, the current information replaces the previous information for that language

table, and both the modal window and the **Store As...** window close. Selecting the **No** button simply closes the modal window and allows you to enter information again in the **Store As...** window.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing language tables appears when this button is selected.

Clear

Selecting **Clear** causes the current language table window to be removed from the screen. It does not, however, delete the language table from the file. If you have not stored the current language table immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the language table is neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear** does not have a hotkey assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and pressing <Enter>.

Clear All

Selecting **Clear All** causes all language table windows currently displayed to be removed from the screen. It does not, however, delete any of those language tables from the file. If there are any language tables that were not stored immediately before selecting **Clear,** a modal window will appear that asks if you want to store them before clearing them from the screen. Selecting **Yes** causes the language tables to be stored and then cleared, selecting **No** causes them to be cleared without storing them first, and selecting **Cancel** simply closes the modal window and the language tables are neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear All** does not have a hotkey assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and pressing <Enter>.

Delete

The **Delete...** option allows you to delete a language table from the current file. Selecting it causes a window similar to the following to appear:
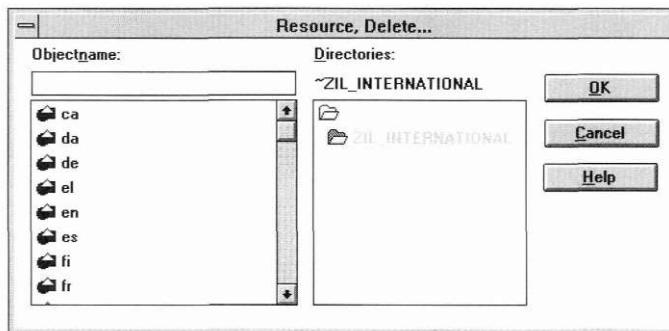

Objectname. Enter the name for the language table to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that language table will automatically appear at the prompt.

Other language tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these language tables causes the name to appear in the **Objectname** field.

Directories. The current language table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

The purpose of this window is to make sure that you want to delete the language table. If you select the **OK** button, the language table indicated at the **Objectname** prompt is deleted from the current file, and both the confirma-

tion window and the *Language, Delete* window close. If you choose the **Cancel** button, the language table is not deleted and just the confirmation window closes.

If the language table entered does not exist, you will receive an error message when the OK button is selected.

If no information has been entered within the window, selecting OK causes an error message to appear.

If the delete operation is successful, the *Language, Delete* window closes, and the language table is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting language tables appears when this button is selected.

Locale

The **Locale** category options allow you to create, modify, and retrieve locale tables in the current file. The locale tables contain formatting information for those OpenZinc objects that need it. Selecting **Locale** causes the following menu to appear:
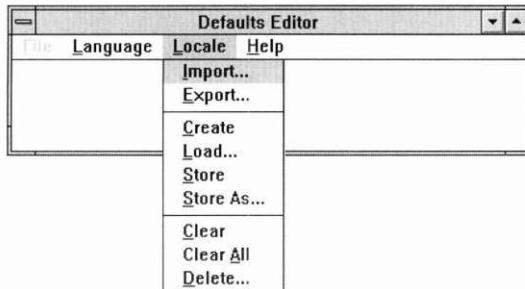
Import

Import allows you to import a locale table from another file. This process involves two simple steps, the first of which is selecting the file containing the desired locale. Consequently, upon selecting **Import,** a window similar to the *File, Open* window appears:
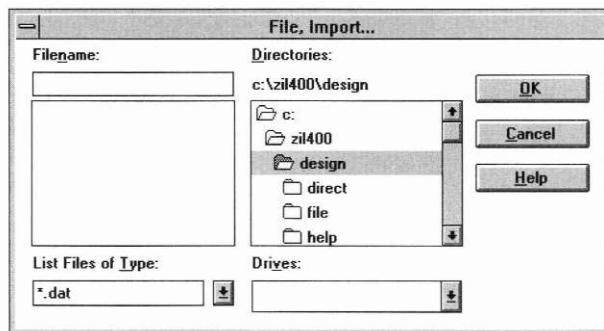
Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to "File" on page 429 for further instructions on interacting with the **File I Import** window.) After the **File I Import** window closes, another window, similar to the following, immediately opens:

This window requests the actual locale table to be imported from the designated file. Interaction with its fields is described below.

Objectname. To import a locale table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the locale table will automatically appear at the prompt.

Other locale tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these locale tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the locale table list will cause that locale table to be imported immediately.

Directories. The current locale directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the locale table specified at the **Objectname** prompt to be imported. If the import procedure is successful, the window will close. If the locale table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about importing locale tables appears when this button is selected.

Once the locale table has been imported, it can be accessed through **Locale I Load...** (Refer to page 424 for more information on loading locale tables.)

Export

Export allows you to export a locale table to another file. This process, much like the importing process, involves two simple steps, the first of which is selecting the file to which you would like to export the locale table. Consequently, upon selecting **Export,** a window similar to the **File I Open** window appears:

Enter the name of the desired file at the **Filename** prompt and select the **OK** button. (Refer to the **File I Open** section on page 197 for further instructions on interacting with the **File I Export** window.) After the **File I Export** window closes, another window, similar to the following, immediately opens:

This window requests the actual locale table to be exported to the designated file. Interaction with its fields is described below.

Objectname. To export a locale table, either enter the name at the **Object-name** prompt, or select it from the list below, and the name of the locale table will automatically appear at the prompt.

Other locale tables that belong to the current directory are listed, in alphabetic order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these locale tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the locale table list will cause that locale table to be exported immediately.

Directories. The current locale directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the locale table specified at the **Objectname** prompt to be exported. If the export procedure is successful, the window will close. If the locale table entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about exporting locale tables appears when this button is selected.

Create

Selecting **Create** places the following window on the screen.
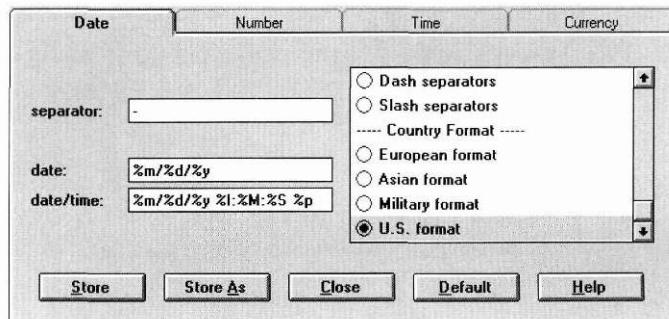
This window allows you to create a new locale table. For details on interaction with its fields, see "The locale window" on page 438.

Load   **Load...** is used to recall a previously created locale table from the current file. Selecting it causes a window similar to the following to appear:
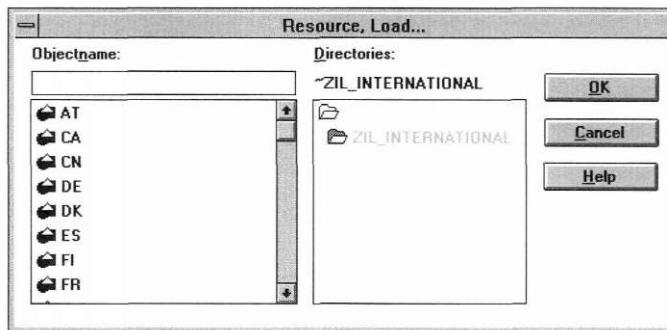
Objectname. To load a locale table, either enter the name at the **Objectname** prompt, or select it from the list below, and the name of the locale table will automatically appear at the prompt.

Other locale tables that belong to the current directory (including those imported from other files) are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these locale tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that locale table to be loaded immediately.

Directories. The current locale table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the locale table specified at the **Objectname** prompt to be loaded. If the load procedure is successful, the *Locale, Load* window will close and the locale table appears on the screen.

If the locale table name entered at the **Objectname** prompt does not exist, or if no information has been entered, you will receive an error message at this time.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about loading locale tables appears when this button is selected.

Once the locale table has been loaded and appears on the screen, it can be modified. When the **Locale I Store** option is subsequently selected, the locale table will be saved in its present condition, replacing the original version. (See the **Store** and **Store** As descriptions in this section for more information on storing locale tables.)

Store

Selecting the **Store** option causes the current locale table to be saved to the current file. If you have not entered a name for the locale table through a **Store** As... operation, you will be asked for a name before you can store the locale table.

NOTE: Each time a store operation is performed, the previous contents of the locale table are completely replaced by the current information.

Store As

Store As... is generally used to store the current locale table under another name. Selecting it causes a window to appear that is similar to the following:
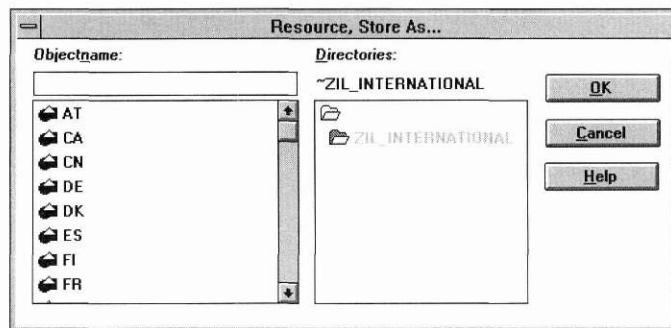
Objectname. Enter a name for the locale table at the **Objectname** prompt, or, if you want to replace a previously created locale table with the current information, select one from the field below, and the name for that locale table will automatically appear at the prompt.

Other locale tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these locale tables causes the name to appear in the **Objectname** field. Double clicking on a name listed in the **Objectname** list will cause that locale table to be stored immediately.

Directories. The current locale table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes the locale table to be stored under the name entered at the **Objectname** prompt. If the store operation is successful, the *Locale, Store As...* window closes.

If no information has been entered within the **Store As...** window and you select the **OK** button, the window will close and no other action will take place.

If you have entered a locale table name that already exists, a modal window will appear, indicating such. If you select the **Yes** button of this window, the current information replaces the previous information for that locale table, and both the modal window and the **Store As...** window close. Selecting the **No** button simply closes the modal window and allows you to enter information again in the **Store As...** window.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about storing locale tables appears when this button is selected.

Clear

Selecting **Clear** causes the current locale table window to be removed from the screen. It does not, however, delete the locale table from the file. If you have not stored the current locale table immediately before, selecting **Clear** causes a modal window to appear that asks if you want to store it before clearing it from the screen. Selecting **Yes** causes it to be stored and then cleared, selecting **No** causes it to be cleared without storing it first, and selecting **Cancel** simply closes the modal window and the locale table is neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear** does not have a hot-key assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Clear All

Selecting **Clear All** causes all locale table windows currently displayed to be removed from the screen. It does not, however, delete any of those locale tables from the file. If there are any locale tables that were not stored imme-

diately before selecting **Clear,** a modal window will appear that asks if you want to store them before clearing them from the screen. Selecting **Yes** causes the locale tables to be stored and then cleared, selecting **No** causes them to be cleared without storing them first, and selecting **Cancel** simply closes the modal window and the locale tables are neither stored nor cleared.

Note that in order to avoid unintentional clearing, **Clear All** does not have a hotkey assignment. It can only be activated by selecting it from the menu with a mouse or by scrolling to it and hitting <Enter>.

Delete

The **Delete...** option allows you to delete a locale table from the current file. Selecting it causes a window similar to the following to appear:
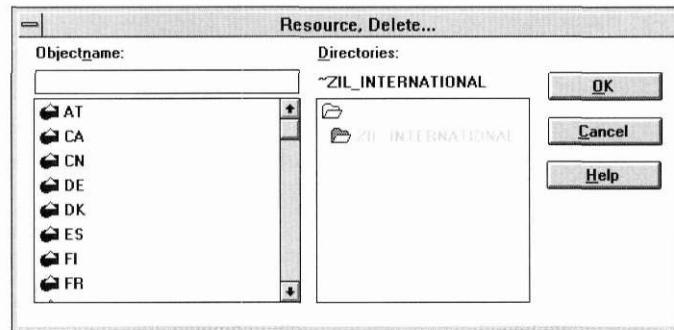
Objectname. Enter the name for the locale table to be deleted at the **Objectname** prompt, or select one from the field below, and the name for that locale table will automatically appear at the prompt.

Other locale tables that belong to the current directory are listed, in alphabetical order, in the scrollable field below the **Objectname** field. As mentioned above, selecting one of these locale tables causes the name to appear in the **Objectname** field.

Directories. The current locale table directory is shown below the **Directories** prompt. Other directories of the current file are listed in the field below the current directory prompt. These two fields are for informational purposes only and cannot be edited.

OK. Selecting this button causes a modal window to appear which is similar to the following:

```
┌─────────────────────────────┐
│ ═│    Resource, Delete...    │
├─────────────────────────────┤
│                             │
│    Delete resource "FR"?    │
│                             │
│  ┌─────────┐   ┌─────────┐  │
│  │  Yes    │   │   No    │  │
│  └─────────┘   └─────────┘  │
└─────────────────────────────┘
```

The purpose of this window is to make sure that you want to delete the locale table. If you select the **OK** button, the locale table indicated at the **Object-name** prompt is deleted from the current file, and both the confirmation window and the **Locale** I **Delete** window close. If you choose the **Cancel** button, the locale table is not deleted and just the confirmation window closes.

If the locale table entered does not exist, you will receive an error message when the **OK** button is selected.

If no information has been entered within the window, selecting **OK** causes an error message to appear.

If the delete operation is successful, the **Locale I Delete** window closes, and the locale table is deleted from the current file.

Cancel. Selecting this button causes the window to close without executing any changes.

Help. Additional information about deleting locale tables appears when this button is selected.

Help

The **Help** category options allow you to get help about various topics in the Defaults Editor.

Index...   Displays a list of all help available in the Defaults Editor. Selecting a topic from this list will display the help for that topic.

File	Displays help for the **File** options.
Language	Displays help for the **Language** options.
Locale	Displays help for the **Locale** options.
System events	Displays a list of OpenZinc system events and their values.
Logical events	Displays a list of OpenZinc logical events and their values.
About Language Editor	Provides an overview of the Language Editor.

The language window

The language window allows you to modify strings that are used by OpenZinc for such purposes as window titles, system menu options, and error messages. Each object that may require a string translation has its own notebook page on the window. Each of these pages is described in detail within this section. All of the pages have several common buttons, described just below.

Common buttons

Store. Selecting the **Store** option causes the current locale table to be saved to the current file. If you have not entered a name for the locale table through a **Store** As... operation, you will be asked for a name before you can store the locale table.

Note that each time a store operation is performed, the previous contents of the locale table are completely replaced by the current information.

Store As. **Store** As... is generally used to store the current locale table under another name. (Refer to "Store" on page 415, or "Store As" on page 416, for further instructions on performing a **Store** As... operation.)

Close. Selecting this button causes the window to close without executing any changes.

Default. Selecting this button returns the entries to their default values.

Help. Additional information about creating locale tables appears when this button is selected.

Day

This page, shown below, contains the strings used for alphanumeric days.
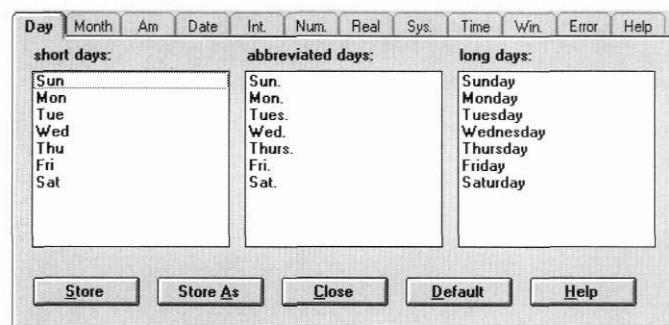
Short days. Contains the strings used when displaying a short day name, such as would appear in a date field that has the *Military format* option set.

Abbreviated days. Contains the strings used when displaying an abbreviated day name, such as would appear in a date field that has the *Short alphanumeric day* option set.

Long days. Contains the strings used when displaying a full day name, such as would appear in a date field that has the *Alphanumeric day-of-week* option set.

Month

Contains the strings used for alphanumeric months. The page is shown below.
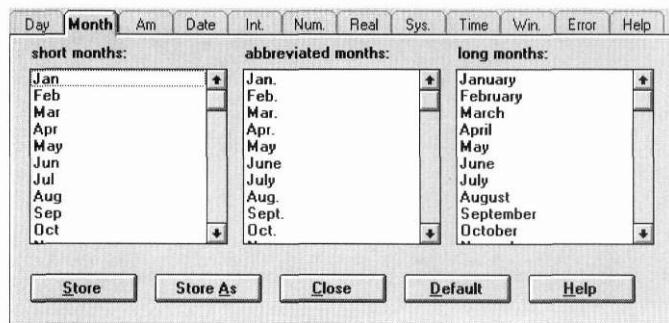
short months. Contains the strings used when displaying a short month name, such as would appear in a date field that has the *Military format* option set.

abbreviated months. Contains the strings used when displaying an abbreviated month name, such as would appear in a date field that has the *Short alphanumeric month* option set.

long months. Contains the strings used when displaying a full month name, such as would appear in a date field that has the *Alphanumeric month* option set.

Am

Contains the strings used to indicate if a time is antemeridian or postmeridian. The page is shown below.
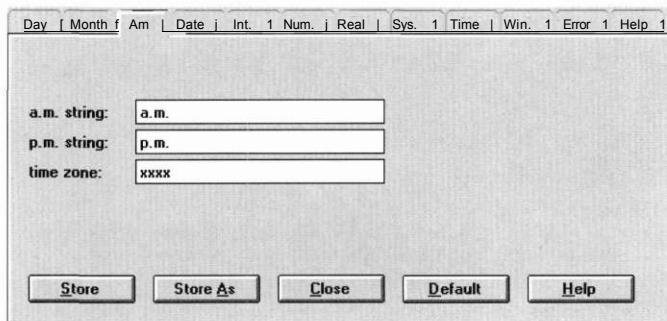
a.m. string. Indicates a time is antemeridian.

p.m. string. Indicates a time is postmeridian.

time zone. This string identifies the time zone.

Date   Contains the strings in error messages for a date field. The page is shown
                            below.
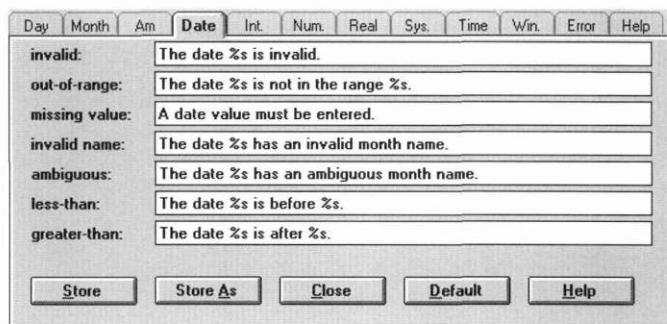
invalid. Indicates the date is in an invalid format.

out-of-range. Indicates the date is out of range for the date field.

missing value. Indicates the date is missing a required value and cannot be
interpreted.

invalid name. Indicates the date has an invalid month name that cannot be
interpreted.

ambiguous. Indicates the date has an ambiguous month name that cannot be
uniquely interpreted.

less-than. Indicates the date is chronologically less than the low end of a
positive, open-ended range of acceptable date values.

greater-than. Indicates the date is chronologically greater than the high end
of a negative, open-ended range of acceptable date values.

Integer

Contains the strings used to display error messages for an integer field. The page is shown below.
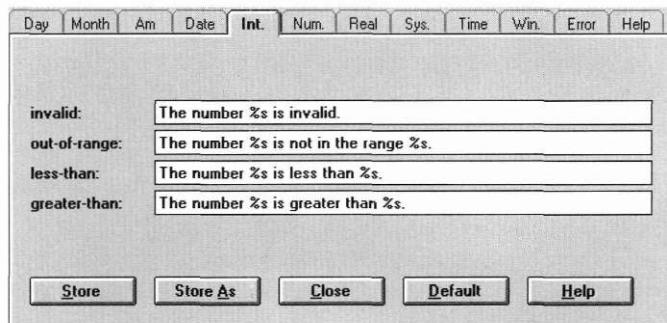
invalid. Indicates the number is in an invalid format.

out-of-range. Indicates the number is out of range for the integer field.

less-than. Indicates the number is less than the low end of a positive, open-ended range of acceptable values.

greater-than. Indicates the number is greater than the high end of a negative, open-ended range of acceptable values.

Number

Contains the strings used to display error messages for a number field. The page is shown below.
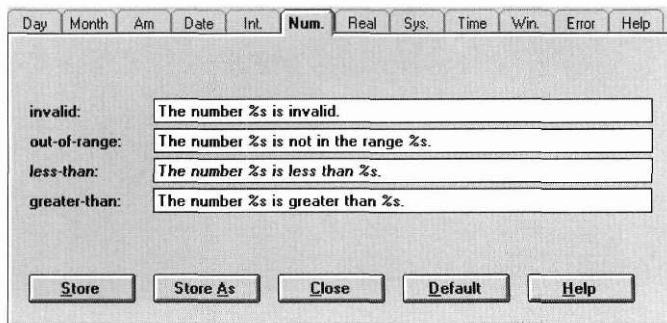
invalid. Indicates the number is in an invalid format.

out-of-range. Indicates the number is out of range for the field.

less-than. Indicates the number is less than the low end of a positive, open-ended range of acceptable values.

greater-thcin. Indicates the number is greater than the high end of a negative, open-ended range of acceptable values.

Real   Contains the strings used to display error messages for a real field. The page is shown below.
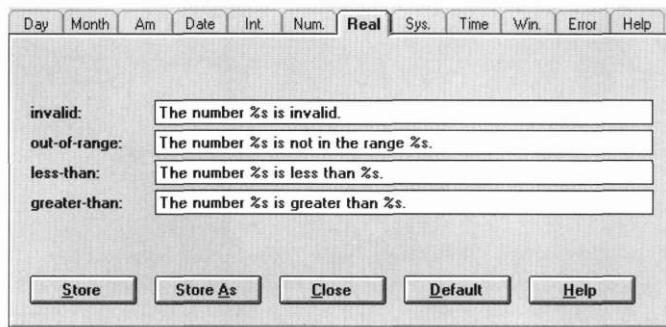
invalid. Indicates the number is in an invalid format.

out-of-range. Indicates the number is out of range for the real field.

less-than. Indicates the number is less than the low end of a positive, open-ended range of acceptable values.

greater-than. Indicates the number is greater than the high end of a negative, open-ended range of acceptable values.

System button   Contains the strings used to display the options in a generic system button menu. The page is shown below.
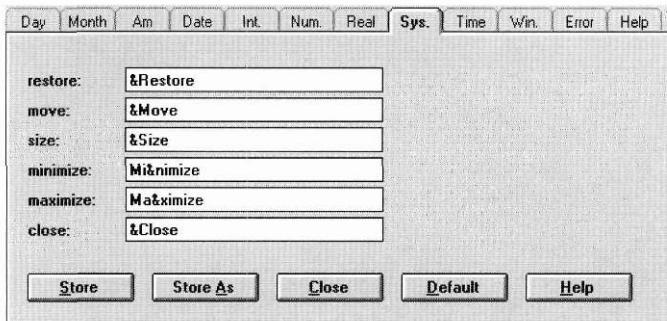
restore. Text used to create the menu option that restores a window from its minimized or maximized state.

move. Text used to create the menu option that allows the window to be moved using the keyboard.

size. Text used to create the menu option that allows the window to be sized using the keyboard.

minimize. Text used to create the menu option that minimizes the window.

maximize. Text used to create the menu option that maximizes the window.

close. Text used to create the menu option that closes the window.

Time

Contains the strings used to display error messages for a time field. The page is shown below.
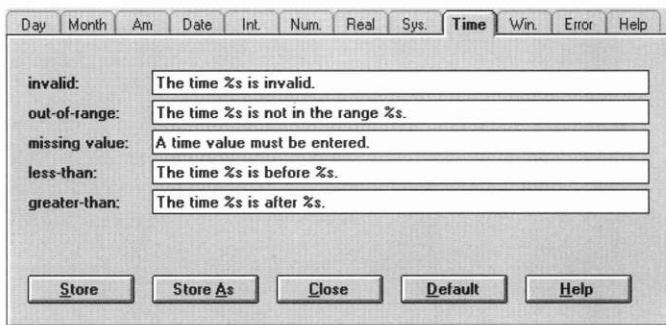
invalid. Indicates the time is in an invalid format.

out-of-range. Indicates the time is out of range for the time field.

missing value. Indicates the time is missing a required value and cannot be interpreted.

less-than. Indicates the time is chronologically less than the low end of a positive, open-ended range of acceptable time values.

greater-than. Indicates the time is chronologically greater than the high end of a negative, open-ended range of acceptable time values.

Window

Contains the strings used to display error messages for a window. The page is shown below.
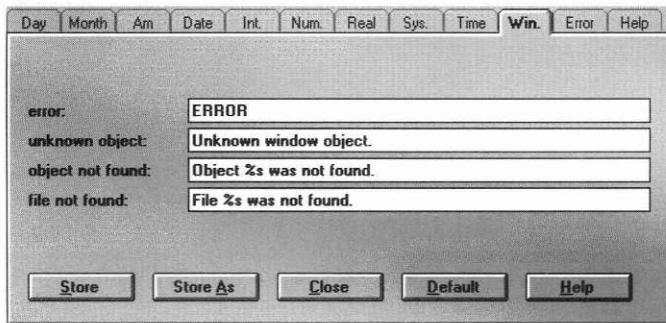
error. International word for error if no other language information is available.

unknown object. String used when loading a persistent window to indicate that an unknown type of object has been located in the **.DAT** file.

object not found. String used when loading a persistent window to indicate that an object could not be located in the **.DAT** file.

file not found. String used when loading a persistent window to indicate that the persistent file could not be found.

Error

Contains the strings used to display objects on an error window. The page is shown below.
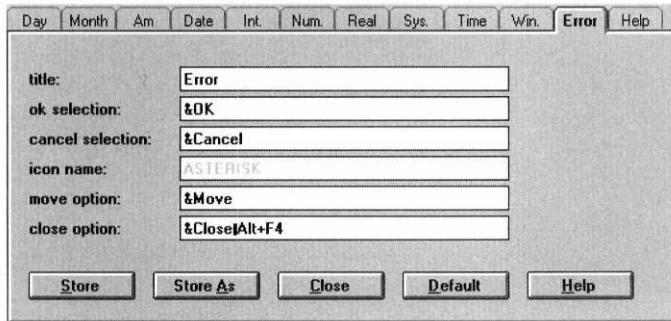
title. Default text that appears in the error window title bar if the application does not supply a new title when calling the error system.

ok selection. Text used to create the **OK** button on the error window.

cancel selection. Text used to create the **Cancel** button on the error window.

icon name. This is the name of the icon that is displayed on the error window as it appears in the **.DAT** file.

move option. Text used to create the system menu option that allows the error window to be moved.

close option. Text used to create the system menu option that allows the error window to be closed.

Help

Contains the strings used to display a help window. The page is shown below.
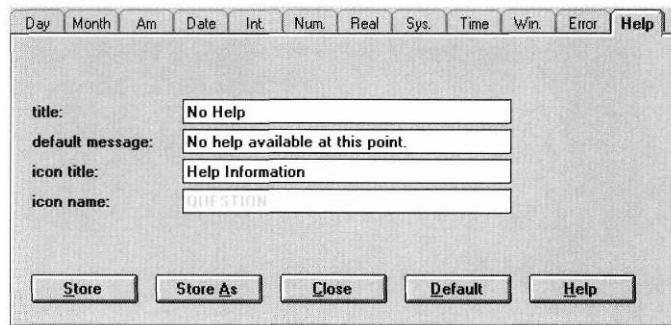
title. Default text that appears in the help window title bar if the application does not supply a help context for the object about which the end-user requested help.

default message. Default text that appears in the help window if the application does not supply a help context for the object about which the end-user requested help.

icon title. Text that appears on the help window's minimize icon.

icon name. Name of the icon that is used as the help window minimize icon as it appears in the **.DAT** file.

The locale window

The locale window allows you to modify formatting information that is used by OpenZinc for such purposes as formatting numbers, dates, or times. Each object that may require formatting data has its own notebook page on the window. Each of these pages is described in detail within this section. The pages have several common buttons, described just below.

Common buttons	*Store.* Selecting the **Store** option saves the current locale table to the current file. **If** you have not entered a name for the locale table through a **Store As...** operation, the Designer will ask for a name before storing the locale table.

Note that each time a store operation is performed, the previous contents of the locale table are completely replaced by the current information.

Store As. **Store As...** is generally used to store the current locale table under another name. (Refer to "Store" on page 415, or "Store As" on page 416, for further instructions on performing a **Store As...** operation.)

Close. Selecting this button causes the window to close without executing any changes.

Default. Selecting this button returns the entries to their default values.

Help. Additional information about creating locale tables appears when this button is selected.

Date Contains the formatting data used to format a date. The page is shown below.
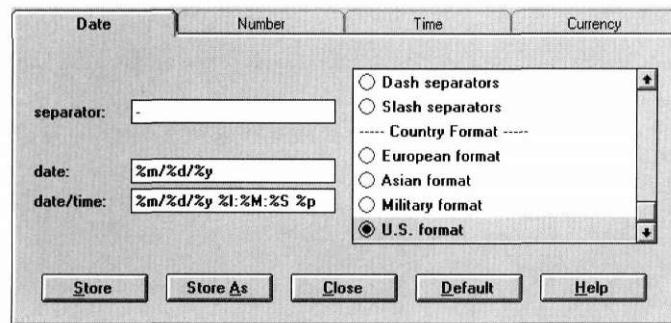
separator. Symbol used to separate the various values of the date.

date. Formatting string that specifies the order in which the date values should be arranged.

date/time. Formatting string that specifies the order in which the values should be arranged in a combined date and time value.

The options that control the presentation of the date object are listed in the field on the right half of the window. The first section presents options for formatting input. The options are:

*   *Short alphanumeric day.* Adds a shortened day-of-week text to the date.
*   *Alphanumeric day-of-week.* Adds an ASCII string day-of-week string to the date.
*   *Short alphanumeric month.* Uses a shortened alphanumeric month in the date.
*   *Alphanumeric month.* Formats the month to be displayed as an ASCII string value.
*   *Short year.* Forces the year to be displayed as a two-digit value.
*   *Format uppercase.* Converts the alphanumeric date to uppercase.
*   *Pad date with zeros.* Forces the year, month and day values to be zero filled when their values are less than 10.
*   *Fill blanks with system value.* Fills a blank date with the system date. For example, if a blank date were entered by the end user and this option were set, the date would be set to the system date.

The second section presents options for separating date values. The options are:

*   *System defaults.* Separates each date value according to the default settings for the current system.
*   *Dash separators.* Separates each date value with a dash, regardless of the default country date separator.
*   *Slash separators.* Separates each date value with a slash, regardless of the default country date separator.

The third section presents options for formatting according to country and military standards. The options are:

- *European format.* Forces the date to be displayed and interpreted in the European format (i.e., *day/month/year),* regardless of the default country information.

- *Asian format.* Forces the date to be displayed and interpreted in the Asian format (i.e., *year/month/day),* regardless of the default country information.

- *Military format.* Forces the date to be displayed and interpreted in the U.S. Military format (i.e., *day month year* where *month* is a 3 letter abbreviated word), regardless of the default country information.

- *U.S. format.* Forces the date to be displayed and interpreted in the U.S. format (i.e., *month/day/year),* regardless of the default country information.

Number

Contains the formatting data used to format a number. The page is shown below.

digit. Number of digits to display after the decimal point on currency values.

int'l digit. Number of digits to display after the decimal point on international currency values.

decimal. Symbol used as the decimal separator.

thousands. Symbol used as the thousands separator.

grouping. Number of digits that are grouped in a number.

left paren. The left symbol used when displaying a negative value using credit symbols.

right paren. The right symbol used when displaying a negative value using credit symbols.

def digits. These are the default digits used to display a number.

alt digits. These are the alternative digits used to display a number.

The *Positive numbers* group contains specifications that affect the display of positive numbers. The *Negative numbers* group contains specifications that affect the display of negative numbers. Both their fields are the same.

separator. An integer used to indicate the spacing of a negative monetary value. The following values can be used:

- 0. Causes no spaces to be placed between the currency symbol and the value.
- 1. Causes a space to be placed between the currency symbol and the value.
- 2. Causes a space to separate the currency symbol and the sign string, if they are adjacent.

currency. An integer used to position the currency symbol for negative monetary values. The following values can be used:

- 0. Causes the currency symbol to follow the value.
- 1. Causes the currency symbol to precede the value.

position. An integer used to position the positive or negative sign for a monetary value. The following values can be used:

- 0. Causes parentheses to enclose the quantity and the currency symbol.
- 1. Causes the sign string to precede the quantity and the currency symbol.
- 2. Causes the sign string to follow the quantity and the currency symbol.
- 3. Causes the sign string to precede the currency symbol.
- 4. Causes the sign string to follow the currency symbol.

sign. Symbol used to indicate that the number is positive or negative, as appropriate.

Time   Contains the formatting data used to format a time. The page is shown below.
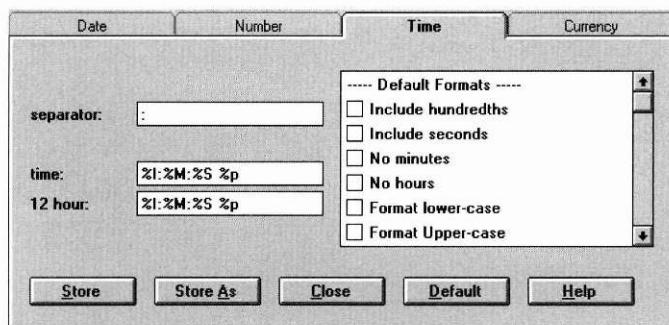
separator. Symbol used to separate the various values of the time.

time. Formatting string that specifies the order in which the time values should be arranged.

12 hour. Formatting string that specifies the order in which the time values should be arranged for a 12 hour format.

The options that control the presentation of the time object are listed in the field on the right half of the window. The first section presents options for default format. The options are:

• *Include hundredths.* Includes the hundredths value in the time. (By default the hundredths value is not included.)

• *Include seconds.* Includes the seconds value in the time. (By default the seconds value is not included.)

• *No minutes.* Does not display nor interpret a minute value for the time object.

• *No hours.* Does not display nor interpret an hour value for the time object.

• *Format lowercase.* Converts the time to lowercase.

• *Format uppercase.* Converts the time to uppercase.

OpenZinc **Designer**   **443**

- *Pad date with zeros.* Forces the hour, minute and second values to be zero filled when their values are less than 10.

- *Fill blanks with system values.* Fills a blank time with the system time. For example, if a blank ASCII time value were entered by the end user and the this option were set, the time would be set to the current system time.

The second section presents options for separating time values. The options are:

- *System defaults.* Separates each time value according to the default settings for the current system.

- *No separators.* Does not use any separator characters to delimit the time values.

- *Colon separators.* Separates each time value with a colon.

The third section presents options for formatting according to country standards. The options are:

- *12 hour.* Forces the time to be displayed and interpreted using a 12-hour clock, regardless of the default country information.

- *24 hour.* Forces the time to be displayed and interpreted using a 24-hour clock, regardless of the default country information.

Currency   Contains the formatting data used to format currency values. The page is shown below.

Date	Number	Time	**Currency**

symbol: $
int'l symbol: USD
decimal: .
thousands: ,
grouping: 3

[Store] [Store As] [Close] [Default] [Help]
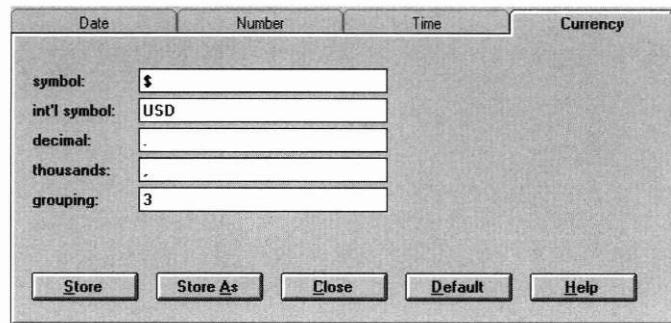
symbol. Symbol used to display the currency type.

int'l symbol. This is the international currency symbol.

decimal. Symbol used as the decimal separator.

thousands. Symbol used as the thousands separator.

grouping. Number of digits that are grouped in a number.

Defaults Editor

String Editor

The String Editor allows you to use characters within your application

which are not available from a standard keyboard. For example, if you need to translate your application into Chinese, the Chinese characters needed can be accessed from within the String Editor. Since this is accomplished using the complete Unicode character set, the String Editor is available only in Unicode mode.

To invoke the String Editor, either restore it from its minimized icon on the bottom of the screen, or in DOS or Windows press <F12> while positioned on any window field within OpenZinc Designer. The latter approach is most commonly used when you are positioned in a field that you would like to translate. Any editing done with the String Editor will be reflected in that field only.

If you are not positioned in a window field containing text when you invoke the String Editor, any editing done can be transferred to a window field by either dragging and dropping the string into a field that accepts dropping, or by copying and pasting the string.

Upon invoking the String Editor, a window similar to the following appears:

String Editor

Original: <untitled>

Current: circle

Page: 0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		I		0	@	P	`	p	I	I		·	À	Ð	à	ð
1	I	I	!	1	A	Q	a	q	I	´	¡	±	Á	Ñ	á	ñ
2	I	I	"	2	B	R	b	r	I	`	¢	²	Â	Ò	â	ò
3	I	I	#	3	C	S	c	s	I	I	£	³	Ã	Ó	ã	ó
4	I	I	$	4	D	T	d	t	I	I	¤	´	Ä	Ô	ä	ô
5	I	I	%	5	E	U	e	u	I	I	¥	µ	Å	Õ	å	õ
6	I	I		6	F	V	f	v	I	I	¦	¶	Æ	Ö	æ	ö
7	I	I	'	7	G	W	g	w	I	I	§	·	Ç	×	ç	÷
8	I	I	(	8	H	X	h	x	I	I	¨	¸	È	Ø	è	ø
9	I	I	)	9	I	Y	i	y	I	I	©	¹	É	Ù	é	ù
A	I	I	*	:	J	Z	j	z	I	I	ª	º	Ê	Ú	ê	ú
B	I	I	+	;	K	[	k	{	I	I	«	»	Ë	Û	ë	û
C	I	I	,	<	L	\	l			I	I	¬	¼	Ì	Ü	ì	ü
D	I	I	-	=	M	]	m	}	I	I		½	Í	Ý	í	ý
E	I	I	.	>	N	^	n	~	I	I	®	¾	Î	Þ	î	þ
F	I	I	/	?	O	_	o	I	I	I	¯	¿	Ï	ß	ï	ÿ

OK   Cancel   Help
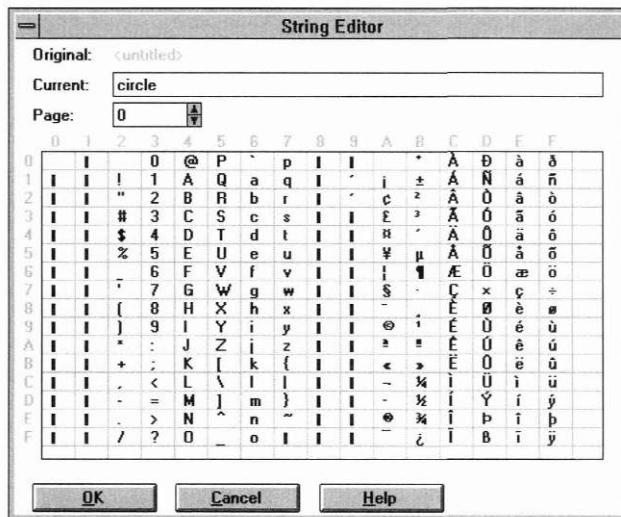
Original This field displays the text that is to be edited. If you invoked the String Editor while positioned in a field with text, the field's text will appear at the **Original** prompt. Otherwise, no text is displayed.

Current The characters selected from the table appear in this field, in the order that they are selected.

Page This spin control field designates the currently displayed Unicode page. It actually displays the upper byte of the Unicode value, or the first two digits of the page number. To change pages, either enter the desired page number, or click on the spinner arrows to increment or decrement a page at a time. The change in pages is reflected immediately in the character table.

Character table This field displays the current page of the character table. To select a character, click on it with the mouse and it will appear at the **Current** prompt. Since not all characters can be rendered in all operating systems, some cells may be blank or display a filler character, depending on the system.

OK Selecting this button causes the string displayed at the **Current** prompt to be saved. If you invoked the editor while positioned on a field, the text of that field will be replaced with the new string.

Cancel	Selecting this button causes the window to close without executing any changes.
Help	Additional information about using the String Editor appears when this button is selected.

String Editor

section four

OpenZinc Designer

appendices

Building the Designer

The Designer is made up of many components, at different levels of abstraction. Some components depend on the existence of other components if they are to function properly, or even at all. Therefore, it is important to build the compiler properly. This appendix discusses the components and their interdependencies on a general level. It does not discuss the details of compiling an application. For details on using for your compiler and environment, see "Appendix A—Compiler Considerations" in the *Getting Started With OpenZinc Programming* manual.

The Designer components

The Designer is a flexible and extensible programming tool. More components may become available after the printing of this manual. The **UPGRADE.TXT** file will list new components as they become available.

The Designer has several components at various levels of abstraction. The Level 1 component, **SERVICE.LIB,** is the most abstract. This component keeps track of the services available in the application and provides those services to components that need and request them. The services are implemented by the other components and include such capabilities as file selection, image editing, etc.

Level 2 components include **STORAGE.LIB, DIRECT.LIB,** and **STREDIT.LIB.** These components, which depend on Level 1 components, must be built after Level 1 components have been built. **STORAGE.LIB** provides persistent object functionality. **DIRECT.LIB** provides file selection, or directory services, capability. **STREDIT.LIB** provides string editing functionality for either 8-bit character mode or 16-bit Unicode mode, depending on the mode of the Designer.

Level 3 components provide the services that are the most visible to the user. In fact, these components can exist as either libraries or as executables. Level 3 components must be built after Level 2 components have been built. The Level 3 components include:

- Image Editor. This component allows you to import, export, create, and modify bitmap, icon, and mouse cursor images.

- Window Editor. This component allows you to import, export, create, and modify windows.

- Help Editor. This component allows you to import, export, create, and modify help contexts.

- Message Editor. This component allows you to import, export, create, and modify message tables. Message tables allow you to remove strings from your code and load them at run-time.

- Internationalization Defaults Editor. This component allows you to import, export, create, and modify language and locale data used for internationalizing applications.

- File Editor. This component allows you to import, export, create, and modify files in different character set formats. For example, you can use this editor to translate a file from Shift-JIS to Unicode.

Compiling   the   components

DOS, Windows, OS/2, Motif, Curses, NEXTSTEP

Each Designer component resides in a subdirectory of **/OpenZinc/DESIGN.** If you wish to build a Level 3 executable, you must go into the desired sub-directory and build that component explicitly as an executable. Building that component, as we mentioned earlier, requires that all Level 1 and Level 2 components already exist.

Makefiles for each environment are provided for each component. These makefiles do not attempt to build other components upon which the component being built depends; they assume that the components exist. If the required components do not exist or cannot be located, you will typically get a compile or link error. If this happens, check that any required components have been built and are in a location accessible by the compiler or linker. (The makefiles usually copy the headers for each component to the **/OpenZinc/INCLUDE** directory and the **<component>.LIB** to the **/OpenZinc/LIB/<compiler version>** directory.)

Most commonly, the Designer will simply be built all at once and its services used only within the context of the Designer. To accommodate this need, there are makefiles for all environments except Macintosh in the **/OpenZinc/DESIGN** directory. (See the section below for details on the Macintosh Designer.) The makefiles build the various components in the proper order producing a single Designer executable.

Macintosh

Since the Macintosh allows only one menu bar per application, OpenZinc Designer on the Macintosh is made up of several applications: Help Editor, Message Editor, Image Editor, I18N Defaults Editor, File Editor, and Window Editor. All these applications are found in the Bin folder. Normally, you won't need to build OpenZinc's Designer, since the installation includes all the Designer applications. But if you should be required to build the Designer, follow these steps:

1. Each piece of the Designer is located in its own folder within the Design folder and has a project file with which to build the piece. Before building a Designer application, you must build the component libraries which that application depends on. All of the Designer applications depend on these libraries, each within its respective folder: **Service, Direct, Storage,** and **StrEdit.** Make aliases of their respective header files and put these aliases into the **SCCPP700 Include** folder: **SERVICE.HPP, DIRECT.HPP, STORAGE.HPP,** and **STREDIT.HPP** Then build these four libraries and copy them into the **SCCPP700 Libraries** folder.

2. Each of the Designer applications also depends on its own component libraries, so build them and copy them into the **SCCPP700 Libraries** folder: **Help, Message, Imagel, Image2,118N, File, Windowl, Window2,** and **Window3.**

3. Each of the Designer applications has a project file that will build the application within its corresponding folder. You may now build any of the Designer applications with these project files. Copy the newly built applications into the **BIN** folder to use them.

GNU Free Documentation License
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
functional and useful document "free" in the sense of freedom: to
assure everyone the effective freedom to copy and redistribute it,
with or without modifying it, either commercially or noncommercially.
Secondarily, this License preserves for the author and publisher a way
to get credit for their work, while not being considered responsible
for modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense. It
complements the GNU General Public License, which is a copyleft
license designed for free software.

We have designed this License in order to use it for manuals for free
software, because free software needs free documentation: a free
program should come with manuals providing the same freedoms that the
software does. But this License is not limited to software manuals;
it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book. We recommend this License
principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that
contains a notice placed by the copyright holder saying it can be
distributed under the terms of this License. Such a notice grants a
world-wide, royalty-free license, unlimited in duration, to use that
work under the conditions stated herein. The "Document", below,
refers to any such manual or work. Any member of the public is a
licensee, and is addressed as "you". You accept the license if you
copy, modify or distribute the work in a way requiring permission
under copyright law.

A "Modified Version" of the Document means any work containing the
Document or a portion of it, either copied verbatim, or with
modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of
the Document that deals exclusively with the relationship of the
publishers or authors of the Document to the Document's overall
subject (or to related matters) and contains nothing that could fall
directly within that overall subject. (Thus, if the Document is in
part a textbook of mathematics, a Secondary Section may not explain
any mathematics.) The relationship could be a matter of historical

connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a

section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which
states that this License applies to the Document. These Warranty
Disclaimers are considered to be included by reference in this
License, but only as regards disclaiming warranties: any other
implication that these Warranty Disclaimers may have is void and has
no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either
commercially or noncommercially, provided that this License, the
copyright notices, and the license notice saying this License applies
to the Document are reproduced in all copies, and that you add no
other conditions whatsoever to those of this License. You may not use
technical measures to obstruct or control the reading or further
copying of the copies you make or distribute. However, you may accept
compensation in exchange for copies. If you distribute a large enough
number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and
you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have
printed covers) of the Document, numbering more than 100, and the
Document's license notice requires Cover Texts, you must enclose the
copies in covers that carry, clearly and legibly, all these Cover
Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on
the back cover. Both covers must also clearly and legibly identify
you as the publisher of these copies. The front cover must present
the full title with all words of the title equally prominent and
visible. You may add other material on the covers in addition.
Copying with changes limited to the covers, as long as they preserve
the title of the Document and satisfy these conditions, can be treated
as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit
legibly, you should put the first ones listed (as many as fit
reasonably) on the actual cover, and continue the rest onto adjacent
pages.

If you publish or distribute Opaque copies of the Document numbering
more than 100, you must either include a machine-readable Transparent
copy along with each Opaque copy, or state in or with each Opaque copy
a computer-network location from which the general network-using
public has access to download using public-standard network protocols
a complete Transparent copy of the Document, free of added material.
If you use the latter option, you must take reasonably prudent steps,
when you begin distribution of Opaque copies in quantity, to ensure
that this Transparent copy will remain thus accessible at the stated
location until at least one year after the last time you distribute an
Opaque copy (directly or through your agents or retailers) of that
edition to the public.

It is requested, but not required, that you contact the authors of the
Document well before redistributing any large number of copies, to
give them a chance to provide you with an updated version of the
Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under
the conditions of sections 2 and 3 above, provided that you release
the Modified Version under precisely this License, with the Modified
Version filling the role of the Document, thus licensing distribution
and modification of the Modified Version to whoever possesses a copy
of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct
   from that of the Document, and from those of previous versions
   (which should, if there were any, be listed in the History section
   of the Document). You may use the same title as a previous version
   if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities
   responsible for authorship of the modifications in the Modified
   Version, together with at least five of the principal authors of the
   Document (all of its principal authors, if it has fewer than five),
   unless they release you from this requirement.
C. State on the Title page the name of the publisher of the
   Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications
   adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
   giving the public permission to use the Modified Version under the
   terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
   and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add
   to it an item stating at least the title, year, new authors, and
   publisher of the Modified Version as given on the Title Page. If
   there is no section Entitled "History" in the Document, create one
   stating the title, year, authors, and publisher of the Document as
   given on its Title Page, then add an item describing the Modified
   Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
   public access to a Transparent copy of the Document, and likewise
   the network locations given in the Document for previous versions
   it was based on. These may be placed in the "History" section.
   You may omit a network location for a work that was published at
   least four years before the Document itself, or if the original
   publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications",
   Preserve the Title of the section, and preserve in the section all
   the substance and tone of each of the contributor acknowledgements
   and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
   unaltered in their text and in their titles. Section numbers

or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
   or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity. If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this
License, under the terms defined in section 4 above for modified
versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and
list them all as Invariant Sections of your combined work in its
license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and
multiple identical Invariant Sections may be replaced with a single
copy. If there are multiple Invariant Sections with the same name but
different contents, make the title of each such section unique by
adding at the end of it, in parentheses, the name of the original
author or publisher of that section if known, or else a unique number.
Make the same adjustment to the section titles in the list of
Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"
in the various original documents, forming one section Entitled
"History"; likewise combine any sections Entitled "Acknowledgements",
and any sections Entitled "Dedications". You must delete all sections

Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other
documents released under this License, and replace the individual
copies of this License in the various documents with a single copy
that is included in the collection, provided that you follow the rules
of this License for verbatim copying of each of the documents in all
other respects.

You may extract a single document from such a collection, and
distribute it individually under this License, provided you insert a
copy of this License into the extracted document, and follow this
License in all other respects regarding verbatim copying of that
document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate
and independent documents or works, in or on a volume of a storage or
distribution medium, is called an "aggregate" if the copyright
resulting from the compilation is not used to limit the legal rights
of the compilation's users beyond what the individual works permit.
When the Document is included in an aggregate, this License does not
apply to the other works in the aggregate which are not themselves
derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these
copies of the Document, then if the Document is less than one half of
the entire aggregate, the Document's Cover Texts may be placed on
covers that bracket the Document within the aggregate, or the
electronic equivalent of covers if the Document is in electronic form.
Otherwise they must appear on printed covers that bracket the whole
aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may
distribute translations of the Document under the terms of section 4.
Replacing Invariant Sections with translations requires special
permission from their copyright holders, but you may include
translations of some or all Invariant Sections in addition to the
original versions of these Invariant Sections. You may include a
translation of this License, and all the license notices in the
Document, and any Warranty Disclaimers, provided that you also include
the original English version of this License and the original versions
of those notices and disclaimers. In case of a disagreement between
the translation and the original version of this License or a notice
or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements",
"Dedications", or "History", the requirement (section 4) to Preserve
its Title (section 1) will typically require changing the actual

title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document
except as expressly provided under this License. Any attempt
otherwise to copy, modify, sublicense, or distribute it is void, and
will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license
from a particular copyright holder is reinstated (a) provisionally,
unless and until the copyright holder explicitly and finally
terminates your license, and (b) permanently, if the copyright holder
fails to notify you of the violation by some reasonable means prior to
60 days after the cessation.

Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License. If your rights have been terminated and not permanently
reinstated, receipt of a copy of some or all of the same material does
not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the
GNU Free Documentation License from time to time. Such new versions
will be similar in spirit to the present version, but may differ in
detail to address new problems or concerns. See
http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number.
If the Document specifies that a particular numbered version of this
License "or any later version" applies to it, you have the option of
following the terms and conditions either of that specified version or
of any later version that has been published (not as a draft) by the
Free Software Foundation. If the Document does not specify a version
number of this License, you may choose any version ever published (not
as a draft) by the Free Software Foundation. If the Document
specifies that a proxy can decide which future versions of this
License can be used, that proxy's public statement of acceptance of a
version permanently authorizes you to choose that version for the
Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any
World Wide Web server that publishes copyrightable works and also
provides prominent facilities for anybody to edit those works. A

public wiki that anybody can edit is an example of such a server. A
"Massive Multiauthor Collaboration" (or "MMC") contained in the site
means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0
license published by Creative Commons Corporation, a not-for-profit
corporation with a principal place of business in San Francisco,
California, as well as future copyleft versions of that license
published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in
part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this
License, and if all works that were first published under this License
somewhere other than this MMC, and subsequently incorporated in whole
or
in part into the MMC, (1) had no cover texts or invariant sections, and
(2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site
under CC-BY-SA on the same site at any time before August 1, 2009,
provided the MMC is eligible for relicensing.