Getting Started

with Zinc Programming

Zinc® Application Framework™
Version 5

Zinc Software Incorporated
Pleasant Grove, Utah

NOTICE

DISCLAIMER

TRADEMARKS

LICENSE AGREEMENTS

ACKNOWLEGEMENTS

The ChartFolio framework used by ZafChart is licensed software ©1994-97 DPC Technology Corporation. The XPM library used by ZafImage on Motif is licensed software ©1989-95 GROUPE BULL. The MetaWINDOW graphics primitives used by ZafDisplay on DOS is licensed software ©1988-96 Metagraphics, Inc.

This manual was generated December 23, 1997.

Contacting Zinc

Worldwide

Sales: info@zinc.com, sales@zinc.com
Technical Support: support@zinc.com
Training and Consulting: services@zinc.com
Web: http://www.zinc.com/
Ftp: ftp://ftp.zinc.com/
CompuServe: GO ZINC

North America

Zinc Software Incorporated
405 South 100 East
Pleasant Grove, Utah 84062 USA
Tel: 1-801-785-8900
Sales: 1-800-638-8665
Support: 1-801-785-8998
Fax: 1-801-785-8996

Zinc Software Services, Inc.
42627 Garfield, Suite 214
Clinton Township, Michigan 48038 USA
Tel: 1-810-228-4900
Fax: 1-810-228-6633

Europe

Zinc Software (UK) Ltd.
106-108 Powis Street
London, SE18 6LU United Kingdom
Tel: +44 (0)181 855-9918
Fax: +44 (0)181 316-2211
BBS: +44 (0)181 317-2310
Email: europe@zinc.com

Table of Contents

Quick Start

Getting Started

Congratulations on your selection of Zinc® Application Framework™ (ZAF), the most powerful cross-platform internationalized application framework available.

What Is Zinc Application Framework?

ZAF is a collection of C++ class libraries with source code, a visual interface design tool called Zinc Designer™, example programs and more. ZAF is the easiest and most elegant C++ user interface API ever developed.

Zinc Application Framework allows a single code base to support multiple platforms, including:

- Microsoft Windows
- X/Motif
- MS-DOS
- Apple Macintosh
- IBM OS/2

Many derivative operating systems are indirectly supported as well. For example, Zinc's MS-DOS support ports easily to embedded and real-time operating systems such as P-SOS while ZAF's X/Motif support ports easily to virtually any Unix or real-time OS supporting X/Motif 1.2 or later. Consult current ZAF 5 readme files for detailed information on tested and certified operating systems.

Zinc Software is well-known for sophisticated internationalization (i18n) technology. Using this technology, ZAF supports virtually any single-, double-, or mixed-byte language worldwide (subject to operating system limitations). Zinc supports ISO-8859-1 and Unicode character encoding standards to provide portable i18n. In addition, ZAF 5 supports any locale (date, time and number formatting).

Using Zinc's i18n features, a single code base may support a completely international application. For example, a single ZAF executable might simultaneously support English, European languages, Japanese and Chinese.

How Does it Work?

Zinc Application Framework defines an abstract user-interface API that is independent of any operating system. This API is then mapped onto native functionality of each operating system to provide a portable access method on each environment. This technique, known as "layering," allows ZAF to be small, fast, and true to the visual and interactive nuances of each operating sys-

tem. Applications developed with ZAF are native, and therefore look and feel like other applications developed using native tools on each OS.

When ZAF defines functionality that is not native to an operating system, ZAF provides the functionality directly. In this way a “superset” of native functionality is assured without the overhead of thick “emulation” APIs.

How Do I Use ZAF?

Zinc Application Framework is written entirely in C++. As such, it requires that programmers be familiar with basic C++ concepts such as inheritance and derivation. While many C type hooks are supplied in ZAF, a knowledge of C++ is essential.

ZAF is an advanced programming tool. It provides a high level of flexibility, extensibility, and scalability to expert users. At the same time, however, ZAF is designed to be easy to use.

Zinc Designer, an interactive visual design tool, is the starting point for most ZAF applications. Using Zinc Designer, a developer lays out the windows, dialogs, and user interface objects that make up an application.

Each object may be customized using “property sheet” editors. ZAF objects contain rich functionality including context-sensitive help, tool tips (pop-up help), color and font selection, bitmap support and more. All this functionality may be accessed and specified without code.

Once an application has been “defined” using Zinc Designer, source code may be generated. This source code can be immediately compiled to test the basic functionality of an application. More sophisticated functionality, including application flow control, may be added at the source code level.

The next section of Getting Started will walk you through simple application scenarios to demonstrate the simplicity and power of Zinc Application Framework. For more complex, real-world applications, study the example programs supplied with ZAF 5.

Hello World 1

Building a Simple Application

The best way to learn ZAF is to use it. Let's begin by building the simple "Hello World" program found in the "example/hello" directory. This program creates a simple window using straight code (without the use of the visual design tool, Zinc Designer). An example using Zinc Designer will follow.

In this chapter we'll be referring to Microsoft Windows. Detailed information on building ZAF programs for each environment is included in the Installation Guide and should be consulted before continuing.

To build the Hello1 application for 32-bit Microsoft Windows, change to the directory containing the source code and type:

```
zmake win32
```

This command invokes "ZMake," a Zinc-supplied make utility and uses a custom make file "zmake.mak." Any make utility and compiler may be used with ZAF, but ZMake is recommended since it is completely compiler and linker independent. If you are using Motif or another platform your make utility may be different.

Now, run the Hello1 application.

The Hello1 application utilizes the basic elements of a ZAF application. Hello1 presents a simple window with appropriate decorations such as a title bar and a border, and a prompt that says "Hello World!" (Note: since this simple example has no nice exit functionality, you'll need to use the system button or ALT-F4 to close it.)

Here is the source to the example. A detailed description of the code will follow:

```
// COPYRIGHT © 1997.  All Rights Reserved. - HELLO1.CPP
// Zinc Software Incorporated.  Pleasant Grove, Utah  USA
// May be freely copied, used and distributed.

#include <zaf.hpp>

int ZafApplication::Main(void)
{
  // Needed for linkers that don't automatically look for
  // unresolved references to main() or WinMain() inside
  // of libraries.
  // (Either main() or WinMain() is found in a ZAF library.)
  LinkMain();
```

```
  // Create a window with generic objects (border, maximize
  // button, minimize button, system button, and title).
  ZafWindow *helloWindow = new ZafWindow(0, 0, 30, 3);
  helloWindow->AddGenericObjects(new ZafStringData("Hello
    Window"));

  // Attach a prompt with the "hello world" text.
  // (The optional ZAF_ITEXT macro guarantees
  // Unicode compatibility.)
  helloWindow->Add(new ZafPrompt(2, 1, 0, ZAF_ITEXT("Hello
    World!")));

  // Center the window on the main monitor.
  zafWindowManager->Center(helloWindow);

  // Attach the window to the window manager
  // (make it appear on the screen).
  zafWindowManager->Add(helloWindow);

  // Process events.
  // (This function passes events from the event manager to the
  // window manager until an S_EXIT is received or no more
  // windows are attached to the window manager.)
  Control();

  // Return an exit code to the OS.
  return (0);
}
```

Let's walk through the ten lines of functional code in detail:

`#include <zaf.hpp>`

The first "real" line of code includes the header file zaf.hpp. This file in turn includes all the header files that define the classes you'll need to use ZAF. All ZAF applications should begin with this #include.

`int ZafApplication::Main(void)`

This example program contains a single method used by all ZAF applications: ZafApplication::Main(). Since every C++ application requires a main() function (or WinMain() in Microsoft Windows), the ZAF libraries automatically include a main() or WinMain() function for you. In your own code, you'll create ZafApplication::Main() (or let Zinc Designer generate it for you) and let ZAF handle the platform specific main() or WinMain().

The ZafApplication class handles many initialization tasks automatically. For example, the following components are initialized prior to ZafApplication::Main() being called:

- ZafErrorSystem (an error handler) is instantiated
- ZafHelpTips (a “pop-up” help device) is instantiated
- ZafI18nData (the core internationalization class) is instantiated and initialized

```
LinkMain();
```

Our ZafApplication::Main() first calls LinkMain(). LinkMain() is a stub method defined in the ZAF libraries along with main() or WinMain(). It is called to assist linkers that don't look for main() or WinMain() in libraries. Some linkers don't require calling LinkMain(), and others will report link errors without it.

```
ZafWindow *helloWindow = new ZafWindow(0, 0, 30, 3);
```

Next we create a new instance of the ZafWindow class. The new window's top left corner is placed at the screen position (0, 0), which is at the top left corner of the screen. The window's width is 30 cells, and its height is 3 cells. A “cell” is basically the average width of a dialog font character, and the height of a string field. (Note: some Motif window managers may override exact window positioning based on user preferences.)

```
helloWindow->AddGenericObjects(new ZafStringData("Hello
    Window"));
```

We want all the normal decorations on the window such as title bar and border, so we call the AddGenericObjects() method. The title “Hello Window” will be used in the window's title bar.

```
helloWindow->Add(new ZafPrompt(2, 1, 0, ZAF_ITEXT("Hello
    World!")));
```

The client area of the window will contain a single prompt object, so we create a new ZafPrompt instance. The prompt will be placed within the client region of the window 2 cells from the left and 1 cell from the top. Passing in a zero for the width causes the prompt to calculate its own width. The prompt will display the text “Hello World!” (Note that the text is passed to the optional “ZAF_ITEXT()” macro to allow automatic conversion of 8-bit characters to 16-bit Unicode characters if the application is built in Unicode mode.)

```
zafWindowManager->Center(helloWindow);
```

We've decided that the window belongs in the center of the screen, so we call the window manager's Center() method to automatically center it for us. To prevent visible movement on the screen we perform the centering prior to displaying the window.

```
zafWindowManager->Add(helloWindow);
```

Next, we add the window to the window manager, which has the effect of displaying it.

```
Control();
```

Like most modern user interfaces, Zinc Application Framework is event-driven. To start the ZAF event system and allow the user to interact with our application, we must now call the Control() method. Control() gets events and causes them to flow through the ZAF system where they will ultimately arrive at the correct object for processing.

Examples of events include a mouse click or a keystroke. These events are passed to the object under the mouse pointer (click), or the object with focus (keystroke) for processing. Control() continues processing events until it receives an S_EXIT event, or until there are no windows on the window manager to which it can pass events.

```
return (0);
```

Finally, our code returns a zero meaning that the application had no errors. (Usually, a C++ application returns -1 if an error occurred.)

Clearly, this is a very simple example, meant to get you started programming with ZAF quickly. You will also want to try using the Zinc Designer for rapid visual development of user interface elements. The next chapter shows this method.

Hello World 2

In the previous chapter, we built a simple application strictly with source code. This technique works well for small applications or for maximum customization and control. For most applications there is a better way.

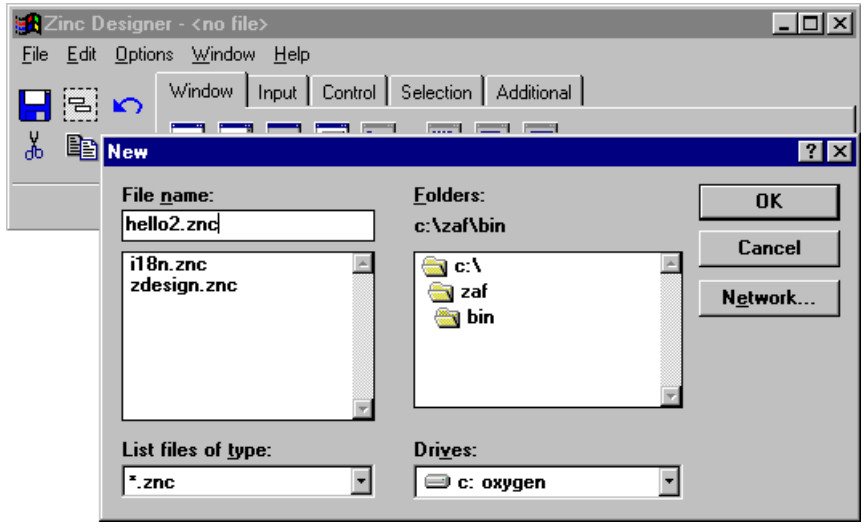
ZAF includes Zinc Designer, an interactive visual design tool, to greatly simplify the task of building a user interface. To build the same application we built in the previous chapter requires no hand-written source code, for example. Hello2, found in the "example/hello" directory, demonstrates this alternative method in which Zinc Designer is used to generate a persistent object data file, "hello2.znc," and to generate the source code necessary to access the object data file at run time.

Using Zinc Designer

Since this is our first experience using Zinc Designer, we'll take things slowly and explain everything in detail. Later tutorials will assume much of the knowledge gained in this chapter.

Start Zinc Designer and create a new file

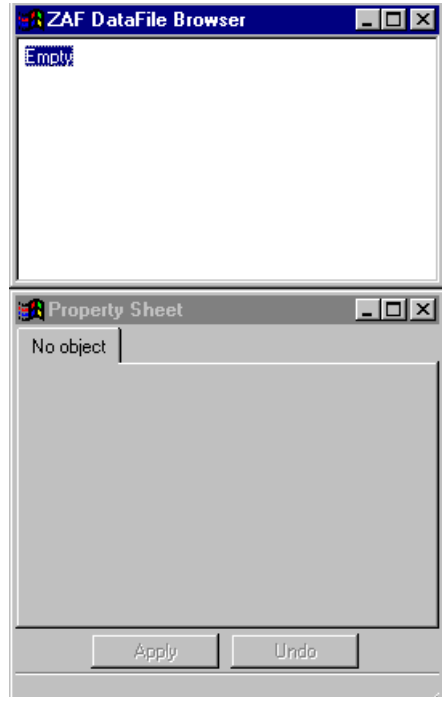
1. To recreate this application from scratch, open the Zinc Designer while in a temporary directory—"work" for example. We don't want to overwrite the Hello2 application shipped with ZAF.
2. In the "File" menu, select "New." Use "hello2.znc" for the file name.



When you exit Zinc Designer (later), this file name will be stored on the main window's "File" menu for easy access in the future. The "File" menu will list the five most recently used files.

Examining the file browser and property sheet

Now that we have a data file to edit, the data file browser and property sheet windows are open. These two windows are most important when using Zinc Designer, but both are empty for now.

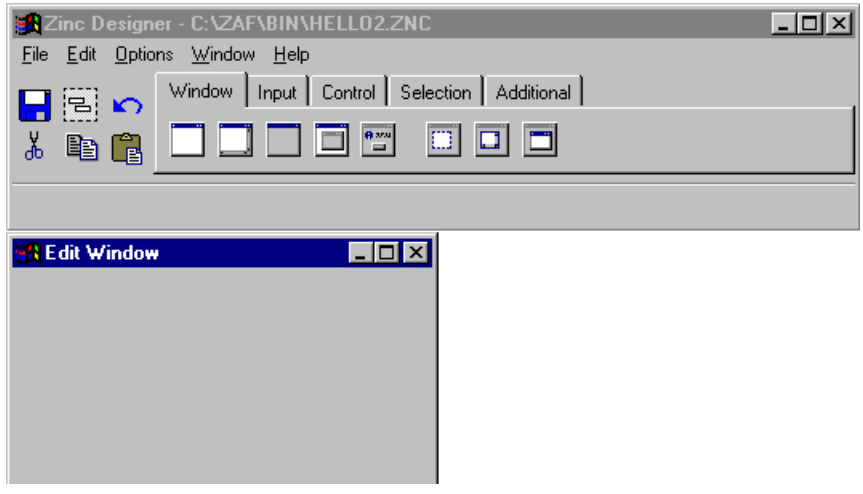


The "Browser" window will display a hierarchy of objects contained in the data file we're using. These objects may be edited by double-clicking on them in the browser. (Note: In some cases this is the only way to edit an object, so keep it in mind.)

The bottom "Property Sheet" window will display all of the properties supported by the current "edit object." These properties are organized both by function and object. As you change object properties you'll make your changes on this window and click "Apply" to save the change. If you are not happy with a change, a one-level undo capability will let you recover from your last "Apply" operation.

Create a window

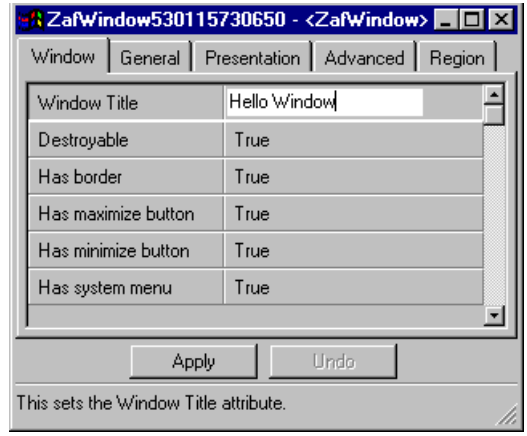
3. Now create a window by selecting the ZafWindow button in the main Designer window. The ZafWindow button is the left-most button on the "Window" page of the toolbar notebook. A new window will now appear. The window may be resized and moved as desired.



Note that the window is placed on the screen immediately after clicking the "Window" button. The first five buttons on the "Window" toolbar page are offset from the others because they share this behavior. All other controls in Zinc Designer must be selected and then *placed* in a specific position on a parent control.

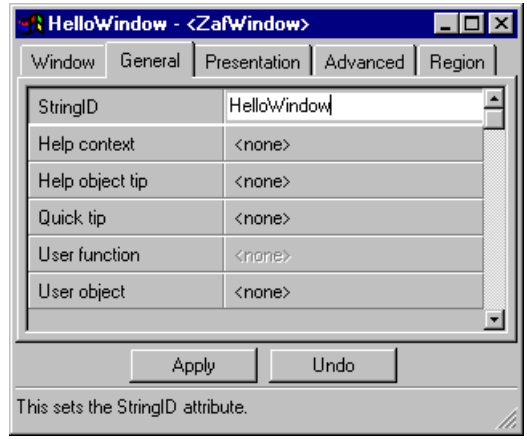
Modify window properties

4. Change the title bar text on the window by selecting the "Window" page of the property sheet. Click the "Window Title" property and replace the default title bar text with "Hello Window." Click the "Apply" button to cause the change to immediately take effect on the window.



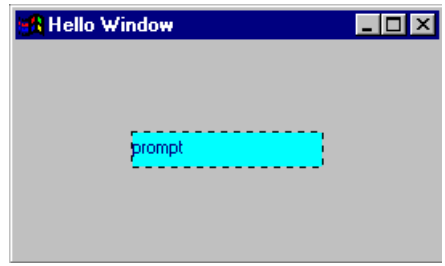
All changes except palette changes (colors and fonts) take effect immediately so you can evaluate the change. If you don't like the change, click the "Undo" button to undo the change.

5. Since we intend to directly access this window at run time (by loading it by name from the data file), we must know its unique identifier, or StringID. To check or change the StringID, select the "StringID" property in the "General" page of the property sheet. To make the StringID easy to remember later, change it to "HelloWindow" (no spaces) and select the "Apply" button again. (Note: we could have changed both the title and the StringID, then selected the "Apply" button just once.)



Add a prompt object

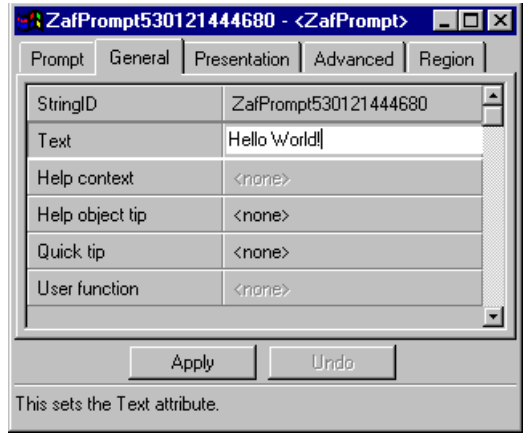
6. Now add a prompt to the window by selecting the ZafPrompt button in the main Designer window. The ZafPrompt button is the left-most button on the "Additional" page of the notebook.



After selecting the ZafPrompt button, click the mouse in the "Hello Window" where the new prompt is to appear. This process is called "placing" the object. After placing the control the mouse returns to normal operation and may be used to select other controls on our edit window. The prompt may now be moved around on the window and sized as desired. (Tip: to rapidly place several objects of the same type you may click the right mouse button to reset the "place" object.)

Modify object properties

7. Now, using the property sheet, change the text of the prompt by selecting the "Text" property on the "General" property sheet page. Change the text to "Hello World!" as in our first application. Select the "Apply" button on the property sheet and watch the change take effect.



You may wish to experiment with some of the other properties as well. Try the "Quick tip" property, for example.

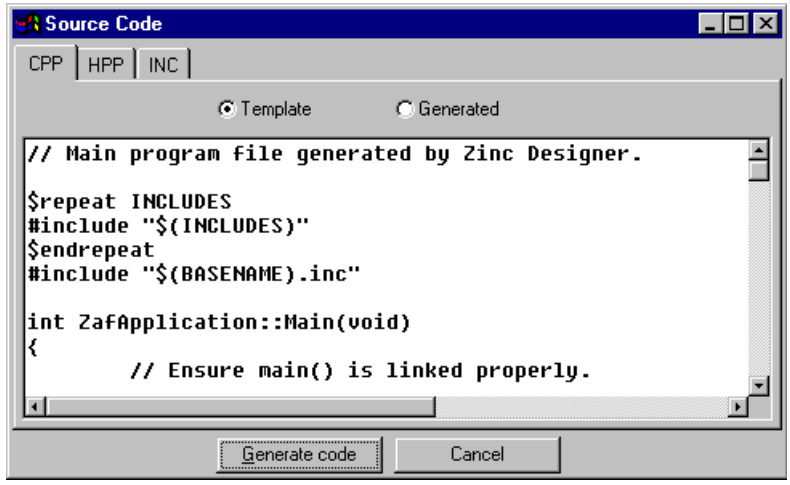
Test the user interface using "Test Mode"

8. Now, let's test the "completed" application. Select "Test Open Windows" from the "Options" menu. All the Zinc Designer windows disappear and only our edit window is left. In this mode Zinc Designer allows the ZAF libraries to take over—the controls now appear and behave exactly as they will in the completed,

compiled application. When finished testing, select the "End Test Mode" button in the lower right of the display, or close your application window.

Generate source code

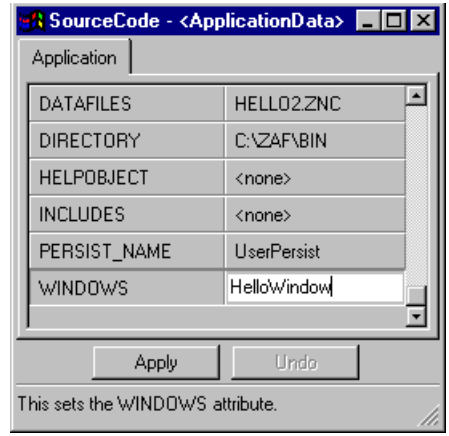
9. Next, select "Code Generation" from the "Options" menu. During this process three source files will be generated by Zinc Designer in addition to the main object data file "hello2.znc." The main code generation window (initially showing the main "CPP" template) is displayed.



Notice that the notebook has three tabs. Each tab corresponds to a file that will be generated by the designer. Each file has a generation "template" that includes macros that will be used to complete the code. The macros are defined from the property sheet currently displayed.

Resolve code generation "macros"

10. Select the "WINDOWS" property. This property specifies the windows that will be loaded and presented on screen when the application starts. Enter "HelloWindow" (no spaces), the StringID we assigned earlier, and select the "Apply" button. Note that other properties were automatically defaulted properly by Zinc Designer and that the code window now displays generated code instead of the template. You may browse the generated source code and templates using the main window. As you



become more familiar with ZAF you may take this opportunity to verify the accuracy and completeness of the code generation (which may be incomplete if the "Application" property sheet is incorrect.)

11. Now select the "Generate code" button on the code generation dialog. Zinc Designer will write the three source files to disk and a message window will appear reporting that the code generation was successful.
12. Finally, select the "Save" item in the "File" menu of the main Designer window and exit the Designer.

Source Code

If you look in the current directory you'll find that the Designer has created five files—the three source files generated, plus two others. The "zdesign.cfg" file is used by the Designer to store configuration information for itself (notably the "most recently used" files list), so we can ignore it. Let's briefly review the others:

"hello2.znc" is the data file that stores the window and other objects we created in the Designer. Zinc Designer may be used to modify this data file without generating new source code. In this way many changes may be made to an application without the need to recompile!

The "hello2.cpp," "hello2.hpp," and "hello2.inc" files contain the source code the Designer generated for the application.

Building the application

"hello2.cpp" must be compiled and linked with the appropriate ZAF libraries to build the final application. To do this we first need a makefile. You may simply copy the example/hello/[makefile] we used in the previous chapter to build our new Hello2 application. Make sure that the "hello2.znc" data file is in the same directory as the application before running it, since that is where our window is stored. If you copy the make file, the make command for 32-bit Microsoft Windows will be:

```
zmake win32
```

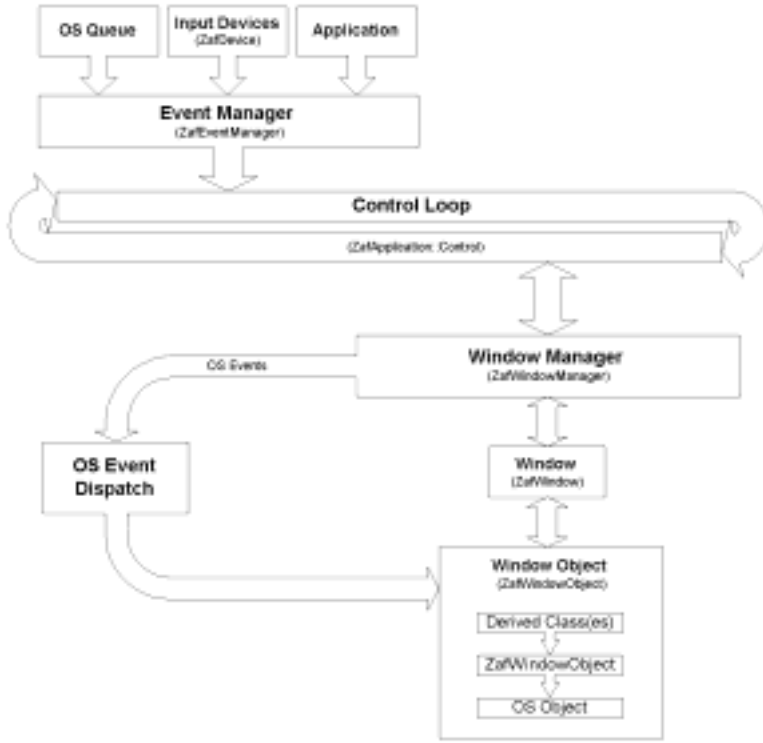
That's it! Run the application you've just built and check it out. You'll find that it runs exactly as it did in the test mode of Zinc Designer. With practice you'll soon be able to create simple applications in just a few minutes.

As you can see after using two techniques for creating applications with Zinc Application Framework, both have advantages. Zinc Designer provides the advantages of application prototyping, rapid interface development, and code generation while "hand coding" provides maximum customization and control. Most developers will combine these techniques when creating real-world applications.

In the next section of "Getting Started" we'll discuss Zinc's basic architectures and try some more complex tutorials to get you up and running quickly.

Architecture Basics

Architecture Basics—Event Flow

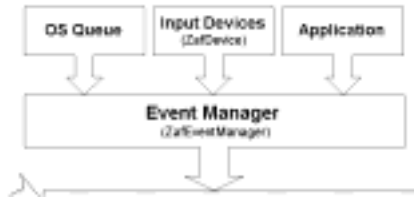


ZAF 5 General Model

Zinc Application Framework is an event-driven system. The general architecture diagram above, or "ZAF General Model," illustrates ZAF's fundamental event-driven architecture. Using this architecture, ZAF obtains events from the operating system if the OS is itself event-driven, directly from input devices, and from application code. These events are then passed through a ZAF application using a well-defined protocol. If you are already familiar with an event-driven operating system such as Microsoft Windows, Motif, Macintosh, or OS/2, you will find ZAF to be quickly understandable, and both easier to use and more powerful than your native API.

An understanding of this architecture is fundamental to programming with Zinc Application Framework, so let's look at the architecture in more detail.

Event Manager



ZAF's event-driven system begins with the event manager (the ZafEventManager class) and its supporting input devices.

For operating environments that don't provide an event-driven system (e.g. MS-DOS), the event manager polls all the attached devices such as the mouse and keyboard, and assembles events for any input information received.

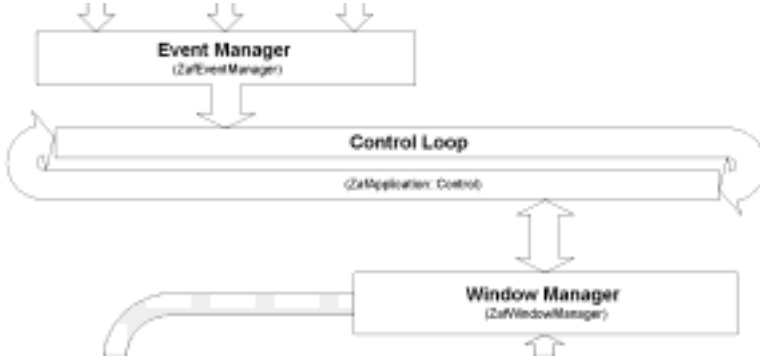
In the more common case, an event-driven operating system provides native events that are intercepted by the ZAF event manager.

As events are received, each is encapsulated in an event structure recognized by ZAF and is placed on ZAF's internal event queue for later processing.

ZAF's event manager also handles events that are generated by the ZAF libraries themselves or created and posted to the queue by the programmer, as represented by the "Application" box in the preceding diagram. ZAF programmers may also provide custom input devices (derived from the ZafDevice class) to communicate with non-standard input devices. See the Programmer's Reference manual for more information about ZafEventManager and event types.

ZafApplication ::Control

Once the event manager has acquired events the main ZAF control process regains control of the application. This process is repeated continually while your application is running.



This section of the model shows that while the event manager manages event acquisition into the event queue, ZafApplication::Control() retrieves individual events from the event manager (via the ZafEventManager::Get() method) and passes them to the window manager (via the ZafWindowManager::Event() method). On event-driven operating systems the Control() loop will "sleep" when no events are available at the operating system, thus allowing other processes to fully utilize system resources.

Event Routing

After an event is passed to the window manager (the ZafWindowManager class), the window manager determines the event's ultimate destination and proper routing, and dispatches it appropriately.

There are two basic types of events processed by ZAF and two different methods of routing these events. The two event types are, roughly, "operating system events" and "ZAF events."

OS Events

Operating system events are generated by an OS and are generally not useful to the programmer without translating them to a portable equivalent. Examples of OS events are mouse movement, redisplay (expose) messages, sizing notifications, etc.

ZAF Events

ZAF events are usually generated by the ZAF libraries or by the programmer. Keyboard events are also considered "ZAF events." These events are generally useful to the programmer in their current state.

Direct Event Routing

In order to provide most efficient event routing, the window manager often allows the native operating system to dispatch native events directly to the appropriate object. This type of event dispatch, indicated by "OS Event Dis-

patch" on the diagram, is called "Direct Event Routing" and is used for all OS events. These events are only rarely useful to the application programmer.

Top-Down Event Routing

Other events are handled by the window manager and dispatched to the appropriate window—usually the window with focus. The window in turn either handles the event if appropriate, or passes the event to the appropriate child for processing—usually the child with focus. ZAF events are commonly accessed by the programmer for application control and response. Top-Down routing allows them to be handled hierarchically—at any level of the user interface.

Exceptions to these event routing rules are made only when requested by the application programmer.

Event Handling

Ultimately each event is received by a window object's Event() method where it is processed. This section of the General Model shows that whether the OS or ZAF dispatches the event, a window object eventually receives it—usually the window object with focus.

The window object handles the event using a hierarchy of Event() methods. The first Event() method called belongs to the most-derived class indicated by "Derived Class(es)" on the diagram. This class may be a ZAF library object, or a programmer derivation.

If the derived class does not handle the event, it is passed to its base class' Event() method for handling. This process may continue until the ultimate base class, ZafWindowObject, receives the event and either processes it or hands it off to the native operating system object for handling. (Remember that ZAF utilizes a "layered" user interface implementation where most ZAF objects have corresponding operating system objects "underneath" them that can handle many operations natively.)

Derivation

ZAF offers two primary methods for customizing event response. The first method (described in the preceding section) is to derive from a ZAF class and overload its Event() method. There the programmer may process any desired events and pass the rest to a base class where the library's default handling can take over. The programmer may also directly call another object's Event() method if appropriate.

Callbacks

The second custom handling method relies on a less object oriented technique—the callback function. A ZAF user function is a C type callback

function that automatically receives a small subset of events if assigned to a ZAF user interface object. This event handling method is suitable for trivial operations and does not require derivation.

Using either event handling method, Zinc's architecture affords both flexibility and extensibility.

Event Mapping

The ZAF General Model processes both native and portable events. To achieve portability, the programmer must therefore translate or "map" native events to portable equivalents prior to interpreting them in an application. ZAF provides the LogicalEvent() method for this purpose.

ZAF provides operating system independence by defining a large set of portable events. LogicalEvent() returns a *context sensitive* mapping of native OS events to portable ZAF events. To accomplish this task, each ZAF class contains a unique table of event mappings that allows objects to translate native events in a specific way for each class. LogicalEvent() also converts event data using similar context sensitivity. For example, mouse events contain pointer coordinates that are converted relative to the top-left corner of the object, and keyboard events contain character data that must be converted relative to the current international character mode (ISO or Unicode).

Event mapping may seem complicated at first glance, but is actually trivial for the application programmer. A simple call to LogicalEvent() prior to processing each event will yield a standardized result across all platforms. (The ZAF libraries do not automatically call LogicalEvent() since they are optimized for maximum performance in each operating system environment and are capable of interpreting native OS events directly.)

The next chapter, Event Window, builds on the concepts discussed in this chapter. It derives a basic "event window" to handle custom user events. Study "Event Window" and other event examples in the ZAF 5 distribution to fully understand ZAF event handling.

Event Window

To experience the ZAF Event Flow Architecture in use, let's create a simple application and watch how it works. "Event Window" will be a simple program that demonstrates the trapping of events. It will have a single window with a pull-down menu that sends user-defined events to the window. These user-defined events will then be trapped to change the background color of the window.

This tutorial builds on the experience gained in the "Hello World 1" and "Hello World 2" tutorials. "Event Window" starts simple with plenty of detail and becomes more advanced as it progresses.

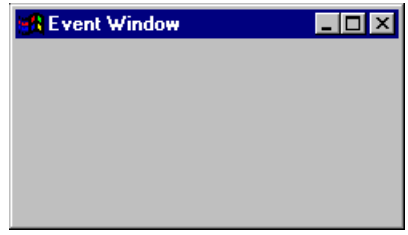
A completed version of the application can be found in "example/event," but we'll create it from scratch in a temporary directory to gain a better understanding of the concepts involved. Before continuing, you may wish to compile and run Zinc's version to get a feel for the end product.

Part One—Using Zinc Designer

1. To start, first create a new directory—"work" for example. We don't want to overwrite the example program shipped with ZAF 5. Eventually this directory will contain source code, header files, a designer data file, a make file and an executable.

Start Zinc Designer and create a window

2. Next, launch Zinc Designer from your temporary directory. We'll be creating a new data file with a simple derived window and a pull-down menu.
3. Create a new data file called "event1.znc." Select "File, New" from the menu.
4. Create a new ZafWindow and move and size it as desired. (Click on the first button in the "Window" page of the toolbar notebook.)



Customize the window

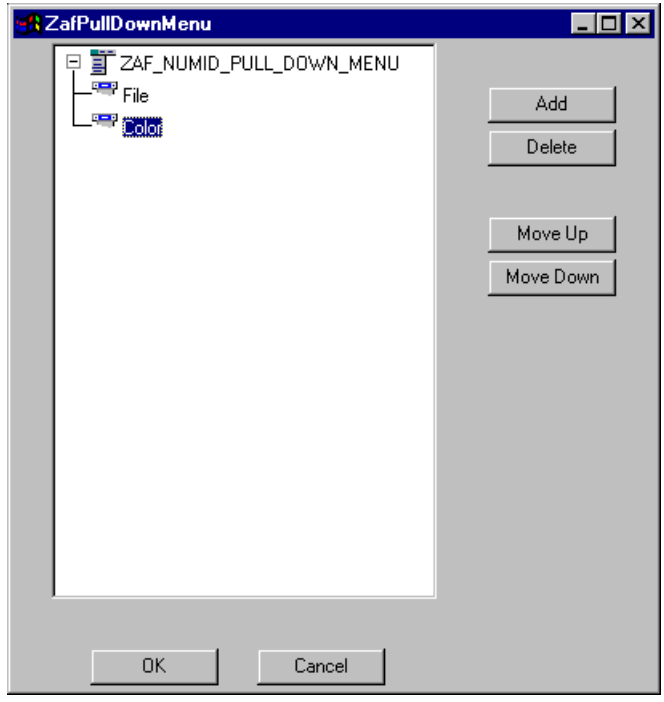
5. Change the window's title to "Event Window." (Use the "Title" property on the "Window" page of the property sheet and select "Apply." Refer to the "Hello World 1" example chapter for a description of property sheet usage.)
6. Change the window's StringID to "EventWindow" (no spaces). We'll use this StringID to refer to the window later. (StringID is on the "General" property sheet page.)
7. Since we'll be deriving a window to handle custom events, we need to specify the derived class name. Later, Zinc Designer will generate code

that loads our derived window using this class name and its StringID. (Class Name is on the “Advanced” property sheet page.)

Add a pull-down menu and pull-down items

8. Now, place a new ZafPullDownMenu on the window. (The third button on the toolbar’s “Control” notebook tab.)

9. Once the pull-down menu is in place, we must invoke the menu editor to modify it and add our custom menu structure. To do this, select the data file browser window and locate the pull-down menu. (You’ll find the pull-down menu in the following location: ZafWindow, EventWindow, ZAF_NUMID_PULL_DOWN_MENU). Now, double-click the menu item to invoke the menu editor.



Add pull-down items

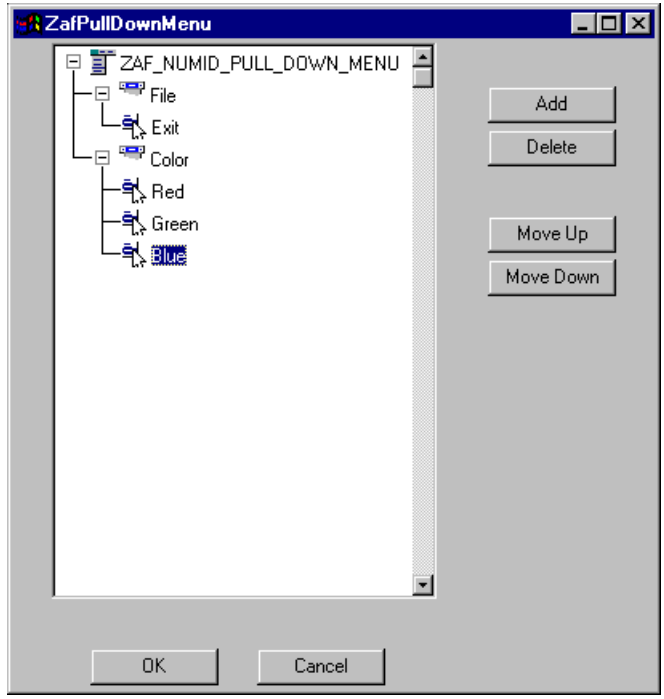
10. Select the “Add” button to add a second ZafPullDownItem to the pull-down menu (the first was automatically added to the menu when it was created). Then, select the first pull-down item in the “ZafPullDownMenu” edit window and using the property sheet change its text to “File.” Select the property sheet “Apply” button to save the change.

11. Next, select the second pull-down item in the “ZafPullDownMenu” edit window and with the property sheet change its text to “Color.”

Add pop-up menu items to hook our custom functionality

12. Now select the “File” menu option in the menu editor and click “Add” again. Notice that a new sub-item was added to the File menu. Edit this new item’s text (using the property sheet) and change it to “Exit.”

13. Using the same technique we used to add the “Exit” item, add three sub-items to the “Color” menu. We want these to read “Red”, “Green”, and “Blue.” When you finish, the menu editor should look similar to the following picture.



Add functionality to the pop-up items

14. Obviously, the “File, Exit” menu item will be used to exit the application. ZAF includes built-in functionality for adding an “exit” trigger to a menu item. To set the exit behavior, change its pop-up item type to “Exit.” Select the property sheet “Apply” button to set the change.

15. Unlike the “Exit” item, the “Red,” “Green,” and “Blue” menu items can’t take advantage of automatic pop-up types to invoke their actions since they invoke unique application functionality. Instead, we’ll cause these items to send programmer-defined events that our code can trap to change the window color. To do this, set the “Send message” property for each of the pop-up items to “true”. “Send message” causes the item to put an event on the ZAF event queue whenever this menu item is selected.

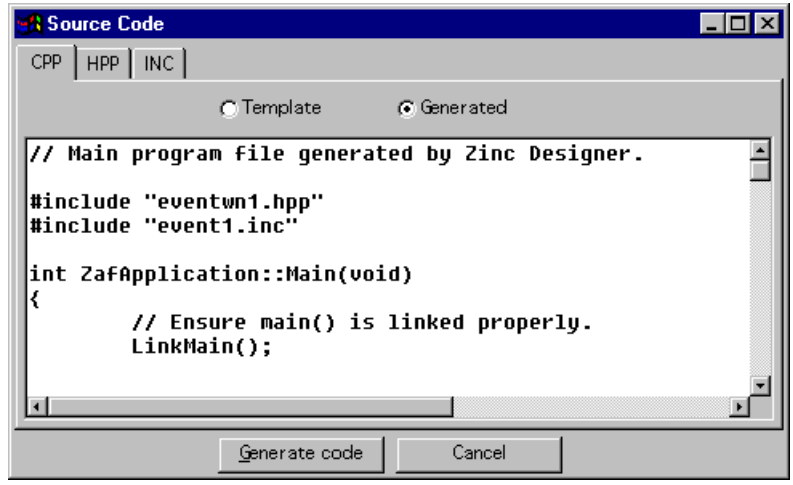
Since ZAF reserves event values above 10,000 for programmer use, we'll start with that value. Set "10000" for the "Red" item's "Value" property, "10001" for the "Green" item's "Value" property, and "10002" for the "Blue" item's "Value" property. Be sure to click "Apply" after each change! Later we'll define these event constants in our header file.

16. Select the "OK" button in the "ZafPullDownMenu" edit window to dismiss the menu editor and finalize the menu changes.

Generate source code

17. With our user interface defined we are ready to generate source code and continue developing our application outside Zinc Designer. To generate code select "Code Generation" from the designer's "Options" menu. You'll see the source code window containing the template used to generate code, and a custom property sheet page used to define the macro symbols used by the code generator.

18. Activate the property sheet. In the property sheet, change "INCLUDES" to "eventwn1.hpp". This is the name of the header file we'll create once we're finished in Zinc Designer. Change the "WINDOWS" property to "EventWindow"—the StringID of the derived window we just created. Finally, select "Apply" to save the macros and apply them to the source code template. You'll see generated source code appear in the source code window:



19. Finally, select "Generate code" to save our generated source code to disk. Save the persistent object data file we've created by using the "File" menu's "Save" option, and exit Zinc Designer.

Congratulations! You're almost done and you've performed some sophisticated tasks in Zinc Designer. Take a break and get ready to do some "real" programming.

Part Two—Source Code

In Part One, Zinc Designer created the following four files:

File	Purpose
`event1.cpp`	Source code to the main process of our application. This code initializes ZAF, loads the window we designed, starts the main control loop, and shuts down gracefully when we're done. "Hello World 1" discusses this code in detail.
`event1.hpp`	Main header file for our application. This code defines a derived persistence class used to access our data file.
`event1.inc`	Static tables containing information used when accessing the Zinc data file. This file is "#include"d by event1.cpp.
`event1.znc`	The Zinc persistent object data file containing our actual user interface definition.

As we continue building "Event Window" we're going to add to the source code created by Zinc Designer. The most critical addition is to define our derived "EventWindow" class.

Let's create new header and source files using the name we specified for "INCLUDES" in the data file: "eventwn1.hpp" and "eventwn1.cpp". Completed versions of these files are listed below with detailed discussions following.

Header file

```
// COPYRIGHT (C) 1997.  All Rights Reserved. - EVENTWN1.HPP
// Zinc Software Incorporated.  Pleasant Grove, Utah  USA
// May be freely copied, used and distributed.

#include <zaf.hpp>

const ZafEventType RED_BACKGROUND   = 10000;
const ZafEventType GREEN_BACKGROUND = 10001;
const ZafEventType BLUE_BACKGROUND  = 10002;

class EventWindow : public ZafWindow
{
```

```
public:
  // --- General members ---
  virtual ~EventWindow(void) {}
  virtual ZafEventType Event(const ZafEventStruct &event);

  // --- Persistent members ---
  EventWindow(const ZafIChar *name, ZafObjectPersistence
    &persist);
};
```

Header file walk-through

`#include <zaf.hpp>`

The header file "zaf.hpp" includes all the header files necessary for defining classes used in a ZAF application. Every ZAF application must include it. (Actually, the code generated by Zinc Designer automatically includes this header, but since we need it earlier in the compile process we'll include it here as well.)

```
const ZafEventType RED_BACKGROUND   = 10000;
const ZafEventType GREEN_BACKGROUND = 10001;
const ZafEventType BLUE_BACKGROUND  = 10002;
```

These constants will allow us easy access to the three user-defined events we need to change colors, and make our code more readable. They must match the values we specified in Zinc Designer.

```
class EventWindow : public ZafWindow
{
public:
  // --- General members ---
  virtual ~EventWindow(void) {}
  virtual ZafEventType Event(const ZafEventStruct &event);

  // --- Persistent members ---
  EventWindow(const ZafIChar *name, ZafObjectPersistence
    &persist);
};
```

Our "EventWindow" class is derived from ZafWindow since that is the base window type we created in Zinc Designer. ZafWindow will give us all the functionality of a normal window and we'll then add a bit more of our own. In our declaration we add three pieces required by our application:

- A virtual destructor is defined. This does nothing—it is even empty—and is not strictly necessary since the compiler will automatically generate one for us if we

forget. Still, it is good coding practice to supply a destructor for all objects and we have done so here.

- An Event() method is defined to intercept the three user-defined events for our color changes.
- A persistent constructor is defined to load our derived window from the data file.

Source file

```
// COPYRIGHT (C) 1997.  All Rights Reserved. - EVENTWN1.CPP
// Zinc Software Incorporated.  Pleasant Grove, Utah  USA
// May be freely copied, used and distributed.

#include "eventwn1.hpp"

EventWindow::EventWindow(const ZafIChar *name,
    ZafObjectPersistence &persist) : ZafWindow(name, persist)
{}

ZafEventType EventWindow::Event(const ZafEventStruct &event)
{
  ZafEventType ccode = LogicalEvent(event);
  switch (ccode)
  {
  case RED_BACKGROUND:
    SetBackgroundColor(ZAF_CLR_RED);
    break;

  case GREEN_BACKGROUND:
    SetBackgroundColor(ZAF_CLR_GREEN);
    break;

  case BLUE_BACKGROUND:
    SetBackgroundColor(ZAF_CLR_BLUE);
    break;

  default:
    // Pass the event to the base class for processing.
    ccode = ZafWindow::Event(event);
    break;
  }

  return (ccode);
}
```

Source file walk-through

`#include "eventwn1.hpp"`

First, we include our header file “eventwn1.hpp” to provide definitions for our constants and derived class.

```
EventWindow::EventWindow(const ZafIChar *name,
    ZafObjectPersistence &persist) : ZafWindow(name, persist)
{}
```

Next, our simple persistent constructor simply calls the base class (ZafWindow) persistent constructor to load our window from the data file. We needn't do anything else here, since we don't define any additional data for the EventWindow class.

```
ZafEventType EventWindow::Event(const ZafEventStruct &event)
{
  ZafEventType ccode = LogicalEvent(event);
  switch (ccode)
  {
  case RED_BACKGROUND:
    SetBackgroundColor(ZAF_CLR_RED);
    break;

  ...

  default:
    // Pass the event to the base class for processing.
    ccode = ZafWindow::Event(event);
    break;
  }

  return (ccode);
}
```

The Event() method first calls LogicalEvent() to translate all incoming events to their portable equivalents, then we trap each event that interests us: RED_BACKGROUND, GREEN_BACKGROUND, and BLUE_BACKGROUND. When we find a matching event, each case calls SetBackgroundColor() to change the background color of the window. In the "default" case, all other events are passed to the base class Event() method for normal processing.

That's it! We've added some simple source code to a Zinc Designer project to create an interesting, working application. In the process we've demonstrated how easy it is to utilize ZAF's sophisticated event architecture.

We're now ready to build an executable and test it.

Finishing Up

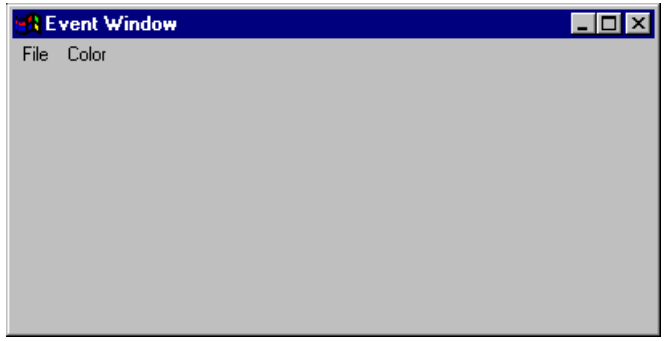
Create a make file and build the application

Our final step is to create a make file. To build one on your own look at the make files included with the ZAF example programs and refer to the documen-

tation for your compiler's make utility, or Zinc's recommended "zmake" if you prefer.

To speed the process along, just copy the file "zmake.mak" (for Windows) and any support files needed (such as "wtest16.def" for 16-bit Microsoft Windows). This will link together the necessary ZAF libraries, along with "event1.cpp," generated by Zinc Designer, and our "eventwn1.cpp."

Now, build the application using zmake or your own make utility and enjoy! You're on your way to becoming a Zinc expert.



Additional Study

For more practice working with events, look at the expansion to this application, "Event Window 2" found in example/event. This example program demonstrates the trapping of system events coming from the mouse and keyboard, shows derived child objects, sends events using ZafEventManager::Put(), uses C type user functions and more.

In the next chapter we'll look at another of ZAF's fundamental architectures—Model / View.

Suggested Study

With an understanding of Zinc's basic architecture, you are ready to begin programming with ZAF. For additional information, see the Programmer's Reference manual and closely examine the example programs provided with ZAF.

Zinc's reference manual is unlike any reference you have used before. It contains a great deal of example code, and provides architectural and practical discussions in an interesting format. In short, the ZAF reference manual is readable!

Before delving deeply into large scale projects using ZAF, you may wish to study the following information:

- ZafWindowObject—the most important base class in ZAF.
- ZafWindow—a critical base class.
- Appendix: Property Matrices—quick reference to the capabilities and limitations of ZAF user interface objects.
- Appendix: Event Definitions—essential information about event types and possibilities.
- Example programs—carefully selected programs that demonstrate important programming techniques. Careful study of these examples will provide the best start to programming with ZAF.

Appendices

Zinc Application Framework Software License Agreement

Professional Version

DO NOT INSTALL OR USE THE ZINC APPLICATION FRAMEWORK SOFTWARE UNTIL YOU HAVE READ AND ACCEPTED THIS LICENSE AGREEMENT. BY INSTALLING OR USING THE SOFTWARE YOU ACCEPT THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THIS LICENSE AGREEMENT: (A) YOU MUST NOT INSTALL OR USE THE SOFTWARE, AND (B) YOU MAY RETURN THE SOFTWARE, INCLUDING ALL PACKAGING, MEDIA, AND DOCUMENTATION, FOR A REFUND, PROVIDED THAT THE RETURN IS MADE WITHIN TEN DAYS OF THE DATE OF PURCHASE OF THIS LICENSE.

Zinc Application Framework, Version 5
Professional Version Software License Agreement

1. ***Developer***. "Developer" is the person who accepts and agrees to this Agreement. If Developer is an employee of a company and intends to use the Software within the scope of his/her employment or to develop Applications for the company, then the "Developer" includes the company, and acceptance of this Agreement is also made on behalf of the company.

2. ***Software***. "Software" shall mean the Zinc Application Framework computer programs provided with this Agreement. The Software consists of "Shared Code" and one or more "Platform Modules." The license certificate provided with this Agreement "designates the Platform Modules which are licensed to Developer. These designated Platform Modules are referred to as the "Licensed Platform Modules." Notwithstanding anything in this Agreement to the contrary, the Software does not include, and Developer has no right to install, use or copy, any Platform Module not designated in the license certificate. If Developer desires to use additional Platform Modules, a license for such additional Platform Modules must first be purchased from Zinc or its authorized reseller. Additional Platform Modules for which a license is purchased shall be governed by this Agreement as Licensed Platform Modules and shall be deemed part of the Software. "Shared Code" means all Software other than Platform Modules. Developer acknowledges that Zinc Software Incorporated ("Zinc") and its licensor(s) own the copyrights and other intellectual property in and to the Software.

3. ***Documentation***. "Documentation" means the online documentation and printed documentation, if any, provided to Developer in connection with this Agreement. Whenever the context reasonably permits, any reference in this Agreement to Software shall also apply to Documentation.

4. ***Applications***. "Applications" mean computer program applications other than competitive computer programs. "Competitive computer programs" means computer programs that are competitive with, or that can be used in lieu of, the Software.

5. ***License***. Subject to the other provisions of this Agreement, Zinc grants to Developer a nonexclusive, nontransferable license (the "License"): (a) to use the Software to develop Applications (as defined above), and (b) to exercise "distribution rights" under Section 6 below. Each Licensed Platform Module may be used by a single user only (i.e., the Licensed Platform Module is restricted to the user) on a single computer running under the operating system designated on the license certificate for the Licensed Platform Module. Developer may not use a Licensed Platform Module on more than one computer at any given time unless an additional license for each additional computer is purchased. The Shared Code may be used by a single user only (i.e., the Shared Code is restricted to the same user) on any computer on which at least one of the users's Licensed Platform Module(s) is used as permitted above. Licenses for additional users may be purchased from Zinc at their then-current prices. Rights not expressly granted are reserved by Zinc.

6. ***Distribution Rights***. The Software includes "Linkable Routines," "Distributable Files," and non-distributable files. Linkable Routines consist of the object code routines in the Software libraries (e.g., *.LIB, lib*.a). Distributable Files consist of those "run-time" files identified in the Software documentation as required during execution of Developer's program applications. The License includes the following distribution rights: (a) authorization for Developer to incorporate Linkable Routines into Applications developed by Developer and to distribute them as part of such Applications to Developer's customers, provided that the Linkable Routines have been incorporated in such a way that they cannot be used apart from the Applications, (b) authorization for Developer to distribute Distributable Files to Developer's customers as part of the Applications developed by Developer, and (c) authorization for Developer to license Developer's customers to use such Linkable Routines and Distributable Files as part of the Applications, but not separate from such Applications.

7. ***Distribution Guidelines***. Except for the Linkable Routines and Distributable Files, no portion of the Software may be distributed or transferred by Developer. The Linkable Routines and Distributable Files may not be distributed as part of any computer program other than Applications as defined in Section 4 without the express written permission of Zinc. Developer must include an appropriate Zinc copyright notice, in accordance with guidelines published by Zinc, on all copies of Developer's Applications in which Linkable Routines are incorporated or with which Distributable Files are distributed. Customers who receive any Linkable Routines or Distributable Files under Section 6 may not use any of the Linkable Routines or Distributable Files independent of Developer's Applications or use any Linkable Routines or Distributable Files for any development pur-

poses. Developer shall ensure that its Application license agreements with customers are consistent with this Agreement.

8. ***Copies***. Developer may make copies of the Software provided that any such copy: (a) is created as an essential step in the utilization of the Software on a computer in accordance with the License and this Agreement, or (b) is only for archival purposes to back-up the licensed use of the the Software. Developer may also make copies of the Software to the extent reasonably needed to exercise rights under the License or this Agreement. All Zinc trademark and copyright notices must be faithfully reproduced and included on copies made by Developer. Developer may not make any other copies of the Software. The online Documentation may be printed by Developer and used by Developer, but only in connection with the licensed use of the Software.

9. ***Protection of the Software***. Except as expressly authorized in this Agreement, Developer may not: (i) disassemble, decompile or otherwise reverse engineer the Software, or (ii) create derivative works based upon the Software, or (iii) rent, lease, sublicense, distribute, transfer, copy, reproduce, or timeshare the Software, or (iv) allow any third party to access or use the Software, or (v) modify the Software (including any deletion of code from or addition of code to the Software).

10. ***Source Code***. "Licensed Source Code" shall mean that portion of the Software's source code which is provided to Developer in connection with this Agreement. The Licensed Source Code is part of the Software and is governed by this Agreement. The License includes authorization for Developer to use the Licensed Source Code to maintain and modify the Software to conform with Developer's needs in creating Applications. All modified Software shall be governed by this Agreement as Software. The Licensed Source Code may not be disclosed or distributed by Developer to any other person. Developer is not entitled to any other Software source code.

11. ***Disclaimer***. Because Zinc has no control over modifications made by Developer, it is not obligated to maintain or support modified versions of the Software and no warranties are applicable to such modified versions. There is no warranty that the Software is suitable for modification and all modifications are undertaken at the risk and discretion of Developer.

12. ***Limited Warranty***.

12.a. ***Media and Documentation***. Zinc warrants that if the media or Documentation provided by Zinc are in a damaged or physically defective condition at the time that the License is purchased and if they are returned to Zinc (postage prepaid) within 90 days of the date this License is purchased, then Zinc will provide Developer with replacements at no charge.

12.b. ***Software***. Zinc warrants that if the Software fails to substantially conform to the specifications in the Software documentation or to any other Software specifications published by Zinc and if the nonconformity is reported in writing by Developer to Zinc within 90 days from the date the License is purchased, then Zinc shall either remedy the nonconformity or offer to refund the purchase price to Developer upon a return of all copies of the Software (including all packaging, media, and Documentation) to Zinc. In the event of a refund the License shall terminate.

13. ***Disclaimers and Limitations***.

13.a. ***Disclaimer of Warranties***. **ZINC MAKES NO WARRANTY, PROMISE OR REPRESENTATION NOT EXPRESSLY SET FORTH IN THIS AGREEMENT. EXCEPT AS EXPRESSLY WARRANTED HEREIN, THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. ZINC DISCLAIMS AND EXCLUDES ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ZINC DOES NOT WARRANT THAT THE SOFTWARE WILL SATISFY DEVELOPER'S REQUIREMENTS OR THAT IT IS WITHOUT DEFECT OR ERROR OR THAT THE OPERATION THEREOF WILL BE UNINTERRUPTED. THIS AGREEMENT GIVES DEVELOPER SPECIFIC LEGAL RIGHTS. DEVELOPER MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.**

13.b. ***Limitation on Liability***. **THE AGGREGATE LIABILITY OF ZINC ARISING FROM OR RELATING TO THIS AGREEMENT OR THE SOFTWARE (REGARDLESS OF THE FORM OF ACTION OR CLAIM--E.G., CONTRACT, WARRANTY, TORT, STRICT LIABILITY, MALPRACTICE, FRAUD AND/OR OTHERWISE) SHALL NOT EXCEED THE TOTAL PAYMENT MADE BY DEVELOPER TO PURCHASE THIS LICENSE. ZINC SHALL NOT IN ANY CASE BE LIABLE FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT OR PUNITIVE DAMAGES, OR FOR LOSS OF PROFIT, REVENUE, DATA, OR PROGRAMS, EVEN IF ZINC HAS BEEN ADVISED OF THE POSSIBILITY THEREOF. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY.**

13.c. ***Responsibility for Decisions***. Developer is responsible for decisions made and actions taken based on the Software. The Software is designed and intended for use by computer professionals experienced in the uses and limitations of computer software and it is Developer's responsibility to ascertain the suitability of the Software.

13.d. ***Non-Parties***. The officers, directors, employees, shareholders and representatives of Zinc are not parties to this Agreement and shall have no obligation or liability to Developer relating to this Agreement or the Software.

14. ***Sole Remedy and Allocation of Risk***. DEVELOPER'S SOLE AND EXCLUSIVE REMEDY IS SET FORTH IN THIS AGREEMENT. This Agreement defines a mutually agreed-upon allocation of risk and the License fees reflect such allocation of risk.

15. ***Governing Law***. This Agreement shall be governed by the laws of the State of Utah and the United States of America without giving effect to conflict of laws. Any litigation between the parties shall be conducted exclusively in Utah.

16. ***Entire Agreement***. This Agreement sets forth the entire understanding and agreement between the parties and may be amended only in a writing signed by both parties. No vendor, distributor, dealer, retailer, sales person or other person is authorized by Zinc to modify this Agreement or to make any warranty, representation or promise which is different than, or in addition to, the warranties, representations or promises of this Agreement.

17. ***Termination***. The License shall automatically terminate if Developer materially breaches this Agreement. Upon termination of the License, Developer shall cease all use of the Software and shall destroy all copies of the Software within the possession or control of Developer and shall return the original Software media and Documentation to Zinc.

18. ***U.S. Government Restricted Rights***. The Software has been developed entirely at private expense and is provided as "Commercial Computer Software" or "restricted computer software" with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government or U.S. Government (sub)contractor is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Zinc Software Incorporated, 405 South 100 East, Pleasant Grove, Utah 84062.

19. ***Export Laws***. Developer shall not export or distribute any Software in violation of any applicable laws or regulations, including the export laws and regulations of the United States.

20. ***Construction***. In the construction and interpretation of this Agreement, no rule of strict construction shall apply against either party.

Zinc Application Framework Software License Agreement

Personal Version

DO NOT INSTALL OR USE THE ZINC APPLICATION FRAMEWORK SOFTWARE UNTIL YOU HAVE READ AND ACCEPTED THIS LICENSE AGREEMENT. BY INSTALLING OR USING THE SOFTWARE YOU ACCEPT THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THIS LICENSE AGREEMENT YOU MUST NOT INSTALL OR USE THE SOFTWARE.

THIS PERSONAL VERSION LICENSE IS OFFERED TO DEVELOPERS WHO DESIRE TO USE THE SOFTWARE FOR PERSONAL USE ONLY. THE LICENSED DEVELOPER IS NOT REQUIRED TO PAY ANY LICENSE FEE OR ROYALTIES FOR THIS PERSONAL VERSION LICENSE. HOWEVER, TO EXERCISE RIGHTS BEYOND THIS PERSONAL VERSION LICENSE, THE DEVELOPER MUST PURCHASE A PROFESSIONAL VERSION LICENSE FROM ZINC.

Zinc Application Framework, Version 5
Personal Version Software License Agreement

1. ***Developer***. "Developer" is the individual person who accepts and agrees to this Agreement. No corporation, partnership, limited liability company or other organization or business entity may be a Developer under this Agreement. They may, however, purchase professional version licenses from Zinc Software Incorporated ("Zinc").

2. ***Software***. "Software" shall mean the Zinc Application Framework computer programs provided with this Agreement. Developer acknowledges that Zinc and its licensor(s) own the copyrights and other intellectual property in and to the Software.

3. ***Documentation***. "Documentation" means the online documentation and printed documentation, if any, provided to Developer in connection with this Agreement. Whenever the context reasonably permits, any reference in this Agreement to Software shall also apply to Documention.

4. ***Personal Applications***. "Personal Applications" mean computer program applications developed by Developer that: (a) are for use by Developer only, and not for use by, or distribution to, any employer, customer or other person, and (b) are not competitive computer programs. "Competitive computer programs" means computer programs that are competitive with, or that can be used in lieu of, the Software.

5. ***License***. Subject to the other provisions of this Agreement, Zinc grants to Developer a nonexclusive, nontransferable license (the "License"): (a) to use the Software to develop Personal Applications (as defined above), and (b) to use such Personal Applications. Rights not expressly granted are reserved by Zinc. The License does not include any right to use the Software in connection with the development of any computer program or application other than Personal Applications. In order to use the Software in connection with the development of computer program applications for use by others, Developer must first purchase a professional version license from Zinc and agree to Zinc's then-current professional version license agreement.

6. ***Linkable Routines and Distributable Files***. The Software includes "Linkable Routines," "Distributable Files," and non-distributable files. Linkable Routines consist of the object code routines in the Software libraries (e.g., *.LIB, lib*). Distributable Files consist of those "run-time" files identified in the Software documentation as required during execution of Developer's program applications. The License includes: (a) authorization for Developer to incorporate Linkable Routines into Personal Applications developed by Developer, provided that the Linkable Routines have been incorporated in such a way that they cannot be used apart from the Personal Applications, and (b) authorization for Developer to include Distributable Files as part of the Personal Applications developed by Developer, and (c) authorization for Developer to use such Linkable Routines and Distributable Files as part of the Personal Applications, but not separate from such Personal Applications. Except as provided in Section 7, Linkable Routines and Distributable Files shall not be distributed or transfered by Developer, not even as part of or with any Personal Application. To distribute Linkable Routines or Distributable Files as part of or with an application, Developer must first purchase a professional version license from Zinc and agree to Zinc's then-current professional version license agreement.

7. ***Distribution Rights***. A copy of the Software in its complete and unmodified form as provided by Zinc may be distributed or transferred by Developer to any other individual person. Such other person shall have no right to install or use the Software unless he/she accepts the same terms and conditions as are in this Agreement. Although such other person's agreement shall be identical to this Agreement, they shall be separate and independent agreements.

8. ***Copies***. Developer may make copies of the Software provided that any such copy: (a) is created as an essential step in the utilization of the Software on a computer in accordance with the License and this Agreement, or (b) is only for archival purposes to back-up the licensed use of the the Software. Developer may also make copies of the Software to the extent reasonably needed to exercise rights under the License or this Agreement (e.g., distribution rights under Section 7). All Zinc trademark and copyright notices must be faithfully reproduced and included on copies made by Developer. Developer may not make any other copies of the Software.

9. ***Protection of the Software***. Except as expressly authorized in this Agreement, Developer may not: (i) disassemble,

decompile or otherwise reverse engineer the Software, or (ii) create derivative works based upon the Software, or (iii) rent, lease, sublicense, distribute, transfer, copy, reproduce, or timeshare the Software, or (iv) allow any third party to access or use the Software, or (v) modify the Software (including any deletion of code from or addition of code to the Software).

10. ***Licensed Source Code***. "Licensed Source Code" shall mean that portion of the Software's source code which is provided to Developer in connection with this Agreement. The Licensed Source Code is part of the Software and is governed by this Agreement. The License includes authorization for Developer to use the Licensed Source Code to maintain and modify the Software to conform with Developer's needs in creating Personal Applications. All modified Software shall be governed by this Agreement as Software. The Licensed Source Code may not be disclosed or distributed by Developer to any other person except as part of a distribution or transfer of a complete and unmodified copy of the Software as provided by Zinc under Section 7. Developer is not entitled to any other Software source code.

11. ***Disclaimer***. Because Zinc has no control over modifications made by Developer, it is not obligated to maintain or support modified versions of the Software and no warranties are applicable to such modified versions. There is no warranty that the Software is suitable for modification and all modifications are undertaken at the risk and discretion of Developer.

12. ***Developer Source Code***. Developer may distribute, transfer, and disclose Developer's source code to Personal Applications, provided that no part of the Licensed Source Code (or modified versions thereof) is distributed, transferred, or disclosed.

13. ***Disclaimer of Warranties***. **ZINC MAKES NO PROMISE OR REPRESENTATION NOT EXPRESSLY SET FORTH IN THIS AGREEMENT. BECAUSE THERE IS NO LICENSE FEE OR ROYALTY, ZINC MAKES NO WARRANTY OF ANY KIND AND THE SOFTWARE IS LICENSED AND PROVIDED TO DEVELOPER STRICTLY ON AN "AS IS" BASIS. ZINC DISCLAIMS AND EXCLUDES ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ZINC DOES NOT WARRANT THAT THE SOFTWARE WILL SATISFY DEVELOPER'S REQUIREMENTS OR THAT IT IS WITHOUT DEFECT OR ERROR OR THAT THE OPERATION THEREOF WILL BE UNINTERRUPTED. THIS AGREEMENT GIVES DEVELOPER SPECIFIC LEGAL RIGHTS. DEVELOPER MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.**

14. ***Limitation on Liability***. **THE AGGREGATE LIABILITY OF ZINC ARISING FROM OR RELATING TO THIS AGREEMENT OR THE SOFTWARE (REGARDLESS OF THE FORM OF ACTION OR CLAIM--E.G., CONTRACT, WARRANTY, TORT, STRICT LIABILITY, MALPRACTICE, FRAUD AND/OR OTHERWISE) SHALL NOT EXCEED TEN DOLLARS. ZINC SHALL NOT IN ANY CASE BE LIABLE FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT OR PUNITIVE DAMAGES, OR FOR LOSS OF PROFIT, REVENUE, DATA, OR PROGRAMS, EVEN IF ZINC HAS BEEN ADVISED OF THE POSSIBILITY THEREOF. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY.**

15. ***Responsibility for Decisions***. Developer is responsible for decisions made and actions taken based on the Software. The Software is designed and intended for use by computer professionals experienced in the uses and limitations of computer software and it is Developer's responsibility to ascertain the suitability of the Software.

16. ***Non-Parties***. The officers, directors, employees, shareholders and representatives of Zinc are not parties to this Agreement and shall have no obligation or liability to Developer relating to this Agreement or the Software.

17. ***Allocation of Risk***. This Agreement defines a mutually agreed-upon allocation of risk and the License fees reflect such allocation of risk.

18. ***Governing Law***. This Agreement shall be governed by the laws of the State of Utah and the United States of America without giving effect to conflict of laws. Any litigation between the parties shall be conducted exclusively in Utah.

19. ***Entire Agreement***. This Agreement sets forth the entire understanding and agreement between the parties and may be amended only in a writing signed by both parties. No vendor, distributor, dealer, retailer, sales person or other person is authorized by Zinc to modify this Agreement or to make any warranty, representation or promise which is different than, or in addition to, the warranties, representations or promises of this Agreement.

20. ***Termination***. The License shall automatically terminate if Developer materially breaches this Agreement. Upon termination of the License, Developer shall cease all use of the Software and shall destroy all copies of the Software within the possession or control of Developer.

21. ***U.S. Government Restricted Rights***. The Software has been developed entirely at private expense and is provided as "Commercial Computer Software" or "restricted computer software" with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government or U.S. Government (sub)contractor is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Zinc Software Incorporated, 405 South 100 East, Pleasant Grove, Utah 84062.

22. ***Export Laws***. Developer shall not export or distribute any Software in violation of any applicable laws or regulations, including the export laws and regulations of the United States.

23. ***Construction***. In the construction and interpretation of this Agreement, no rule of strict construction shall apply against either party.

Index

A

B

C

D

E

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Z