Chapter 25

Graphical User Interfaces

Chapter Goals

- To learn about event-driven programming
- To learn how to use an application framework
- To implement menus and buttons and their associated actions
- To understand the concept of layout management for graphical user interface components
- To understand window repainting
- To be able to implement simple applications with graphical user interfaces

You know how to implement console programs that read input from `cin` and send output to `cout`. However, these programs are hardly typical of today's applications. Modern applications have a user interface with menus, buttons, scroll bars, and other elements. Those programs are often called graphical user interface (GUI) applications. Many programmers pronounce GUI as "gooey".

There is an essential difference between a console application and a GUI application. A console application is in complete control of the user input. The program asks the user questions in an order that is convenient for processing the input. The user must supply the responses in exactly the same order. In contrast, the user of a GUI application can click on buttons, pull down menus, and type text in

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

any order. The user is in charge of providing input, and the program must adapt to the user. For that reason, GUI applications are much more difficult to program than the console applications you have seen so far. In this chapter, you will learn how to create programs with a graphical user interface.

Chapter Contents

25.1 The wxWidgets Toolkit

GUIs can be programmed through low-level libraries specific to an operating system, or through higher-level application frameworks.

Modern operating systems provide libraries for GUI programming. However, those libraries are typically complex and hard to use. Most programmers use a toolkit that provides an object-oriented abstraction layer over the low-level graphical services. By far the most commonly used C++ GUI toolkit is MFC (Microsoft Foundation Classes). MFC is used to write Microsoft Windows programs, and it is a part of the Microsoft Visual C++ compiler.

However, in this chapter, we will use a different toolkit, called wxWidgets. The wxWidgets toolkit is conceptually very similar to MFC, but it has a number of advantages for our purposes.

- wxWidgets is freely available.
- wxWidgets runs on several platforms, not just Microsoft Windows. In particular, it runs on Linux and the Macintosh OS.
- wxWidgets works with a large number of compilers.

- wxWidgets is more transparent to the beginning programmer. MFC is tightly integrated with the Visual Studio environment, and it hides quite a bit of magic behind wizards and builder tools.
- wxWidgets is structurally very similar to MFC. Thus, the skills that you learn in this chapter transfer immediately to MFC and other GUI toolkits.

The wxWidgets framework is suitable for GUI programming with various platforms and compilers.

You can download the wxWidgets software from `www.wxWidgets.org`. The web site for this book contains detailed instructions on how to compile wxWidgets programs on a number of platforms.

The following sections contain a step-by-step guide to user interface programming, beginning with a very simple program that displays an empty window and ending up with a GUI implementation of the clock game from Chapter 22. Here are the steps:

1. Create an empty frame window—your first GUI program.
2. Add a text control that can be used to display or enter text.
3. Add menus to the top of the frame window.
4. Add event handling code that is executed when the user selects a menu item.
5. Lay out buttons and other user interface controls.
6. Paint geometric shapes inside a window.
7. Handle mouse input.
8. Use dialog boxes to obtain user input.
9. Put it all together with the clock game.

25.2 Frames

A frame is a window with the decorations provided by the windowing system.

To get started with wxWidgets, you will write a very simple program that simply puts up a *frame*, a window with the typical decorations that the windowing system provides. The decorations depend on the windowing system—Figures 1 through Figure 3 show the same frame under Linux, Macintosh OS X, and Microsoft Windows. If you look carefully, you can see that the icons for common window operations, such as minimizing and closing the window, are different. These differences are not important for GUI programming, so you shouldn't worry if the figures shown here look slightly different from your programs.

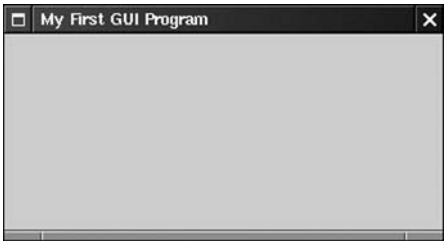

Figure 1
A Frame in Linux

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

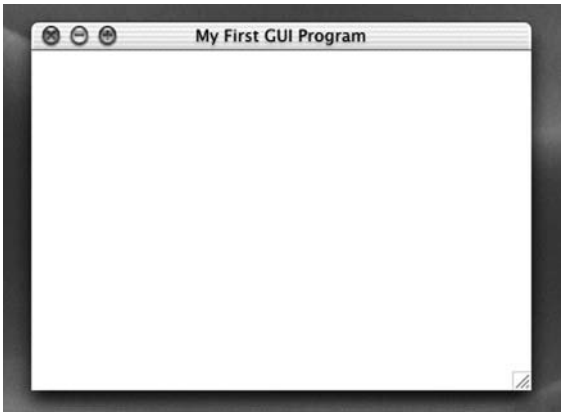

Figure 2 The Same Frame in Macintosh OS X

The program listing at the end of this section shows the program that displays a blank frame (see page 6). As you can see, the program is fairly simple. Let's walk through its features.

Your application class should be derived from the wxApp class.

To use the wxWidgets toolkit in your program, you need to include the header file `wx/wx.h`. You then define a class that contains details about the workings of your application. This class must be derived from the class `wxApp` that the wxWidgets toolkit provides.

```
class BasicApp : public wxApp
{
public:
   virtual bool OnInit();
private:
   wxFrame* frame;
};
```

Here you specify that your application is identical to the default application defined by `wxApp`, except that your application has a frame, and that you want to override the `OnInit` function.

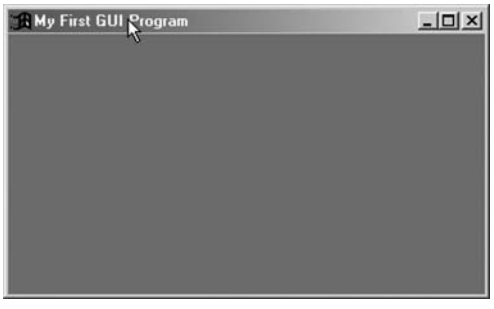

Figure 3 The Same Frame in Windows

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

In the `OnInit` function, you construct and show the frame and return `true` to indicate that the initialization was successful.

```
bool BasicApp::OnInit()
{
   frame = new wxFrame(NULL, -1, "My First GUI Program");
   frame->Show(true);
   return true;
}
```

Note the window title `"My First GUI Program"` in the `wxFrame` constructor. The other constructor parameters specify that the window has no parent window and a default window ID.

The names of the wxWidgets framework functions (such as `OnInit` and `Show`) start with an uppercase letter. That differs from the convention in the standard C++ library, but it is the convention used in MFC. You need to be careful about capitalization when overriding or calling wxWidgets functions.

Finally, note that the program listing contains the lines

```
DECLARE_APP(BasicApp)
IMPLEMENT_APP(BasicApp)
```

These are two preprocessor macros, defined in the `wx/wx.h` header file. These macros carry out some magic that the framework needs to make an application out of the `BasicApp` class. If you distribute the code for an application class into separate header and implementation files, then you need to place the `DECLARE_APP` macro into the header (`.h`) file and the `IMPLEMENT_APP` macro into the implementation (`.cpp`) file.

Our `BasicApp` class inherits a large number of functions from the `wxApp` class. One of those functions will, at the appropriate time, call the `OnInit` function. When the user closes the frame, another function of the `wxApp` class will be called to take care of necessary cleanup. All of this is entirely transparent to the programmer.

You use inheritance to describe the differences between an application framework's generic classes and the functionality required by your application.

The wxWidgets toolkit supplies several base classes (such as `wxApp`) from which programmers derive classes to specify the behavior of their application. Such a toolkit is called an *application framework*. An application framework contains classes that perform a fair amount of complex work, such as interfacing with the operating system and the window environment. However, application programmers need not know about these technical details. They must simply supply their derived classes, according to the rules of the framework.

One of the rules of the wxWidgets framework is that you must initialize the top window of an application in the `OnInit` function. It is usually quite difficult for a programmer who learns a new framework to know what exactly is required to build an application. See Productivity Hint 25.1 on page 6 for some tips.

Compiling a wxWidgets program is not as simple as compiling a console program, and the instructions differ quite a bit, depending on your compiler and platform. The web site for this book has quick-start instructions for several popular compilers and platforms. For more detailed information, turn to the wxWidgets documentation. You should set aside some time to install the toolkit and compile this simple program—see Productivity Hint 25.2 on page 7.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Of course, this program isn't terribly exciting—it just shows an empty frame. You will see in the following sections how to add menus, buttons, and text fields, and how to display graphical images in the frame.

ch25/basic.cpp

```
#include <wx/wx.h>

/**
   A basic application that shows an empty frame.
*/
class BasicApp : public wxApp
{
public:
   /**
      Constructs and shows the frame.
      @return true
   */
   virtual bool OnInit();
private:
   wxFrame* frame;
};

DECLARE_APP(BasicApp)

IMPLEMENT_APP(BasicApp)

bool BasicApp::OnInit()
{
   frame = new wxFrame(NULL, -1, "My First GUI Program");
   frame->Show(true);
   return true;
}
```

PRODUCTIVITY HINT 25.1

Learning About a New Toolkit

When you are faced with learning a new framework, you want to look for the following information:

- A tutorial that gives you step-by-step instructions for building simple applications. This chapter is such a tutorial.
- Sample applications that show techniques used in more ambitious programs. The wxWidgets distribution contains a number of sample applications.
- Documentation that describes the details of the various classes and functions. For example, you can look up the meaning of the parameters of the `wxFrame` constructor in the wxWidgets documentation.

The rules for using a framework are necessarily somewhat arbitrary. It is not necessary or possible to completely understand the details of every function call in the routine code that is

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

needed for initialization and other mundane tasks. Framework programmers engage in a fair amount of "copy and paste" from tutorials and samples. You should do the same when you work with wxWidgets. Just start with an application that is similar to the one that you want to create, and modify it.

Productivity Hint 25.2

Becoming Familiar with a Complex Tool

When you first use a complex tool, such as the wxWidgets framework, you will likely face quite a few difficulties when trying to compile your first program. Because you don't yet know the "rules of the game", you are likely to make lots of mistakes and it may appear as if success will never come. This is a particularly frustrating experience, and many beginners give up in disgust. Here are a few tips.

- *Set aside plenty of time.* This is the most important tip of all. Learning a new tool is time-consuming for beginners and professionals alike. Trying to do it in a hurry adds a tremendous amount of stress.
- *Expect mistakes.* You will make plenty of mistakes before you hit upon the right pathway. If you hope for instant success, it is very easy to become frustrated and demoralized.
- *Don't think you are stupid.* Even professional programmers find it difficult and frustrating to learn a new environment.
- *Start with an easy task.* Find an extremely simple program, preferably one that you are absolutely certain is correct. Get it to compile and run.
- *Read the error messages.* With a complex tool, lots of things can go wrong. Just saying "it didn't work" will get you nowhere. Of course, the error messages may be confusing. Be on the lookout for clues and for red herrings, just like a detective.
- *Keep a log.* You will likely try several approaches, spread out over hours or days, and observe more details than you can remember. Open your text editor, start a new file, and keep track of the commands that you tried. Paste in all error messages that you got. Include links to promising parts of the documentation.
- *Browse the documentation.* Most programs come with "readme" or installation instructions that contain tips for quick start and troubleshooting. Find them and look at them. It is usually pointless to try to understand everything, but knowing what's where can be very helpful when you get stuck.
- *Try something else.* It is extremely common for beginners to give up because they got stuck with their initial approach. You'll be amazed how often a breakthrough comes from trying some slight variation, no matter how improbable. Of course, the variation won't lead to instant success, but watching how the error messages change can give you invaluable clues.
- *Work with a friend.* It is much easier to tolerate errors and come up with creative approaches when working with someone else.
- *Ask someone who knows.* The Internet has lots of useful discussion groups where people help each other. However, nobody likes to spend time helping a lazy person, so you have

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

to do your own research first. Read through other people's problems and solutions, and formulate your question so that it is clear that you tried your best.

25.3 Adding a Text Control to the Frame

Use a `wxTextCtrl` for text input and output.

The frame of the preceding example was completely empty. In this section, you will see how to make the frame more interesting. Since you want a more interesting frame than the basic `wxFrame`, you use inheritance and define your own frame class. As you can see from Figure 4, the frame of this sample program contains a text area into which users can type any text.

Here is the definition of your derived frame class:

```
class TextFrame : public wxFrame
{
public:
   TextFrame();
private:
   wxTextCtrl* text;
};
```

The `TextFrame` constructor initializes the base class and the text control:

```
TextFrame::TextFrame()
   : wxFrame(NULL, -1, "TextFrame")
{
   text = new wxTextCtrl(this, -1, "Type some text here!",
      wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
}
```

The first construction parameter of the `wxTextCtrl` class denotes that the parent of the text control is this frame. The text control moves wherever its parent moves. (Recall that the frame has no parent—it is a top-level window.) As with the `wxFrame` constructor, `-1` denotes a default window ID. The third construction parameter is the initial contents of the text control. The next two parameters specify a default size and position, and the final parameter turns on the "multiline" style, which allows the text control to hold multiple lines of text.

Figure 4 A Frame with a Text Control

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

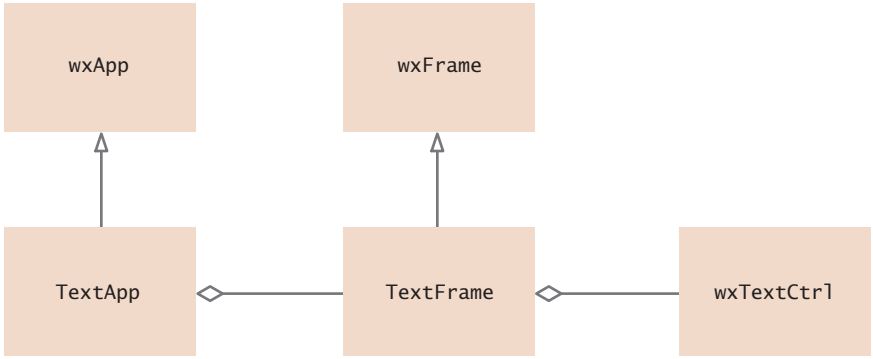

Figure 5 The Classes of the Text Program

Now you need to make a slight change to the application class. This time, you want it to show a `TextFrame`, not a `wxFrame`.

```
class TextApp : public wxApp
{
public:
   virtual bool OnInit();
private:
   TextFrame* frame;
};

bool TextApp::OnInit()
{
   frame = new TextFrame();
   frame->Show(true);
   return true;
}
```

This application uses inheritance in two places, to customize the application and to customize a frame (see Figure 5). Here is the complete program.

ch25/text.cpp

```
1  #include <wx/wx.h>
2
3  /**
4     A frame that contains a text control.
5  */
6  class TextFrame : public wxFrame
7  {
8  public:
9     /**
10       Constructs the text control.
11    */
12    TextFrame();
13 private:
14    wxTextCtrl* text;
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
15 };
16
17 /**
18    An application that shows a frame with a text control.
19 */
20 class TextApp : public wxApp
21 {
22 public:
23    /**
24       Constructs and shows the frame.
25       @return true
26    */
27    virtual bool OnInit();
28 private:
29    TextFrame* frame;
30 };
31
32 DECLARE_APP(TextApp)
33
34 IMPLEMENT_APP(TextApp)
35
36 TextFrame::TextFrame()
37    : wxFrame(NULL, -1, "TextFrame")
38 {
39    text = new wxTextCtrl(this, -1, "Type some text here!",
40       wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
41 }
42
43 bool TextApp::OnInit()
44 {
45    frame = new TextFrame();
46    frame->Show(true);
47    return true;
48 }
```

25.4 Menus

In this step, you will add a menu to your sample application (see Figure 6).

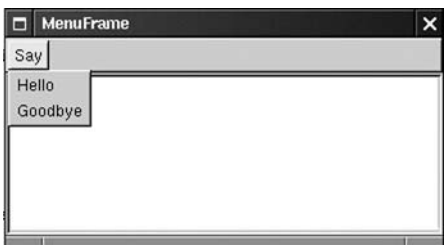

Figure 6 An Application with a Menu

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

If you look carefully at Figure 6, you will see that the frame now has a *menu bar*. The menu bar contains the names of the top-level menus. In this case, there is a single top-level menu named "Say". That menu contains two *menu item*s, "Hello" and "Goodbye".

Each menu item must have an integer ID number. The ID numbers are used to match up menu items with their actions—you will see the details in the next section. It doesn't matter what the numbers are, as long as the numbers for different actions are distinct. It is customary to give the constants names that start with `ID_`.

```
const int ID_SAY_HELLO = 1000;
const int ID_SAY_GOODBYE = 1001;
```

Now construct a menu and append the two menu items to it.

```
wxMenu* menu = new wxMenu();
menu->Append(ID_SAY_HELLO, "Hello");
menu->Append(ID_SAY_GOODBYE, "Goodbye");
```

Finally, construct a menu bar object, set it as the menu bar of the frame, and append the menu:

```
wxMenuBar* menu_bar = new wxMenuBar();
SetMenuBar(menu_bar);
menu_bar->Append(menu, "Say");
```

No further changes to the program are required. Here is the complete program listing. Compile the program and verify that the menu works!

ch25/menu.cpp

```
#include <wx/wx.h>

const int ID_SAY_HELLO = 1000;
const int ID_SAY_GOODBYE = 1001;

/**
   A frame with a simple menu and a text control.
*/
class MenuFrame : public wxFrame
{
public:
   /**
      Constructs the menu and text control.
   */
   MenuFrame();
private:
   wxTextCtrl* text;
};

/**
   An application with a frame that has a menu and text control.
*/
class MenuApp : public wxApp
{
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
25  public:
26     /**
27        Constructs and shows the frame.
28        @return true
29     */
30     virtual bool OnInit();
31  private:
32     MenuFrame* frame;
33  };
34
35  DECLARE_APP(MenuApp)
36
37  IMPLEMENT_APP(MenuApp)
38
39  MenuFrame::MenuFrame()
40     : wxFrame(NULL, -1, "MenuFrame")
41  {
42     text = new wxTextCtrl(this, -1, "",
43        wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
44
45     // Initialize menu
46     wxMenu* menu = new wxMenu();
47     menu->Append(ID_SAY_HELLO, "Hello");
48     menu->Append(ID_SAY_GOODBYE, "Goodbye");
49
50     // Add menu to menu bar
51     wxMenuBar* menu_bar = new wxMenuBar();
52     SetMenuBar(menu_bar);
53     menu_bar->Append(menu, "Say");
54  }
55
56  bool MenuApp::OnInit()
57  {
58     frame = new MenuFrame();
59     frame->Show(true);
60     return true;
61  }
```

25.5 Event Handling

GUI programs are event-driven. Event handlers are functions that are called when events occur.

In the preceding section, you saw how to attach a menu bar with menus to a frame. However, when you run the program and select a menu item, nothing happens. Now you will attach actions to the menu items.

Define a function for each action. It is customary to give these functions names that begin with `On`, such as `OnSayHello`. Here is the function for the "Hello" menu option. It appends a greeting to the text control.

```
void EventFrame::OnSayHello(wxCommandEvent& event)
{
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
   text->AppendText("Hello, World!\n");
}
```

In wxWidgets, event tables map events to functions.

Whenever the program user selects the menu option, you want this function to run. In order to do that, you need to build an *event table*. Here is an example of an event table. It routes menu events with an ID of `ID_SAY_HELLO` to the `OnSayHello` function.

```
BEGIN_EVENT_TABLE(EventFrame, wxFrame)
   EVT_MENU(ID_SAY_HELLO, EventFrame::OnSayHello)
END_EVENT_TABLE()
```

These entries are again macros, somewhat similar to the `IMPLEMENT_APP` macro that causes the application to start by constructing a particular application object. The details of capturing user interface events differ among platforms. These macros automatically produce the correct code to capture the events and call the designated functions when the events occur. An event table has the format

```
BEGIN_EVENT_TABLE(ClassName, BaseClassName)
   EVT_TYPE(parameters, function)
   ...
END_EVENT_TABLE()
```

You use different table entries for each event type (such as menu, button, or mouse events).

There are several different event types; you will encounter a couple of them in this chapter. The `MENU` event type requires a menu ID as a parameter. Other event types may require different information. The final parameter of the event macro is the name of the function that should be called when the event occurs.

The event handler functions have parameters that describe the triggering event. For example, as you have seen, the handler for the "Hello" menu item has a parameter of type `wxCommandEvent`. That particular handler function has no interest in the event description, but you must nevertheless declare the handler function with the appropriate event type. Otherwise, you will get a bewildering error message when the event macro generates code that doesn't match the rules of the framework.

You need to know which class handles a particular event. Menu events are handled by the frame that contains the menu bar. Later you will see that button events are handled by the window that contains the button.

Finally, you need to insert another macro into the definition of each class that has an event table. For example,

```
class EventFrame : public wxFrame
{
   ...
private:
   wxTextCtrl* text;
   DECLARE_EVENT_TABLE()
};
```

That macro generates the necessary data fields and function declarations for the event table.

Following is the code for the complete program. Select the "Hello" and "Goodbye" menu items and observe how they append text to the text control.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

ch25/event.cpp

```
#include <wx/wx.h>

const int ID_SAY_HELLO = 1000;
const int ID_SAY_GOODBYE = 1001;

/**
   A frame with a simple menu that adds greetings to a
   text control.
*/
class EventFrame : public wxFrame
{
public:
   /**
      Constructs the menu and text control.
   */
   EventFrame();
   /**
      Adds a "Hello, World!" message to the text control.
      @param event the event descriptor
   */
   void OnSayHello(wxCommandEvent& event);
   /**
      Adds a "Goodbye, World!" message to the text control.
      @param event the event descriptor
   */
   void OnSayGoodbye(wxCommandEvent& event);
private:
   wxTextCtrl* text;
   DECLARE_EVENT_TABLE()
};

/**
   An application to demonstrate the handling of menu events.
*/
class EventApp : public wxApp
{
public:
   /**
      Constructs and shows the frame.
      @return true
   */
   virtual bool OnInit();
private:
   EventFrame* frame;
};

DECLARE_APP(EventApp)

IMPLEMENT_APP(EventApp)

BEGIN_EVENT_TABLE(EventFrame, wxFrame)
   EVT_MENU(ID_SAY_HELLO, EventFrame::OnSayHello)
   EVT_MENU(ID_SAY_GOODBYE, EventFrame::OnSayGoodbye)
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
54  END_EVENT_TABLE()
55
56  EventFrame::EventFrame()
57     : wxFrame(NULL, -1, "EventFrame")
58  {
59     text = new wxTextCtrl(this, -1, "",
60        wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
61
62     // Initialize menu
63     wxMenu* menu = new wxMenu();
64     menu->Append(ID_SAY_HELLO, "Hello");
65     menu->Append(ID_SAY_GOODBYE, "Goodbye");
66
67     // Add menu to menu bar
68     wxMenuBar* menuBar = new wxMenuBar();
69     SetMenuBar(menuBar);
70     menuBar->Append(menu, "Say");
71  }
72
73  void EventFrame::OnSayHello(wxCommandEvent& event)
74  {
75     text->AppendText("Hello, World!\n");
76  }
77
78  void EventFrame::OnSayGoodbye(wxCommandEvent& event)
79  {
80     text->AppendText("Goodbye, World!\n");
81  }
82
83  bool EventApp::OnInit()
84  {
85     frame = new EventFrame();
86     frame->Show(true);
87     return true;
88  }
```

25.6 Layout Management

The next sample program is similar to the preceding one, except that you use buttons instead of menus to add greetings to a text control (see Figure 7).

Figure 7
An Application with Two Buttons

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Conceptually, buttons are very similar to menu items. When the user presses a button, a function is called that carries out the button action. To associate the button with its action, you use an event table. Each button has an ID, and the event table matches the ID with a function.

When you construct a button, you specify the parent window, the button ID, and the button label:

```
wxButton* hello_button = new wxButton(this,
   ID_SAY_HELLO, "Say Hello");
```

In the event table of the parent window, you use the `EVT_BUTTON` macro to specify the event handler function.

```
BEGIN_EVENT_TABLE(ButtonFrame, wxFrame)
   EVT_BUTTON(ID_SAY_HELLO, ButtonFrame::OnSayHello)
END_EVENT_TABLE()
```

The event handler has the same form as a menu event handler.

```
void ButtonFrame::OnSayHello(wxCommandEvent& event)
{
   text->AppendText("Hello, World!\n");
}
```

That's all you need to do to activate a button. There is just one additional problem—you need to make sure the buttons are placed correctly inside the frame. It turns out that placing buttons is more complex than arranging menus. Menus, after all, have a simple layout, on top of the frame. Buttons, on the other hand, can be located anywhere inside a frame.

When placing user interface elements in a window, you need to specify the layout of the components.

Some user interface toolkits supply a graphical layout tool to define the placement of buttons, text controls, and other user interface elements in a frame. Such a tool makes it simple to design a user interface with a few "drag and drop" operations. However, the resulting design tends to be fragile. If the sizes of the components change, then the components no longer line up, and someone has to run the tool again to fix the layout. Why would the component sizes change? There are two common reasons. First, if an application is translated to another language, text strings can change dramatically in length and may no longer fit. For example, a button that holds "Goodbye" may be too small to hold the German equivalent "Auf Wiedersehen". Furthermore, if an application is ported to another windowing system, the sizes of buttons, scroll bars, and other elements is likely to change.

If you only write applications for a single language and a single platform, then a "drag and drop" tool is a good solution. But for more robust layouts, you want to describe the logic behind the placement of the user interface elements. Consider for example the layout of Figure 7. We have a text control that expands to fill the entire frame, except for a horizontal row of buttons on the bottom. Those buttons don't expand. The row of buttons is centered horizontally.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

The wxWidgets framework uses sizers to specify the sizing rules so that programs can be ported to different languages and windowing environments.

In wxWidgets, you use objects of the `wxSizer` class or one of its derived classes to specify the layout of user interface elements. One subclass is `wxBoxSizer`. It defines a horizontal or vertical arrangement. For example, here is how you line up the buttons horizontally:

```
wxBoxSizer* button_sizer = new wxBoxSizer(wxHORIZONTAL);
button_sizer->Add(hello_button);
button_sizer->Add(goodbye_button);
```

You use a second sizer to place the text control on top of the button row.

```
wxBoxSizer* frame_sizer = new wxBoxSizer(wxVERTICAL);
frame_sizer->Add(text, 1, wxGROW);
frame_sizer->Add(button_sizer, 0, wxALIGN_CENTER);
```

The second parameter of the `Add` member function is a value that tells the sizer by how much to grow the component vertically. A value of `0` does not grow the component—the button row stays at its natural size. You can use different non-zero weights to indicate which components should grow fastest. For example, if you specified a value of 2 for the text control and 1 for the button bar, then the text control would grow twice as fast. If you have only one expanding component, simply give it a weight of 1.

The third parameter of the `Add` member function describes the horizontal growth behavior. We want the text control to grow to take up all horizontal space. But the button bar shouldn't grow—it is kept at its normal size and centered.

Finally, turn on auto layout and tell the frame which sizer to use. Then the frame consults the sizer when it is first shown, and whenever the user resizes it.

```
SetAutoLayout(true);
SetSizer(frame_sizer);
```

Here is the complete program. Run the program and resize the frame. Observe how the sizers grow the text control and keep the button bar centered.

ch25/button.cpp

```
1  #include <wx/wx.h>
2
3  const int ID_SAY_HELLO = 1000;
4  const int ID_SAY_GOODBYE = 1001;
5
6  /**
7     A frame with buttons that add greetings to a
8     text control.
9  */
10 class ButtonFrame : public wxFrame
11 {
12 public:
13    /**
14       Constructs and lays out the text control and buttons.
15    */
16    ButtonFrame();
17
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
18       /**
19          Adds a "Hello, World!" message to the text control.
20          @param event the event descriptor
21       */
22       void OnSayHello(wxCommandEvent& event);
23
24       /**
25          Adds a "Goodbye, World!" message to the text control.
26          @param event the event descriptor
27       */
28       void OnSayGoodbye(wxCommandEvent& event);
29    private:
30       wxTextCtrl* text;
31       DECLARE_EVENT_TABLE()
32    };
33
34    /**
35       An application to demonstrate button layout.
36    */
37    class ButtonApp : public wxApp
38    {
39    public:
40       /**
41          Constructs and shows the frame.
42          @return true
43       */
44       virtual bool OnInit();
45    private:
46       ButtonFrame* frame;
47    };
48
49    DECLARE_APP(ButtonApp)
50
51    IMPLEMENT_APP(ButtonApp)
52
53    BEGIN_EVENT_TABLE(ButtonFrame, wxFrame)
54       EVT_BUTTON(ID_SAY_HELLO, ButtonFrame::OnSayHello)
55       EVT_BUTTON(ID_SAY_GOODBYE, ButtonFrame::OnSayGoodbye)
56    END_EVENT_TABLE()
57
58    ButtonFrame::ButtonFrame()
59       : wxFrame(NULL, -1, "ButtonFrame")
60    {
61       text = new wxTextCtrl(this, -1, "",
62          wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
63
64       wxButton* hello_button = new wxButton(this,
65          ID_SAY_HELLO, "Say Hello");
66
67       wxButton* goodbye_button = new wxButton(this,
68          ID_SAY_GOODBYE, "Say Goodbye");
69
70       wxBoxSizer* button_sizer = new wxBoxSizer(wxHORIZONTAL);
71       button_sizer->Add(hello_button);
72       button_sizer->Add(goodbye_button);
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
73
74       wxBoxSizer* frame_sizer = new wxBoxSizer(wxVERTICAL);
75       frame_sizer->Add(text, 1, wxGROW);
76       frame_sizer->Add(button_sizer, 0, wxALIGN_CENTER);
77
78       SetAutoLayout(true);
79       SetSizer(frame_sizer);
80    }
81
82    void ButtonFrame::OnSayHello(wxCommandEvent& event)
83    {
84       text->AppendText("Hello, World!\n");
85    }
86
87    void ButtonFrame::OnSayGoodbye(wxCommandEvent& event)
88    {
89       text->AppendText("Goodbye, World!\n");
90    }
91
92    bool ButtonApp::OnInit()
93    {
94       frame = new ButtonFrame();
95       frame->Show(true);
96       return true;
97    }
```

25.7 Painting

A GUI program receives paint events whenever the contents of a window need to be painted.

In this section, you will see how to draw images in a GUI program. Drawing images in a windowing environment is not quite as straightforward as you may think. Consider the program in Figure 8. The program draws an ellipse that fills the entire window. When does the program need to draw the ellipse? Of course, the drawing must happen when the program's frame is first displayed. But that may not be enough. If the user resizes the frame, or minimizes and restores it, or if another frame pops up over it and then vanishes again, the program must redraw the image. The program has no idea when these events will happen. But the window manager knows when the contents of a window have been corrupted. Whenever that happens,

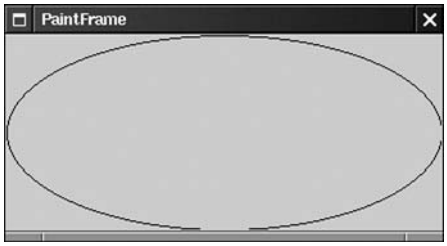

Figure 8 A Program That Draws a Graphical Shape

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

the program receives a paint event. Thus, the program needs to draw the image not just once, but *every time a paint event occurs*.

Therefore, you need to place all drawing instructions into a function, and set that function as the target of paint events. Place an entry such as the following into the event table:

```
EVT_PAINT(EllipseWindow::OnPaint)
```

Here is the paint function. It obtains a *device context*, an object that represents the surface of the window. By default, the device context fills the inside of a geometric shape with a fill color. For compatibility with this book's drawing library, we turn that feature off by setting the brush to a transparent brush.

The device context class has drawing functions such as `DrawLine`, `DrawEllipse`, and `DrawText`. Use the `DrawEllipse` function to draw an ellipse that fills the entire window.

```
void EllipseWindow::OnPaint(wxPaintEvent& event)
{
   wxPaintDC dc(this);
   dc.SetBrush(*wxTRANSPARENT_BRUSH);
   wxSize size = GetSize();
   int x = 0;
   int y = 0;
   int width = size.GetWidth();
   int height = size.GetHeight();
   dc.DrawEllipse(x, y, width, height);
}
```

The `DrawEllipse` function is a bit odd. You don't specify the center of the ellipse but the top left corner of the bounding box (see Figure 9).

The device context coordinates are in pixels. The (0, 0) point is the top left corner, and the y-coordinates increase towards the bottom of the screen (see Figure 10).

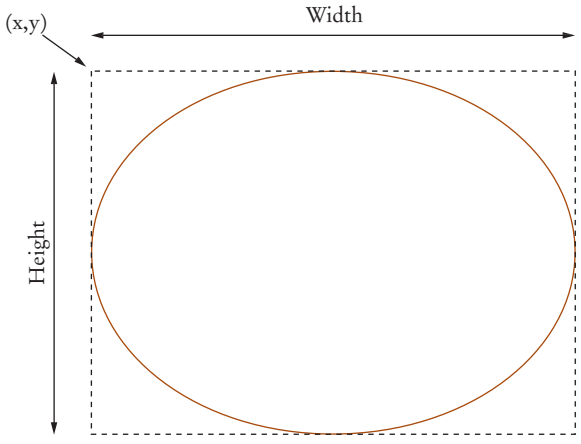

Figure 9 Specifying the Bounding Box of an Ellipse

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

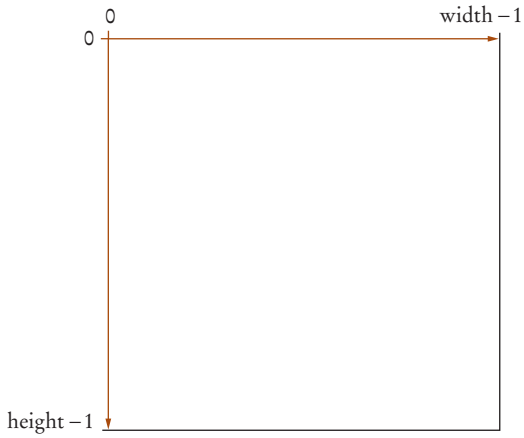

Figure 10 The Device Context Coordinate System

That is a common source of confusion, because it is the opposite of the convention in mathematics (and this book's graphics library).

The device context drawing operations are somewhat less object-oriented than those of this book's graphics library. There are no classes for lines, ellipses, and so on. Instead, you call functions whenever you want to draw a shape. On the other hand, the device context supports many advanced features. You can easily change brush colors, pen sizes, and text fonts. If you are interested in creating fancy drawings, check out the wxWidgets documentation for more information.

Finally, note that the `OnPaint` function does not draw directly on the application's frame but on a separate window of type `EllipseWindow`, derived from `wxWindow`. We take the attitude that an "ellipse window" is a user interface element, just like a text control or button, and that it deserves its own class. In Section 25.10, you will see a more realistic example where the paint function draws a clock in its own `ClockWindow`. That clock window is then placed inside a frame, along with text controls and buttons.

When constructing the window class, we need to tell the base class that this window needs to be fully repainted when the window is resized. By default, only newly exposed parts of the window are repainted, but that does not work if the contents of the window depends on the window size. Full repainting is achieved with the following call to the base class constructor:

```
EllipseWindow::EllipseWindow(wxWindow* parent)
   : wxWindow(parent, wxID_ANY, wxDefaultPosition, wxDefaultSize,
      wxFULL_REPAINT_ON_RESIZE)
{
}
```

Figure 11 shows the classes of the sample program. Note that we have three derived classes—this program specializes the application, frame, and window classes from the application framework.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

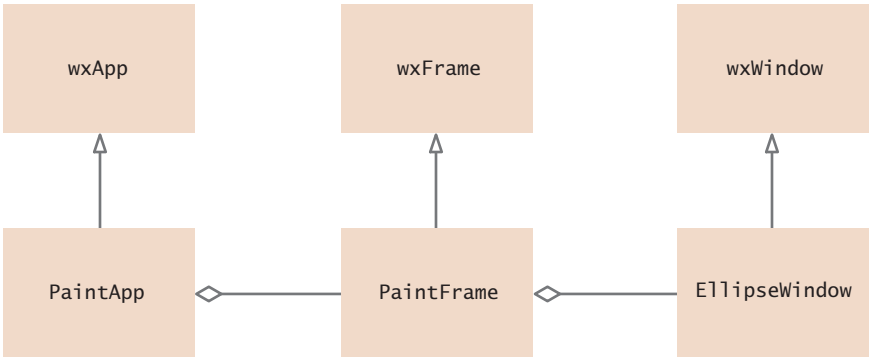

Figure 11 The Classes of the Paint Program

Here is the complete program. Run the program and resize the frame. Note how the ellipse is automatically repainted to fit the new frame size.

ch25/paint.cpp

```
#include <wx/wx.h>

/**
   A window onto which an ellipse is painted.
*/
class EllipseWindow : public wxWindow
{
public:
   /**
      Initializes the base class.
      @param parent the parent window
   */
   EllipseWindow(wxWindow* parent);

   /**
      Draws an ellipse on the window.
      @param event the event descriptor
   */
   void OnPaint(wxPaintEvent& event);
private:
   DECLARE_EVENT_TABLE()
};

/**
   A frame with a window that shows an ellipse.
*/
class PaintFrame : public wxFrame
{
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
29  public:
30     /**
31        Constructs the window.
32     */
33     PaintFrame();
34  private:
35     EllipseWindow* window;
36  };
37
38  /**
39     An application to demonstrate painting.
40  */
41  class PaintApp : public wxApp
42  {
43  public:
44     /**
45        Constructs and shows the frame.
46        @return true
47     */
48     virtual bool OnInit();
49  private:
50     PaintFrame* frame;
51  };
52
53  DECLARE_APP(PaintApp)
54
55  IMPLEMENT_APP(PaintApp)
56
57  BEGIN_EVENT_TABLE(EllipseWindow, wxWindow)
58     EVT_PAINT(EllipseWindow::OnPaint)
59  END_EVENT_TABLE()
60
61  EllipseWindow::EllipseWindow(wxWindow* parent)
62     : wxWindow(parent, wxID_ANY, wxDefaultPosition, wxDefaultSize,
63        wxFULL_REPAINT_ON_RESIZE)
64  {
65  }
66
67  void EllipseWindow::OnPaint(wxPaintEvent& event)
68  {
69     wxPaintDC dc(this);
70     dc.SetBrush(*wxTRANSPARENT_BRUSH);
71     wxSize size = GetSize();
72     int x = 0;
73     int y = 0;
74     int width = size.GetWidth();
75     int height = size.GetHeight();
76     dc.DrawEllipse(x, y, width, height);
77  }
78
79  PaintFrame::PaintFrame()
80     : wxFrame(NULL, -1, "PaintFrame")
81  {
```

Big C++, *Second Edition*, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
82       window = new EllipseWindow(this);
83    }
84
85    bool PaintApp::OnInit()
86    {
87       frame = new PaintFrame();
88       frame->Show(true);
89       return true;
90    }
```

25.8 Mouse Events

To handle mouse input in a graphical window, you install a function that is notified when mouse events occur. There are several kinds of mouse events:

- motion
- dragging (moving while depressing a mouse button)
- mouse button going down
- mouse button going up
- clicking (mouse button going down and up within a short period)
- double-clicking

You install the mouse handler with the `EVT_MOUSE_EVENTS` macro. In the notification function, you can query the `wxMouseEvent` parameter about the event type. For example, the function `ButtonDown` returns `true` for a "button down" event. You can also obtain the mouse position by calling the `GetX`/`GetY` functions of the `wxMouseEvent` class.

In our sample program, we allow a user to specify a triangle by clicking on the three corners. The principal difficulty in this program lies in the fact that the mouse handler is called separately for each mouse press. With each press, we record the mouse position. Then we need to carry out some drawing to give visual feedback to the user.

It is not a good idea to do the drawing in the mouse handler. All drawing should happen in the paint handler so that the logic for drawing is contained in a single location. Depending on the number of corners that have already been specified, the paint handler draws

- a small circle for the first mouse click
- a line after the first two mouse clicks
- a triangle after three mouse clicks

See Figure 12.

The mouse handler stores the corner points and then calls the `Refresh` function. That function generates a paint event, which eventually causes the paint function to

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Figure 12 The Three Phases of the Mouse Program

be called. You should never call the paint function directly but always call `Refresh` to request repainting.

```
void TriangleWindow::OnMouseEvent(wxMouseEvent& event)
{
   if (event.ButtonDown() && corners < 3)
   {
      x[corners] = event.GetX();
      y[corners] = event.GetY();
      corners++;
      Refresh();
   }
}
```

Here is the paint function:

```
void TriangleWindow::OnPaint(wxPaintEvent& event)
{
   const int RADIUS = 2;
   wxPaintDC dc(this);
   if (corners == 1)
      dc.DrawEllipse(x[0] - RADIUS, y[0] - RADIUS,
         2 * RADIUS, 2 * RADIUS);
   if (corners >= 2)
      dc.DrawLine(x[0], y[0], x[1], y[1]);
   if (corners >= 3)
   {
      dc.DrawLine(x[1], y[1], x[2], y[2]);
      dc.DrawLine(x[2], y[2], x[0], y[0]);
   }
}
```

This program is very typical for event-driven programming. Each mouse event causes a small change in the program state, increasing the `corners` counter and adding values to the `x` and `y` arrays. Whenever a paint event occurs, then the paint function consults that state to carry out the drawing operations. It is immaterial whether the paint event is the consequence of a mouse event or some other event.

Whenever you design such a program, it is a good idea to "work backwards" from the paint handler. What are the various kinds of drawings that the paint handler needs to create? What values does it need to have available to create these drawings? Those values need to be a part of the window state. Then ask yourself which events update those values. The code for updating the values needs to be placed into mouse handlers, button handlers, or other event handlers.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

ch25/mouse.cpp

```
#include <wx/wx.h>

/**
   A window on which the program user can draw
   a triangle by clicking on the three corners.
*/
class TriangleWindow : public wxWindow
{
public:
   /**
      Initializes the base class.
      @param parent the parent window
   */
   TriangleWindow(wxWindow* parent);

   /**
      Paints the corners and lines that have already been
      entered.
      @param event the event descriptor
   */
   void OnPaint(wxPaintEvent& event);

   /**
      Adds another corner to the triangle.
      @param event the event descriptor
   */
   void OnMouseEvent(wxMouseEvent& event);
private:
   int x[3];
   int y[3];
   int corners;
   DECLARE_EVENT_TABLE()
};

/**
   A frame with a window that shows a triangle.
*/
class MouseFrame : public wxFrame
{
public:
   /**
      Constructs the window.
   */
   MouseFrame();
private:
   TriangleWindow* window;
};

/**
   An application to demonstrate mouse event handling.
*/
class MouseApp : public wxApp
{
```

Big C++, *Second Edition*, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
54  public:
55     /**
56        Constructs and shows the frame.
57        @return true
58     */
59     virtual bool OnInit();
60  private:
61     MouseFrame* frame;
62  };
63
64  DECLARE_APP(MouseApp)
65
66  IMPLEMENT_APP(MouseApp)
67
68  BEGIN_EVENT_TABLE(TriangleWindow, wxWindow)
69     EVT_MOUSE_EVENTS(TriangleWindow::OnMouseEvent)
70     EVT_PAINT(TriangleWindow::OnPaint)
71  END_EVENT_TABLE()
72
73  TriangleWindow::TriangleWindow(wxWindow* parent)
74     : wxWindow(parent, wxID_ANY)
75  {
76     corners = 0;
77  }
78
79  void TriangleWindow::OnMouseEvent(wxMouseEvent& event)
80  {
81     if (event.ButtonDown() && corners < 3)
82     {
83        x[corners] = event.GetX();
84        y[corners] = event.GetY();
85        corners++;
86        Refresh();
87     }
88  }
89
90  void TriangleWindow::OnPaint(wxPaintEvent& event)
91  {
92     const int RADIUS = 2;
93     wxPaintDC dc(this);
94     dc.SetBrush(*wxTRANSPARENT_BRUSH);
95     if (corners == 1)
96        dc.DrawEllipse(x[0] - RADIUS, y[0] - RADIUS,
97           2 * RADIUS, 2 * RADIUS);
98     if (corners >= 2)
99        dc.DrawLine(x[0], y[0], x[1], y[1]);
100    if (corners >= 3)
101    {
102       dc.DrawLine(x[1], y[1], x[2], y[2]);
103       dc.DrawLine(x[2], y[2], x[0], y[0]);
104    }
105 }
106
```

Big C++, *Second Edition*, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
107 MouseFrame::MouseFrame()
108    : wxFrame(NULL, -1, "MouseFrame")
109 {
110    window = new TriangleWindow(this);
111 }
112
113 bool MouseApp::OnInit()
114 {
115    frame = new MouseFrame();
116    frame->Show(true);
117    return true;
118 }
```

25.9 Dialog Boxes

When designing a user interface, it is generally preferred to minimize *modes*. A mode restricts what a user can do at any given time, or interprets a user input in a way that depends on the mode. One example of a mode is the overtype mode in a word processor. In overtype mode, the typed characters replace existing characters instead of inserting themselves before the cursor. However, experience has shown that modes burden program users. To anticipate the behavior of the program, the user must expend some mental effort and keep track of the current mode. Mode errors are common. For example, if you accidentally activate overtype mode in a word processor, you delete text and must spend time to correct your error.

Modal dialog boxes interrupt a GUI program and force the user to fill in the dialog before going on.

Another example of a special program mode is a dialog box that requires immediate input from the user. The user can do nothing else except fill in or cancel the dialog box. This too can be burdensome for the user. Perhaps the user doesn't want to fill in all the information right now. Suppose you fill out a dialog box in the word processor, and then you remember that you need to make a change to the document. You can abandon the dialog box, losing the information that you already typed. Or you can complete the dialog box, and hopefully you then still remember what changes you wanted to make. Issues such as these can subject users to a certain amount of stress, and good user interface designers will want to minimize stressful situations. One alternative is to make a dialog box modeless, allowing users to switch back and forth between dialog windows and other windows.

Nevertheless, modal dialog boxes are necessary whenever a program simply cannot proceed without user intervention. They are also very easy to program, so you see them quite often in many applications, perhaps more often than good user interface design would suggest.

The wxWidgets toolkit makes it very easy to program several kinds of common dialog boxes. You can display a message for the user as follows:

```
wxMessageDialog* dialog = new wxMessageDialog(parent, message);
dialog->ShowModal();
dialog->Destroy();
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

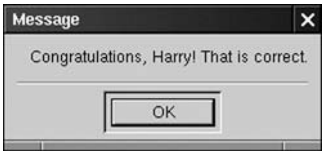

Figure 13
A Message Dialog Box

Then a dialog box pops up (see Figure 13). The dialog box is displayed until the user clicks the "OK" button. No other program window receives input until the user dismisses the dialog box.

The parent parameter is a pointer to the parent window. The dialog box is placed over its parent window. Often, the code that pops up the dialog box is a member function of a class derived from `wxFrame` or `wxWindow`. Then you pass `this` as the parent window pointer.

The message parameter is of type `wxString`, a class that is similar to the standard `string` type. You occasionally encounter such library classes that replicate standard library classes, usually because the library was older than the C++ standard. The `wxString` class has a constructor that accepts a C style (`char*`) string. We recommend that you use the standard string class for all string computations, then use the `c_str` function to convert to a C string, which is then automatically converted to a `wxString`. For example,

```
string message = "Hello, " + name;
dialog = new wxMessageDialog(this, message.c_str());
```

When you are done with a dialog box, you should destroy it. That function carries out a "delayed `delete`". It waits until all user interface messages to the dialog box have been processed, and then deletes the memory. (You don't delete or destroy frames, windows, buttons, and menus that are a permanent part of the program.)

Another convenient dialog box is the text entry dialog box that asks the user to supply a single line of text (see Figure 14). For example,

```
wxTextEntryDialog* dialog = new wxTextEntryDialog(this,
   "What is your name?");
dialog->ShowModal();
string name = dialog->GetValue().c_str();
dialog->Destroy();
```

The `GetValue` function returns a `wxString`. That class has a `c_str` function to convert to a C string, which you can immediately convert to a standard `string` object.

Figure 14 A Text Entry Dialog Box

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

ADVANCED TOPIC 25.1

Custom Dialog Boxes

If you want to show a custom dialog box, such as the one in Figure 15, you need to derive a class from the `wxDialog` class. Supply OK and Cancel buttons with the standard IDs `wxID_OK` and `wxID_CANCEL`. If the dialog is made up of labeled text fields, you can use a `wxFlexGridSizer` to lay them out in two columns. Finally, call the `Fit` function of the dialog sizer to give the dialog box the exact size needed to lay out the component.

```
class PlayerInfoDialog : public wxDialog
{
public:
   PlayerInfoDialog(wxWindow* parent);
   string get_name() const;
   int get_level() const;
private:
   wxTextCtrl* name_text;
   wxTextCtrl* level_text;
};

PlayerInfoDialog::PlayerInfoDialog(wxWindow* parent)
   : wxDialog(parent, -1, wxString("Player information"))
{
   name_text = new wxTextCtrl(this, -1);
   level_text = new wxTextCtrl(this, -1);

   wxFlexGridSizer* text_sizer = new wxFlexGridSizer(2);
   text_sizer->Add(new wxStaticText(this, -1, "Name:"));
   text_sizer->Add(name_text);
   text_sizer->Add(new wxStaticText(this, -1, "Level:"));
   text_sizer->Add(level_text);

   wxBoxSizer* button_sizer = new wxBoxSizer(wxHORIZONTAL);
   button_sizer->Add(new wxButton(this, wxID_OK, "OK"));
   button_sizer->Add(new wxButton(this, wxID_CANCEL, "Cancel"));
   wxBoxSizer* dialog_sizer = new wxBoxSizer(wxVERTICAL);
   dialog_sizer->Add(text_sizer, 1, wxGROW);
   dialog_sizer->Add(button_sizer, 0, wxALIGN_CENTER);

   SetAutoLayout(true);
   SetSizer(dialog_sizer);
   dialog_sizer->Fit(this);
}
```

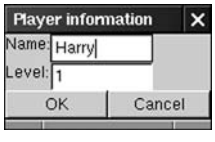

Figure 15
A Custom Dialog Box

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Then you call the `ShowModal` function in the usual way. That function returns `wxID_OK` if the user clicked on OK or `wxID_CANCEL` if the user cancels the dialog box.

```
PlayerInfoDialog* dialog = new PlayerInfoDialog(this);
if (dialog->ShowModal() == wxID_OK)
{
      player.set_name(dialog->get_name());
      player.set_level(dialog->get_level());
}
dialog->Destroy();
```

25.10 Case Study: A GUI for the Clock Game

In the final example of this chapter, we will put together a longer program, namely a wxWidgets version of the clock game in Chapter 22. The program has menus, buttons, text fields, dialog boxes, and a paint function. It is a good exercise for you to go through the program code and identify the various event handlers and their purposes. Figure 17 shows a diagram of all classes in the program.

Because of the event-driven nature of GUI programming, several modifications had to be made to the program logic. For example, the original program asks the user for a guess, then asks again if the guess was not correct. In this program, each guess is communicated to the program in the handler of the "Guess" button. The program must keep track whether the guess is the first or second guess. A data field `tries` has been added for that purpose.

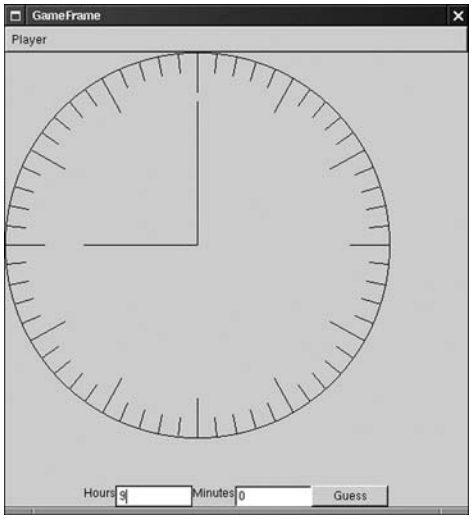

Figure 16 The Clock Game

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

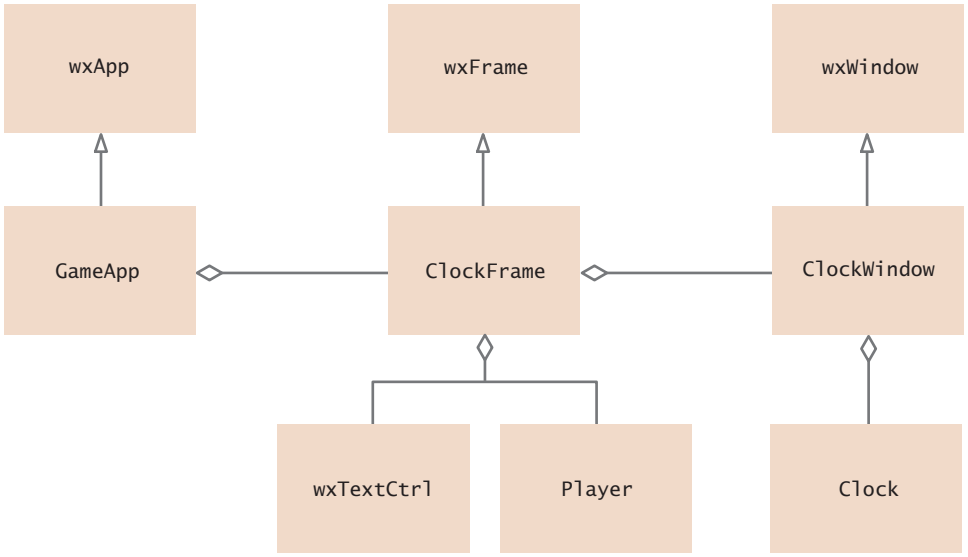

Figure 17 The Classes of the Clock Game

The original program queried the user name and level upon startup. The GUI version uses a different mechanism. The user selects menu options that lead to dialog boxes for entering this information. That is actually a better solution—a user can now change the level at any time during the game.

To simplify compilation, the nonstandard `Point` and `Time` classes have been eliminated from this program.

Here is the complete program. As you can see, the GUI programming strategies in this chapter allow you to produce professional looking applications with a relatively modest amount of programming. That is a tribute to the power of C++, classes, inheritance, and application frameworks. By using the wxWidgets framework, you inherit a tremendous amount of general purpose functionality, which leaves you to focus on the tasks that are specific to your application.

ch25/game.cpp

```
1  #include <wx/wx.h>
2  #include <string>
3  #include <cstdlib>
4  #include <cmath>
5
6  using namespace std;
7
8  const double PI = 3.141592653589793;
9
10 const int ID_GUESS = 1000;
11 const int ID_PLAYER_NAME = 1001;
12 const int ID_PLAYER_LEVEL = 1002;
13
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
14  /**
15     A clock that can draw its face.
16  */
17  class Clock
18  {
19  public:
20        Sets the current time.
21        @param h the hours to set
22        @param m the minutes to set
23     */
24     void set_time(int h, int m);
25
26     /**
27        Sets the size of this clock.
28        @param width the width of the enclosing window
29        @param height the width of the enclosing window
30     */
31     void set_size(int width, int height);
32
33     /**
34        Draws the clock face, with tick marks and hands.
35        @param dc the device context to draw on
36     */
37     void draw(wxDC& dc) const;
38  private:
39     /**
40        Draws a tick mark (hour or minute mark).
41        @param dc the device context to draw on
42        @param angle the angle in minutes (0...59, 0 = top)
43        @param length the length of the tick mark, as a fraction
44        of the radius (between 0.0 and 1.0)
45     */
46     void draw_tick(wxDC& dc, double angle, double length) const;
47
48     /**
49        Draws a hand, starting from the center.
50        @param dc the device context to draw on
51        @param angle the angle in minutes (0...59, 0 = top)
52        @param length the length of the hand, as a fraction
53        of the radius (between 0.0 and 1.0)
54     */
55     void draw_hand(wxDC& dc, double angle, double length) const;
56
57     int hours;
58     int minutes;
59     int centerx;
60     int centery;
61     int radius;
62  };
63
64  /**
65     The player of the clock game.
66  */
67  class Player
68  {
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
69  public:
70     /**
71        Constructs a player with name "Player",
72        level 1, and score 0.
73     */
74     Player();
75
76     /**
77        Increments the score. Moves to next level if current
78        level complete
79     */
80     void increment_score();
81
82     /**
83        Gets the current level.
84        @return the level
85     */
86     int get_level() const;
87
88     /**
89        Gets the player's name.
90        @return the name
91     */
92     string get_name() const;
93
94     /**
95        Sets the player's level.
96        @param l the level
97     */
98     void set_level(int l);
99
100    /**
101       Sets the player's name.
102       @param n the name
103    */
104    void set_name(string n);
105 private:
106    string name;
107    int score;
108    int level;
109 };
110
111 /**
112    The window that shows the clock.
113 */
114 class ClockWindow : public wxWindow
115 {
116 public:
117    /**
118       Constructs a clock window.
119       @param parent the parent window
120    */
121    ClockWindow(wxWindow* parent);
122
```

Big C++, *Second Edition*, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
123     /**
124        Sets the time of the clock and repaints it.
125        @param h the hours
126        @param m the minutes
127     */
128     void set_time(int h, int m);
129
130     /**
131        Paints the clock.
132        @param event the event descriptor
133     */
134     void OnPaint(wxPaintEvent& event);
135  private:
136     Clock clock;
137     DECLARE_EVENT_TABLE()
138  };
139
140  /**
141     The frame that contains the clock window and the
142     fields for entering a guess.
143  */
144  class GameFrame : public wxFrame
145  {
146  public:
147     /**
148        Constructs the game frame.
149     */
150     GameFrame();
151
152     /**
153        Starts a new round, with a new clock time.
154     */
155     void new_round();
156
157     /**
158        Processes the player's guess.
159        @param event the event descriptor
160     */
161     void OnGuess(wxCommandEvent& event);
162
163     /**
164        Prompts the player to enter a name.
165        @param event the event descriptor
166     */
167     void OnPlayerName(wxCommandEvent& event);
168
169     /**
170        Prompts the player to enter a level.
171        @param event the event descriptor
172     */
173     void OnPlayerLevel(wxCommandEvent& event);
174  private:
175     ClockWindow* window;
176     wxTextCtrl* hour_text;
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
177     wxTextCtrl* minute_text;
178     Player player;
179     int current_hours;
180     int current_minutes;
181     int tries;
182     DECLARE_EVENT_TABLE()
183  };
184
185  /**
186     The clock game application.
187  */
188  class GameApp : public wxApp
189  {
190  public:
191     /**
192        Constructs and shows the frame.
193        @return true
194     */
195     virtual bool OnInit();
196  private:
197     GameFrame* frame;
198  };
199
200  DECLARE_APP(GameApp)
201
202  IMPLEMENT_APP(GameApp)
203
204  BEGIN_EVENT_TABLE(ClockWindow, wxWindow)
205     EVT_PAINT(ClockWindow::OnPaint)
206  END_EVENT_TABLE()
207
208  BEGIN_EVENT_TABLE(GameFrame, wxFrame)
209     EVT_BUTTON(ID_GUESS, GameFrame::OnGuess)
210     EVT_MENU(ID_PLAYER_NAME, GameFrame::OnPlayerName)
211     EVT_MENU(ID_PLAYER_LEVEL, GameFrame::OnPlayerLevel)
212  END_EVENT_TABLE()
213
214  /**
215     Sets the seed of the random number generator.
216  */
217  void rand_seed()
218  {
219     int seed = static_cast<int>(time(0));
220     srand(seed);
221  }
222
223  /**
224     Returns a random integer in a range.
225     @param a the bottom of the range
226     @param b the top of the range
227     @return a random number x, a <= x and x <= b
228  */
229  int rand_int(int a, int b)
230  {
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
231    return a + rand() % (b - a + 1);
232 }
233
234 void Clock::set_time(int h, int m)
235 {
236    hours = h;
237    minutes = m;
238 }
239
240 void Clock::set_size(int width, int height)
241 {
242    centerx = width / 2;
243    centery = height / 2;
244    if (width < height)
245       radius = width * 0.45;
246    else
247       radius = height * 0.45;
248 }
249
250 void Clock::draw_tick(wxDC& dc, double angle,
251    double length) const
252 {
253    double alpha = -PI / 2 + 6 * angle * PI / 180;
254    dc.DrawLine(
255       centerx + static_cast<int>(
256          cos(alpha) * radius * (1 - length)),
257       centery + static_cast<int>(
258          sin(alpha) * radius * (1 - length)),
259       centerx + static_cast<int>(cos(alpha) * radius),
260       centery + static_cast<int>(sin(alpha) * radius));
261 }
262
263 void Clock::draw_hand(wxDC& dc, double angle,
264    double length) const
265 {
266    double alpha = -PI / 2 + 6 * angle * PI / 180;
267    dc.DrawLine(centerx, centery,
268       centerx + static_cast<int>(cos(alpha) * radius * length),
269       centery + static_cast<int>(sin(alpha) * radius * length));
270 }
271
272 void Clock::draw(wxDC& dc) const
273 {
274    dc.DrawEllipse(centerx - radius, centery - radius,
275       2 * radius, 2 * radius);
276    const double HOUR_TICK_LENGTH = 0.2;
277    const double MINUTE_TICK_LENGTH = 0.1;
278    const double HOUR_HAND_LENGTH = 0.6;
279    const double MINUTE_HAND_LENGTH = 0.75;
280    for (int i = 0; i < 12; i++)
281    {
282       draw_tick(dc, i * 5, HOUR_TICK_LENGTH);
283       int j;
284       for (j = 1; j <= 4; j++)
285          draw_tick(dc, i * 5 + j, MINUTE_TICK_LENGTH);
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
286      }
287      draw_hand(dc, minutes, MINUTE_HAND_LENGTH);
288      draw_hand(dc, (hours + minutes / 60.0) * 5, HOUR_HAND_LENGTH);
289   }
290
291   Player::Player()
292   {
293      name = "Player";
294      level = 1;
295      score = 0;
296   }
297
298   void Player::set_level(int l)
299   {
300      level = l;
301   }
302
303   void Player::set_name(string n)
304   {
305      name = n;
306   }
307
308   int Player::get_level() const
309   {
310      return level;
311   }
312
313   string Player::get_name() const
314   {
315      return name;
316   }
317
318   void Player::increment_score()
319   {
320      score++;
321      if (score % 5 == 0 && level < 4)
322         level++;
323   }
324
325   ClockWindow::ClockWindow(wxWindow* parent)
326      : wxWindow(parent, wxID_ANY, wxDefaultPosition, wxDefaultSize,
327         wxFULL_REPAINT_ON_RESIZE)
328   {
329   }
330
331   void ClockWindow::OnPaint(wxPaintEvent& event)
332   {
333      wxPaintDC dc(this);
334      dc.SetBrush(*wxTRANSPARENT_BRUSH);
335
336      wxSize size = GetSize();
337      clock.set_size(size.GetWidth(), size.GetHeight());
338
339      clock.draw(dc);
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
340 }
341
342 void ClockWindow::set_time(int h, int m)
343 {
344    clock.set_time(h, m);
345    Refresh();
346 }
347
348 GameFrame::GameFrame()
349    : wxFrame(NULL, -1, "GameFrame")
350 {
351    // Initialize menu
352    wxMenu* menu = new wxMenu();
353    menu->Append(ID_PLAYER_NAME, "Name");
354    menu->Append(ID_PLAYER_LEVEL, "Level");
355
356    // Add menu to menu bar
357    wxMenuBar* menu_bar = new wxMenuBar();
358    SetMenuBar(menu_bar);
359    menu_bar->Append(menu, "Player");
360
361    window = new ClockWindow(this);
362
363    hour_text = new wxTextCtrl(this, -1);
364    minute_text = new wxTextCtrl(this, -1);
365
366    wxButton* guess_button = new wxButton(this,
367       ID_GUESS, "Guess");
368
369    wxBoxSizer* bottom_sizer = new wxBoxSizer(wxHORIZONTAL);
370    bottom_sizer->Add(new wxStaticText(this, -1, "Hours"));
371    bottom_sizer->Add(hour_text);
372    bottom_sizer->Add(new wxStaticText(this, -1, "Minutes"));
373    bottom_sizer->Add(minute_text);
374    bottom_sizer->Add(guess_button);
375
376    wxBoxSizer* frame_sizer = new wxBoxSizer(wxVERTICAL);
377    frame_sizer->Add(window, 1, wxGROW);
378    frame_sizer->Add(bottom_sizer, 0, wxALIGN_CENTER);
379
380    SetAutoLayout(true);
381    SetSizer(frame_sizer);
382
383    new_round();
384 }
385
386 void GameFrame::OnGuess(wxCommandEvent& event)
387 {
388    tries++;
389    int hours = atoi(hour_text->GetValue().c_str());
390    int minutes = atoi(minute_text->GetValue().c_str());
391    if (hours < 1 || hours > 12)
392    {
```

Big C++, *Second Edition*, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
393         wxMessageDialog* dialog = new wxMessageDialog(this,
394            "Hours must be between 1 and 12");
395         dialog->ShowModal();
396         dialog->Destroy();
397         return;
398      }
399      if (minutes < 0 || minutes > 59)
400      {
401         wxMessageDialog* dialog = new wxMessageDialog(this,
402            "Hours must be between 1 and 12");
403         dialog->ShowModal();
404         dialog->Destroy();
405         return;
406      }
407      if (current_hours == hours && current_minutes == minutes)
408      {
409         string text = "Congratulations, " + player.get_name()
410            + "! That is correct.";
411         wxMessageDialog* dialog = new wxMessageDialog(this,
412            text.c_str());
413         dialog->ShowModal();
414         dialog->Destroy();
415         player.increment_score();
416         new_round();
417      }
418      else
419      {
420         string text = "Sorry, " + player.get_name()
421            + "! That is not correct.";
422         wxMessageDialog* dialog = new wxMessageDialog(this,
423            text.c_str());
424         dialog->ShowModal();
425         dialog->Destroy();
426         if (tries == 2) new_round();
427      }
428   }
429
430   void GameFrame::new_round()
431   {
432      tries = 0;
433      int level = player.get_level();
434      if (level == 1) current_minutes = 0;
435      else if (level == 2) current_minutes = 15 * rand_int(0, 3);
436      else if (level == 3) current_minutes = 5 * rand_int(0, 11);
437      else current_minutes = rand_int(0, 59);
438      current_hours = rand_int(1, 12);
439      window->set_time(current_hours, current_minutes);
440   }
441
442   void GameFrame::OnPlayerName(wxCommandEvent& event)
443   {
444      wxTextEntryDialog* dialog = new wxTextEntryDialog(this,
445         "What is your name?");
446      dialog->ShowModal();
447      player.set_name(dialog->GetValue().c_str());
```

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

```
448    dialog->Destroy();
449 }
450
451 void GameFrame::OnPlayerLevel(wxCommandEvent& event)
452 {
453    wxTextEntryDialog* dialog = new wxTextEntryDialog(this,
454       "At what level do you want to play? (1-4)");
455    dialog->ShowModal();
456    int level = atoi(dialog->GetValue().c_str());
457    dialog->Destroy();
458    if (level < 1 || level > 4)
459    {
460       wxMessageDialog* dialog = new wxMessageDialog(this,
461          "The level must be between 1 and 4");
462       dialog->ShowModal();
463       dialog->Destroy();
464       return;
465    }
466    player.set_level(level);
467 }
468
469 bool GameApp::OnInit()
470 {
471    rand_seed();
472    frame = new GameFrame();
473    frame->Show(true);
474    return true;
475 }
```

Random Fact 25.1

Visual Programming

Programming as you know it involves typing code into a text editor and then running it. A programmer must be familiar with a programming language to write even the simplest of programs. When programming menus or buttons, one must write code to direct the layout of these user interface elements.

A *visual* style of programming makes this much easier. When you use a visual programming environment, you use your mouse to specify where buttons, text fields, and other fields should appear on the screen (see Figure 18). You still need to do some programming. You need to write code for every event. For example, you can drag a button to its desired location, but you still need to specify what should happen when the user clicks on that button.

Visual programming offers several benefits. It is much easier to lay out a screen by dragging buttons and menu items with the mouse than it is to write the layout code. Most visual programming environments are also very extensible. You can add user interface elements from third parties, many with sophisticated behavior. For example, a calendar element can show the current month's calendar, with buttons to move to the next or previous month. All of that has been preprogrammed by someone (usually the hard way, using a traditional programming language), but you can add a fully working calendar to your program simply by dragging it off a toolbar and dropping it into your program.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

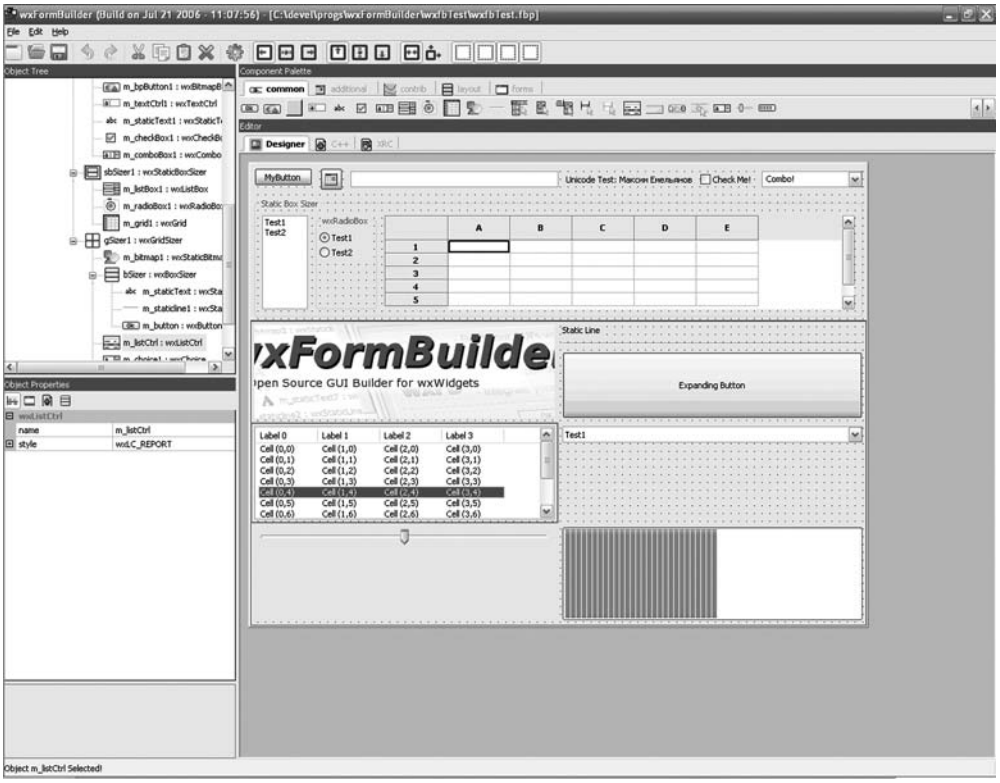

Figure 18 A Visual Programming Environment

A prebuilt component, such as a calendar chooser, usually has a large number of *properties* that you can simply choose from a table. For example, you can simply check whether you want the calendar to be weekly or monthly. The provider of the calendar component had to work hard to include code for both cases, but the developer using the component can customize the component without any programming.

You should select visual GUI builders with care. Some environments force you to use mouse clicks even when editing a text file would be much faster. For example, it is nice for a beginner to drag and drop menu trees, but experienced programmers find it much easier to modify a text file. A good environment should offer both options. Some environments only remember your mouse clicks, and not the intentions behind them. Then it can be tedious to adapt your visual design to other languages or platforms.

A good GUI builder can make building an effective GUI *much* easier than writing the equivalent code in C++. These systems are highly recommended for professional user interface programming.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

CHAPTER SUMMARY

1. GUIs can be programmed through low-level libraries specific to an operating system, or through higher-level application frameworks.
2. The wxWidgets framework is suitable for GUI programming with various platforms and compilers.
3. A frame is a window with the decorations provided by the windowing system.
4. Your application class should be derived from the `wxApp` class.
5. You use inheritance to describe the differences between an application framework's generic classes and the functionality required by your application.
6. Use a `wxTextCtrl` for text input and output.
7. GUI programs are event-driven. Event handlers are functions that are called when events occur.
8. In wxWidgets, event tables map events to functions.
9. You use different table entries for each event type (such as menu, button, or mouse events).
10. When placing user interface elements in a window, you need to specify the layout of the components.
11. The wxWidgets framework uses sizers to specify the sizing rules so that programs can be ported to different languages and windowing environments.
12. A GUI program receives paint events whenever the contents of a window need to be painted.
13. Modal dialog boxes interrupt a GUI program and force the user to fill in the dialog before going on.

REVIEW EXERCISES

Exercise R25.1. What is the essential difference in control flow between graphical user interface applications and console applications?

Exercise R25.2. List at least eight user interface elements that you have encountered in commonly used GUI programs.

Exercise R25.3. What is an application framework?

Exercise R25.4. What is the essential difference between a frame and a window?

Exercise R25.5. When do you form derived classes of `wxFrame` and when do you form derived classes of `wxWindow`?

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Exercise R25.6. Into which class do you place the event table of a menu? Of a button? Of paint events? Of mouse events?

Exercise R25.7. What happens if you forget to place the `DECLARE_APP` or `IMPLEMENT_APP` macro into your application? Try it out and explain the error message that you get.

Exercise R25.8. What happens if your event table maps an event to a function with the wrong signature? Try it out and explain the error message that you get.

Exercise R25.9. What is the difference between a single-line and a multi-line text control? How do you construct each kind?

Exercise R25.10. Explain why you need sizers to lay out buttons but you don't need them to lay out menus.

Exercise R25.11. Explain under which circumstances paint events are generated. You may want to place a print statement into the paint function of a sample program and find out when it is called.

Exercise R25.12. List the different kinds of mouse events. Find out from the documentation of the `wxMouseEvent` class how you can tell them apart.

Exercise R25.13. What is a mode in a program? Give three examples of modes in commonly used applications.

Exercise R25.14. What is the difference between a modal and a modeless dialog box?

Exercise R25.15. Which wxWidgets objects do you allocate on the stack? Which do you allocate with `new`? Which of them do you destroy?

Exercise R25.16. How do you convert between standard strings and `wxString` objects? Why are there two separate classes?

PROGRAMMING EXERCISES

Exercise P25.1. Implement a program that shows the growth of a $10,000 investment that earns interest at 5 percent per year. Supply a menu called "Bank" and a menu item called "Add Interest". When the user selects that menu item, add the interest to the current balance and append a message showing the current balance to a text control.

Exercise P25.2. Add menu options to change the current balance and interest rate to Exercise P25.1.

Exercise P25.3. Write a wxWidgets program that displays a square. Initially, the square is displayed in the center of the window. Supply a menu called "Move" and four menu items called "Left", "Right", "Up", and "Down" that move the square by 10 pixels in the indicated direction. *Hint:* In the event handlers, change the position of the square, then call `Refresh`.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.

Exercise P25.4. Implement the same functionality as in Exercise P25.3, except supply a row of four buttons to move the square.

Exercise P25.5. Write a wxWidgets program that displays the temperature chart of Section 2.8.2. Since you can't change the coordinate system of the window, you must manually transform logical units to pixel units.

Exercise P25.6. Add a menu option to the program in Exercise P25.5 that changes the temperature value of a given month.

Exercise P25.7. *Implement a tic-tac-toe board.* Draw the grid lines and process mouse events. When the user clicks on a field, draw an "x" mark for all even moves and an "o" mark for all odd moves. You don't have to check for illegal moves.

Exercise P25.8. Refine the program of Exercise P25.7 so that it checks for illegal moves, pronounces winners and ties, and resets the game after a win or tie.

Exercise P25.9. Write a program that paints a clock face at the current time. That is, in the paint handler, get the current time and draw the clock's hands accordingly.

Exercise P25.10. Consult the wxWidgets documentation to find out about timer events. Add a timer event handler to the preceding program that refreshes the clock window once a second, so that it always shows the correct time.

Exercise P25.11. Change the clock game by showing the current level in a text control. Place a button "Set level" next to the text control so that the user can change the level at any time.

Exercise P25.12. Write a wxWidgets program that implements a different game, to teach arithmetic to your younger brother. The program tests addition and subtraction. In level 1 it tests only addition of numbers less than 10 whose sum is less than 10. In level 2 it tests addition of arbitrary one-digit numbers. In level 3 it tests subtraction of one-digit numbers with a nonnegative difference. Generate random problems and get the player input. The player gets up to two tries per problem. Advance from one level to the next when the player has achieved a score of five points.

Big C++, Second Edition, Horstmann and Budd, Copyright © 2009 John Wiley & Sons, Inc. All Rights Reserved.