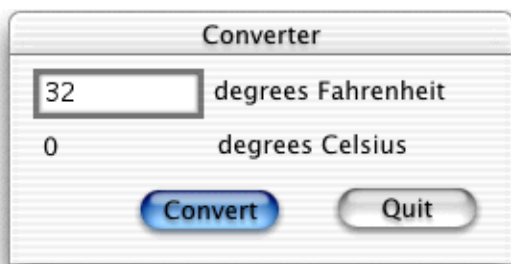


Converter: Creating a User Interface with Interface Builder

This tutorial shows you how to use Interface Builder to create the interface for a simple application that converts Fahrenheit temperatures to Celsius. You'll see how Interface Builder is closely tied to the Carbon Event Manager and lets you create a useful application with very little code.

In this tutorial, you'll create a simple application that converts a Fahrenheit temperature to Celsius. When you're done, you'll have an application that looks like this:



This tutorial contains three sections, which you should complete in order:

1. "Create the Converter Window" (page 2) teaches you how to create a simple interface and shows how much the Carbon Event Manager does for you, even when you don't write any code.
2. "Add the Convert Button Handler" (page 14) shows you what an application that uses the Carbon Event Manager looks like and teaches you how to create a command handler
3. "Create the Convert Menu Item" (page 19) shows you how to add a menu to an application and how the Carbon Event Manager lets you extend an application's easily.

CHAPTER 1

This tutorial does not attempt to teach you Project Builder or Mac OS X programming.

Create the Converter Window

In this section, you'll create the application's project and interface. You'll learn how to create a simple interface, and find out what the Carbon Event Manager can do for you, even if you don't write any code.

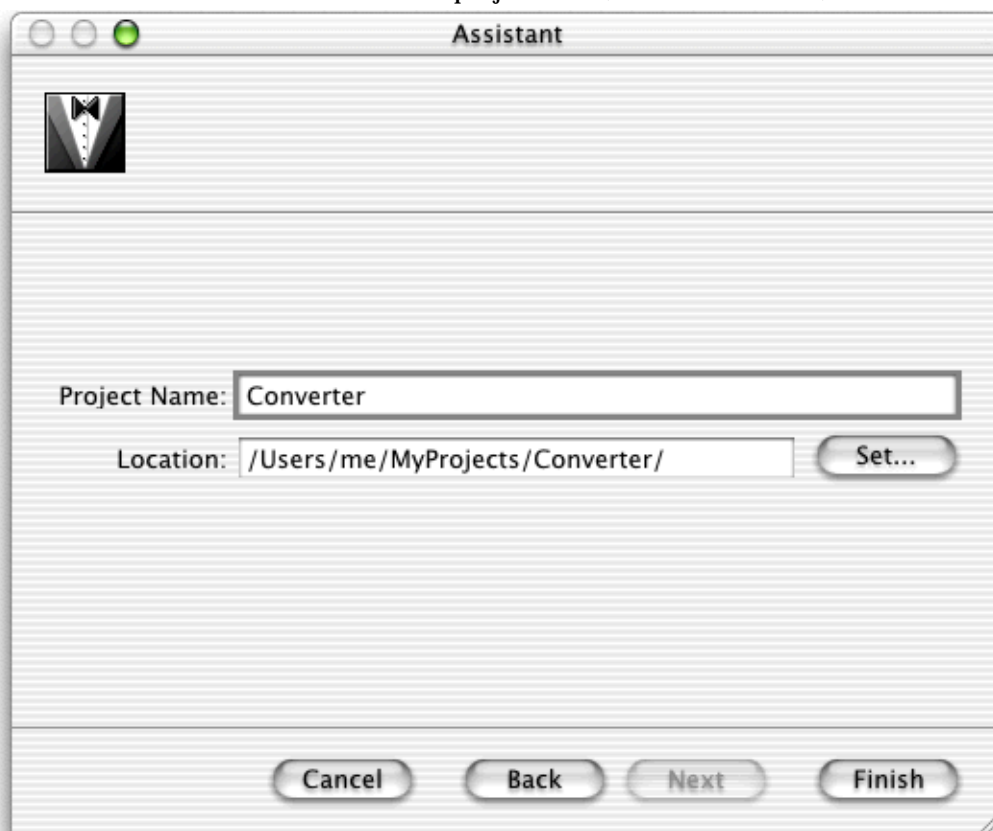
1. ["Create the Project"](#) (page 2)
2. ["Open the Nib File"](#) (page 3)
3. ["Add the Fahrenheit Field and Label"](#) (page 5)
4. ["Add the Celsius Field and Label"](#) (page 7)
5. ["Add a Convert Button"](#) (page 8)
6. ["Add a Quit Button"](#) (page 10)
7. ["Set the Window's Attributes"](#) (page 11)
8. ["Build and Run the Application"](#) (page 13)

Create the Project

Start Project Builder by double-clicking its icon. You can find it in /Developer/Applications/.

CHAPTER 1

Choose File > New Project. Project Builder displays a dialog box with several template projects to choose from. Select Carbon Application (Nib Based), and click Next. Then enter Converter as the project name, choose a location, and click Finish.



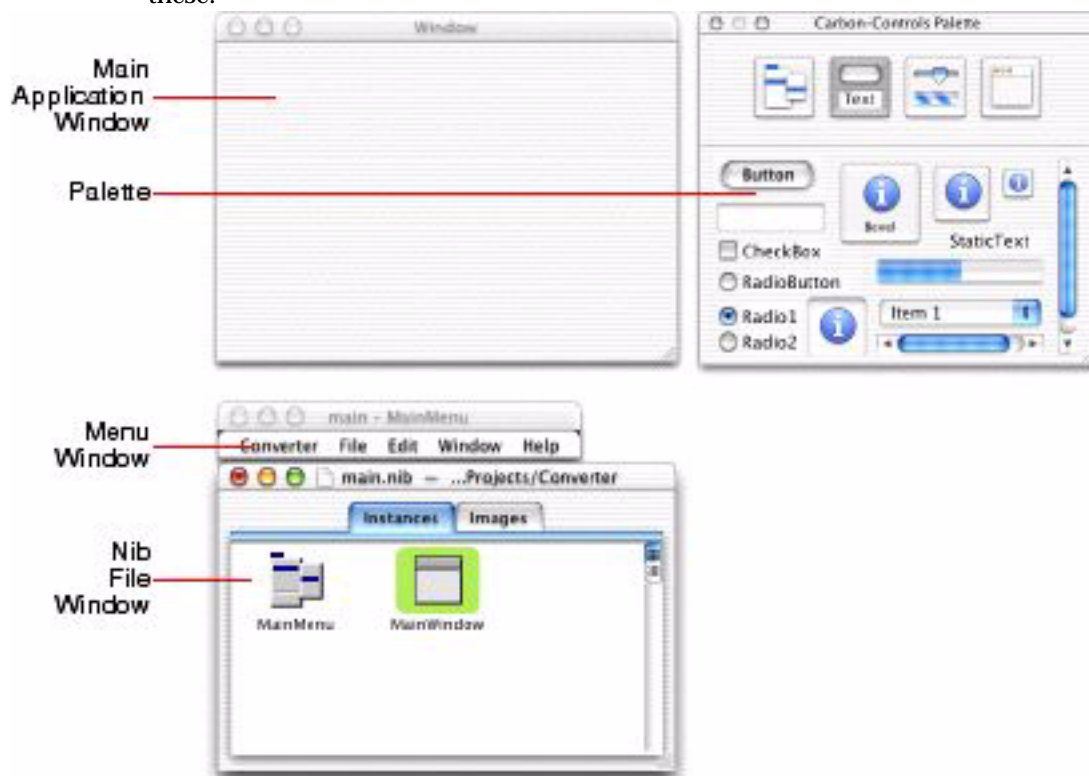
Open the Nib File

A nib file is a way to store your application's resources that's new with Carbon. It's a XML text file that describes your application's windows, menus, buttons, text fields and other user interface elements. Your application can open a nib file just as easily as it opens a resource file or a resource fork.

In the file list, double-click the main.nib file. If you can't see the main.nib file, click the Files tab and open the Resources group.

CHAPTER 1

Interface Builder launches and opens the file. It displays several windows, like these:



- The Main Application Window is the window displayed when your application runs. An application can use any number of windows, but this template contains just one.
- The Palette contains controls that you can add to your interface, just by dragging them onto a window or a menu.
- The Menu window lets you edit the items that appear in your application's menu bar. It already contains most of the commands applications need, such as About, Quit, Save, Close, Copy, and Paste. Many of them do the right thing without your needing to write any code, such as Quit, Copy, and Paste. For some, you do need to write code, such as About and Save.
- The Nib file window displays all the top level items in your nib file. This boilerplate nib file contains a window and a menu bar.

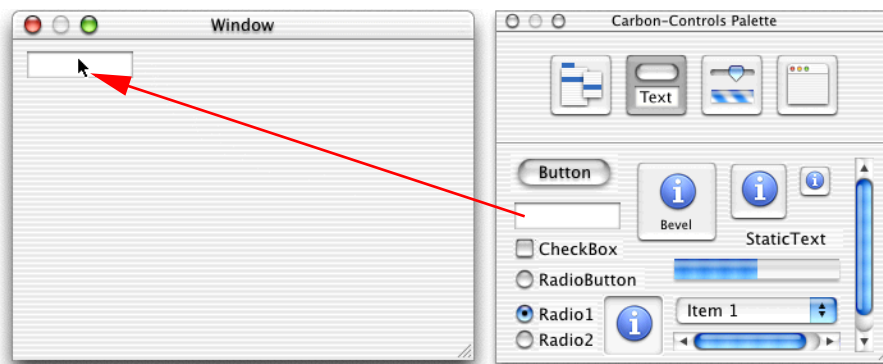
Add the Fahrenheit Field and Label

Now you'll add the text field and the label where the user will enter the Fahrenheit temperature.

1. Drag a text field to the top left corner the window.

This will be the field where the user enters the Fahrenheit temperature.

In the palette click the Button/Text button. The text field is the white box under the push button labeled "Button." Drag a text field to the upper left corner of the window.



2. Enter 32 as the text field's value.

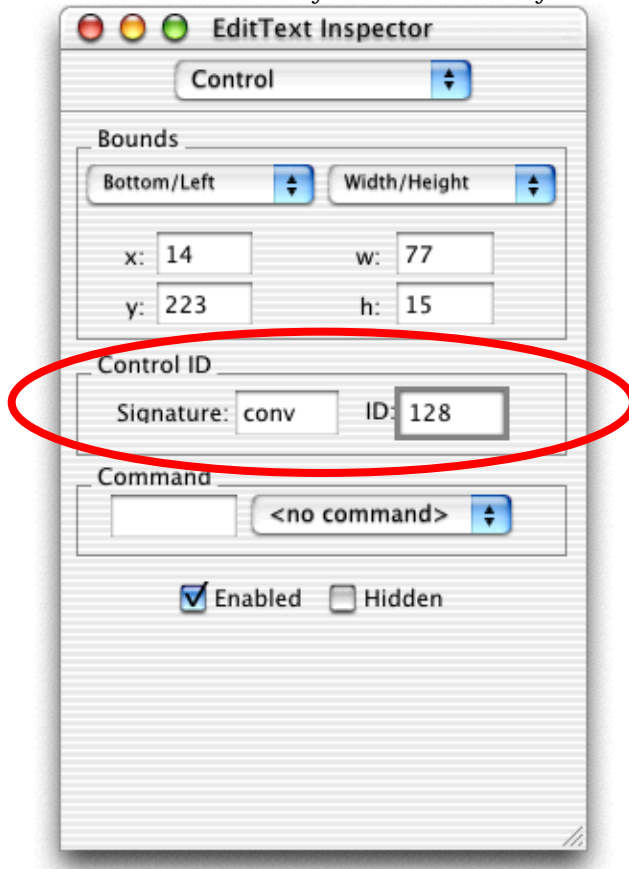
Double-click the field so a blinking insertion point appears. Enter 32 and press Return.

3. Enter the conv as text field's signature and 128 as the text field's ID.

Select the field, choose Tools > Inspector. In the Inspector, you can set all of a control's properties. In the Attributes section, you set attributes specific to a particular control, such as whether a button contains a graphic or whether a window is resizeable. In the Controls section, you set attributes that can apply to any Carbon controls, such as its size, position, and identification information.

CHAPTER 1

From the pop-up menu at the top of the Inspector, select Control. In the Control ID section, enter `conv` in the Signature field and 128 in the ID field. You'll use these values later when you need to identify this field to read its value.

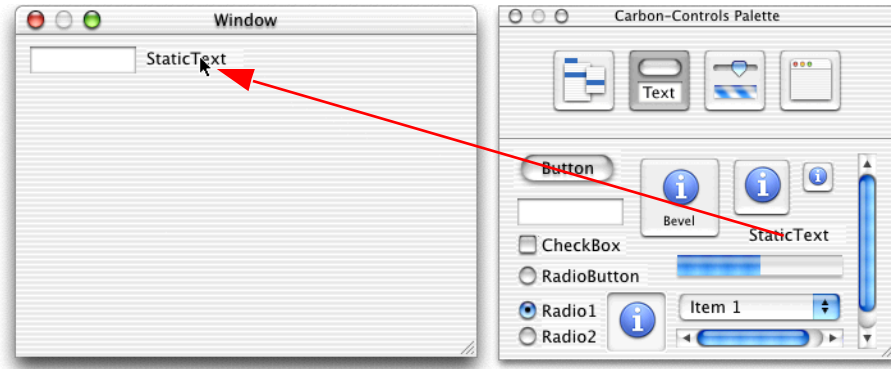


When you're done, don't close the Inspector. You'll be using it later. Interface Builder has one Inspector and it displays the attributes of whatever control is selected.

4. Drag a static text field to the right of the Fahrenheit field.

CHAPTER 1

From the palette, drag the item named Static Text to the right of the field you just created.



5. Enter degrees Fahrenheit as the text field's value.

Double-click the field so a blinking insertion point appears. Enter degrees Fahrenheit and press Return.

Part of the text may be cut off since the field is too small. If that's the case, select the field and choose Layout > Size to Fit. Interface Builder resizes the field so it's just large enough to contain its text.

Add the Celsius Field and Label

Now you'll add the static text field and the label where the Celsius temperature will appear.

1. Drag a static text field below the Fahrenheit text field.

This will be the field where the Celsius field appears. It's a static text field since the user won't be able to change its value.

From the palette, drag the item named Static Text to the area below the Fahrenheit text field.

2. Enter 0 as the text field's value.

Double-click the field so a blinking insertion point appears. Enter 0 and press Return.

3. Enter the conv as text field's signature and 129 as the text field's ID.

CHAPTER 1

From the pop-up menu at the top of the Inspector, select Control. In the Control ID section, enter `conv` in the Signature field and 129 in the ID field.

4. Drag a static text field to the right of the Fahrenheit field.

From the palette, drag the item named Static Text to the right of the field you just created.

5. Enter `degrees Celsius` as the text field's value.

Double-click the field so a blinking insertion point appears. Enter “degrees Celsius” and press Return. If the text is cut off, select the field and choose Layout > Size to Fit.

Add a Convert Button

Now you'll add the button that actually performs the conversion. In a later section, you'll write the handler that's called when the user presses this button.

1. Drag a push button below the Celsius field and label.

In the palette, the push button is in the top left corner and is labeled “Button.” Drag it so it's centered under the space between the Celsius field and label.

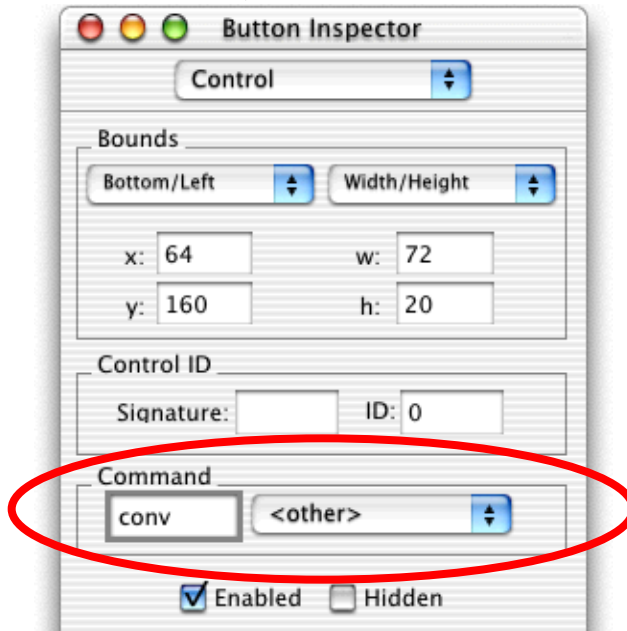
2. Enter `Convert` as the button's text.

Double-click the button so a blinking insertion point appears. Enter `Convert` and press Return.

3. Enter `conv` as the button's command.

CHAPTER 1

From the pop-up menu at the top of the Inspector, select Control. In the Command section, enter conv in the text field.



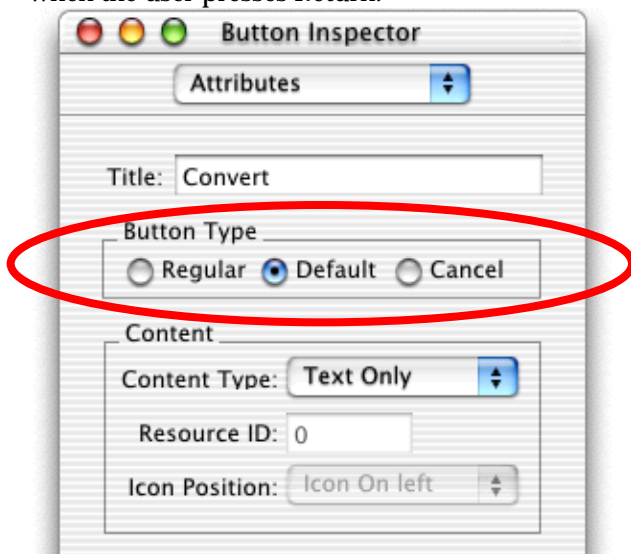
When the user presses this button, it will send a 'conv' command. This sort of command is new to the Carbon Event Manager and is very powerful. You can assign the 'conv' command to any control and when the user selects that control, the Carbon Event Manager calls the handler for it. Later on, you'll define the handler for it. And in the last section of this tutorial you'll add a menu command that also uses the 'conv' command

Notice the pop-up menu besides this field contains items for many common menu commands, including Cut, Copy, Paste, Preferences, and Hide. These commands are handled automatically by the Carbon Event Handler. You don't need to write a handler for them.

4. Make this the window's default button.

CHAPTER 1

From the pop-up menu at the top of the Inspector, select Attributes. In the Button Type section, select Default. Now, this button will pulse and be selected when the user presses Return.



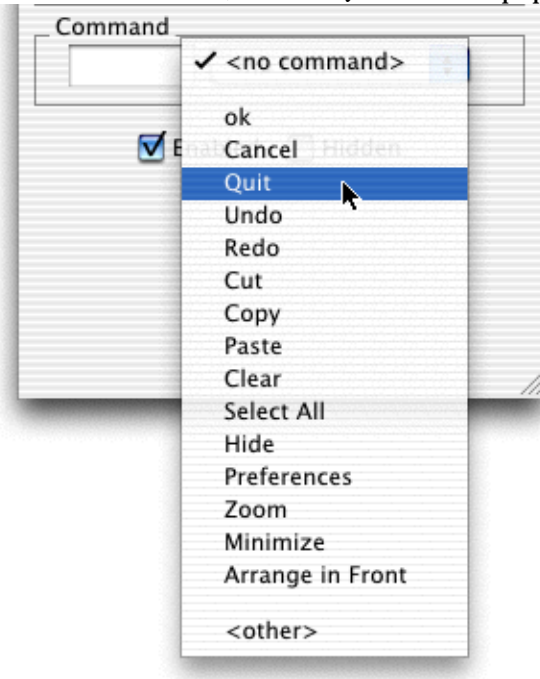
Add a Quit Button

Now you'll add a button that quits the application when the user presses it.

1. Drag a push button to the right of the Convert button.
From the palette, drag the push button to the right of the Convert button.
2. Enter Quit as the button's text.
Double-click the button so a blinking insertion point appears. Enter Quit and press Return.
3. Choose Quit as the button's command.

CHAPTER 1

From the pop-up menu at the top of the Inspector, select Control. In the Command section, Choose Quit from the pop-up menu.



When the user presses this button, it will send a Quit command which the Carbon Event Manager will handle for you by quitting the application. You don't need to write any code to accomplish this.

Set the Window's Attributes

Now you'll resize the window and change its name and type.

1. Resize the window

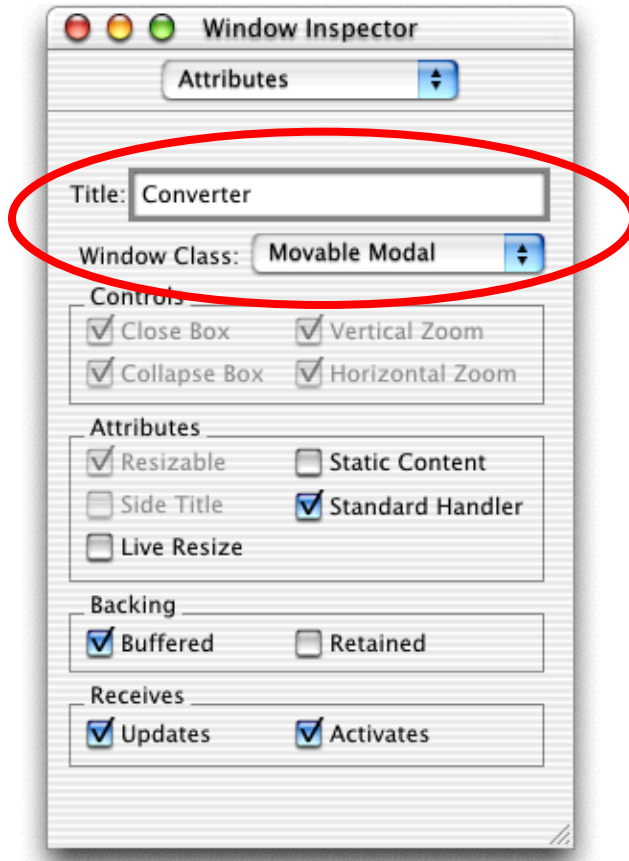
Click and drag the window's resize box so the window is just big enough to hold its items.

2. Change the window's name to Converter and its class to Movable Modal.

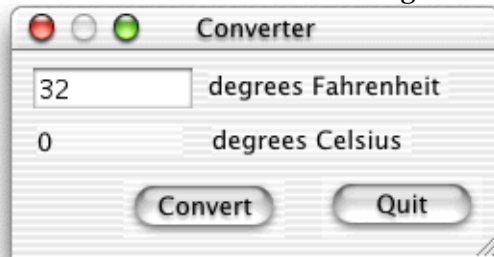
To select the window, click anywhere in its background; that is, any spot that doesn't contain a control. From the pop-up menu at the top of the Inspector,

CHAPTER 1

select Attributes. Enter Converter as the Title, and choose Movable Modal as the Window Class.



The window should look something like this:



Build and Run the Application

Now you'll run the application, and see how much the Carbon Event Manager does for you, even when you don't write any code.

1. In Project Builder, build the application.

Click the Build button at the top of the project window.



Project Builder may ask you if you want to save your modified files before building. If so, click Save All.

The Build panel slides down from the top of your project window. When the build is done, "Build succeeded" appears at the bottom of the project window. To remove the Build panel, click the Build tab.

2. Run the application.

Click the Run button.



The Run panel slides down and the application starts running.

3. Play with the running application.

The window appears, with the Fahrenheit field selected and the Convert button pulsing. Try typing into the text field, selecting some text, copying it, cutting it, and pasting it. All that works without your needing to write any code.

The menu bar has a full set of menus with the most common commands. Some of them, such as Open or Save As, are just stubs until you write code for them. Others, such as Cut, Copy, Paste, Quit, and Hide, work without any additional code.

Note that if you press the Convert button, nothing happens. You'll enable it in the next section.

4. Quit the application.

Now press the Quit button. The application quits as though you chose the Quit menu item, all without needing to write any code.

Add the Convert Button Handler

In this section, you'll write the code that handles the 'conv' event. You'll see what an application that uses the Carbon Event Manager looks like and learn how to create an command handler.

1. [“Look at the Existing Code”](#) (page 14)
2. [“Declare the Event Handler”](#) (page 15)
3. [“Install the Event Handler”](#) (page 15)
4. [“Write the Main Window Event Handler”](#) (page 16)
5. [“Write the ‘conv’ Event Handler”](#) (page 17)
6. [“Run and Build the New Application”](#) (page 18)

Look at the Existing Code

Now, go back to Project Builder by clicking its icon in the dock.

To look at the code, click on main.c in the project window's file list. If you can't see the main.c file, click the Files tab and open the Sources group.

You only need six functions to write an application that uses the Carbon Event Manager. Here they are, from the sample main.c file that's in your project with the error checking and comments removed:

```
CreateNibReference( CFSTR("main"), &nibRef );
SetMenuBarFromNib( nibRef, CFSTR("MainMenu") );
CreateWindowFromNib( nibRef, CFSTR("MainWindow"), &window );
DisposeNibReference( nibRef );
ShowWindow( window );
RunApplicationEventLoop();
```

Here's what the statements do:

CHAPTER 1

- `CreateNibReference` searches your application's package for a file called `main.nib` and opens it.
- `SetMenuBarFromNib` and `CreateWindowFromNib` set up the menu bar and main window from the nib file.
- `DisposeNibReference` closes the nib file.
- `ShowWindow` displays the main window, since it was set up to be hidden by default.
- `RunApplicationEventLoop` runs the main event loop.

Notice that you don't need to initialize any of the toolboxes, nor do you need to write your own event loop. All that's handled for you automatically.

Declare the Event Handler

Towards the top of `main.c`, after `#include <Carbon/Carbon.h>`, type (or copy and paste) this code:

```
#define kConvertCommand 'conv'
#define kConvertSignature 'conv'
#define kFahrenheitFieldID 128
#define kCelsiusFieldID 129
pascal OSStatus MainWindowCommandHandler( EventHandlerCallRef handlerRef,
    EventRef event, void *userData );
pascal void ConvertCommandHandler();
```

The `#define` statements create macros for the command, IDs, and the signature you entered earlier. And the declarations are for the functions that handle the 'conv' command.

Install the Event Handler

Now you'll install the handler that will handle the 'conv' command.

1. Declare some variables.

Towards the beginning of the main function, right after the `OSStatus err` statement, type (or copy and paste) this code:

```
EventTypeSpec commSpec = { kEventClassCommand, kEventProcessCommand };
```

CHAPTER 1

`controlSpec` specifies what kind of event handler you're installing. In this case, `kEventClassCommand` means it's in the class of command events, and `kEventProcessCommand` means the handler should be called when the command needs to be processed.

2. Install the handler.

Between the `ShowWindow` and the `RunApplicationEventLoop` functions, type (or copy and paste) this code:

```
InstallWindowEventHandler( window,
    NewEventHandlerUPP(MainWindowCommandHandler),
    1, &commSpec, (void *) window, NULL );
```

`InstallEventHandler` tells the Carbon Event Manager to call `MainWindowCommandHandler` whenever it receives a command from the main window. Notice that the second to the last parameter is for user data and that we're passing in a pointer to the main window. When the Carbon Event Model calls the command handler, it will pass this pointer as an argument, so the handler can access the text fields that are on the window.

Write the Main Window Event Handler

At the bottom of `main.c`, after the `main` function, type (or copy and paste) this function:

```
pascal OSStatus MainWindowCommandHandler( EventHandlerCallRef handlerRef,
    EventRef event, void *userData )
{
    OSStatus err = eventNotHandledErr;
    HICCommand command;

    GetEventParameter( event, kEventParamDirectObject, typeHICCommand,
        NULL, sizeof(HICCommand), NULL, &command );

    switch( command.commandID ) {
        case kConvertCommand:
            ConvertCommandHandler( (WindowRef) userData );
            err = noErr;
            break;
    }
    return err;
}
```


CHAPTER 1

```
}
```

This function is called whenever the Carbon Event Manager receives a command from the main window. It tries to handle the command, and returns `noErr` if it can or `eventNotHandledErr` if it can't. `GetEventParameter` retrieves the command and the switch statement checks whether this function can handle the command. If the command ID is `kConvertCommand`, it calls `ConvertCommandHandler`, which is described below, and returns `noErr`. If it can't handle the command, it returns `eventNotHandledErr` and the Carbon Event Manager tries to find someone else to handle the command.

Write the 'conv' Event Handler

After the `MainWindowCommandHandler`, type (or copy and paste) this function:

```
pascal void ConvertCommandHandler(WindowRef window)
{
    ControlHandle    fahrenheitField, celsiusField;
    ControlID        fahrenheitControlID =
                        { kConvertSignature, kFahrenheitFieldID };
    ControlID        celsiusControlID =
                        { kConvertSignature, kCelsiusFieldID };
    CFStringRef      text;
    Size             actualSize;
    double           fahrenheitTemp, celsiusTemp;

    GetControlByID( window, &fahrenheitControlID, &fahrenheitField );
    GetControlByID( window, &celsiusControlID, &celsiusField );

    GetControlData( fahrenheitField, 0, kControlEditTextCFStringTag,
                    sizeof(CFStringRef), &text, &actualSize );
    fahrenheitTemp = CFStringGetDoubleValue( text );
    CFRelease( text );

    celsiusTemp = ( fahrenheitTemp - 32.0 ) * 5.0 / 9.0;

    text = CFStringCreateWithFormat( NULL, NULL, CFSTR("%g"), celsiusTemp );
    SetControlData( celsiusField, 0, kControlEditTextCFStringTag,
                    sizeof(CFStringRef), &text );
    CFRelease( text );
    DrawOneControl( celsiusField );
}
```

CHAPTER 1

```
}
```

This function reads the value from the Fahrenheit text field, converts it to Celsius, and writes the value back out to the Celsius text field. Here's line-by-line description of what's happening:

- The two `GetControlByID` functions retrieve the Fahrenheit and Celsius text fields. The variables `fahrenheitControlID` and `celsiusControlID` contain the signature and IDs you entered.
- `GetControlData` retrieves the text in the Fahrenheit field as a `CFString`.
- `CFStringGetDoubleValue` converts that string to an floating-point number, and `CFRelease` releases the memory that the string used.
- The assignment statement computes the Celsius temperature.
- `CFStringCreateWithFormat` converts that floating-point number to a string, using the `printf`-style format string `"%g"`.
- `SetControlData` sets the Celsius text field to that string, and `CFRelease` releases the memory that the string used.
- `DrawOneControl` redraws the text field with its new value.

Run and Build the New Application

Now you'll run the application and watch your new command handler at work.

1. In Project Builder, build the application.

Click the Build button at the top of the project window.



Project Builder may ask you if you want to save your modified files before building. If so, click Save All.

2. Run the application.

Click the Run button.



The Run panel slides down and the application starts running.

CHAPTER 1

3. Play with the running application.

Try entering different temperatures into the Fahrenheit field and pressing Convert. Watch the equivalent Celsius temperature appear below.

4. Quit the application.

Create the Convert Menu Item

In this section, you'll add a menu item that performs the same command as the Convert button, and you won't need to write any code to do it. You'll learn how to use Interface Builder to add a menu.

Create a Commands Menu

First, you'll create the menu that will contain the Convert command.

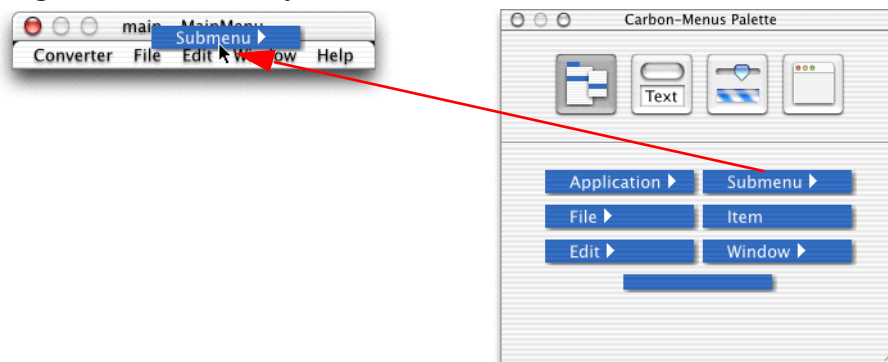
1. Go to Interface Builder by clicking its icon in the dock.
2. In the palette window, click the Menus button, which is the left-most button.

Here's what the menu palette contains:

- The Application, File, Edit, and Window elements are fully-loaded menus that you can drop into your application. Note that the template application already contains them.
- The Submenu element can be either a top-level menu that you add to your menu bar or a hierarchical menu that you add to another menu.
- The Item element is a single menu item that you can add to any menu.
- The blank element is a separator that you can add to any menu.

CHAPTER 1

3. Drag a Submenu item to your menu window, between Edit and Window.



4. Rename the menu to Commands.

Double-click the word Submenu so a blinking insertion point appears. Enter Commands and press Return.

Create a Convert Menu Item

Now, you'll add the Convert command to the menu and give it a command-key equivalent.

1. Open the Commands menu.

Click on the word Command, and Interface Builder displays the items it contains.

2. Rename the menu item to Convert.

Double-click the menu item so a blinking insertion point appears. Enter Convert and press Return.

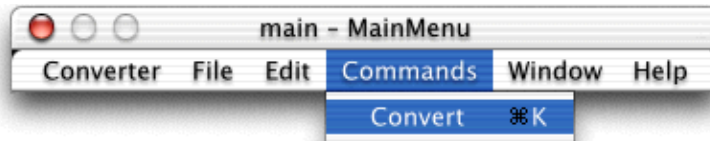
3. Assign Command-K as the menu item's command-key equivalent.

Double-click to the right of the Convert menu item so a box appears and type K.

You may have to try double-clicking a few places before you succeed. Try double-clicking from the right edge of the word Convert all the way to the right edge of the menu.

CHAPTER 1

The menu window should look something like this:



4. Enter conv as the menu item's command.

From the pop-up menu at the top of the Inspector, select Attributes. In the Command section, enter conv in the text field.



CHAPTER 1

Run and Build the New Application

Now you'll run the application, and try out the new menu command.

1. In Project Builder, build the application.

Click the Build button at the top of the project window.



Project Builder may ask you if you want to save your modified files before building. If so, click Save All.

2. Run the application.

Click the Run button.



The Run panel slides down and the application starts running.

3. Play with the running application.

Now try entering different temperatures into the Fahrenheit field and convert them by choosing the Convert command or pressing Command-K. Watch the equivalent Celsius temperature appear below.

4. Quit the application.

You're done!