HelloWorld: Creating a Project With Project Builder

This tutorial shows how to create a simple project with Project Builder. In it, you'll create a new project from a template, build and run it, then add a function to it, and build and run the new application. Along the way, you'll learn how to edit files, search for text, and fix errors.

This tutorial does not attempt to teach you Mac OS X programming. You'll create a Carbon application, but you do not need to be familiar with Carbon to benefit.

Create the Project

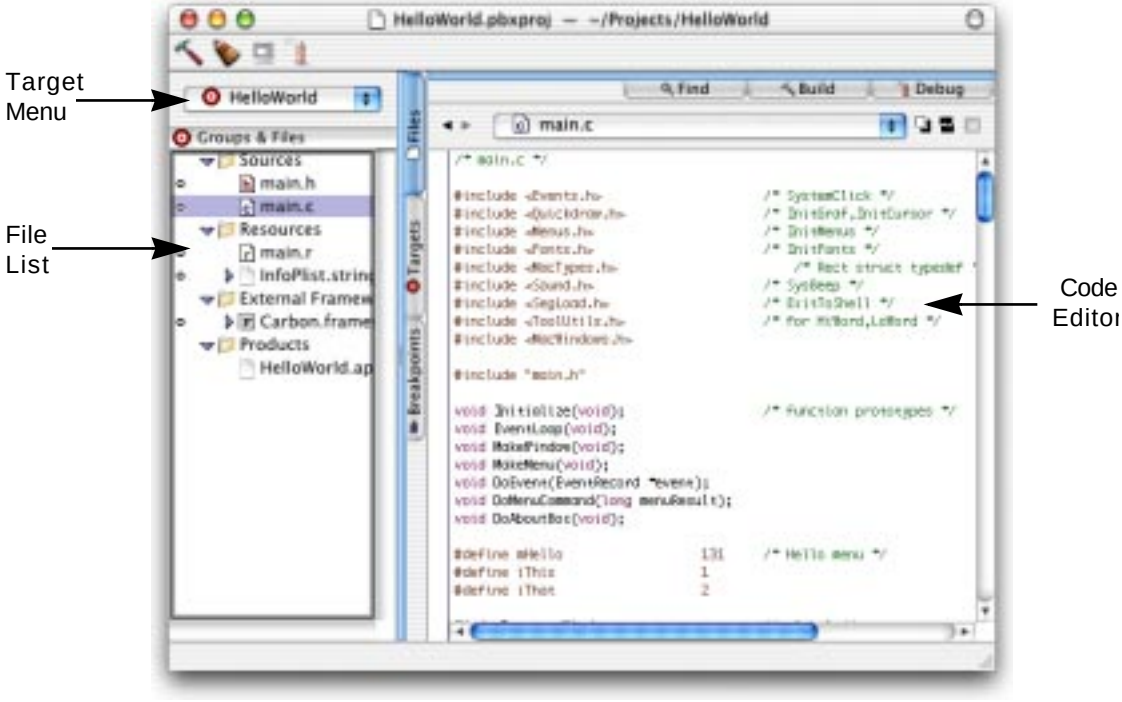
Choose File > New Project. Project Builder displays a dialog box with several template projects to choose from. Select Carbon Application, and click Next. Then enter `HelloWorld` as the project name, choose a location, and click Finish.

Project Builder creates a directory for you, places a project file and some source files in it, and opens the project's window. The project already contains sample source files that you can compile and run without change.

Now take some time to look at what's in this Project Builder project. If you already understand what projects, targets, and frameworks are, you can move ahead to "Build and Run the Sample Application" (page 18).

A Project Builder project contains two types of items: file references and targets. The files are in a list at the left of the project window and the targets are in a pop-up menu above the file list.

- Its files can be references to source files, resource files, libraries, and frameworks. The files themselves aren't in your poject. You can place related files together in groups. Moving files into groups does not change where they are located on disk.
- Its targets are things you can build from your project's files. A simple project, like this one, has just one target that builds an application. A complex project may contain several. For example, a project for a client-server software package could contain targets for a client application, a server application, a private framework that both applications use, and command-line tools that you can use instead of the applications. Another tutorial (to be written) describes how to manage projects with multiple targets.



This template project puts its files in four groups: Sources, Resources, External Frameworks and Libraries, and Products. These are also the four main types of files that a Project Builder project can contain. To see what's in them, click the triangles beside them.

- Sources: These are files that are compiled to produce object code. In Project Builder, header files are listed in the project window and are in this group. This project contains only two source files: `main.c` and `main.h`.
- Resources: These are files that contain resources or that can be compiled to produce resources. This project contains two: `main.r`, which contains the application's resources, and `InfoPlist.strings`,which contains some strings that can be localized. You'll learn more about `InfoPlist.strings` in the tutorial "AboutBox: Creating a Framework With Project Builder."
- External Frameworks and Libraries: These are different types of libraries. In Mac OS X, a framework is a shared library that's bundled with its header files and resources. If you click the triangle beside a framework, you'll see a list of its header files. This project contains one: `Carbon.framework`.
- Products: These are the items that the targets in your project can produce. It contains one file: `HelloWorld.app`.

You can move the files into any groups you want. The groups are there solely for your convenience and do not affect Project Builder's ability to find or compile the files.

Now go back to the Finder and look at the files in your project's folder.

`HelloWorld.pbxproj` is a bundle that keeps track of your project's files and targets. (It may be called `HelloWorld.pbproj`.) If it appears as a single file, you can double-click it to open your project. If it appears as a folder, open it and double-click the `project.pbxproj` file that's inside it.

`English.lproj` contains resources that are localized into English. In this case, it contains `InfoPlist.strings`. You can have `lproj` folders for other languages, too, like `French.lproj` or `Japanese.lproj`.

You'll notice that although `main.r` is in a different group from `main.c` and `main.h`, they're all in the same folder. Also notice that `Carbon.framework` isn't in here. It's in `/System/Library/Frameworks`. The project contains a reference to it.

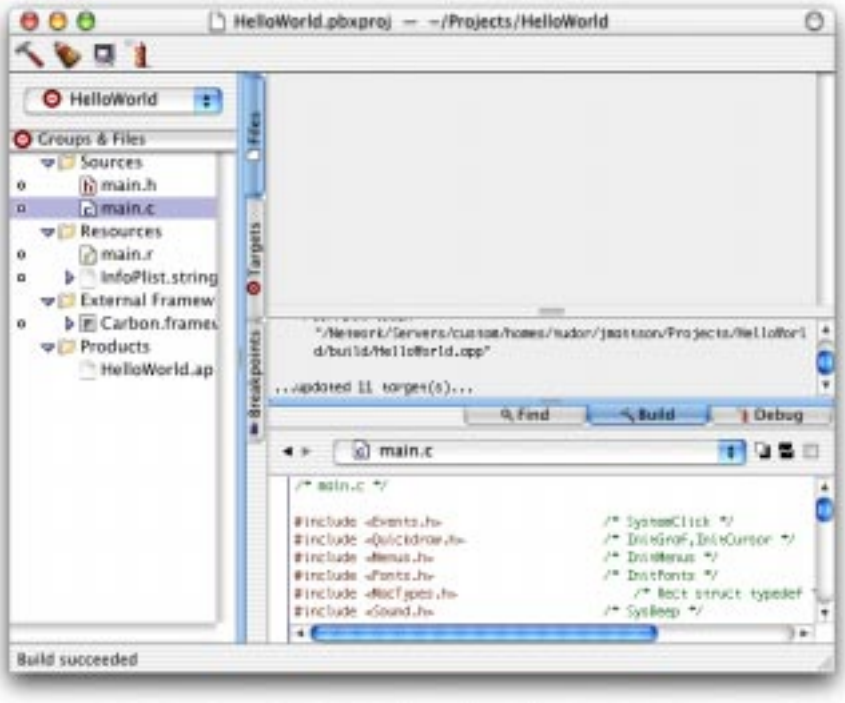
Build and Run the Sample Application

First, you'll build and run the sample application without modifying it, just to get a feel for Project Builder.

1. Build the application.

Click the Build button. It's the hammer in the upper-left corner of the project window.

The Build panel slides down from the top of your project window. The top part displays error messages and the bottom part displays the build's status. Because you haven't changed the source files, there shouldn't be any error messages.



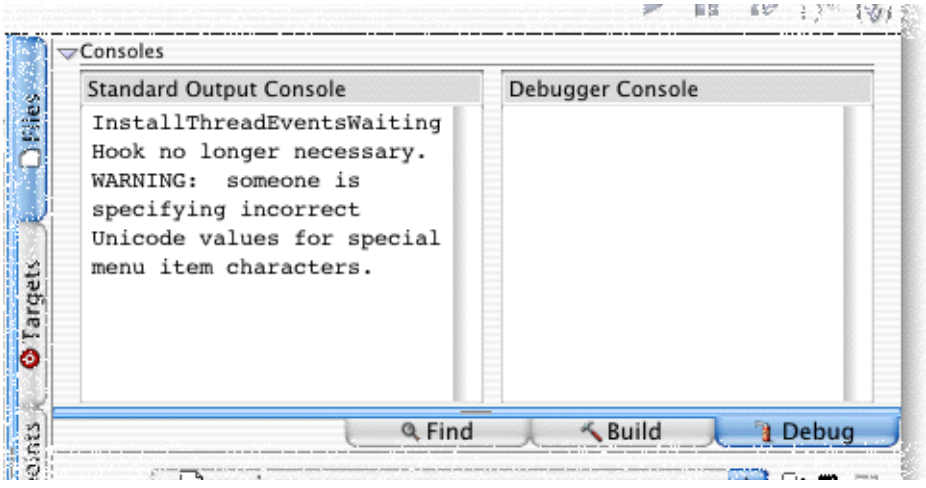
When the build is done, "Build Succeeded" appears at the bottom of the project window. To remove the Build panel, click the Build tab.

2. Run the application.

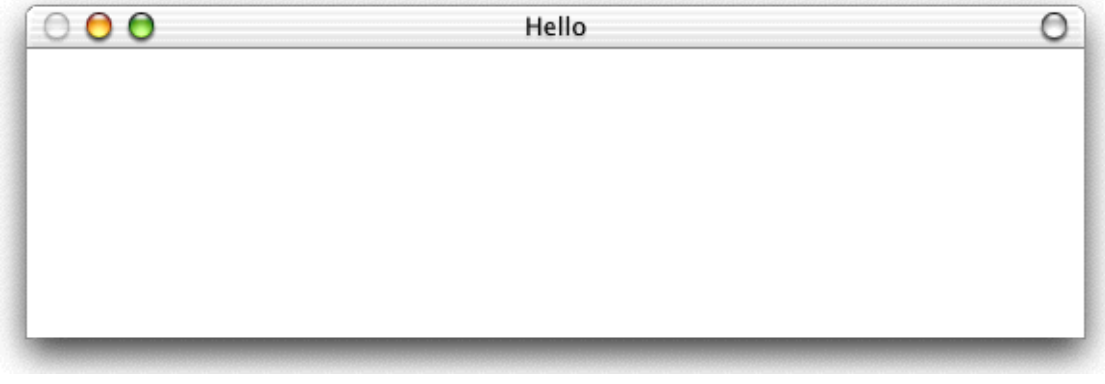
   Click the Run button. It's the computer monitor in the upper-left corner of the project window.

   Project Builder launches the application and drops down the Debug panel to display messages written to `stdout` and `stderr`. Sometimes it can be handy to

perform simple debugging by writing status information to these streams. Project Builder also lets you perform more sophisticated debugging, as shown in the tutorial "DebugApp: Debugging an Application with Project Builder."



The application displays an empty window. To quit it, press Command-Q.



Write the New Function

Now you'll write a function that prints "Hello World!" in that empty window.

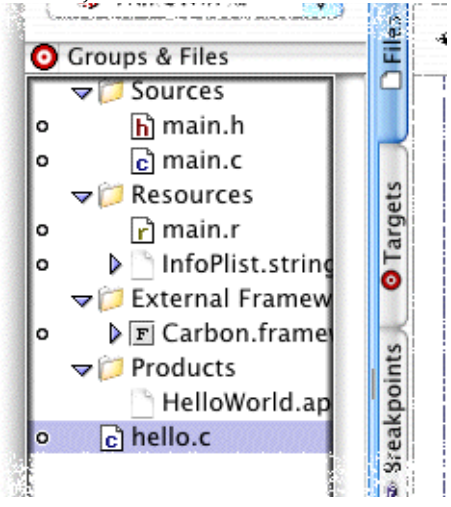
1. "Write the Source File" (page 21)
2. "Write the Header File" (page 23)

Write the Source File

1. Create the source file.

   Choose File > New File. Select Empty File and click Next. Enter `hello.c` as the filename and click Finish.

   Project Builder creates a new file and places a reference to it in your project



The file is automatically opened in the project window's editing pane. If you want you can open it in a separate window by double clicking it in the file list.

2. Enter the source code.

   You can either type in this code or copy and paste it from this document.

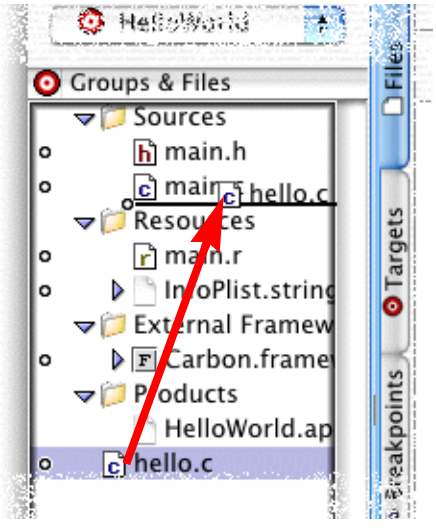
Listing 2-1 The hello.c file

```
#include <QuickDraw.h>
#include <MacWindows.h>

void printHello(WindowPtr aWindow)
{
    short int fNum;

    SetPort(GetWindowPort(aWindow)) // <<-- ERROR! No semicolon
    GetFNum("\pTimes", &fNum);
    TextFont(fNum);
```

```
    TextFace(bold + italic);
    TextSize(48);
    MoveTo(5,50);
    DrawString("\pHello World!");
}
```

Notice that the function contains a syntax error. You'll fix it later on.

If you want, you can move the file into the Sources group. Just drag the file's icon.



Write the Header File

1. Create the header.

   Choose File > New File. Select Empty File and click Next. Enter `hello.h` as the filename and click Finish.

2. Enter the header's code.

   Again, you can either type in this code or copy and paste it from this document.

Listing 2-2 The hello.h file

```
void printHello(WindowPtr aWindow);
```

If you want, you can move the file into the Sources group as before.

Call the New Function From the Main Application

Now you'll add calls to `printHello` in the `main.c` file. You'll use Project Builder's Find command to determine where to put the calls.

1. "Include the New Header File" (page 24)
2. "Update MakeWindow" (page 24)
3. "Update DoEvent" (page 25)

Include the New Header File

Scroll to the beginning of the file `main.c`. After `#include <MacWindows.h>` (the last #include statement), add the line `#include "hello.h"`

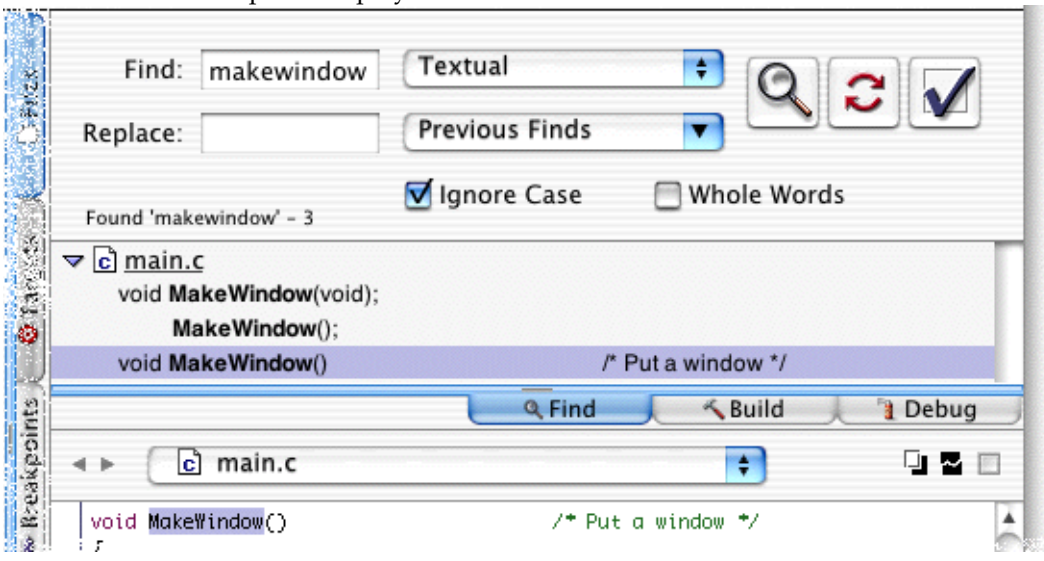
Update MakeWindow

In this step, you'll edit `MakeWindow` to call `printHello`, so the application prints "Hello World!" when it's first started.

1. Find the function definition for `MakeWindow`.

   Click the Find tab, near the top of the project window. The Find panel drops down. In Project Builder, the Find panel appears right in the project window, so you can find code and change it without opening several windows. The top of

the Find panel contains fields and buttons for performing the search. The bottom of the Find panel displays a list of search results.



Enter `MakeWindow` and click the Find button (the magnifying glass). The results list displays every occurrence of the string in your project.

Click the entry for the definition of `MakeWindow`. (It's the one without a semicolon at the end of the line.) The code for `MakeWindow` appears in the code editor, and its first line is selected.

2. Replace the call to `SetPort` with a call to `printHello`.

   Scroll down until you see this line. It appears after an else statement:

   ```
   SetPort(GetWindowPort(myWindow));
   ```

   Replace it with this line:

   ```
   printHello(myWindow);
   ```

Update DoEvent

In this step, you'll edit `DoEvent` to call `printHello`, so the application prints "Hello World!" whenever it updates its window.

1. Find the function definition for `DoEvent`.

Search for `DoEvent` and click the entry for its definition.

2. Add a call to `printHello`.

   Scroll down until you see the lines `BeginUpdate((WindowPtr)event->message);` and `EndUpdate((WindowPtr)event->message);` . They should be near the end of the function. Between these two lines, add this line:

   ```
   printHello(myWindow);
   ```

Fix, Build, and Run the New Application

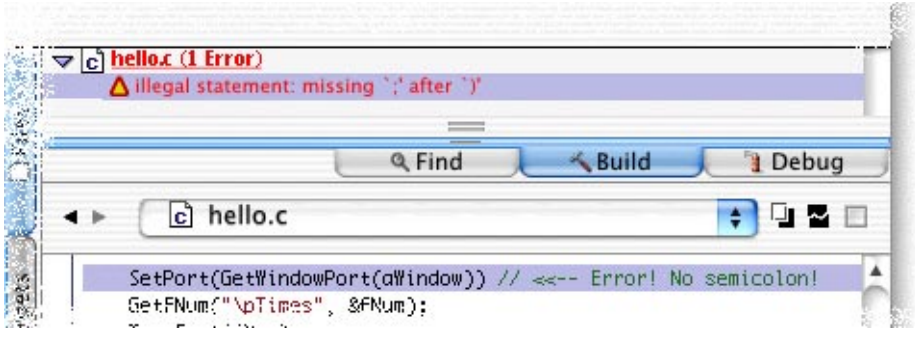
Finally, you'll fix the error in the `hello.c` file, build your application, and run it.

1. Build the application.

   Click the Build button.

   The Build panel drops down and Project Builder starts building your project. An error message appears in the results list.

   Each file with errors is listed in the results list, with its errors listed underneath.

2. Fix the error.

   Click the error message. The line with the error is selected in the code editor. Add a semicolon at the end of the line.

3. Build the application again.

Click the Build button again. Now, Project Builder builds the project without any problem.

4. Run the application.

   Click the Run button.

   Project Builder runs the application, which now displays a window that says "Hello World!"