Using Project Builder to Build and Debug a Carbon Application

This document describes how to build and debug a Carbon application on Mac OS X. The first two sections describe how to create a Mach-O Carbon application that you can debug in Project Builder, Apple's development environment for Mac OS X. These sections tell you how to proceed through the rest of the document:

- Building a Mach-O Carbon Application in Project Builder describes how to build a Carbon project in Project Builder. This is best if your application will take advantage of features available only in Mac OS X and will run only under Mac OS X.
- Using a CodeWarror Carbon Project in Project Builder describes how to use CodeWarrior to build a Carbon application that you can debug in Project Builder. This is best if you eventually want to create an application that runs under both Mac OS 8 and Mac OS X. This method requires that you run Project Builder in Mac OS X and run CodeWarrior in either the Blue Box or in Mac OS 8.

If your application requires CFM (for example, it uses plug-ins), see this section:

- Debugging a CFM Application with GDB describes a work-around that lets you debug a CFM application with GDB. Note that you will be able to perform low-level debugging only. You cannot view the symbol names in your application or step through your application line-by-line.

If you have access to two machines networked to each other, one running Mac OS 8 and the other running Mac OS X, you may be able to use CodeWarrior both to build and to debug your application. For more information, contact Metrowerks.

Building a Mach-O Carbon Application in Project Builder

You can both build and debug a Mach-O Carbon application in Project Builder, Apple's development environment for Mac OS X.

Creating a Carbon application in Project Builder is best if you're creating an application that will be used only under Mac OS X and that will take advantage of the features that are only found in Mac OS X. You won't need to use the Blue Box or switch between Mac OS 8 and Mac OS X. Note that Project Builder cannot currently create Mac OS 8 applications.

To import a CodeWarrior project:

1. Import it using Creating a Project Builder Project from a CodeWarrior Project.
2. Edit your code until you can compile it under Project Builder. For more information on Project Builder and its compiler, choose Help > Project Builder Help, and follow the link for "Using the Tools."
3. Debug it using the Debugging A Mach-O Carbon Application

To create a new Carbon project:

1. Launch Project Builder, choose Project > New and choose Carbon Application as the Project Type.
2. Write your application. For more information on Project Builder and its compiler, choose Help > Project Builder Help, and follow the link for "Using the Tools."
3. Debug it using the Debugging A Mach-O Carbon Application

Using a CodeWarror Carbon Project in Project Builder

This section describes how to create a CodeWarrior Carbon project that you can debug with Project Builder. To do this, you'll use both CodeWarrior and Project Builder. Your CodeWarrior and Project Builder projects will use the same source code and reside in the same folder.

These are the steps you'll follow:

Step 1: Preparing Your CodeWarrior Project

Step 2: Creating a Debuggable Mach-O Application

Step 3: Creating a Project Builder Project from a CodeWarrior Project

Step 4: Debugging A Mach-O Carbon Application

Preparing Your CodeWarrior Project

Before you start, port your project to Carbon following the directions in "The Carbon Porting Guide." Then, follow these recommendations, which ensure that Project Builder and CodeWarrior will be able to share the same source files:

- Remove any spaces from the names of the folders and volumes that lead up to your project. For example, if your project is in the sub-folder "My Carbon App" in the folder "My Projects" on the hard disk "Mac HD", you'll need to change the names to something like "MyCarbonApp", "MyProjects", and "MacHD".
- If possible, remove any spaces and non-alpha-numeric characters from the names of the files and folders that the project contains.

  If any spaces or non-alpha-numeric characters remain in a file name, Project Builder will create a symbolic link to that item. The link's name will be the same as the original's name with the offending characters changed to underscores (_).

Caution: If the names of two items differ only by non-alpha-numeric characters, Project Builder will treat them as the same item.

Note that a symbolic link is *not* an alias. Under Mac OS 8, a symbolic link looks like an ordinary file and does not point to the original item.

- Organize your project so related files are in the same folder. You may even consider putting all the files in the same folder.

  Unlike CodeWarrior, Project Builder always shows your source files in their folders. You cannot see files from different folders in the same list.

- You may want to change your application so it always opens an untitled document when launched. Carbon applications currently do not appear in Mac OS X's Application menu, so the only way to bring a Carbon application to the front is to click one of its windows.

Creating a Debuggable Mach-O Application

To debug your application in Project Builder, you need to set certain options before building it. This section explains how.

1. Open your project in CodeWarrior, and select the Mach-O target.
2. Choose Project > Enable Debugger.
3. Open the Settings dialog for your Mach-O target, and in the PPC Mach-O Linker panel, turn off the "Emit full path in Stab debug info" option. Click Save and close the dialog
4. Choose Project > Remove Object Code.
5. Choose Project > Make.

Creating a Project Builder Project from a CodeWarrior Project

This section explains how to import your CodeWarrior project into Project Builder. When you're done, you'll have both CodeWarrior and Project Builder projects in your project folder.

1. Open Project Builder. It's in /System/Developer/Applications.

2. Choose Project > New from Source. The New from Source dialog appears.
3. In "Locate Project Path," click Browse, and choose the folder that contains your CodeWarrior project. When you're finished, click Next.

Note: If the New from Source dialog disappears, choose Project Builder from the Application menu. It should re-appear.

4. In "Define Project Types," make sure that Project Builder chose the correct projects types for your project and its folders, as follows:
   - The topmost project folder should be "Carbon Application."
   - Any library should be "Library."
   - Any subfolder that contains source files should be "Carbon Component."
   - Any subfolder that doesn't contain source files should be "(Ignored)."

   When you're finished, click Next.
5. In "Locate the directories that contain header files," choose the folders that contain header files, and turn on the "Also look for header files in subdirectories" option.

   All the folders that contain headers for your application should be in the bottom list. To move a folder from the top list to the bottom list, double click it. To add a folder that isn't listed, click the Browse button.

   When you're finished, click Next.
6. In "Choose Prefix File," enter the name of your project's prefix file. This field needs a value only if you plan to build your application within Project Builder. You can find your project's prefix file in CodeWarrior. Open the Settings dialog for your project, go to the C/C++ Language panel, and look at the "Prefix File" field.
7. In "Locate the executables created by this project," double-click the Mach-O debuggable executable that CodeWarrior created, so it moves to the bottom list. When you're finished, click Next.
8. When Project Builder has finished importing your project, click Finish. In the confirmation dialog that appears, click Yes.

Debugging A Mach-O Carbon Application

This section describes how to run your application in Project Builder's debugger, GDB. To debug an application:

1. Open the Launch Panel.

   In the Project window, click the Launch Panel button.

   

2. Make sure the proper executable and source directories are selected.

   In the Launch panel, click the Launch Options button.

   

   In the top view, select the debuggable Mach-O executable. If you can't see it in the list, use the Add button.

   In the bottom list, click the Sources tab and make sure all your source directories are listed. If you can't see them, use the Add button.

3. In the Launch Panel, click the Debug button to start the debugger.

   

   The Launch Panel expands to show three views:

   - The stack view lists your program's stack frames. Click a frame to view its source code.
   - The source view displays the source code for the currently selected frame.
   - The GDB Command view displays GDB's status. Anytime the debugger is running, you can enter a GDB command here. For a list of GDB help topics,

enter help. For more information on a GDB command, enter help <command-name>

Project Builder loads the GDB debugger and displays its status in the GDB Command view. When you see the "(gdb)" prompt, the debugger has finished loading but hasn't started your application yet.

4. Set some breakpoints.

   In this version, the Suspend button doesn't always work. It's especially useful to set a breakpoint right at the beginning of your application's main routine. For more information, see Using Breakpoints.

5. Click the Start button, to start your application.

   If the debugger cannot run your application, the application may not have executable privileges set. In the Workspace Manager, select your application, choose Tools > Inspector > Access, and make sure each of the Execute boxes have checkmarks in them. Close the Inspector window and press the Start button again.

6. Debug your application.

   You can find help in these sections:

   Navigating Code

   Using Breakpoints

   Controlling Execution

   Setting and Viewing Data

Note: Carbon applications currently do not appear in the Application menu. To bring a Carbon application to the front, click one of its windows. If your application isn't displaying a window, you cannot bring it to the front.

7. When you're done, click the Stop button.

8. Revise your application and debug it again.

Using CodeWarrior in the Blue Box, revise and rebuild your application, then return to Project Builder to debug it again.

Note: If you add or remove files from your CodeWarrior project, you'll have to update your Project Builder project as well. Project Builder does not update it automatically. To add a file, choose Project > Add File, choose the file's type, and then choose the file. To remove a file, select it and choose Project > Remove Files. If you need to make many changes, it may be easier to re-import the project, as described in Creating a Project Builder Project from a CodeWarrior Project

Navigating Code

This section describes how to find functions and code in Project Builder.

To find a source file in your project folder:

- In the project window's browser, click Sources, then click your file's name.

To find a source file in a subfolder within your project folder:

- In the project window's browser, click Subprojects, click the subfolder's name, click Sources, then click your file's name.

To search your project for a function or variable definition:

- Press Command-Shift-F, choose Definitions from the pop-up menu, and enter the symbol's name.

  You can also find Mac OS Toolbox declarations this way. However, at this time, you cannot find C++ methods.

To search your project for a string:

- Press Command-Shift-F, choose Textually from the pop-up-menu, and enter the string.

To go to the function or method for a frame in your application's stack:

- Click the frame in the Stack view.

Using Breakpoints

This section describes how to set regular breakpoints, temporary breakpoints, and conditional breakpoints. Take note of the following:

- Project Builder does not save breakpoint information. Each time you start the debugger, re-set your breakpoints.
- When the debugger pauses at a breakpoint, there may be a significant pause before it brings Project Builder to the front.

To set a breakpoint:

- In a source code view, double-click in the gray bar beside the line. (If there is no gray bar, the debugger isn't running. See Debugging A Mach-O Carbon Application.)

To remove a breakpoint:

- Drag the breakpoint's arrow out of the gray bar.

  OR

- Click the Task Inspector button, and use the Breakpoints tab view.

To view, enable, or disable breakpoints:

- Click the Task Inspector button, and use the Breakpoints tab view.

To see complete information on your breakpoints, including their ID numbers, the number of times they've been hit, and the conditions or commands that are attached to them:

- In the GDB Command view, enter info breakpoints.

To ignore a breakpoint for a specified number of times:

- In the GDB Command view, enter ignore <bp-id> <count>.

<bp-num> is the breakpoint's ID number, which you can find in the Task Inspector. <count> is the number of times to ignore it before pausing at it again. For example, ignore 3 10 ignores breakpoint number 3 the next 10 times it's reached, and then pauses at it on the 11th time.

To set a temporary breakpoint, which GDB pauses at once and then removes:

- In the GDB Command view, enter tbreak <spec>.

  <spec> is either a function name or a filename and line number, separated by a colon. For example, tbreak main.c:10 breaks at the tenth line of main.c, and tbreak PrintStatus breaks at the first line of the function PrintStatus.

To attach a condition to a breakpoint, so GDB pauses at it only when the condition is true:

- In the GDB Command view, enter condition <bp-id> <cond>.

  <bp-id> is the breakpoint's ID number, which you can find in the Task Inspector. <cond> is a valid boolean expression in the language you're debugging. For example, condition 3 (ptr==nil) stops at the breakpoint number 3 only if ptr is nil.

To attach GDB commands to a breakpoint, so GDB executes them whenever it reaches the breakpoint:

- In the GDB Command view, enter command <bp-id> , your commands, and then end.

  <bp-id> is the breakpoint's ID number, which you can find in the Task Inspector. The commands can be any valid GDB commands. Enter each one on a separate line. Some useful ones are print, to print an expression's value, and continue, to continue without pausing. If silent is the first command, GDB prints only your commands' output and nothing else.

  For example, this sample prints the value of i whenever it hits breakpoint number 3. It prints nothing else and continues immediately.

```
command 3
silent
print i
continue
end
```

Controlling Execution

To continue executing after a pause (such as hitting a breakpoint):

- Click the Continue button. (If there is no Continue button, the application is already executing.)

To step over a function or method:

- Click the Step Over button.

To step into a function or method:

- Click the Step Into button.

To step out of a function or method:

- In the GDB Command view, enter finish.

Setting and Viewing Data

To print a variable that appears in source code:

- Select the variable in a source view, and click the Print Value, Print Reference, or Print Object button.

  The Print Value button prints the variable's value.

The Print Reference button prints the value that the variable points to.

The Print Object button prints the self-description for the variable's object. It's useful only when you're debugging Java or Objective-C code.

To print the value of any valid expression:

- In the GDB Command view, enter print <expr>.

  <expr> is a valid expression in the language you're debugging. For example, print aRect.width and print *ptr.

  Here are some useful functions:

  - GDBShowControlHierarchy displays the control hierarchy for the specified window. For example p GDBShowControlHierarchy(myWindow).
  - GDBShowControlInfo displays the contents of the specified control. For example, p GDBShowControlInfo(myButton).
  - GDBShowDialogInfo displays the contents of the specified dialog. For example, p GDBShowDialogInfo(myDialog).
  - GDBShowMenuList displays the standard and hierarchical menus currently inserted in the menu bar. For example, p GDBShowMenuList().
  - GDBShowMenuInfo displays the contents of the specified menu. For example, p GDBShowMenuInfo(fontMenu).
  - GDBShowMenuItemInfo displays the contents of the specified menu item. For example, p GDBShowMenuItemInfo(editMenu, cutItem).
  - GDBShowPasteboardTypes displays the data types that are currently on the pasteboard. For example, p GDBShowPasteboardTypes().

To change the value of any variable:

- In the GDB Command view, enter set <var> = <expr>.

  <var> is the name of a variable in your program. <expr> is a valid expression in the language you're debugging. For example, set i = 10 .

Debugging a CFM Application with GDB

Although GDB cannot directly debug a CFM application, there is a work-around that lets you perform low-level debugging on a CFM application. You'll use GDB to debug LaunchCFMApp, a Mach-O program that launches CFM applications.

This work-around has these features and limitations:

- The application must be on a UFS volume. If you built your application on an HFS or HFS+ volume, move it to a UFS volume using FTP.
- You can set breakpoints at Mach-O functions. Since the Carbon library is Mach-O code, you can set breakpoints at Carbon functions. However, you cannot set breakpoints at CFM functions, including those in your application.
- You can see a function backtrace, which includes both CFM and Mach-O functions, with the bt command. Your CFM functions won't have names associated with them, though, since GDB cannot use the symbol names in a CFM application.
- You can examine the memory contents at any address with the x command. However, you cannot view variables or expressions, since GDB cannot use the symbol names in a CFM application.
- You cannot step through your application's code.

To debug your CFM application:

1. Launch the Terminal application, in /System/Administration/Terminal.app.
2. Enter gdb /usr/Carbon/bin/LaunchCFMApp.

   GDB loads the LaunchCFMApp program.
3. If you want, set breakpoints at any Carbon function with the br command.

4. At the GDB prompt, enter r <app-pathname>, where <app-pathname> is the full pathname for your CFM application.

   LaunchCFMApp launches your application.

   To pause your application's execution at any time, press Control-C in the Terminal application. To continue your application, enter cont. For more information on GDB, enter help.