

# Player/Stage Xcode iPhone Project Setup

Paul Solt

[PaulSolt@gmail.com](mailto:PaulSolt@gmail.com)

Release 1.0: 8-20-09

The goal of this project was to use the Player/Stage robotics code on the iPhone to communicate and control robots. I discuss how to setup the Xcode development environment. There are two example Xcode projects. The first one is an Objective-C project that wraps around the C++ Player/Stage code. The second project is a very primitive C++ program running on the iPhone without any UI. Both of these Xcode projects are fully documented and will serve as a starting point to iPhone Player/Stage development.

## Table of Contents

<b>Player/Stage Xcode iPhone Project Setup .....</b>	<b>1</b>
<b>Requirements .....</b>	<b>2</b>
<b>Setup the Source Folders .....</b>	<b>2</b>
<b>Global Xcode Settings: .....</b>	<b>2</b>
<b>Project Settings.....</b>	<b>3</b>
<b>Create Multiple Build Targets .....</b>	<b>5</b>
<b>Setup Library Dependencies, Linking, and Header Search Paths.....</b>	<b>5</b>
<b>iPhone Developer Profile Settings.....</b>	<b>8</b>
<b>How to Run the App: .....</b>	<b>8</b>
<b>RobotSample .....</b>	<b>9</b>
<b>BasicRobotSample.....</b>	<b>10</b>
<b>References.....</b>	<b>11</b>

# Requirements

1. Install Player/Stage 2.03 or 2.1
  - a. Use Macports: <http://www.macports.org/>
  - b. <http://paulsolt.com/2009/04/playerstage-macports-and-iphone/>
2. Install the Xcode iPhone 3.0 SDK
  - a. <http://developer.apple.com/iphone/>
3. Get a iPhone developers license if developing for the actual device.  
Developing with the simulator is free.

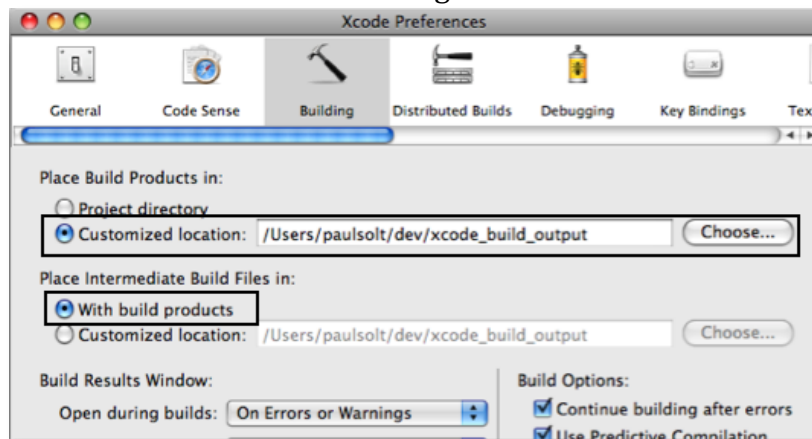
## Setup the Source Folders

1. Create a dev folder on your machine at /Users/USERNAME/dev/
2. Place the projects within the folder with the directory structure.
  - a. /Users/USERNAME/dev/xcode\_build\_output
  - b. /Users/USERNAME/dev/player\_2\_1
  - c. /Users/USERNAME/dev/player\_2\_0\_3
  - d. /Users/USERNAME/dev/ROBOT\_PROJECTS

## Global Xcode Settings:

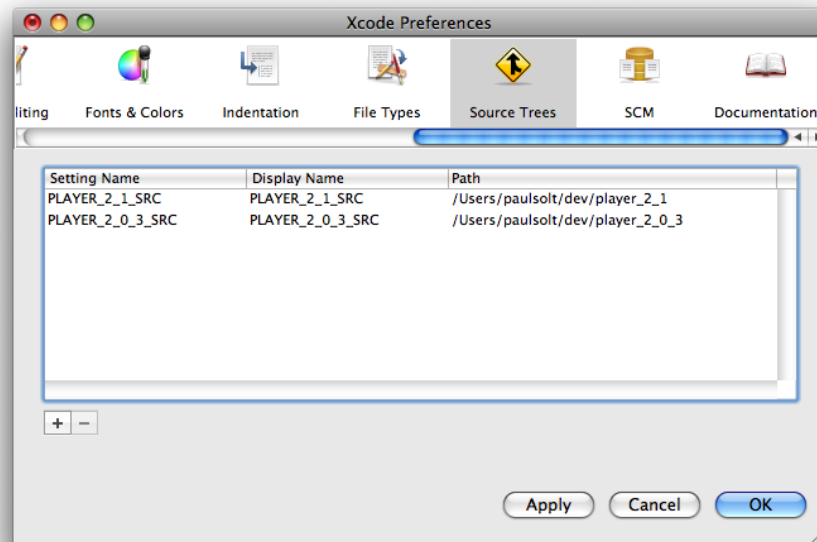
The iPhone projects use cross-project references to link against the iPhone Player/Stage static libraries that are included. There is basic support for Player/Stage versions: 2.1 and 2.03. The first step is to create a common build location for your Xcode projects. Below are the steps to setup the **Shared Build Output**, in addition to setting up **Source Tree** variables to use in the project settings. These variables can be used within Xcode project settings like so: \$(VARIABLE)

1. Use a **Shared Build Output** directory
  - a. Xcode -> Preferences -> Building



- i. Set *Place Build Products in:* to *Customized location:* and choose the directory path
- ii. Set *Place Intermediate Build Files in:* to *With build products*

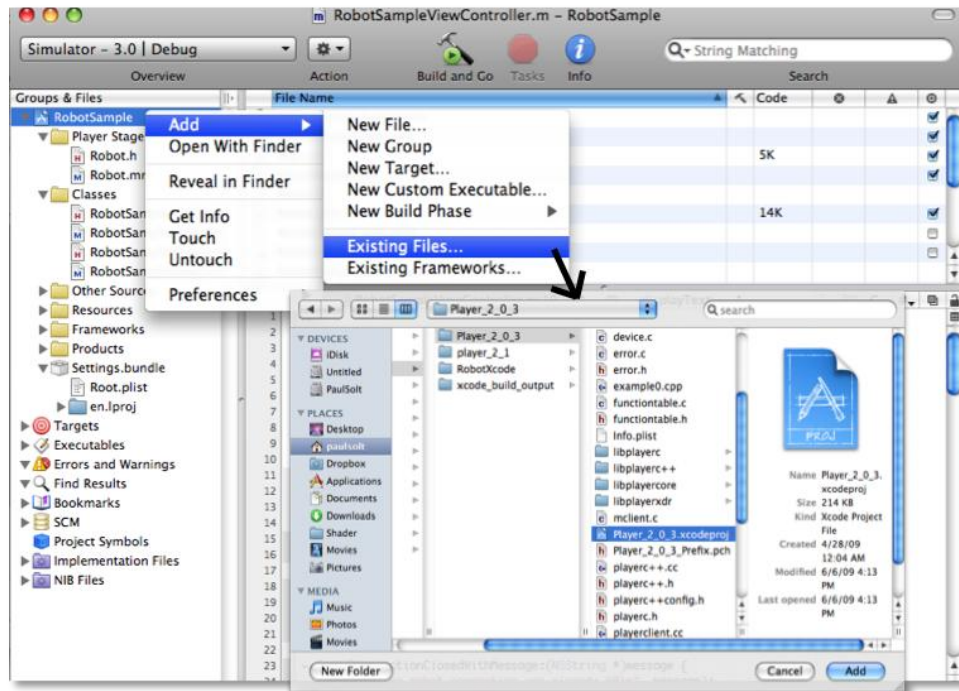
2. Add **Source Tree** variables for the reference project locations
  - a. Xcode -> Preferences -> Source Trees



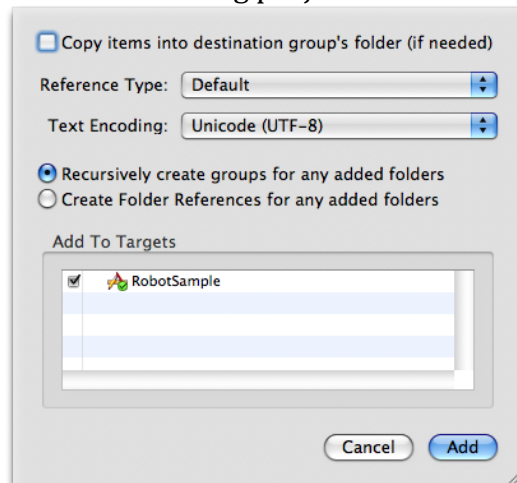
- i. Add new variables by pressing the “+” button and use the path locations from Setup the Source Folders with the following variables.
  1. PLAYER\_2\_1\_SRC
  2. PLAYER\_2\_0\_3\_SRC
  3. **Note:** Don’t use special characters, since Xcode has issues with them.

## Project Settings

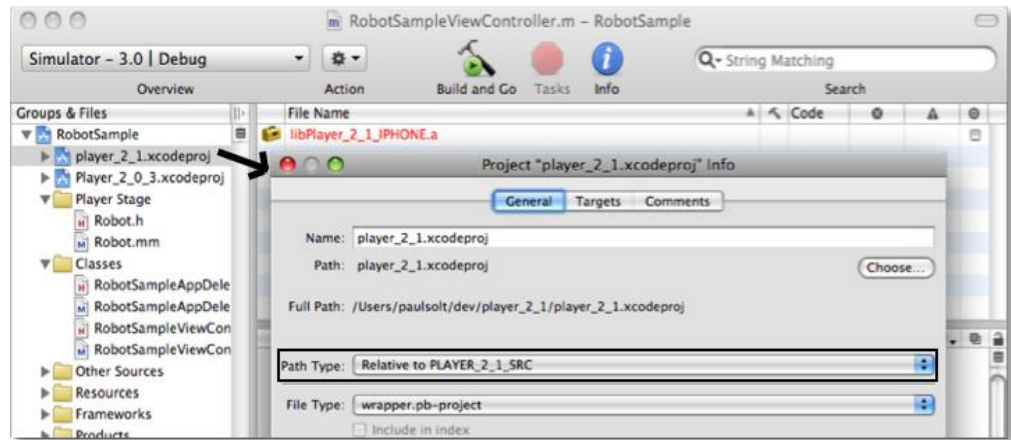
1. Setup the **Cross-Project Reference(s)**
  - a. Add the existing Xcode projects. Look under *Groups and Files* and right-click on the **Project Name** (RobotSample) -> *Add -> Existing Files...*



- b. Navigate to your Player/Stage Xcode projects and add both of the *Player\_2\_0\_3.xcodeproj* and *player\_2\_1.xcodeproj* Xcode projects.
- c. Note: Do **NOT** check the *Copy items into destination group's folder* since we are using project references and not a physical copy.



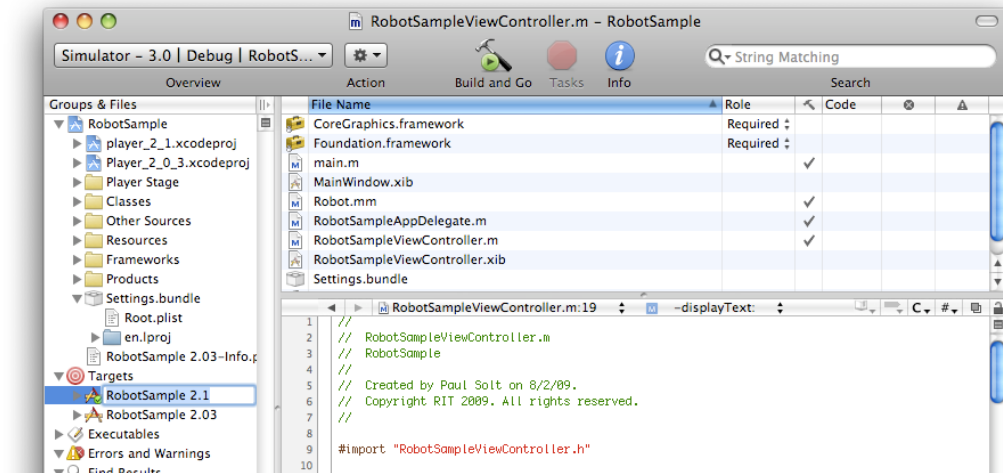
- d. Select the cross project references for **both** Player projects. Double-click on the project reference and then select *Get Info* (press Cmd+i). Change *Path Type* to be **relative** and use the **Source Tree** variables we setup in the previous section.



## Create Multiple Build Targets

Depending on what you need you may want either Player 2.03 or Player 2.1. In these projects I added support to both libraries. The different robots I've worked with use different versions of Player/Stage. To support these different versions we'll use Build Targets. A **Build Target** contains the settings to create the executable for your application.

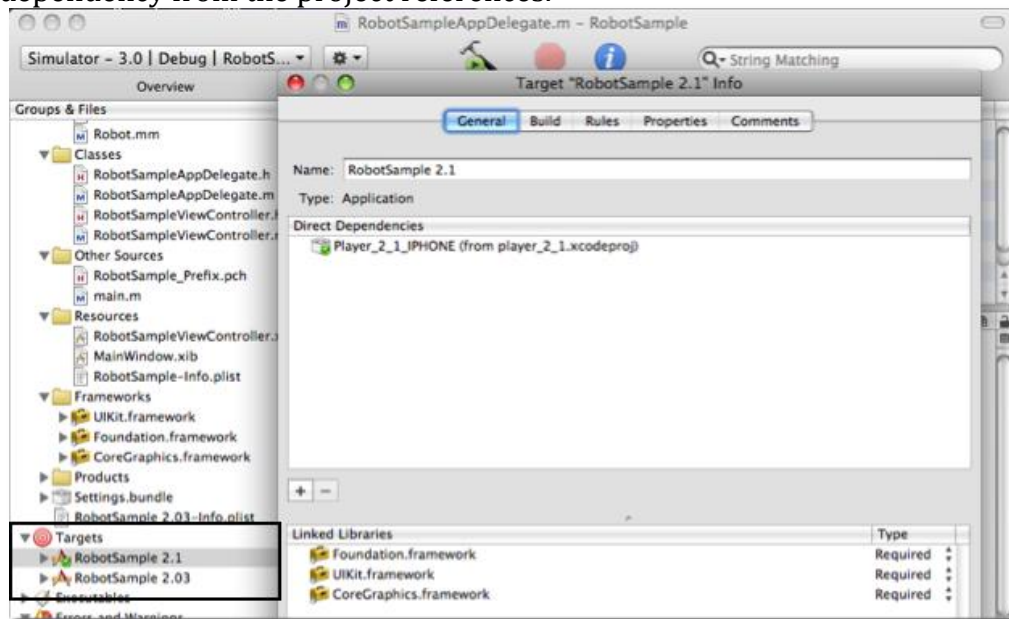
1. Duplicate the *RobotSample* Target and rename it *Robot Sample 2.1*
2. Rename the *RobotSample* Target to *RobotSample 2.03*



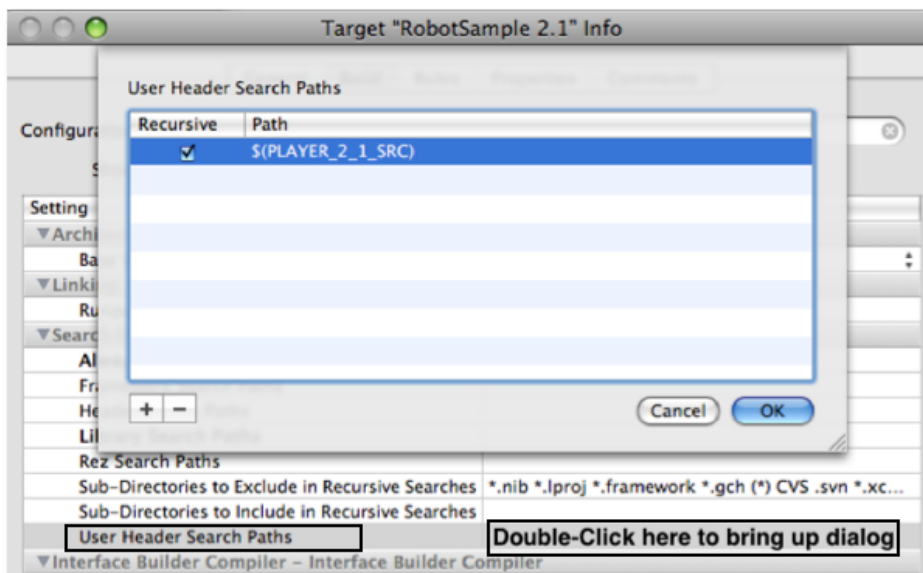
## Setup Library Dependencies, Linking, and Header Search Paths

The Build Targets are not linked to the Player/Stage static libraries. I will show how to finish linking and referencing the static libraries.

1. On the left pane under *Groups & Files*, click on **Targets** and then double-click on the RobotSample 2.1 target to access the *Get Info* view (Cmd+i). In the *Direct Dependencies* pane press the “+” and add the Player\_2\_1\_IPHONE dependency from the project references.

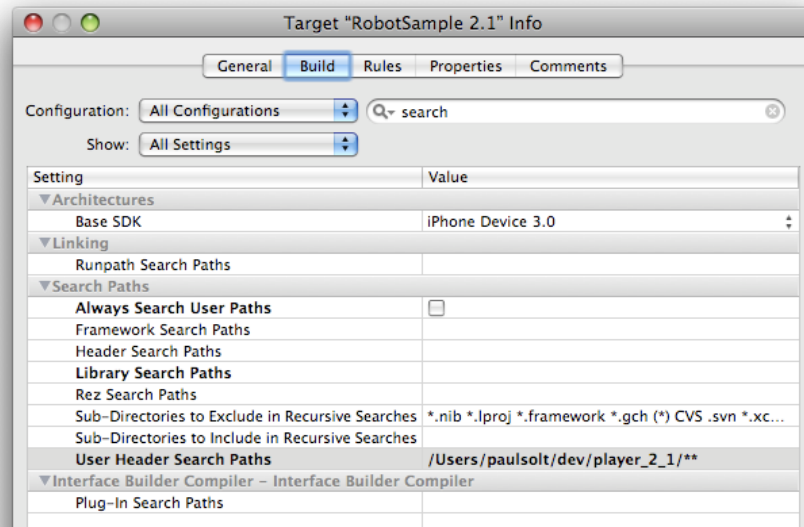


2. Click on the *Build* tab and search for “**Search**”. Double-click next to *Header Search Path* to edit the value. Set the path to the **Source Tree** variable **\$(PLAYER\_2\_1\_SRC)** and select the checkbox for **Recursive**.

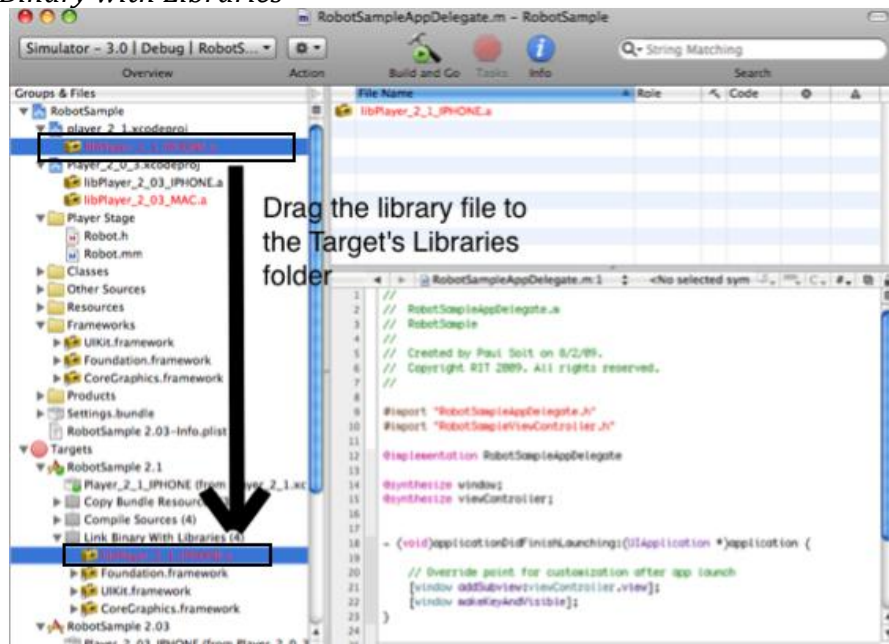


3. You should see the full path with “\*\*” appended to the end, which indicates the search path is recursive. For example:

/Users/paulsolt/dev/player\_2\_1/\*\*



4. **Repeat** the previous steps for the *RobotSample 2.03* target. It should use the **Player\_2\_03\_IPHONE** Direct Dependency and the **\$(PLAYER\_2\_03\_SRC)** Source Tree variable with **Recursive** checked.
5. Select and drag the static libraries from the cross-project references to the matching targets.
  - a. Drag **libPlayer\_2\_1\_IPHONE.a** to *Targets -> RobotSample 2.1 -> Link Binary with Libraries*

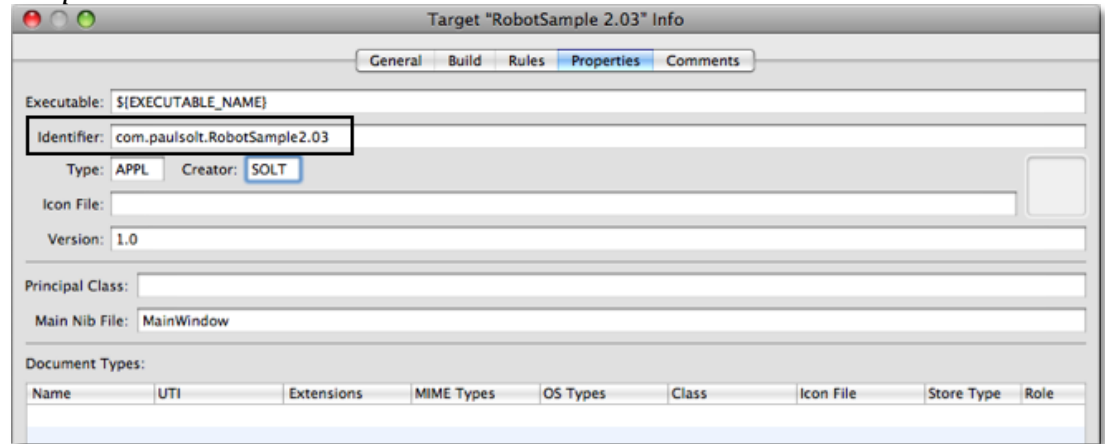


- b. Repeat the process and drag **libPlayer\_2\_03\_IPHONE.a** to *Targets -> Robot Sample 2.03 -> Link Binary with Libraries*



# iPhone Developer Profile Settings

1. Edit the **Build Target Identifier**. This value must match the format of your **Developer Profile's** App ID address if you are going to deploy to devices.
  - a. The value should be unique across your different applications. The RobotSample 2.1 Target has the *Identifier* value: *com.paulsolt.RobotSample2.03* since my base address is *com.paulsolt.\**

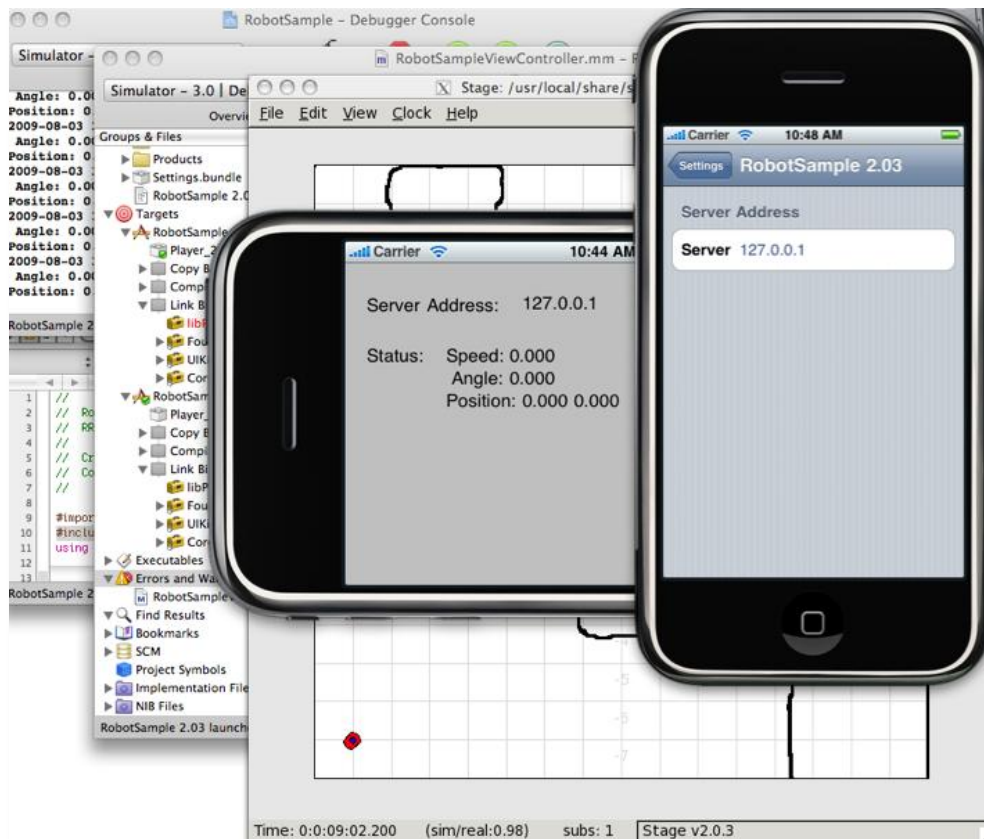


2. In the *Build Target* settings, sign the project with your *Developer Profile*.
3. **\*Note\***: The code signing process can be buggy. I always seem to run into difficulties trying to sign a project. **Clean** and rebuild the project after changing these settings. As long as the targets *Developer Profile* is set, you should be able to deploy to the device. There is a project wide *Developer Profile* setting that isn't always in sync with the *Build Targets*.

## How to Run the App:

1. Open *Terminal* and start a *Player/Stage* server using the command:  
\$ player /usr/local/share/stage/worlds/simple.cfg
2. Run the *RobotSample* version target in the *iPhone Simulator* with the 127.0.0.1 local server address. (Cmd + Enter)
3. The *RobotSample* makes it easy to change the IP address on the device/simulator using iPhone's *Settings* App. On iPhone go to *Settings* -> *Robot Sample* -> *Server Address*





4. You are ready to control real robots remotely! Just run the application on the iPhone and change the server address to point to the correct IP addresses.

## RobotSample

The RobotSample demonstrates how to use C++ and Objective-C in an iPhone application. The Robot class wraps the C++ Player/Stage code in a “run” method. It provides a basic wrapper around the robot code using Objective-C. I demonstrate several useful features:

1. Multi-threaded communication with iPhone UI – The Player/Stage code runs in a separate thread from the User Interface.
2. Bundle Settings – Use the iPhone Settings App to control the server address.
3. Basic User Interface connections with Interface Builder
4. NSTimer for timer-based programming – Run logic code in an update function at 60 frames per second.

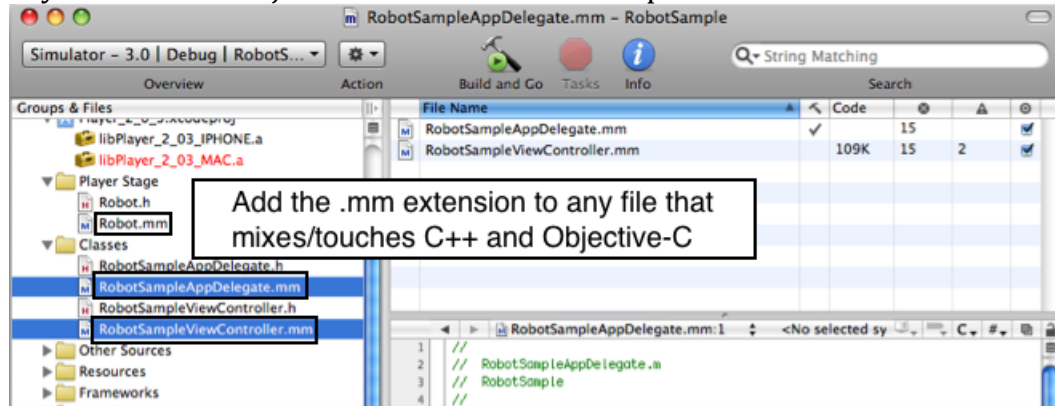
### How to Use C++ with Objective-C

To use C++ with Objective-C rename any .m file .mm if it will “touch” the C++ code. The extension will cause the compiler to compile both languages together.

More information is available in Apple’s guide on *Using C++ with Objective-C*

[http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Articles/ocCPlusPlus.html#//apple\\_ref/doc/uid/TP30001163-CH10-SW1](http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Articles/ocCPlusPlus.html#//apple_ref/doc/uid/TP30001163-CH10-SW1)

to use both languages in the compiler. This is needed if you try and wrap up the Player code with Objective-C like the RobotSample did.

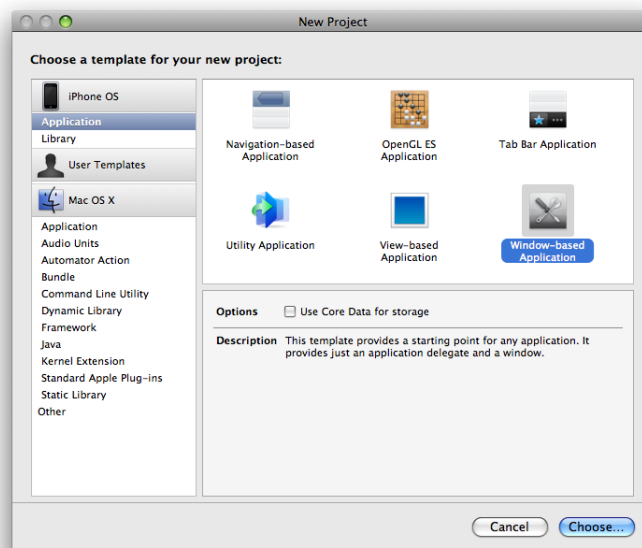


## BasicRobotSample

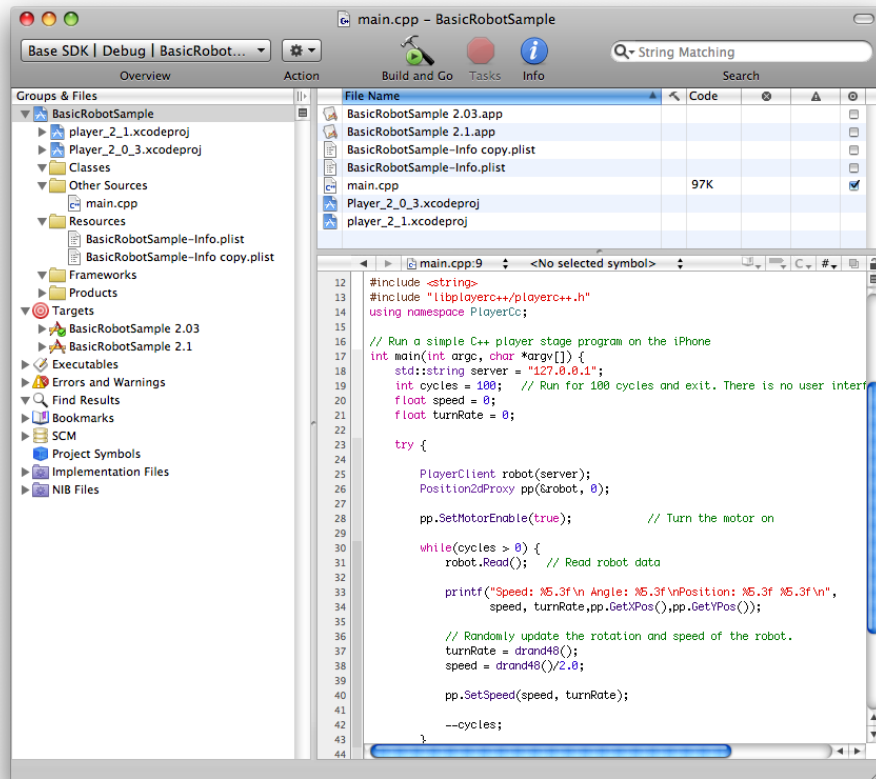
The basic sample is just a C++ program running on the iPhone without a user interface. It's an example starting point to run non-interactive algorithms on the iPhone platform.

There are some settings you should be aware of when working with or recreating this type of Xcode project:

1. Start with the iPhone OS: Window Based Application



2. Remove all iPhone related code, resources, and libraries This project will only contain *main.cpp*, *.plist* files, and your project files.



3. Remove the **precompiled header** from the Target Build settings. Search for .pch

## References

Harris, Clint. Easy, Modular Code Sharing Across iPhone Apps: Static Libraries and Cross-Project References. 2009. <<http://www.clintharris.net/2009/iphone-app-shared-libraries/>>.