Check out the tutorial, resources and examples that are available for MacRuby.

MACRUBY EVENTS »

19-21 Nov 2009 » RubyConf
San Francisco, CA
Laurent Sansonetti presents MacRuby

19-21 Nov 2009 » RubyConf
San Francisco, CA
Matt Aimonetti talks about writing 2D video games with MacRuby

Test Driven Development in Objective-C with MacRuby

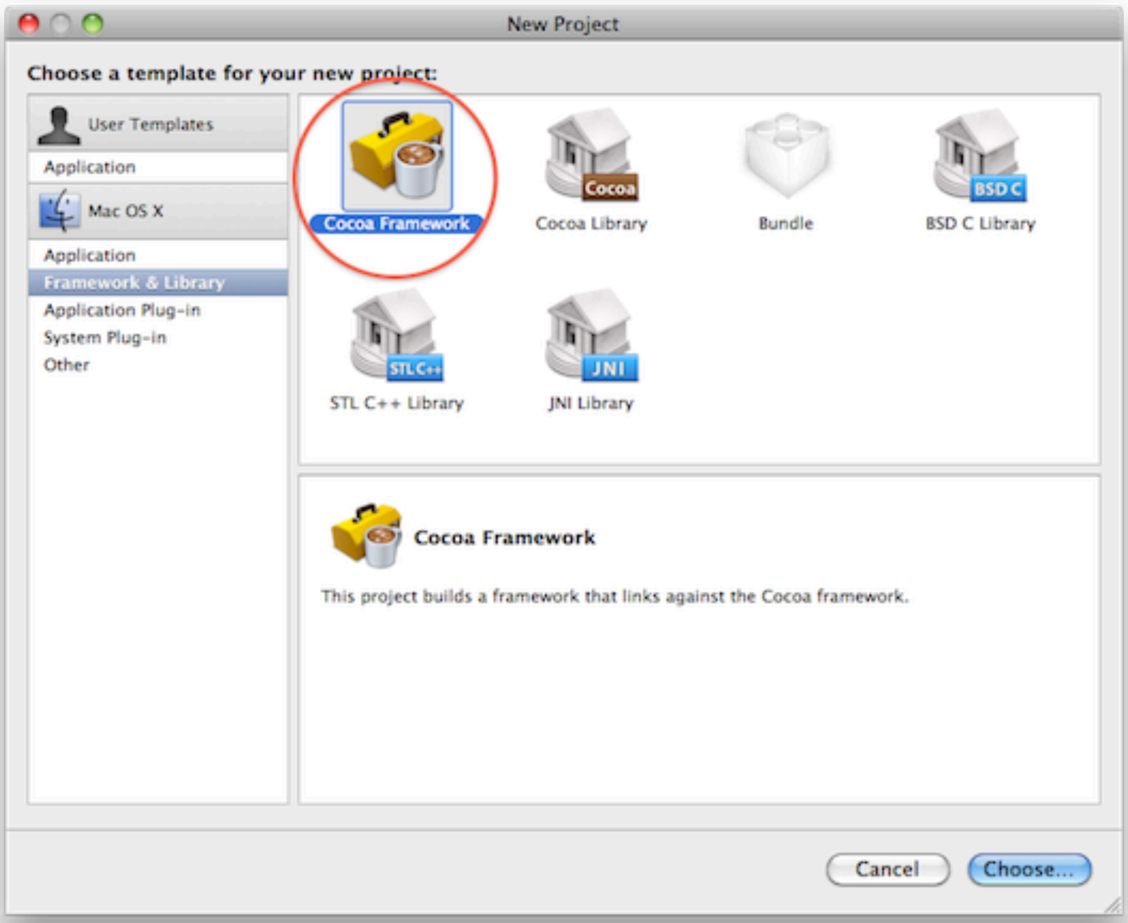
By Joshua Ballanco

Testing Envy

Test::Unit, RSpec, Shoulda, Minitest, Bacon, Cucumber… If there's one thing that the Ruby community has no shortage of, it's testing frameworks. Ever wish you could take advantage of these tools to do Test Driven Development in Cocoa? Well, wish no longer! With MacRuby, any Ruby testing framework instantly becomes an Objective-C testing framework. Let's see how…

Getting Started

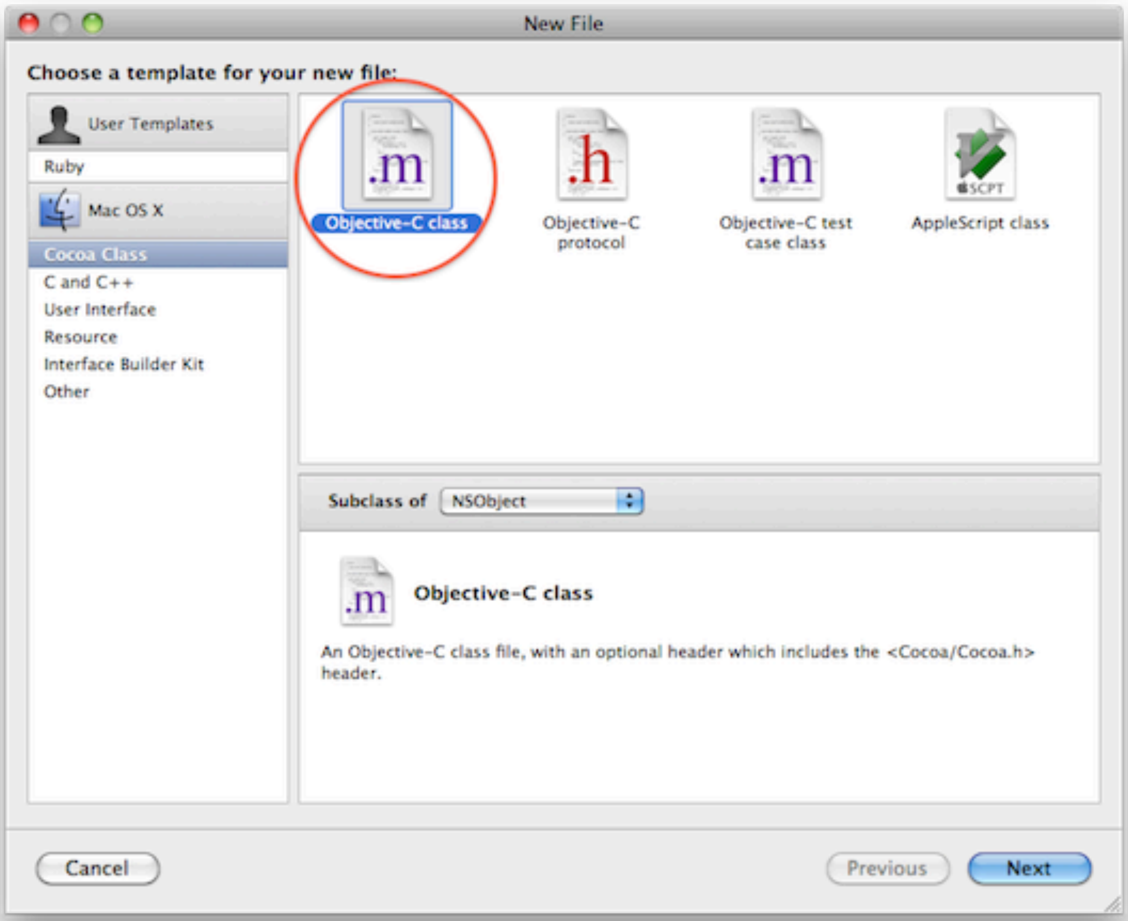
In this recipe, we're going to do test driven development of a Pig Latin Translator framework. That means the first step is to open up Xcode and start an empty Cocoa Framework project.



Cocoa Framework

This project builds a framework that links against the Cocoa framework.

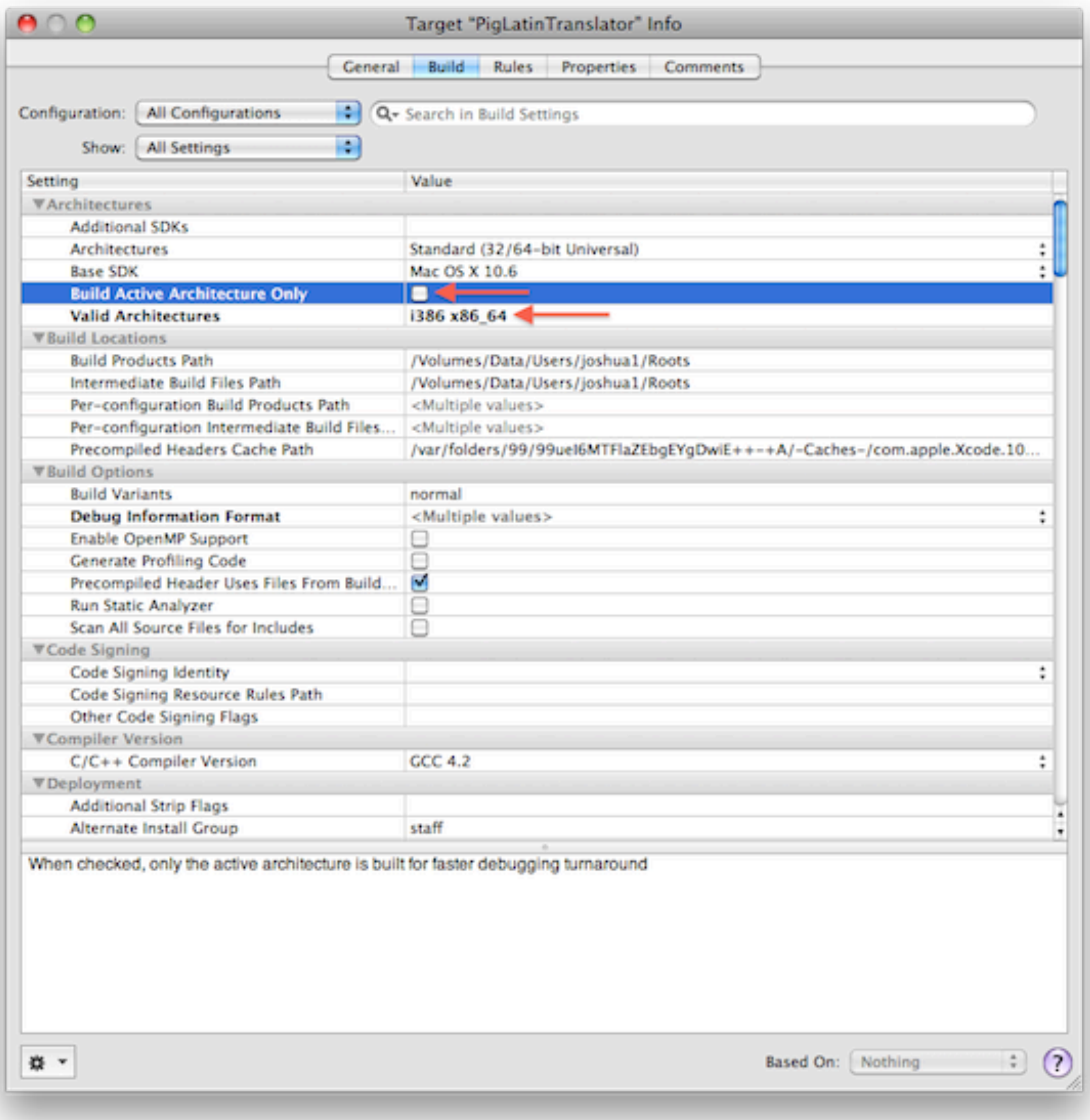
Name the project "PigLatinTranslator" and save it. Now we need something to go into this framework. From the File menu, select "New File…". Choose the Objective-C class template, name the class "PLTranslator", and add it to the "PigLatinTranslator" target.



Objective-C class

An Objective-C class file, with an optional header which includes the <Cocoa/Cocoa.h> header.

In order to do TDD with MacRuby, we'll have to adjust a few build settings for the framework target. From the main Xcode project page, open the "Targets" group, highlight the "PigLatinTranslator" target, and click the Inspector button. Then, under the "Build" tab we need to turn off "Build Active Architecture Only" and adjust the "Valid Architectures".



Configuration: All Configurations

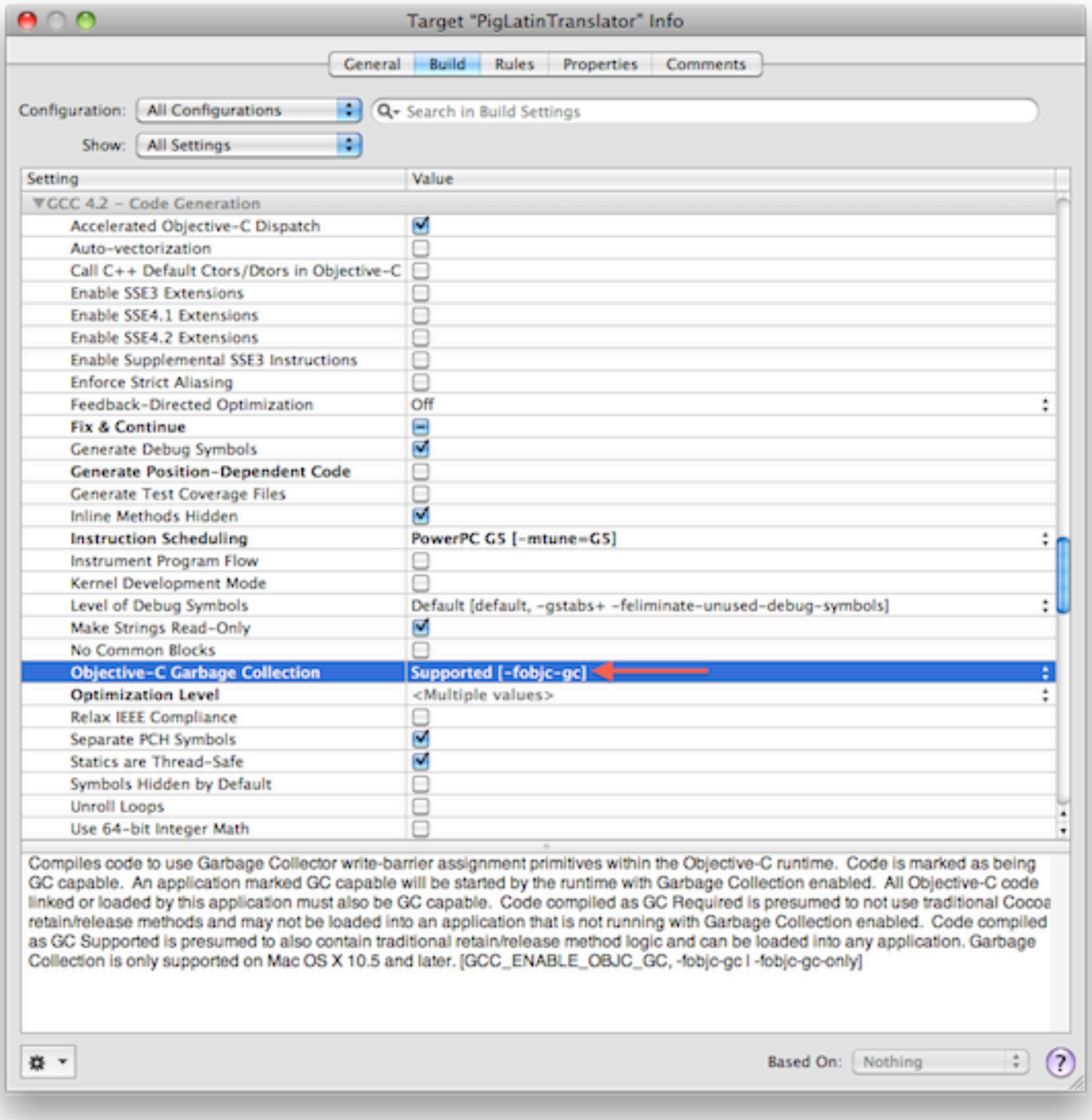
Show: All Settings

Setting	Value
▼Architectures	
Additional SDKs	
Architectures	Standard (32/64-bit Universal)
Base SDK	Mac OS X 10.6
Build Active Architecture Only	☐
Valid Architectures	**i386 x86_64**
▼Build Locations	
Build Products Path	/Volumes/Data/Users/joshua1/Roots
Intermediate Build Files Path	/Volumes/Data/Users/joshua1/Roots
Per-configuration Build Products Path	<Multiple values>
Per-configuration Intermediate Build Files...	<Multiple values>
Precompiled Headers Cache Path	/var/folders/99/99ueI6MTFlaZEbgEYgDwiE++-+A/-Caches-/com.apple.Xcode.10...
▼Build Options	
Build Variants	normal
Debug Information Format	<Multiple values>
Enable OpenMP Support	☐
Generate Profiling Code	☐
Precompiled Header Uses Files From Build...	✓
Run Static Analyzer	☐
Scan All Source Files for Includes	☐
▼Code Signing	
Code Signing Identity	
Code Signing Resource Rules Path	
Other Code Signing Flags	
▼Compiler Version	
C/C++ Compiler Version	GCC 4.2
▼Deployment	
Additional Strip Flags	
Alternate Install Group	staff

When checked, only the active architecture is built for faster debugging turnaround

The reason for this is that, in the interest of speed, Xcode prefers to only build the "i386" architecture by default. If you're on a 64-bit system, though, MacRuby will run in 64-bit mode by default and it will look for the "x86_64" architecture in your framework (and complain if it doesn't find it).

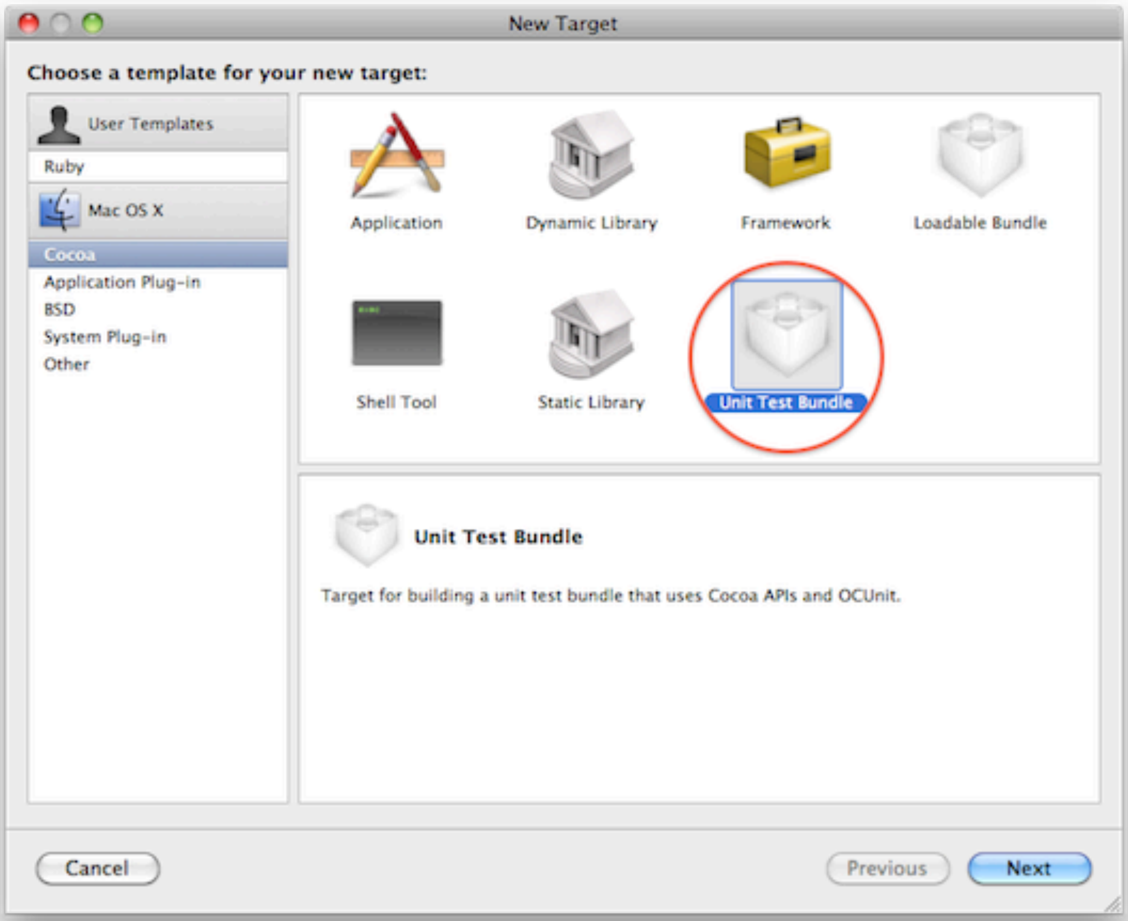
We'll also need to make sure that our framework builds with support for garbage collection since MacRuby can only load frameworks compatible with the Objective-C Garbage Collector.



Setting	Value
▼GCC 4.2 – Code Generation	
Accelerated Objective-C Dispatch	✓
Auto-vectorization	☐
Call C++ Default Ctors/Dtors in Objective-C	☐
Enable SSE3 Extensions	☐
Enable SSE4.1 Extensions	☐
Enable SSE4.2 Extensions	☐
Enable Supplemental SSE3 Instructions	☐
Enforce Strict Aliasing	☐
Feedback-Directed Optimization	Off
Fix & Continue	−
Generate Debug Symbols	✓
Generate Position-Dependent Code	☐
Generate Test Coverage Files	☐
Inline Methods Hidden	✓
Instruction Scheduling	**PowerPC G5 [-mtune=G5]**
Instrument Program Flow	☐
Kernel Development Mode	☐
Level of Debug Symbols	Default [default, -gstabs+ -feliminate-unused-debug-symbols]
Make Strings Read-Only	✓
No Common Blocks	☐
Objective-C Garbage Collection	**Supported [-fobjc-gc]**
Optimization Level	<Multiple values>
Relax IEEE Compliance	☐
Separate PCH Symbols	✓
Statics are Thread-Safe	✓
Symbols Hidden by Default	☐
Unroll Loops	☐
Use 64-bit Integer Math	☐

Compiles code to use Garbage Collector write-barrier assignment primitives within the Objective-C runtime. Code is marked as being GC capable. An application marked GC capable will be started by the runtime with Garbage Collection enabled. All Objective-C code linked or loaded by this application must also be GC capable. Code compiled as GC Required is presumed to not use traditional Cocoa retain/release methods and may not be loaded into an application that is not running with Garbage Collection enabled. Code compiled as GC Supported is presumed to also contain traditional retain/release method logic and can be loaded into any application. Garbage Collection is only supported on Mac OS X 10.5 and later. [GCC_ENABLE_OBJC_GC, -fobjc-gc | -fobjc-gc-only]

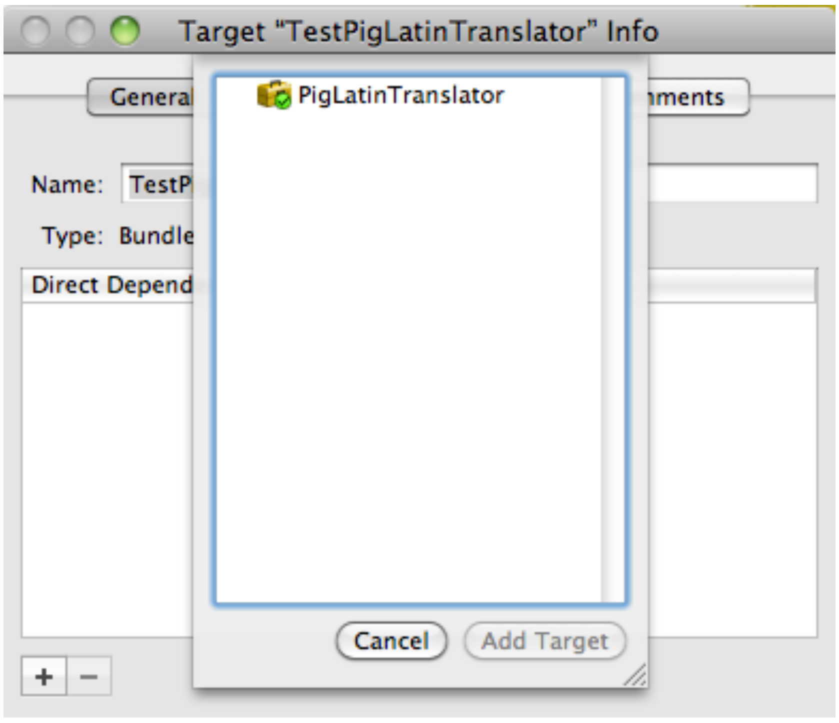
At this point we have an empty Cocoa Framework project just waiting for some fresh code. But, since we're good TDD practitioners, before we write any framework code we need write a test. We'll put these tests together in a new Xcode build target. From the Project menu, choose "New Target…", select a "Unit Test Bundle", and name it "TestPigLatinTranslator".



Unit Test Bundle

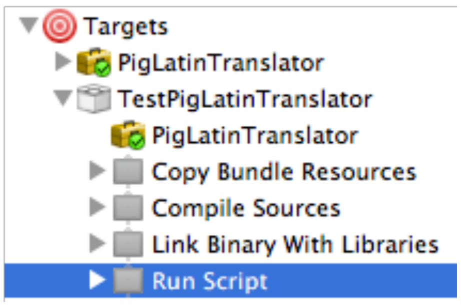
Target for building a unit test bundle that uses Cocoa APIs and OCUnit.

Xcode will then open the inspector for this new target. Under the “General” tab, click the plus button at the bottom of the “Direct Dependencies” section and add the PigLatinTranslator target as a dependency.



This will ensure that each time we build the test target, Xcode will first rebuild the framework target.

Now, if you've been paying attention, you might be wondering why we've chosen a "Unit Test Bundle" target. After all, the description of claims this target is for testing with OCUnit, which is decidely **not** a Ruby testing framework. That's quite alright. If we click on the disclosure triangle next to our new testing target, we find that it's nothing more than a series of copy, compile, and link steps followed up with a simple shell script.



If you double-click on the "Run Script" step, you'll find that the test target is calling out to a "RunUnitTests" script. We'll modify that to fit our needs in just a moment.

First, we need a MacRuby file for our specs. We're going to be using Bacon in this recipe, so if you don't already have it installed, you'll first need to run "sudo macgem install bacon". Once that's done we can add another new file to our project, this time picking a "Ruby File" from the "User Templates" section. Name the new file "pltranslator_spec.rb", and make sure to add it to the TestPigLatinTranslator target and not the framework target.

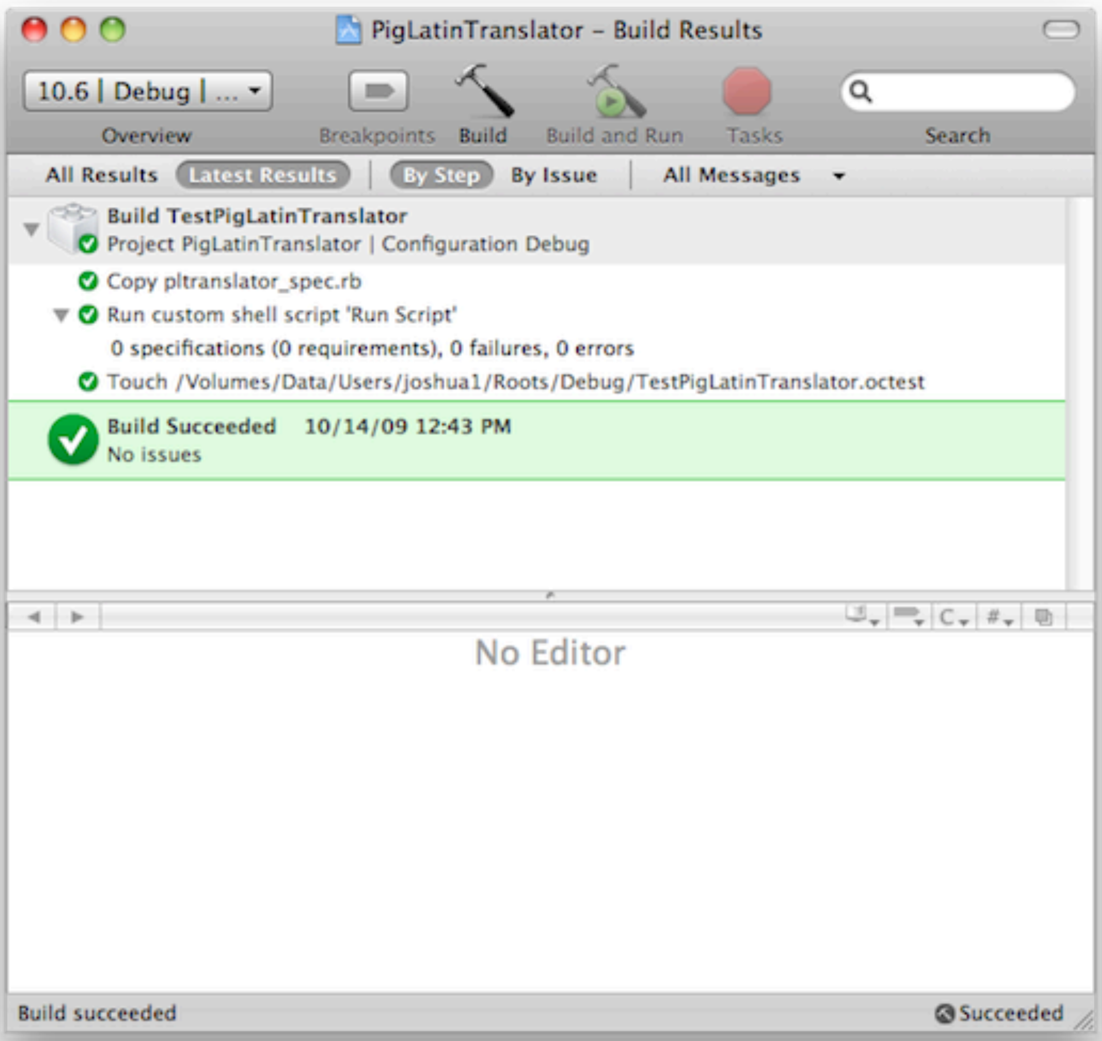
Now we need to modify that script that our test target will run. Double-click on the "Run Script" entry under the test target to bring up the build step inspector. The first thing we need to do is tell Xcode to run the test script with MacRuby. Change the "/bin/sh" entry for shell to "/usr/local/bin/macruby" (or wherever you have MacRuby installed; you can get this info by typing "which macruby" in terminal).

Then, we need to replace the default test runner script with a script that will run our Bacon specs:

```
# Include the Bacon libraries
require "rubygems"
require "bacon"

# Tell MacRuby where to find our framework
ENV['DYLD_FRAMEWORK_PATH'] = ENV['BUILT_PRODUCTS_DIR']

# Setup Bacon to report on tests at the end
Bacon.summary_on_exit

# Load all of the *_spec test files in the test bundle
Dir.glob('./*_spec.rb').each do |test_file|
  require test_file
end
```

This script will go through all of the Ruby **_spec** files in the test bundle and call them with MacRuby. We also need to set the DYLD_FRAMEWORK_PATH so that MacRuby knows where to look for our PigLatinTranslator framework.

Once you've made this change, set the Xcode target to "TestPigLatinTranslator" and build! Don't worry if you don't see anything. Xcode normally won't show the build results unless something goes wrong. If you want to double-check and see what Xcode did, choose "Build Results" from the Build menu.



Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Copy pltranslator_spec.rb
Run custom shell script 'Run Script'
0 specifications (0 requirements), 0 failures, 0 errors
Touch /Volumes/Data/Users/joshua1/Roots/Debug/TestPigLatinTranslator.octest
Build Succeeded 10/14/09 12:43 PM
No issues
Build succeeded
Succeeded

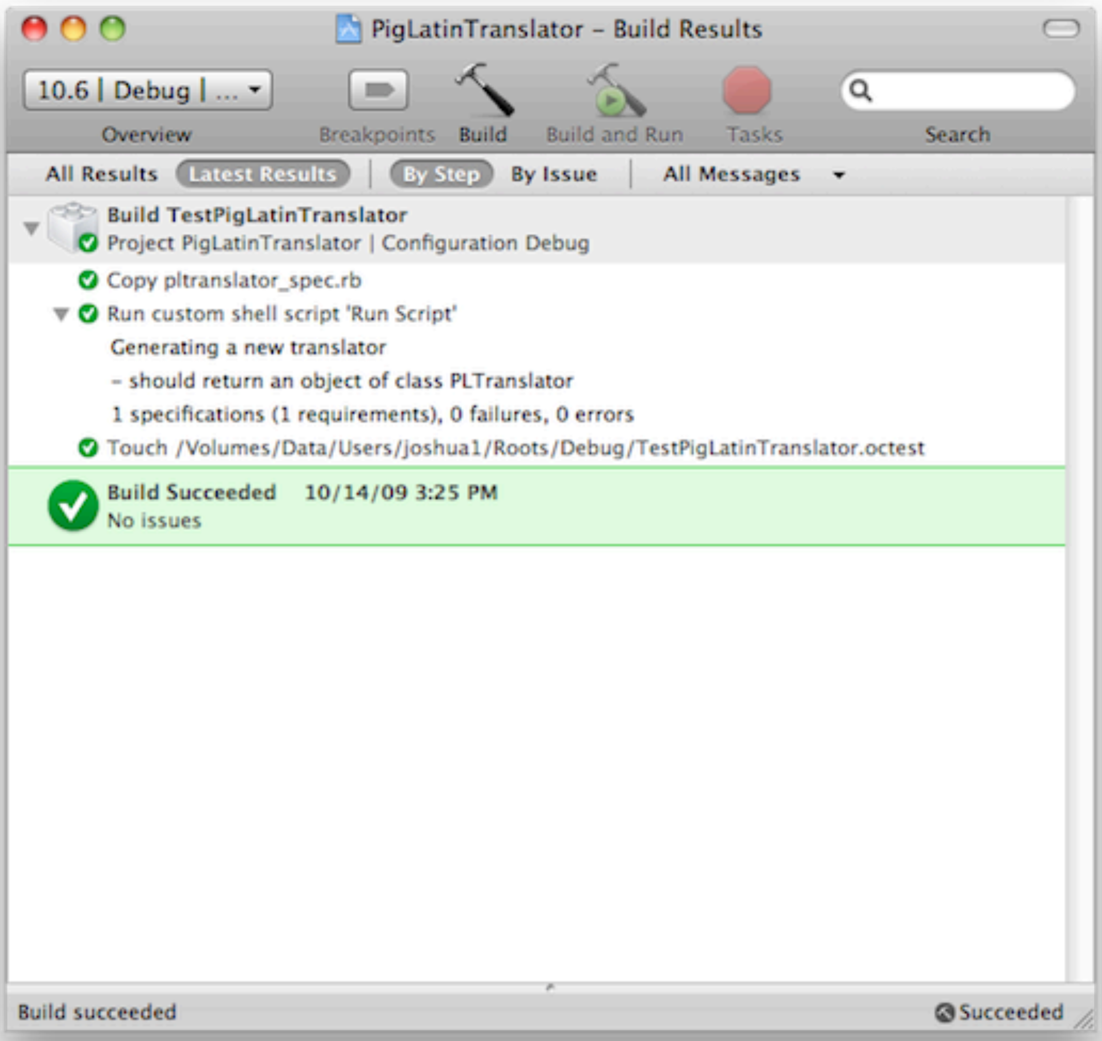
The First Iteration

Now that we have everything set-up, it's time to get to work. Since this is our first time doing TDD of Objective-C with MacRuby, we'll take it slow. Our first spec will simply test instantiation of a translator object. Go back to the Xcode editor, and add the following to your pltranslator_spec.rb file:

```
# pltranslator_spec.rb
# PigLatinTranslator

framework "PigLatinTranslator"

describe "Generating a new translator" do
  it "should return an object of class PLTranslator" do
```

```
    t = PLTranslator.new
    t.class.should.equal PLTranslator
  end
end
```

This will load up the framework and attempt to instantiate a PLTranslator object (Note: if you're having trouble with loading the framework, make sure that you set the DYLD_FRAMEWORK_PATH in the runner script). A quick CMD-B to build and… nothing. Wait, this isn't how TDD is supposed to work. We wrote a test before writing any code, so the test should fail right? Maybe there is an issue with the runner script? Let's check the build results:



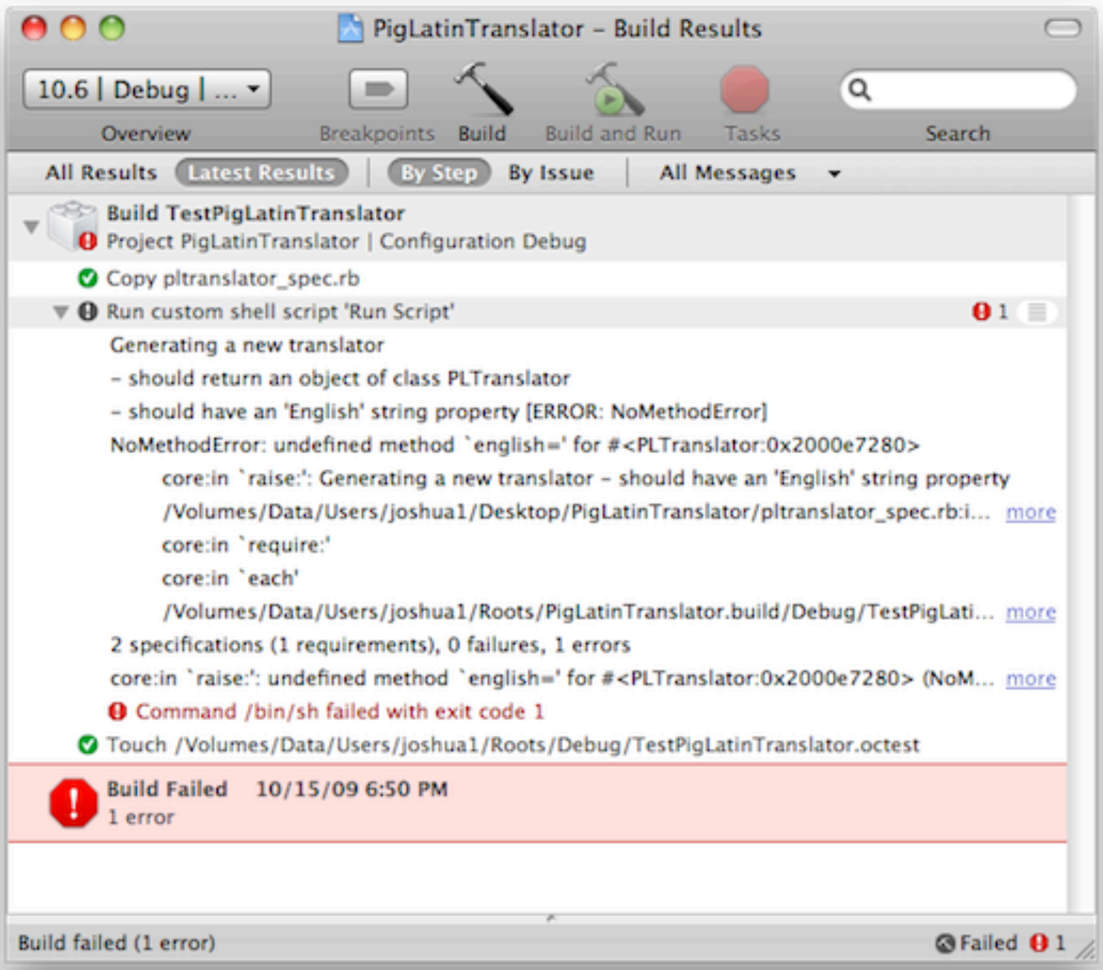
```
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Copy pltranslator_spec.rb
Run custom shell script 'Run Script'
  Generating a new translator
  - should return an object of class PLTranslator
  1 specifications (1 requirements), 0 failures, 0 errors
Touch /Volumes/Data/Users/joshua1/Roots/Debug/TestPigLatinTranslator.octest
Build Succeeded    10/14/09 3:25 PM
No issues
Build succeeded                                        Succeeded
```

Huh? So the test passed without us doing anything? If you're an old hand at Xcode, this should be no surprise. Go back and look at the PLTranslator.m file we added to our project at the beginning. You'll see that Xcode has already created an empty interface and implementation for the PLTranslator class.

This means we can skip straight to testing some properties and methods of the PLTranslator class. Our design calls for instances of the PLTranslator class to have two string properties representing the English and Pig Latin forms of a word. Our object also needs methosds to convert the English to Pig Latin and vice versa. Let's start by testing the getting and setting of the properties, English first:

```
# pltranslator_spec.rb
# PigLatinTranslator

framework "PigLatinTranslator"

describe "Generating a new translator" do
  before do
    @t = PLTranslator.new
  end

  it "should return an object of class PLTranslator" do
    @t.class.should.equal PLTranslator
  end

  it "should have an 'English' string property" do
    @t.english = "Hello"
    @t.english.should == "Hello"
  end
end
```

We've refactored the spec to add a "before" block, and added a requirement regarding the English string property. Save these changes and build, and we'll get:



```
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Copy pltranslator_spec.rb
Run custom shell script 'Run Script'                                    1
  Generating a new translator
  - should return an object of class PLTranslator
  - should have an 'English' string property [ERROR: NoMethodError]
  NoMethodError: undefined method `english=' for #<PLTranslator:0x2000e7280>
      core:in `raise:': Generating a new translator - should have an 'English' string property
      /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator/pltranslator_spec.rb:i... more
      core:in `require:'
      core:in `each'
      /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/TestPigLati... more
  2 specifications (1 requirements), 0 failures, 1 errors
  core:in `raise:': undefined method `english=' for #<PLTranslator:0x2000e7280> (NoM... more
  Command /bin/sh failed with exit code 1
Touch /Volumes/Data/Users/joshua1/Roots/Debug/TestPigLatinTranslator.octest
Build Failed    10/15/09 6:50 PM
1 error
Build failed (1 error)                                          Failed  1
```

Ah, that's more like it. Our spec failed, so now we can write some code to make it pass.

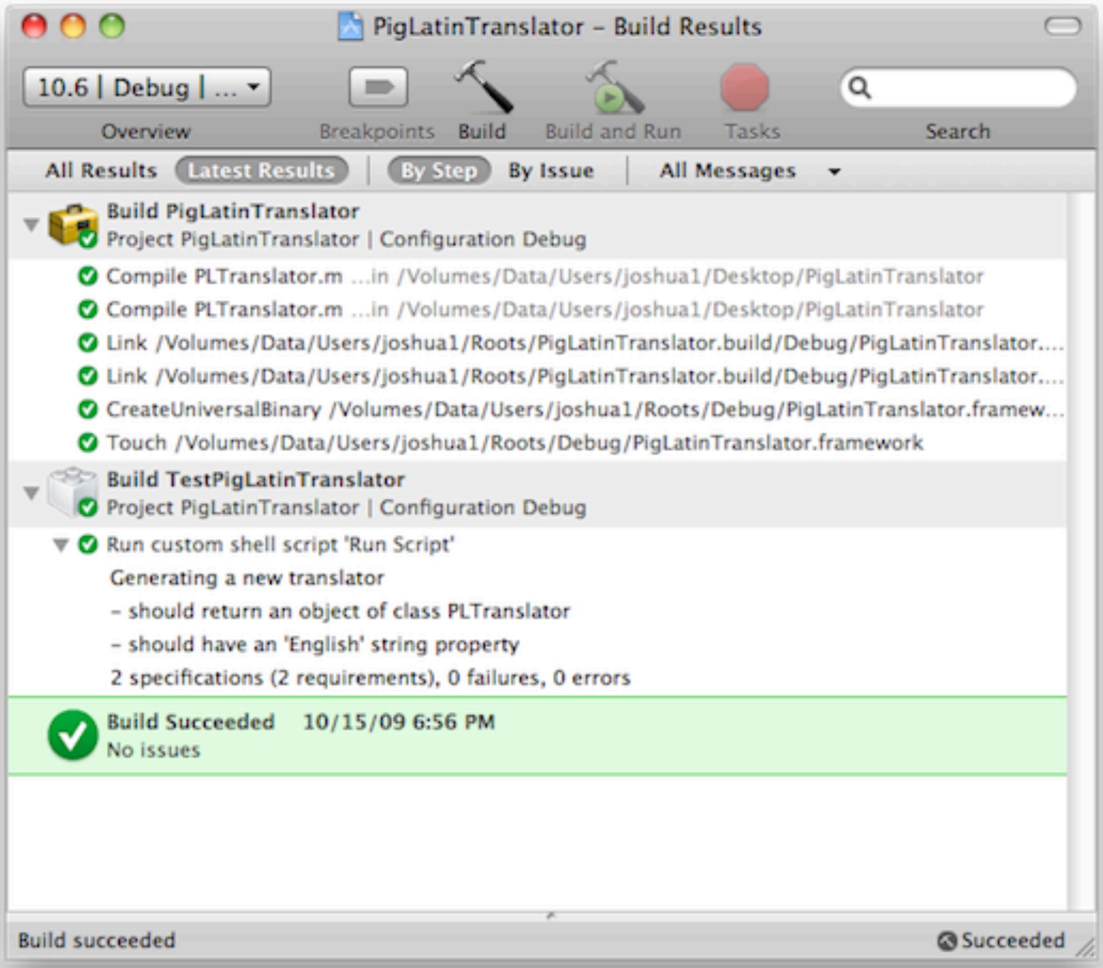
Opening up PLTranslator.h we'll add the property declaration:

```
//  PLTranslator.h
//  PigLatinTranslator

#import <Cocoa/Cocoa.h>

@interface PLTranslator : NSObject {
  NSString *english;
}

@property(copy) NSString *english;

@end
```

…and in PLTranslator.m we'll tell the compiler to synthesize the property for us:

```
//  PLTranslator.m
//  PigLatinTranslator

#import "PLTranslator.h"

@implementation PLTranslator

@synthesize english;

@end
```

Now we can build, and opening up the build results window we see:

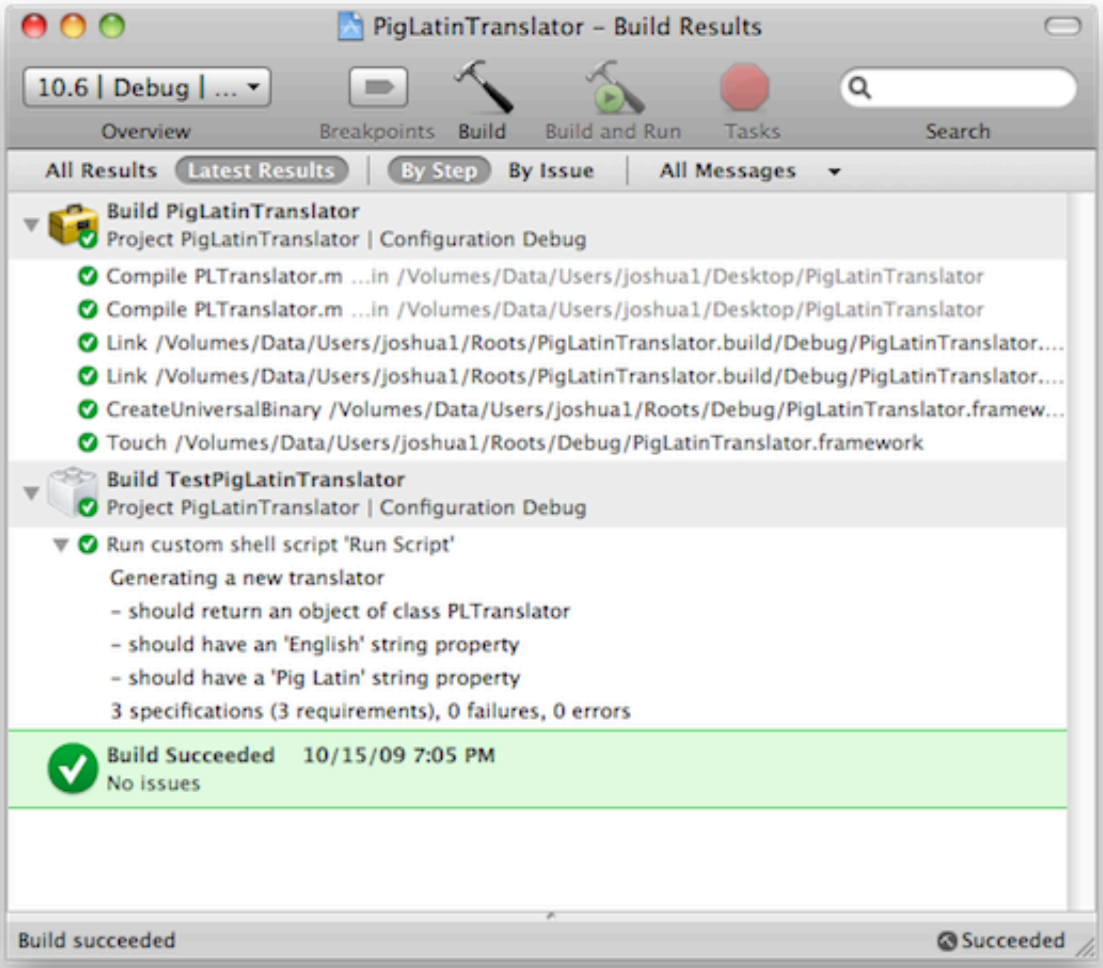


```
Build PigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
CreateUniversalBinary /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framew...
Touch /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framework
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Run custom shell script 'Run Script'
Generating a new translator
- should return an object of class PLTranslator
- should have an 'English' string property
2 specifications (2 requirements), 0 failures, 0 errors
Build Succeeded    10/15/09 6:56 PM
No issues
Build succeeded                                           Succeeded
```

Good! All green. Now go back and complete the first specification with the Pig Latin string property:

```
it "should have a 'Pig Latin' string property" do
  @t.pigLatin = "ello-hay"
  @t.pigLatin.should == "ello-hay"
end
```

(Note that for an Objective-C object, we use the Cocoa camelCase naming convention, not the Ruby under_case stlye.) Build, and this one will fail. Add property defining code to PLTranslator same as we did with the English string property, build, and we're back in the green:



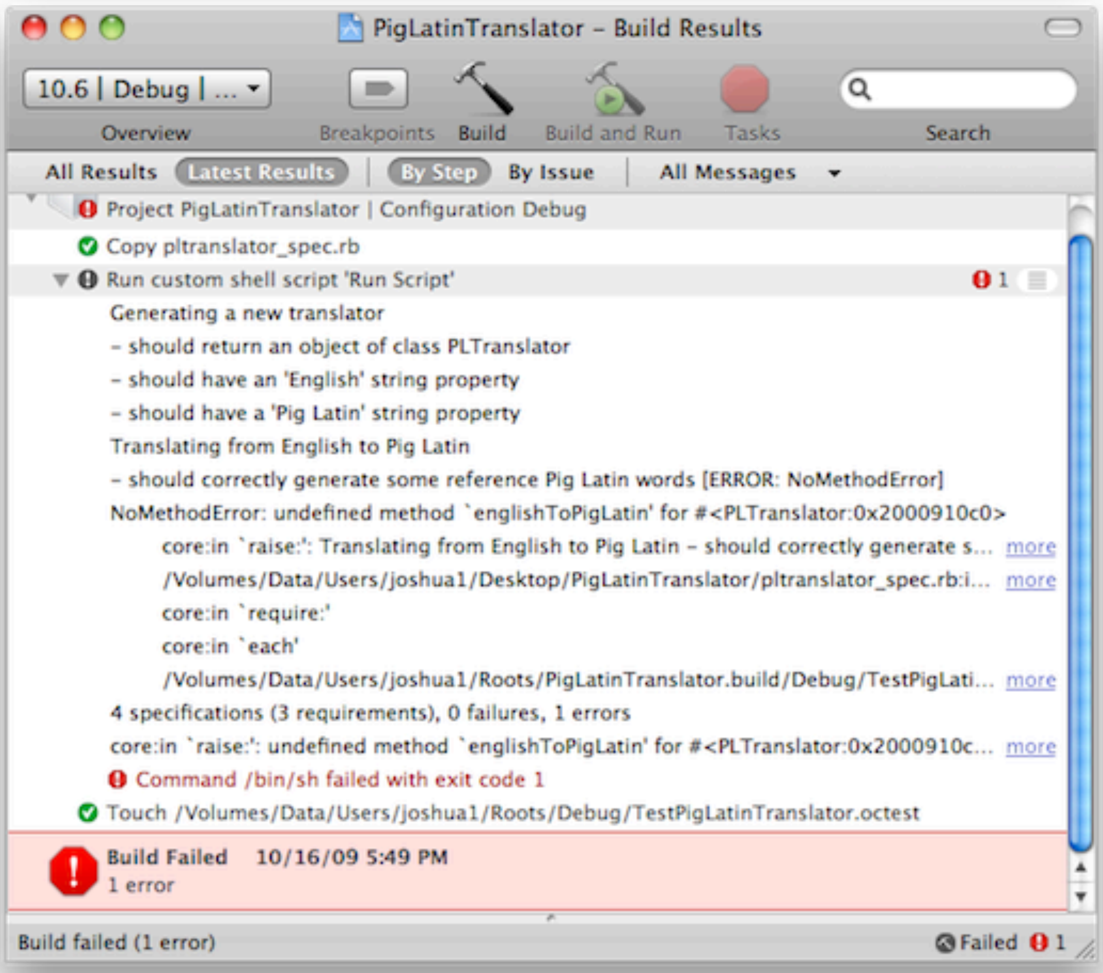
```
Build PigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
CreateUniversalBinary /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framew...
Touch /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framework
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Run custom shell script 'Run Script'
 Generating a new translator
 - should return an object of class PLTranslator
 - should have an 'English' string property
 - should have a 'Pig Latin' string property
 3 specifications (3 requirements), 0 failures, 0 errors
Build Succeeded    10/15/09 7:05 PM
No issues
Build succeeded                                                  Succeeded
```

You're Doing TDD Now!

OK, now we're ready to make things a bit more interesting. We can now set the value of the English or Pig Latin strings, but how to we get a translation from one to the other? We could put some of that logic in the setters, but to keep things simple for the time being, let's spec out a method that will take the English string, and populate the Pig Latin string with the proper translation. To test translation from English to Pig Latin we'll use a few sample words lifted from the Pig Latin Wikipedia Page

```
describe "Translating from English to Pig Latin" do
  before do
    @t = PLTranslator.new
  end

  it "should correctly generate some reference Pig Latin words" do
    { "beast" => "east-bay",
      "dough" => "ough-day",
      "happy" => "appy-hay",
      "question" => "estion-quay",
      "star" => "ar-stay",
      "three" => "ee-thray" }.each do |eng, pl|
        @t.english = eng
        @t.englishToPigLatin

        @t.pigLatin.should.equal pl
    end
  end
end
```

Building our project now, we get the expected spate of failures:



```
Project PigLatinTranslator | Configuration Debug
Copy pltranslator_spec.rb
Run custom shell script 'Run Script'
    Generating a new translator
    - should return an object of class PLTranslator
    - should have an 'English' string property
    - should have a 'Pig Latin' string property
    Translating from English to Pig Latin
    - should correctly generate some reference Pig Latin words [ERROR: NoMethodError]
    NoMethodError: undefined method `englishToPigLatin' for #<PLTranslator:0x2000910c0>
        core:in `raise:': Translating from English to Pig Latin - should correctly generate s... more
        /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator/pltranslator_spec.rb:i... more
        core:in `require:'
        core:in `each'
        /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/TestPigLati... more
    4 specifications (3 requirements), 0 failures, 1 errors
    core:in `raise:': undefined method `englishToPigLatin' for #<PLTranslator:0x2000910c... more
    Command /bin/sh failed with exit code 1
Touch /Volumes/Data/Users/joshua1/Roots/Debug/TestPigLatinTranslator.octest
Build Failed   10/16/09 5:49 PM
1 error
Build failed (1 error)                                                Failed 1
```

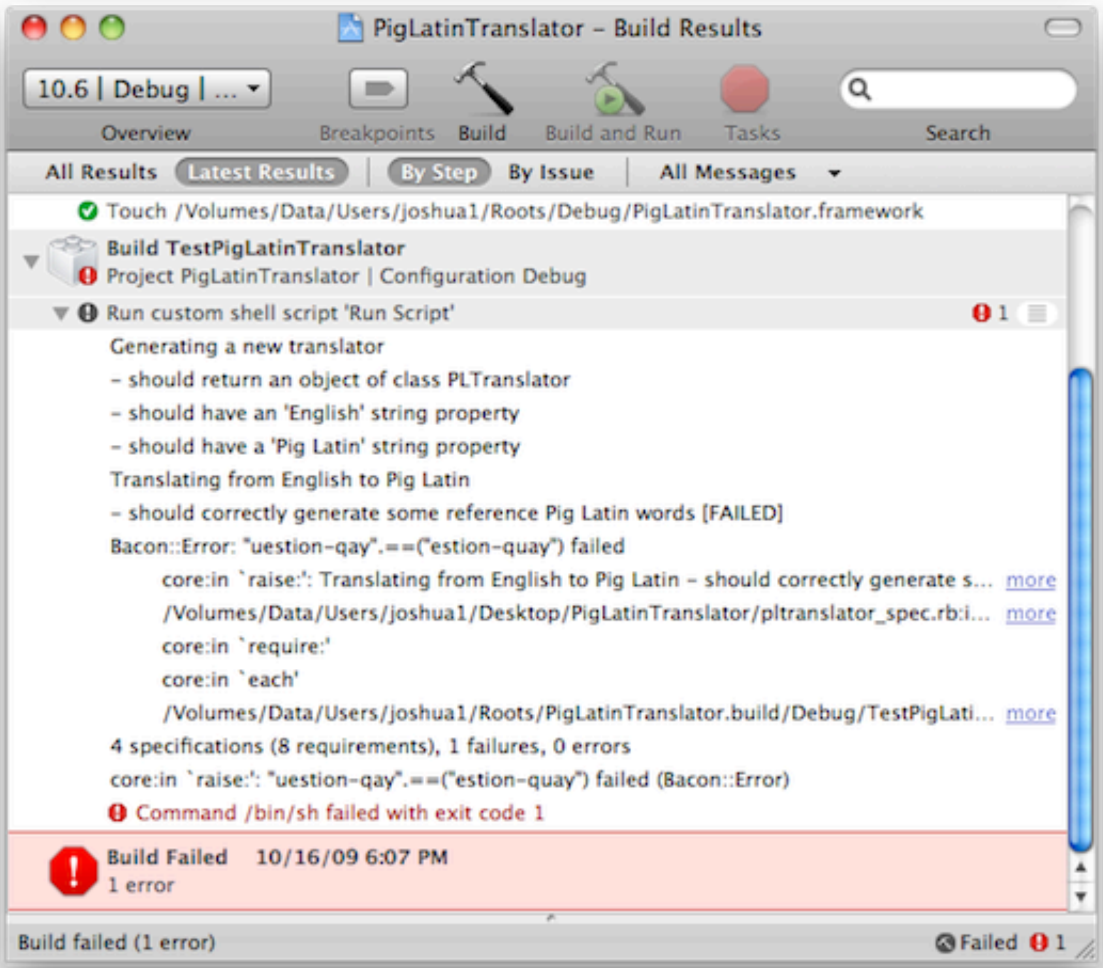
To get our spec passing again, we'll need to implement the "englishToPigLatin" method. First, let's add the function prototype to PLTranslator.h:

```
- (void)englishToPigLatin;
```

Then we'll implement the method in PLTranslator.m:

```
- (void)englishToPigLatin
{
  // Decompose the English string
  NSString *leader, *rest;
  int firstVowel = [english rangeOfCharacterFromSet:
                     [NSCharacterSet characterSetWithCharactersInString:@"aeiou"]
                    ].location;
  leader = [english substringToIndex:firstVowel];
  rest = [english substringFromIndex:firstVowel];

  // ...and recompose as Pig Latin
  self.pigLatin = [rest stringByAppendingString:
                    [@"-" stringByAppendingString:
                          [leader stringByAppendingString:@"ay"]]];
  return;
}
```

Lookin' good! Let's take this baby for a spin…

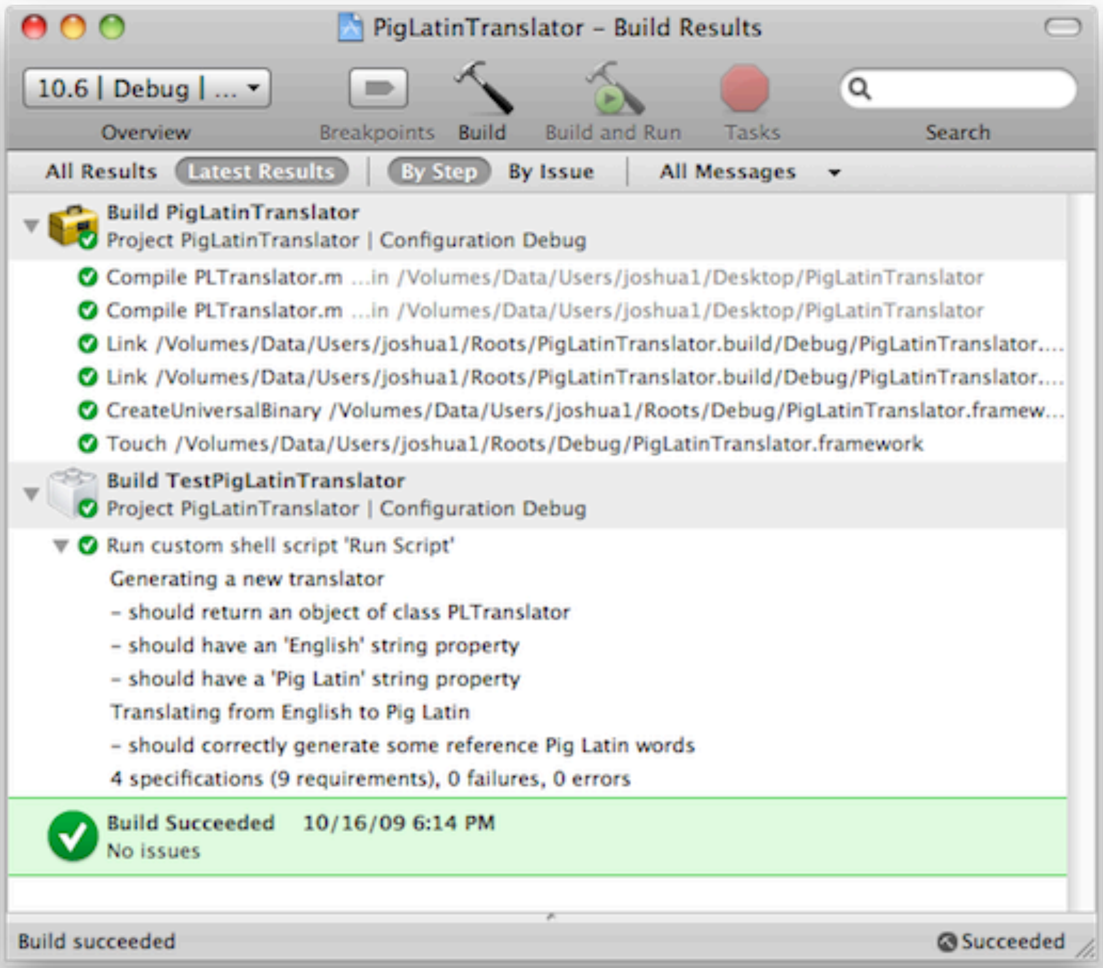


```
Touch /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framework
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Run custom shell script 'Run Script'    1
Generating a new translator
- should return an object of class PLTranslator
- should have an 'English' string property
- should have a 'Pig Latin' string property
Translating from English to Pig Latin
- should correctly generate some reference Pig Latin words [FAILED]
Bacon::Error: "uestion-qay".==("estion-quay") failed
    core:in `raise:': Translating from English to Pig Latin - should correctly generate s... more
    /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator/pltranslator_spec.rb:i... more
    core:in `require:'
    core:in `each'
    /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/TestPigLati... more
4 specifications (8 requirements), 1 failures, 0 errors
core:in `raise:': "uestion-qay".==("estion-quay") failed (Bacon::Error)
Command /bin/sh failed with exit code 1
Build Failed    10/16/09 6:07 PM
1 error
Build failed (1 error)                                    Failed 1
```

Drat! We were so close. Apparently "Q" needs some special casing. Let's go back and modify the translation method slightly:

```
// Decompose the English string
NSString *leader, *rest;
int firstVowel = [english rangeOfCharacterFromSet:
                  [NSCharacterSet characterSetWithCharactersInString:@"aeiou"]
                 ].location;
if (firstVowel > 0 &&
    ('q' == [english characterAtIndex:(firstVowel - 1)] ||
     'Q' == [english characterAtIndex:(firstVowel - 1)])) {
  if ('u' == [english characterAtIndex:firstVowel]) {
    firstVowel++;
  }
}
leader = [english substringToIndex:firstVowel];
rest = [english substringFromIndex:firstVowel];
```

Now when we build and open the build results window…



```
Build PigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Compile PLTranslator.m ...in /Volumes/Data/Users/joshua1/Desktop/PigLatinTranslator
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
Link /Volumes/Data/Users/joshua1/Roots/PigLatinTranslator.build/Debug/PigLatinTranslator....
CreateUniversalBinary /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framew...
Touch /Volumes/Data/Users/joshua1/Roots/Debug/PigLatinTranslator.framework
Build TestPigLatinTranslator
Project PigLatinTranslator | Configuration Debug
Run custom shell script 'Run Script'
  Generating a new translator
  - should return an object of class PLTranslator
  - should have an 'English' string property
  - should have a 'Pig Latin' string property
  Translating from English to Pig Latin
  - should correctly generate some reference Pig Latin words
  4 specifications (9 requirements), 0 failures, 0 errors
Build Succeeded    10/16/09 6:14 PM
No issues
Build succeeded                                        Succeeded
```

Ah-HA! Success!

Going further…

Hopefully, by now you're starting to see just how useful testing Objective-C with MacRuby can be. As an exercise to improve your skills, try implementing the reverse translation. You can start by reversing the keys and values we used in the test above. What other corner cases can you think of? Can you write some specs to flush out the behavior of our framework class?

Be creative! Now that you can access Objective-C objects from MacRuby, it's not only possible to use Cocoa to improve your Ruby, but it's also possible to use Ruby to improve your Objective-C. For example, once you've got a pretty good set of specs covering the PLTranslator class, throw it away and rewrite it in Ruby! Try this for another Cocoa class, or even take a Ruby library with extensive test coverage, and use those tests as a scaffold to rewrite the library in Objective-C.

Also, don't forget that, while Ruby requires that a framework be compatible with the Objective-C garbage collector, the framework does not have to be strictly garbage collected. So, you could even use Bacon to spec out an Objective-C framework running with the GC turned on, then add in "retain"s and "release"s and use that same framework on the iPhone.

Bonus Credit

I’ve never been a huge fan of the throw-away demo, so just to prove that this exercise isn’t completely pointless, let’s go ahead and make an application using our framework. First, we need a new “Application” target for our project. If you make the “TestPigLatinTranslator” target a direct dependency of your application, then not only will you rebuild your framework every time you build the app, but you’ll also be re-running the tests. Don’t forget to add the PigLatinTranslator.framework as a Linked Library.

We need a “main.m” for the application, so from the new file window choose an “Objective-C Class” and add it to the application target. We only need the *.m file, so you can skip creation of the corresponding *.h header. Fill in “main.m” with the standard Cocoa start-up code:

```
#import <Cocoa/Cocoa.h>

int main(int argc, const char **argv)
{
  return NSApplicationMain(argc, argv);
}
```

Now we need an interface for the app, so add another new file, this time from the new file window choose an “Application XIB” from the “User Interface” section. Name this file “MainMenu.xib” and double-click on it in Xcode to open up Interface Builder. Go ahead and drag in some labels and text fields (you can uncheck the “Editable” option on the Pig Latin text field for now to hide the fact that PigLatin to English isn’t implemented yet), and a button to trigger the translation.

Then take an NSObject from the library, add it to your project, and set its Class Identity to “PLTranslator”. Before you can hook everything up, there’s a change you need to make to the framework we just wrote. We’d like our “englishToPigLatin” function to be available directly to the interface, but in order for that to be the case, we need to alter its method signature slightly:

```
- (IBAction)englishToPigLatin:(id)sender;
```

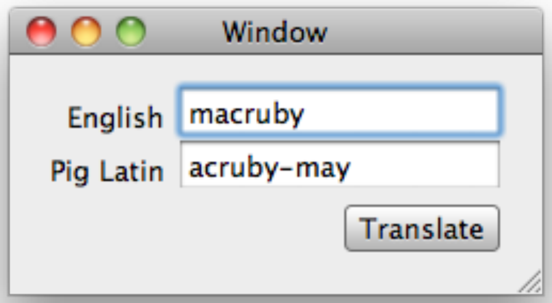
Notice that this will break our tests (you’d find this out as soon as you attempted a build), so we will also need to tweak our test:

```
@t.englishToPigLatin(self)
```

Then, we can go back to Interface Builder, right-click on the button and drag a connection to the Translator object to connect that button to “englishToPigLatin:”. Then, for the text fields we’ll take advantage of Cocoa’s bindings. In the info panel, set the binding of the English text field to the Translate object’s “english” key. Also make sure to check the “Continuously

Updates Value” and “Validates Immediately” boxes. Set the Pig Latin text field’s binding likewise, but this time using the key “pigLatin”.

…and that’s it! Save everything, go back to Xcode, and do a “Build and Run” and enjoy the fun!



English: macruby
Pig Latin: acruby-may

MacRuby is a free software project by Apple Inc. Sources are available under the Ruby license.

Hosting provided by Mac OS Forge. Use of this site is subject to the Mac OS Forge Terms of Use.

Website designed by John Athayde and created with Webby.