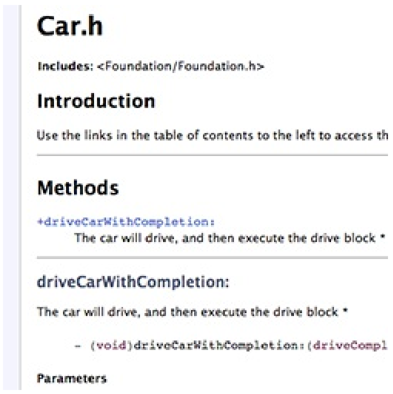
Documenting in Xcode with HeaderDoc Tutorial - Ray Wenderlich


Car.h

Includes: <Foundation/Foundation.h>

Introduction

Use the links in the table of contents to the left to access th

Methods

+driveCarWithCompletion:
The car will drive, and then execute the drive block *

driveCarWithCompletion:

The car will drive, and then execute the drive block *

```
- (void)driveCarWithCompletion:(driveCompl
```

Parameters

Learn the best practices for documenting your code!

Learn the best practices for documenting your code!

When Xcode 5 and iOS 7 were announced, a small addition was mentioned that most people might have missed: HeaderDoc.

HeaderDoc is a command line tool that you can use to automatically generate nice HTML documentation for your code – as long as you structure your comments in a particular way.

In addition, Xcode parses HeaderDoc-style comments behind the scenes to automatically present your documentation in quick look panels.

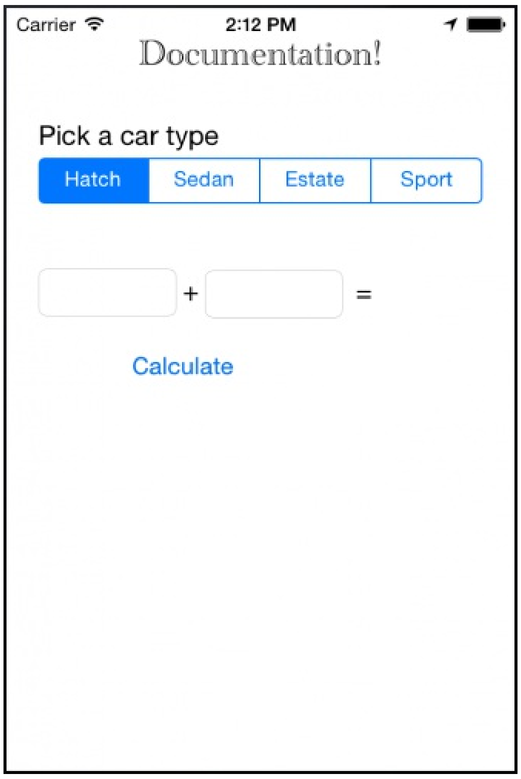
In this HeaderDoc tutorial, you will learn how to:

1. Write HeaderDoc-style comments
2. Preview your documentation in Xcode
3. Generate HTML documentation from your comments
4. Learn about a third-party tool to make documentation even easier (*VVDocumenter-Xcode*)

Let’s get documenting!

Getting Started

Download the [starter project]() for this tutorial, and build and run. It should look like this:



RW_Documentation_Initial

This is a simple little test project that includes two main helper classes:

- **Car: Contains a few properties, and a method to "drive" a car and run a block upon completion**
- **MathAPI: Contains a helper method to add two numbers**

Right now, these files are barely documented. Let's see how you can use HeaderDoc to make creating documentation for these classes super easy.

HeaderDoc Comments

HeaderDoc is a tool that you can either run from the command-line, or is automatically run by Xcode. Basically it scans your files for comments made in a particular style.

There are three styles of comments that HeaderDoc scans for:

Option 1:

```
/// Your documentation comment will go here
```

Option 2:

```
/** * Your documentation comment will go here */
```

Option 3:

```
/*! * Your documentation comment will go here */
```

Note that these are similar to "normal" comments, except that there's an extra / in option 1, and an extra character in the first line of options 2 and 3 (* and !, respectively).

Note: In options two and three, there is an additional asterisk on each line in-between the required opening and closing lines. This is purely aesthetic, and is not required.
All three syntaxes will produce the same result, if done in Xcode.
Since we could start a small war over which way is the “best” way to document your code, we’ll stick with the following rules for the purposes of this tutorial:

- For large comment blocks, use the style used in Apple’s documentation (/*!)
- For single line comments, use the triple forward slash syntax (///) comment to save room

HeaderDoc Tags

Once HeaderDoc finds a comment in one of the above styles, it will search that comment for **tags** for more information. You will use tags to mark-up your code.
A tag starts with the @ symbol and a keyword, followed by a space, and a string that contains a description relative to that keyword (such as **@param foo**). There are two levels of tags:

- **Top-Level Tags: These are tags that declare the type of thing you are commenting (Headers, classes, methods, etc.)**
- **Second-Level Tags: These tags help to give more detail about the specific thing you are commenting.**

An example of a top-level tag is **@typedef**, which you use to indicate typedefs for things like enums, structs and function pointers.
HeaderDoc is pretty good at adding Top-Level tags automatically via the context, so they are generally optional. In this tutorial, you’ll focus mostly on Second-Level tags.
Let’s take a look at some helpful Second-Level tags:

- **@brief: Quickly describes the data type, method, etc. that you are documenting.**
- **@abstract: Equivalent to @brief.**
- **@discussion: Similar to @abstract and @brief, but it allows multiple lines. It’s not required to actually write this keyword out; but it is good to use for clarity’s sake.**
- **@param: The name and description of a parameter to a method, callback or function.**
- **@return: A description of what a method or function returns. (You can also use the equivalent @result)**

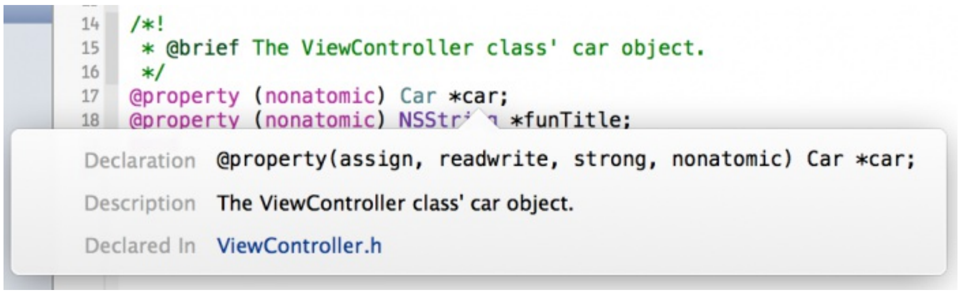
This is by no means all of the tags, and you can find all of the available tags in the HeaderDoc User Guide.
For the sake of this tutorial, all comments will be placed in the header files, just to keep the implementation files clean.

Documenting Properties

Let’s start by documenting a few properties.
Back in Xcode, in the *DocumentationExamples* project, open **ViewController.h**. You’ll see two properties that are lacking some documentation. Just above the declaration for the **car** property, add a comment block so that it looks like this:

/*! * @brief The ViewController class' car object. */@property(nonatomic) Car *car;

Now, build your project. When it finishes, hold the option key, and click on the **car** variable name. You should see a popover with your comment.



```
/*!
 * @brief The ViewController class' car object.
 */
@property (nonatomic) Car *car;
@property (nonatomic) NSStr   *funTitle;
```

Declaration @property(assign, readwrite, strong, nonatomic) Car *car;
Description The ViewController class' car object.
Declared In ViewController.h

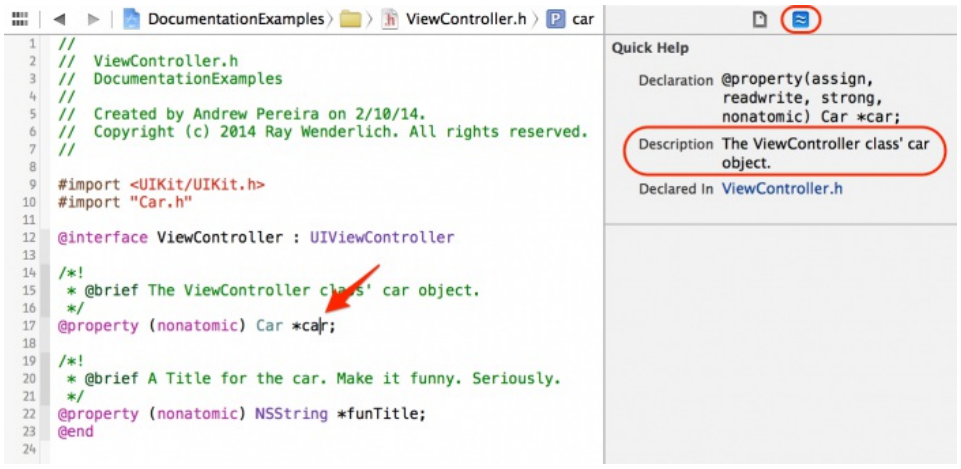
RW_Documentation_FirstComment

Troubleshooting: If your popover didn' t contain your comment, you may not have waited long enough for the project to build. If it has finished, try restarting Xcode, just to make sure.

Since that was so easy, try adding a documentation comment to the other property in **ViewController** on your own. It should say something about how this is a title that is supposed to be funny.

Solution Inside: Solution	Show

Before you complete the remainder of your documentation, check out another way to view documentation in Xcode. Open the Utilities panel' s Quick Help Inspector. Now **click on the "car" variable** name. In the Quick Help Inspector, you should see your comment appear:



```
//
//  ViewController.h
//  DocumentationExamples
//
//  Created by Andrew Pereira on 2/10/14.
//  Copyright (c) 2014 Ray Wenderlich. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Car.h"

@interface ViewController : UIViewController

/*!
 * @brief The ViewController class' car object.
 */
@property (nonatomic) Car *car;

/*!
 * @brief A Title for the car. Make it funny. Seriously.
 */
@property (nonatomic) NSString *funTitle;
@end
```

Declaration @property(assign, readwrite, strong, nonatomic) Car *car;
Description The ViewController class' car object.
Declared In ViewController.h

RW_Documentation_QuickHelp

Now, there are two more classes that need to be documented: **MathAPI**, and **Car**.

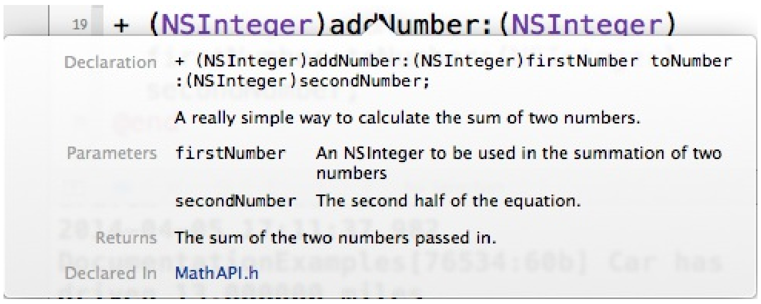
Documenting Methods

MathAPI includes a method that needs to be commented, so open MathAPI.h.

Examine the method **addNumber:toNumber:**. It returns a value and takes two parameters. So you' ll need to add a **@description** tag, two **@param** tags and a **@return** tag. Go ahead and create a comment that looks like the following:

```
/*! * @discussion A really simple way to calculate the sum of two numbers. * @param firstNumber An NSInteger to be used in the summation of two numbers * @param secondNumber The second half of the equation.  * @return The sum of the two numbers passed in. */+(NSInteger)addNumber:(NSInteger)firstNumber toNumber:(NSInteger)secondNumber;
```

Now, **Build** the project again then option-click on part of the method name to see your handiwork.



```
+ (NSInteger)addNumber:(NSInteger)
```

Declaration + (NSInteger)addNumber:(NSInteger)firstNumber toNumber :(NSInteger)secondNumber;

A really simple way to calculate the sum of two numbers.

Parameters	firstNumber	An NSInteger to be used in the summation of two numbers
	secondNumber	The second half of the equation.

Returns The sum of the two numbers passed in.

Declared In MathAPI.h

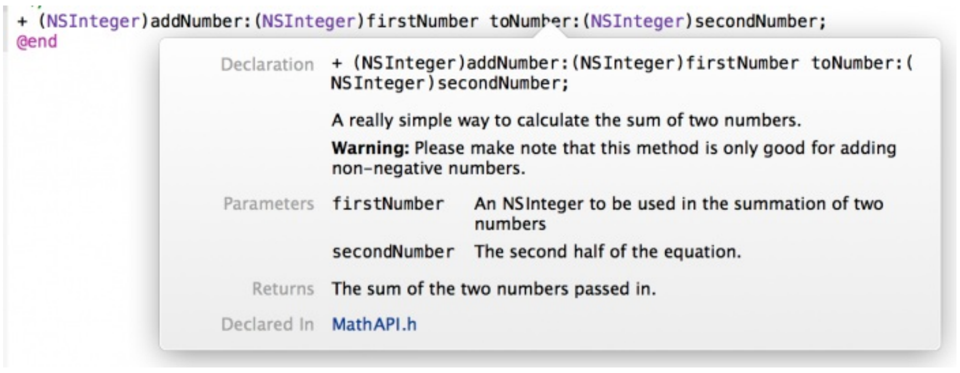
Screen Shot 2014-04-05 at 5.31.52 PM

Troubleshooting: Many symbols are option-clickable in the Xcode text editor. Make sure you click on the thing you want to get help on. In the above example, you' ll need to click on either "addNumber:" or "toNumber:" to see the correct help.

Now, unbeknownst to you, I' ve created a pretty terrible implementation of this method. It will only really work with non-negative numbers. To inform the user of this method of that, you can add a little more information to your comment. In this case, let' s add a @warning tag. **Add the following text** just above the *@return* tag:

```
* @warning Please make note that this method is only good for adding non-negative numbers.
```

Now, build your project again, and option-click on the method name when it finishes. You' ll see a nice warning to the user.



```
+ (NSInteger)addNumber:(NSInteger)firstNumber toNumber:(NSInteger)secondNumber;
@end
```

Declaration + (NSInteger)addNumber:(NSInteger)firstNumber toNumber:(NSInteger)secondNumber;

A really simple way to calculate the sum of two numbers.

Warning: Please make note that this method is only good for adding non-negative numbers.

Parameters	firstNumber	An NSInteger to be used in the summation of two numbers
	secondNumber	The second half of the equation.
Returns	The sum of the two numbers passed in.	
Declared In	MathAPI.h	

RW_Documentation_SecondComment

Making Your Life Easier: Code Snippets

That was really fast, no? But what if you could make that process faster?

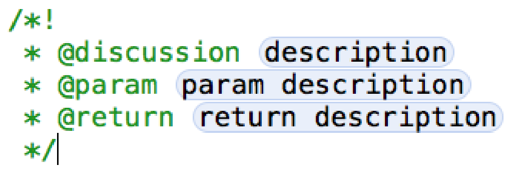
There’ s something that’ s been mentioned before on the site, and on the the raywenderlich.com podcast: code snippets.

Code snippets are one of the unsung heroes of Xcode. A snippet is a block of code that you can save, to a snippet library, and reuse over and over again. Snippets can even contain placeholders for you to know what you need to fill in. What’ s that you say? You want a snippet for documenting your code? Fabulous idea.

In **MathAPI.h**, copy and paste in the following, just before the comment you already have:

```
/*! * @discussion <#description#> * @param <#param description#> * @return <#return description#> */
```

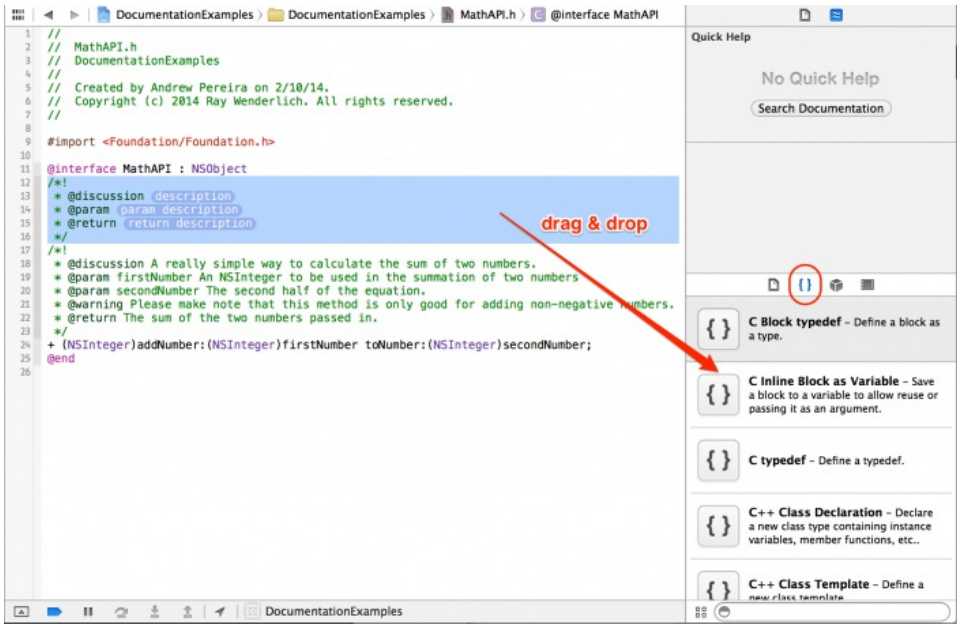
Notice that when you copied or wrote out the code, the parts that had “<# #>” around them became tokens that you could tab between. It’ s exactly the same thing when you use autocomplete for your code.



```
/*!
 * @discussion description
 * @param param description
 * @return return description
 */
```

Screen Shot 2014-04-05 at 5.35.35 PM

The next part may be a little tricky. You’ ll need to select the Code Snippet Library in the Utilities panel, highlight your comment block, and drag the text to the Code Snippet Library section (by dragging from one of the placeholder text areas like <#description#>):



```
//
//  MathAPI.h
//  DocumentationExamples
//
//  Created by Andrew Pereira on 2/10/14.
//  Copyright (c) 2014 Ray Wenderlich. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface MathAPI : NSObject
/*!
 * @discussion description
 * @param param description
 * @return return description
 */
/*!
 * @discussion A really simple way to calculate the sum of two numbers.
 * @param firstNumber An NSInteger to be used in the summation of two numbers
 * @param secondNumber The second half of the equation.
 * @warning Please make note that this method is only good for adding non-negative numbers.
 * @return The sum of the two numbers passed in.
 */
+ (NSInteger)addNumber:(NSInteger)firstNumber toNumber:(NSInteger)secondNumber;
@end
```

C Block typedef - Define a block as a type.

C Inline Block as Variable - Save a block to a variable to allow reuse or passing it as an argument.

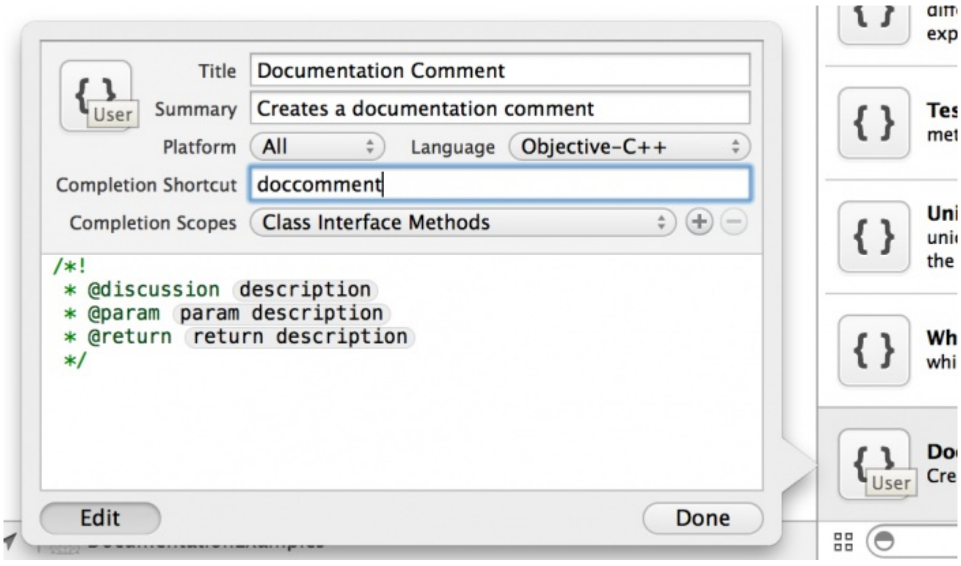
C typedef - Define a typedef.

C++ Class Declaration - Declare a new class type containing instance variables, member functions, etc..

C++ Class Template - Define a new class template

RW_Documentation_CreateSnippet

A popover will appear that lets you edit some information about the snippet and create an autocompletion shortcut. It will even let you edit the code. Fill out the information as you see it below:



Title	Documentation Comment
Summary	Creates a documentation comment
Platform	All
Language	Objective-C++
Completion Shortcut	doccomment
Completion Scopes	Class Interface Methods

```
/*!
 * @discussion description
 * @param param description
 * @return return description
 */
```

RW_Documentation_EditSnippet

If you ever need to change the code or autocomplete shortcut, you can always go back and do it later. Feel free to change it however you' d like, or make new ones. To edit the snippet, click on it in the Code Snippet Library, then click the **Edit** button.

To see your new snippet in action, delete the comments you added for **addNumber:toNumber:**, place your cursor to the left of the "+", type **doccomment**, and hit enter. Your snippet text will appear.

Tab through the three tokens filling out each one to complete your documentation as follows:

```
/*! * @discussion A really simple way to calculate the sum of two numbers. * @param firstNumber An NSInteger to be used in the summation of two numbers. * @param secondNumber The second half of the equation. * @warning Please make note that this method is only good for adding non-negative numbers. * @return The sum of the two numbers passed in. */
```

Note you' ll need to still manually add a second **@param** tag and the **@warning** tag, but this snippet still saves a ton of time.

You' re doing great so far. See, this documentation stuff isn' t so bad after all!

Documenting Typedefs

Open Car.h. You' ll see there are a few more things here than the previous class. There' s an **NS_ENUM**, typedef enum, a block, multiple properties, and a void method. Don' t freak out - it' s still really easy to document a class with more than one or two things in it. :]

Remember the **@typdef** tag? This Top-Level tag is a little special. It can be used to document things like **enum** typedefs and **struct** typedefs. But depending on what you are documenting, it should only contain Second-Level tags that match the type being defined.

For **enum**, you' ll want to include the **@constant** tag for each constant you have. (For **struct**, you would include **@field** tags.)

Find the enum **OldCarType**. It has two constants, and these should always be really old cars. Above the typedef declaration, replace the existing comment so that it looks like this:

```
/*! * @typedef OldCarType * @brief A list of older car types. * @constant OldCarTypeModelT A cool old car. * @constant OldCarTypeModelA A sophisticated old car. */typedefenum {   OldCarTypeModelT,   OldCarTypeModelA } OldCarType;
```

Build, and option+click on **OldCarType**. You' ll notice your popover has the information from your **@brief** tag. Now, option+click on **OldCarTypeModelA**. Did you see your comment? Queue the sad music. :[

But fear not, you can still get your information where you need it - we' ve got the trusty triple forward slash in our tool belt. Add the comments to the **enum** like you see here:

```
typedefenum{/// A cool, old car.   OldCarTypeModelT, /// A sophisticated older car. OldCarTypeModelA} OldCarType;
```

When you build, and option+click, you' ll now see your comment.

Since there' s an **NS_ENUM** in the class, go head try documenting it on your own. The constants are already documented, it just needs to be commented overall.

Solution Inside: Solution	Show

Note: Since this enum is created through a macro, sadly Xcode doesn' t actually give you the same documentation features as a traditional typedef enum, even though **NS_ENUM** is the recommended way to make enums. Perhaps one day this will change, but for now, that' s the way it is.

Now take a minute to document the **carType** property. **Add the following line** above it' s declaration, so it looks like this:

```
/// Indicates the kind of car as enumerated in the "CarType" NS_ENUM@property(nonatomic, assign) CarType carType;
```

Build again and then view your shorthand documentation by option-clicking the **`carType`** variable name.

Moving on, there’ s also a typedef block in Car.h. Commenting a block is no more difficult than anything you’ ve done so far. Add the text below so it looks like this:

```
/*! * @brief A block that makes the car drive. * @param distance The distance is equal to a distance driven when the block is ready to execute. It could be miles, or kilometers, but not both. Just pick one and stick with it. ;] */ typedefvoid(^driveCompletion)(CGFloat distance);
```

Notice that it’ s not too different from anything else. It has:

- A **`@brief`** tag, giving a very simple explanation about the block.
- A **`@param`** tag, to let you know that you’ll need to pass something as a parameter when you call the block

Adding Formatted Code to Your Documentation

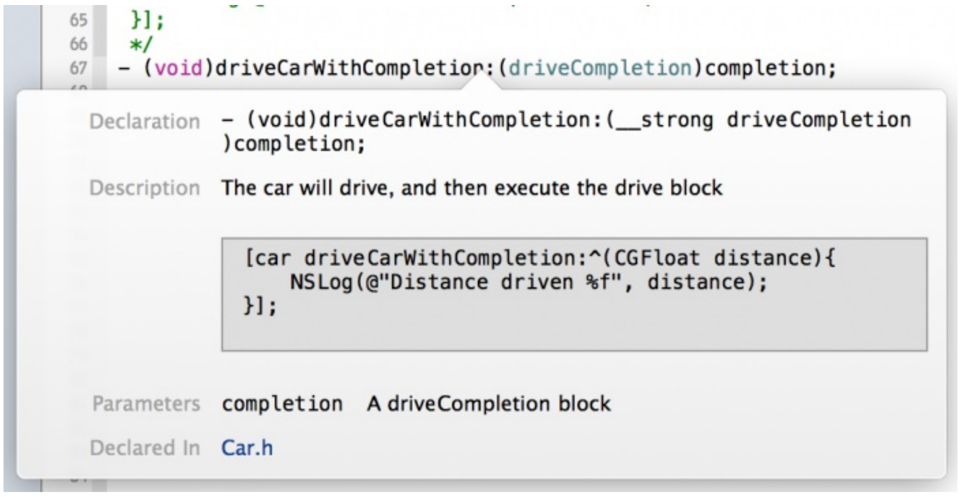
Sometimes, it’ s nice for a programmer to have an example of how to use a method.

To try this out, you’ re going to document the **`Car`** class’ **`driveCarWithCompletion:`** method. This method takes a block as a parameter, and it might be a good idea to show a programmer how to use it – since blocks seem to be a little hard for new developers.

For this, you can use a **`@code`** tag. Add the following in Car.h above the **`driveCarWithCompletion:`** declaration:

```
/*! * @brief The car will drive, and then execute the drive block * @param completion A driveCompletion block * @code [car driveCarWithCompletion:^(CGFloat distance){ NSLog(@"Distance driven %f", distance); }]; */
```

Build, and **option+click** the method name. You should see a nicely formatted code block in the popover. Beauty comes in many forms, readers.



```
}];
*/
- (void)driveCarWithCompletion:(driveCompletion)completion;
```

Declaration - (void)driveCarWithCompletion:(__strong driveCompletion)completion;

Description The car will drive, and then execute the drive block

```
[car driveCarWithCompletion:^(CGFloat distance){
    NSLog(@"Distance driven %f", distance);
}];
```

Parameters completion A driveCompletion block

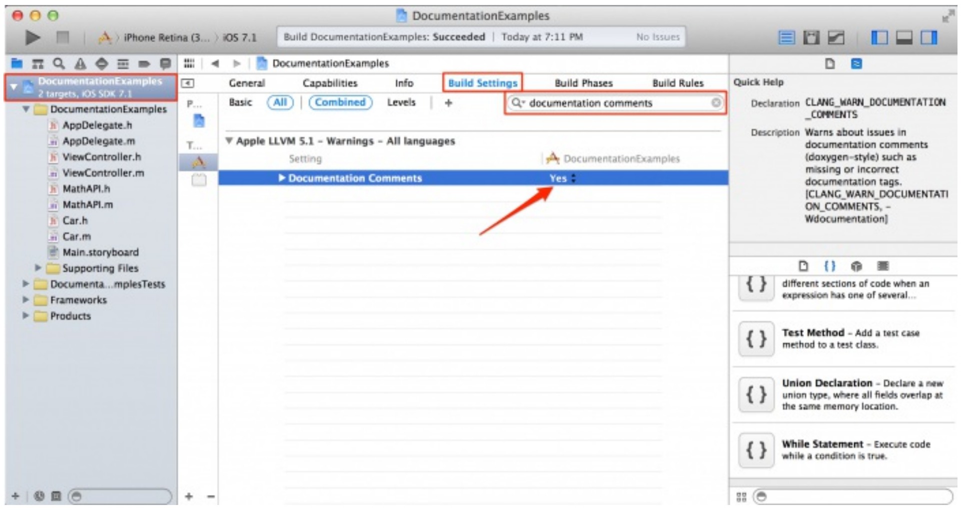
Declared In Car.h

RW_Documentation_Code

Checking Yourself

Now that you’ve learned how to comment, wouldn’t it be nice if Xcode would check your work, much the same way it checks if your code is typed correctly? Good news! Clang has a flag (“CLANG_WARN_DOCUMENTATION_COMMENTS”) that you can set to make Xcode check your HeaderDoc style comments.

In Xcode, open the DocumentationExamples **Project Settings**, click the **Build Settings** button at the top, search for **Documentation Comments**, and set the value to **YES**.


Build Settings

Basic | All | Combined | Levels

documentation comments

Apple LLVM 5.1 - Warnings - All languages

Setting	DocumentationExamples
Documentation Comments	Yes

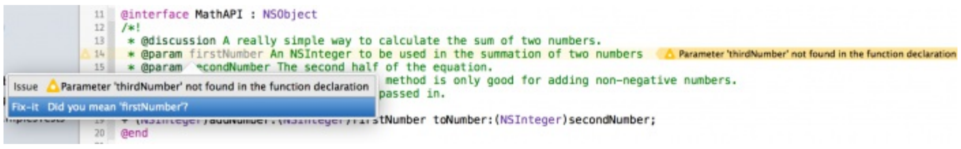
Quick Help

Declaration CLANG_WARN_DOCUMENTATION_COMMENTS

Description Warns about issues in documentation comments (doxygen-style) such as missing or incorrect documentation tags. [CLANG_WARN_DOCUMENTATION_COMMENTS, -Wdocumentation]

RW_Documentation_Warnings

To see it in action, open to **MathAPI.h**, and change your first @param tag’ s parameter name to **thirdNumber**, instead of **firstNumber**, then Build.


```
@interface MathAPI : NSObject
/*!
 * @discussion A really simple way to calculate the sum of two numbers.
 * @param firstNumber An NSInteger to be used in the summation of two numbers
 * @param secondNumber The second half of the equation.
```
Parameter 'thirdNumber' not found in the function declaration

Issue Parameter 'thirdNumber' not found in the function declaration

Fix-it Did you mean 'firstNumber'?

RW_Documentation_WarningEx

Notice that a warning was generated and even gave you an auto-fix option to change the variable to the correct name. While it won’ t catch everything, it can be very useful to double check yourself.

Special Types of Comments

While using HeaderDoc style comments to document your code is useful for yourself and others, there are a few other special types of comments Xcode supports. These are really useful for you, or others that work in your code.

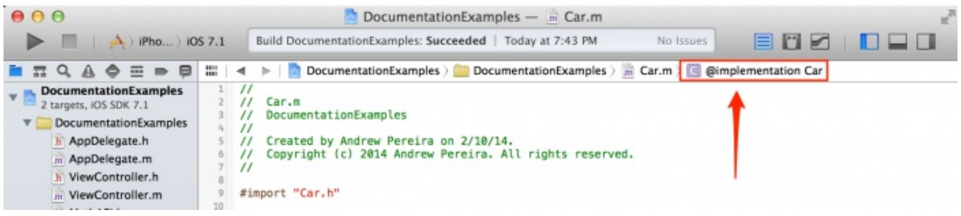
Open **Car.m** and in **driveCarWithCompletion:**, paste the following three comment lines above the call to **completion**:

```
// FIXME: This is broken// !!!: Holy cow, it should be checked!// ???: Perhaps check if the block is not nil first?
```

you have added three different types of comments here:

- **FIXME: A comment to yourself that something needs to be fixed.**
- **!!!: A comment that something needs attention.**
- **???: A comment that there is a question about the code. Or, the code is questionable.**

Not only are these types of comments useful to see as you go through code, Xcode recognizes them, and will show them in the Jump Bar at the top of the editor. Select the method menu from the Jump Bar, shown here:

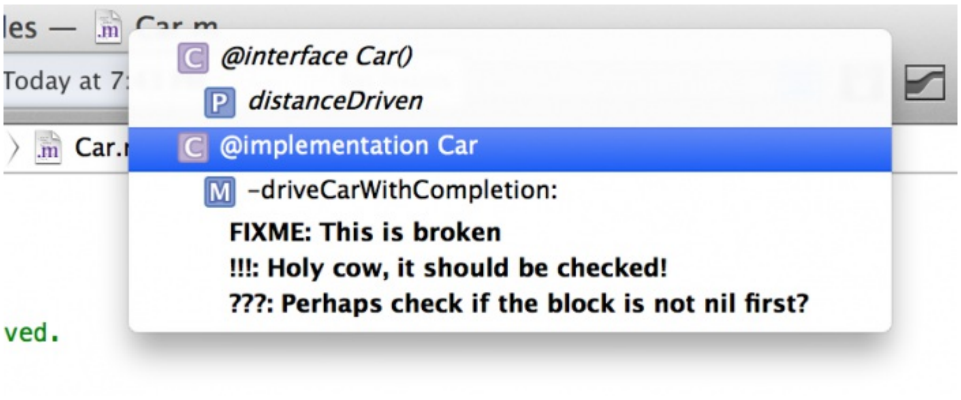

DocumentationExamples — Car.m
iPho... 〉 iOS 7.1
Build DocumentationExamples: Succeeded | Today at 7:43 PM
No Issues
DocumentationExamples 〉 DocumentationExamples 〉 Car.m 〉 @implementation Car
DocumentationExamples
2 targets, iOS SDK 7.1
DocumentationExamples
AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m

```
//
//  Car.m
//  DocumentationExamples
//
//  Created by Andrew Pereira on 2/10/14.
//  Copyright (c) 2014 Andrew Pereira. All rights reserved.
//

#import "Car.h"
```

RW_Documentation_JumpBar

Once you do, you’ll see the three comments you entered, bolded, with the text of your comment:



FIXME: This is broken
!!!: Holy cow, it should be checked!
???: Perhaps check if the block is not nil first?

RW_Documentation_JumpBarSelect

With all of this, you’ve learned enough to document the rest of this project.

Take some time to practice on the other properties and methods included in the project, and try adding a few things yourself. See what happens when you change things around in your comment blocks, or if you leave a tag out. It’s important to know how your formatting will impact your documentation.

Creating HTML Documentation with headerdoc2html

All of this documentation magic is done with a tool called HeaderDoc. It’s already installed on your computer if you have Xcode installed. Not only does it parse your comments, making those cool popovers and Quick Help panels, it can also create HTML, XML and man pages containing your documentation.

You’ll focus on creating HTML files in this section. But if you want to learn more about all of the options for creating stand-alone documentation with HeaderDoc, check out Appple’s HeaderDoc User Guide.

Open Terminal by going to **/Applications/Utitlities/Terminal**. Once you do, navigate to the DocumentationExamples project folder. You can do this by typing in the following:

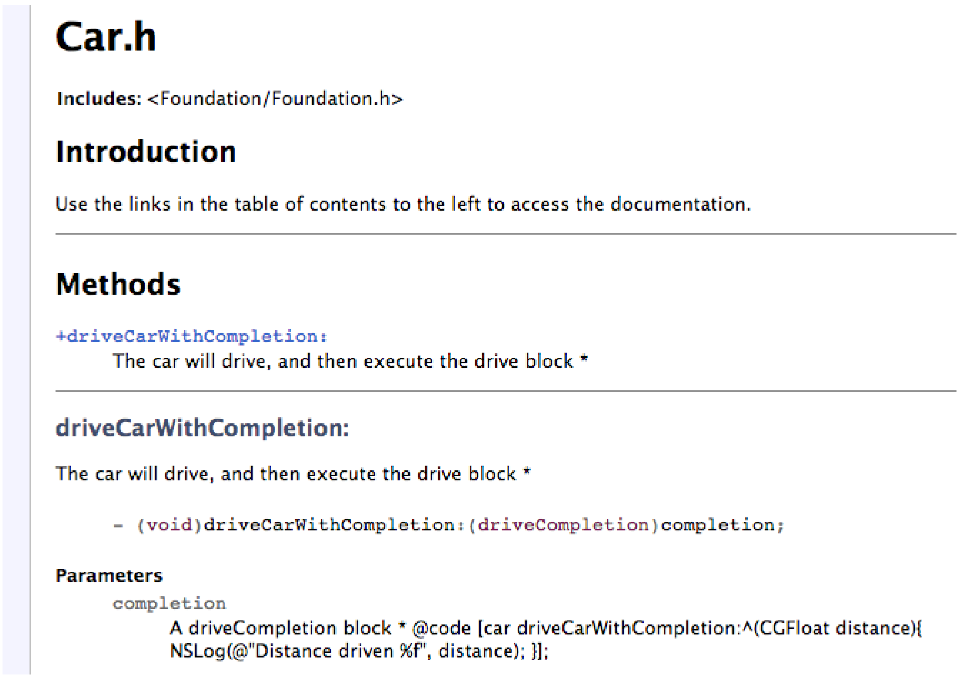
```
cd /path/to/your/folder
```

Make sure to enter the path to the top level folder that contains the Xcode project file. (“DocumentationExamples.xcodeproj”)

Now, create a set of HTML documentation files by typing in the following in Terminal:

```
headerdoc2html -o ~/Desktop/documentation DocumentationExamples/
```

You’ll see a lot of output in the Terminal. Once it finishes, go to your Desktop, and you' ll see a new directory named **documentation**. Open it up, and navigate to **Car_h**, and open **index.html**. Awesome – you have pretty docs!


Car.h

Includes: <Foundation/Foundation.h>

Introduction

Use the links in the table of contents to the left to access the documentation.

Methods

+driveCarWithCompletion:
The car will drive, and then execute the drive block *

driveCarWithCompletion:

The car will drive, and then execute the drive block *

```
- (void)driveCarWithCompletion:(driveCompletion)completion;
```

Parameters

completion
A driveCompletion block * @code [car driveCarWithCompletion:^(CGFloat distance){ NSLog(@"Distance driven %f", distance); }];

Screen Shot 2014-04-05 at 5.58.18 PM

So what just happened? Well, you ran the *headerdoc2html* script with 2 options:

- **-o ~/Desktop/documentation – This told the script to output its results to the documentation on your desktop.**
- **DocumentationExamples/ – This told the script to only parse files in the DocumentationExamples/ folder inside the project folder. (There are other folders in the project directory that do not contain code files.)**

Troubleshooting: You may find that with the latest versions of *headerdoc2html* and Google Chrome, that opening index.html does not correctly show the Table of Contents toc.html in the left sidebar. However, opening index.html in Safari should show both the documentation content and the Table of Contents correctly.

Also, you may have noticed that the documentation for the **carType** property you added earlier did not appear. It looks like the latest version of *headerdoc2html* does not correctly parse /// styled comments. For now, you might want to stick the /*! style.

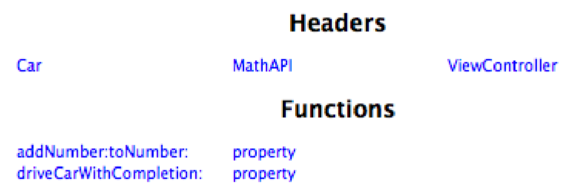
That' s all really cool, but it gets better. Instead of having to navigate down into the output directory you want, HeaderDoc can create a master table of contents page for you. Back in Terminal, navigate to your new documentation directory by entering the following:

```
cd ~/Desktop/documentation
```

Now, to create the table of contents, enter the following:

```
gatherheaderdoc .
```

gatherheaderdoc will now search your documentation directory, indicated by the period (representing the current Terminal directory) in the command. In Finder, go back to your documentation directory. You' ll notice there' s a new file named **masterTOC.html**. Open this file in Safari. You' ll see a page that contains links to all of your documented properties, methods, enums and blocks.


Headers

Car MathAPI ViewController

Functions

addNumber:toNumber: property
driveCarWithCompletion: property

Screen Shot 2014-04-05 at 6.01.35 PM

Now, you can host all of this HTML content on a web server, and give access to your documentation to anyone!

VVDocumenter-Xcode

Last up in our roundup of documentation tips is *VVDocumenter-Xcode*, a third party Xcode plugin that will make documenting even easier than using the Code Snippet you created earlier.

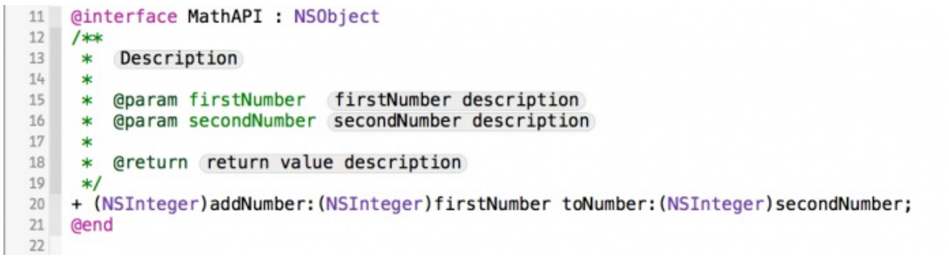
To get started, download the plugin from Github.

All you need to do is open the project, and build. As it builds, it will automatically install the plugin for you in your **~/Library/Application Support/Developer/Shared/Xcode/Plug-ins** directory.

Once it finishes, restart Xcode, and open your DocumentationExamples project if it is not already open. In **MathAPI.h** delete the entire comment block for **addNumber:toNumber:**. Now, type the following above the method declaration:

```
///
```

Once you do, *VVDocumenter-Xcode* will automatically create a comment block with all of the necessary @param tags, with autocomplete tokens between all necessary fields.



```
@interface MathAPI : NSObject
/**
 *  Description
 *
 *  @param firstNumber  firstNumber description
 *  @param secondNumber secondNumber description
 *
 *  @return return value description
 */
+ (NSInteger)addNumber:(NSInteger)firstNumber toNumber:(NSInteger)secondNumber;
@end
```

RW_Documentation_VVDocumentor

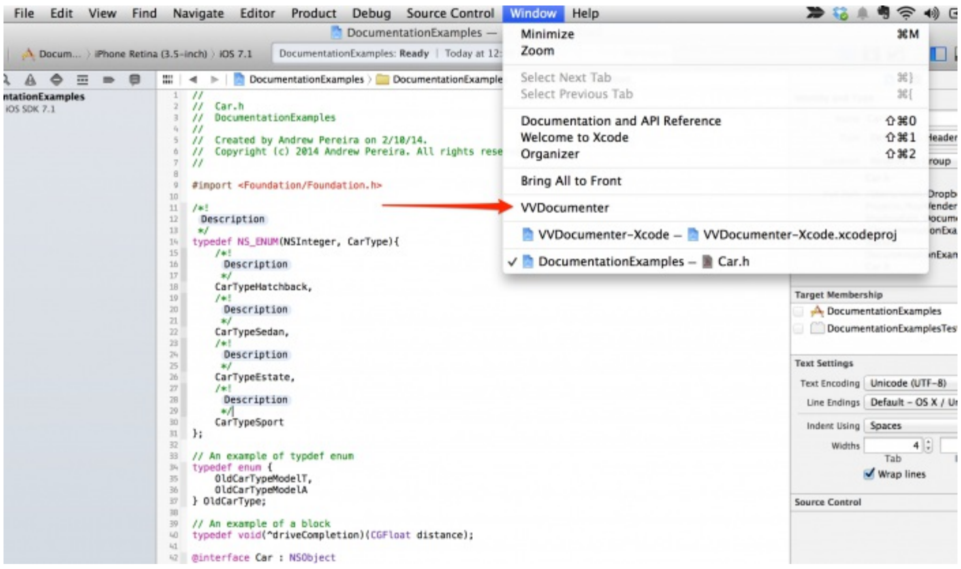
Convenient, eh?

Now, go to **Car.h** and delete all of your comments for the CarType **NS_ENUM**, even the comments for each constant. Above the **NS_ENUM** declaration, type:

```
///
```

This time, it created the discussion section above the enum, and even put the necessary comments above each constant!

VVDocumenter-Xcode can make your life so much easier. If you want to customize VVDocumenter-Xcode, in Xcode, go to **Window>VVDocumenter**.



```
//
//  Car.h
//  DocumentationExamples
//
//  Created by Andrew Pereira on 2/10/14.
//  Copyright (c) 2014 Andrew Pereira. All rights rese

#import <Foundation/Foundation.h>

/*!
 Description
 */
typedef NS_ENUM(NSInteger, CarType){
    /*!
     Description
     */
    CarTypeHatchback,
    /*!
     Description
     */
    CarTypeSedan,
    /*!
     Description
     */
    CarTypeEstate,
    /*!
     Description
     */
    CarTypeSport
};

// An example of typdef enum
typedef enum {
    OldCarTypeModelT,
    OldCarTypeModelA
} OldCarType;

// An example of a block
typedef void(^driveCompletion)(CGFloat distance);

@interface Car : NSObject
```

VW_Documentation_VVDocPrefs

Here, you can change the autocomplete keyword, the comment style, and more. Feel free to customize it however you’d like. I’ve found it saves a ton of time.

Where To Go From Here?

You can download the finished project from this HeaderDoc tutorial here.

As a challenge, go through some of your own code, and create documentation for yourself. Try using the code snippet you made, and *VVDocumentor*. See what fits your style, and how you can work it in to your workflow.

Also, be sure to check out Appple’s HeaderDoc User Guide for more information now that you know the basics.

If you have any questions or comments on this tutorial, please join the discussion below!