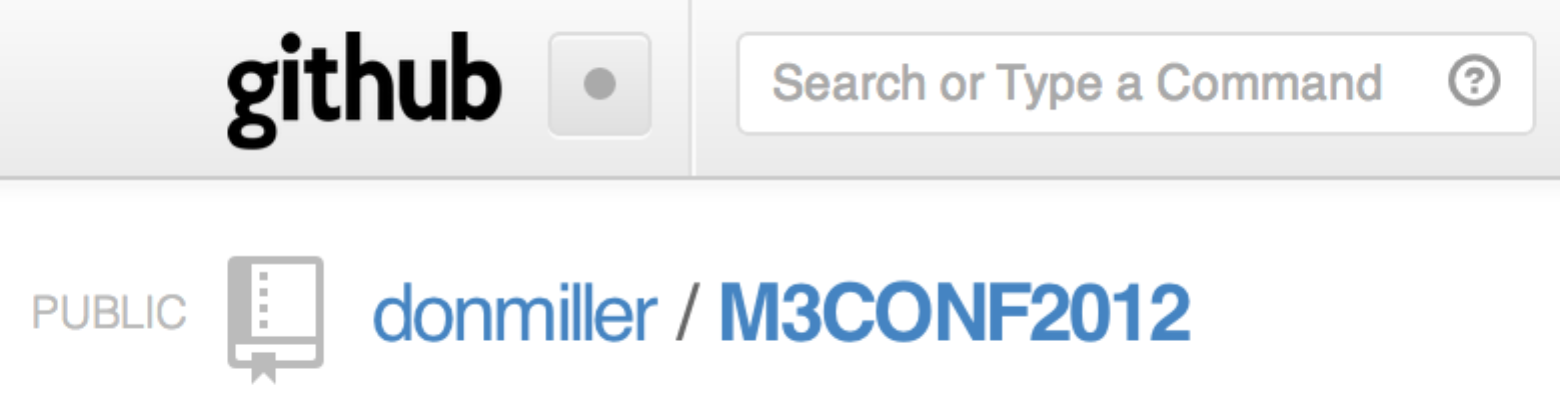
Don Miller @donmiller Toledo, OH



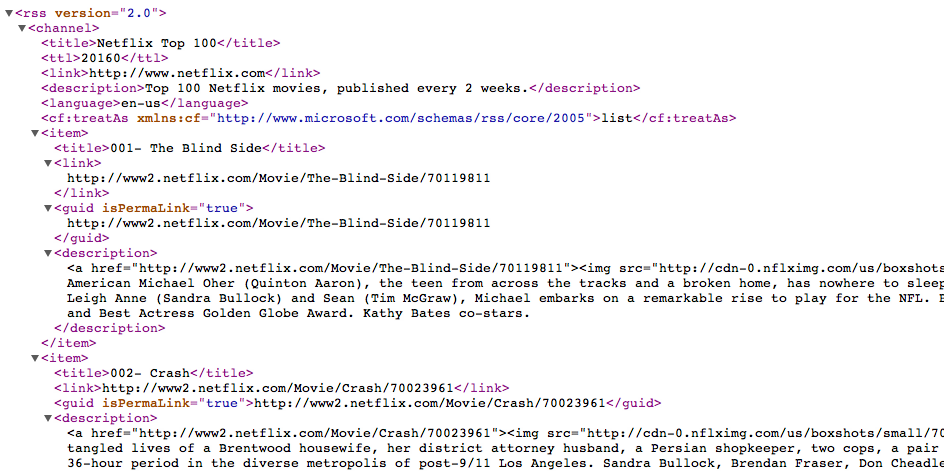
  



Parsing an XML RSS Feed

1.0: Understanding What You Are Parsing

[RSS](http://en.wikipedia.org/wiki/RSS) stands for Really Simple Syndication. It is a web feed put into a standard XML format with certain tags that are used to pull information easily from a web site. Most RSS feeds are public and the topics distributed can range from blog feeds, podcasts, video casts, or other information that is updated regularly. It has a URL that can be used to find the latest updates of information. An example of an RSS feed can be the Netflix Top 100 movies feed. This feed obviously changes all of the time and we can write an iPhone app that displays the top 100 movies and some detail information on each movie. The below example is the XML output of the Netflix Top 100 RSS feed: <http://rss.netflix.com/Top100RSS>.



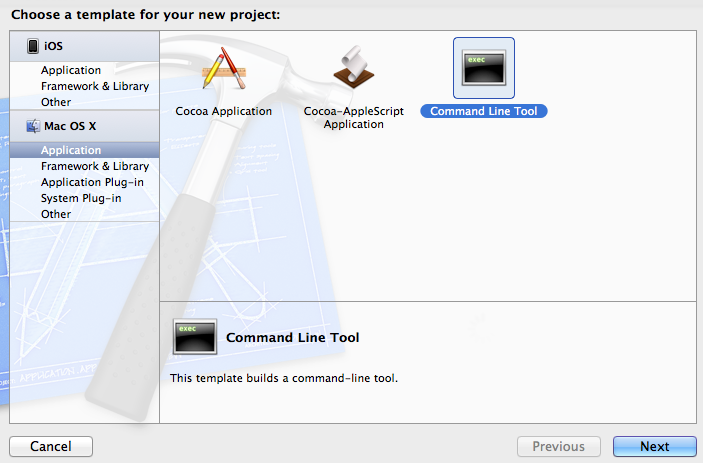
The channel element consists of some header information on the feed; such as title, link, and description. Then there is a list of 100 items that consists of elements; such as title, link to movie thumbnail, and description.

2.0: Creating a Class to Parse an XML File

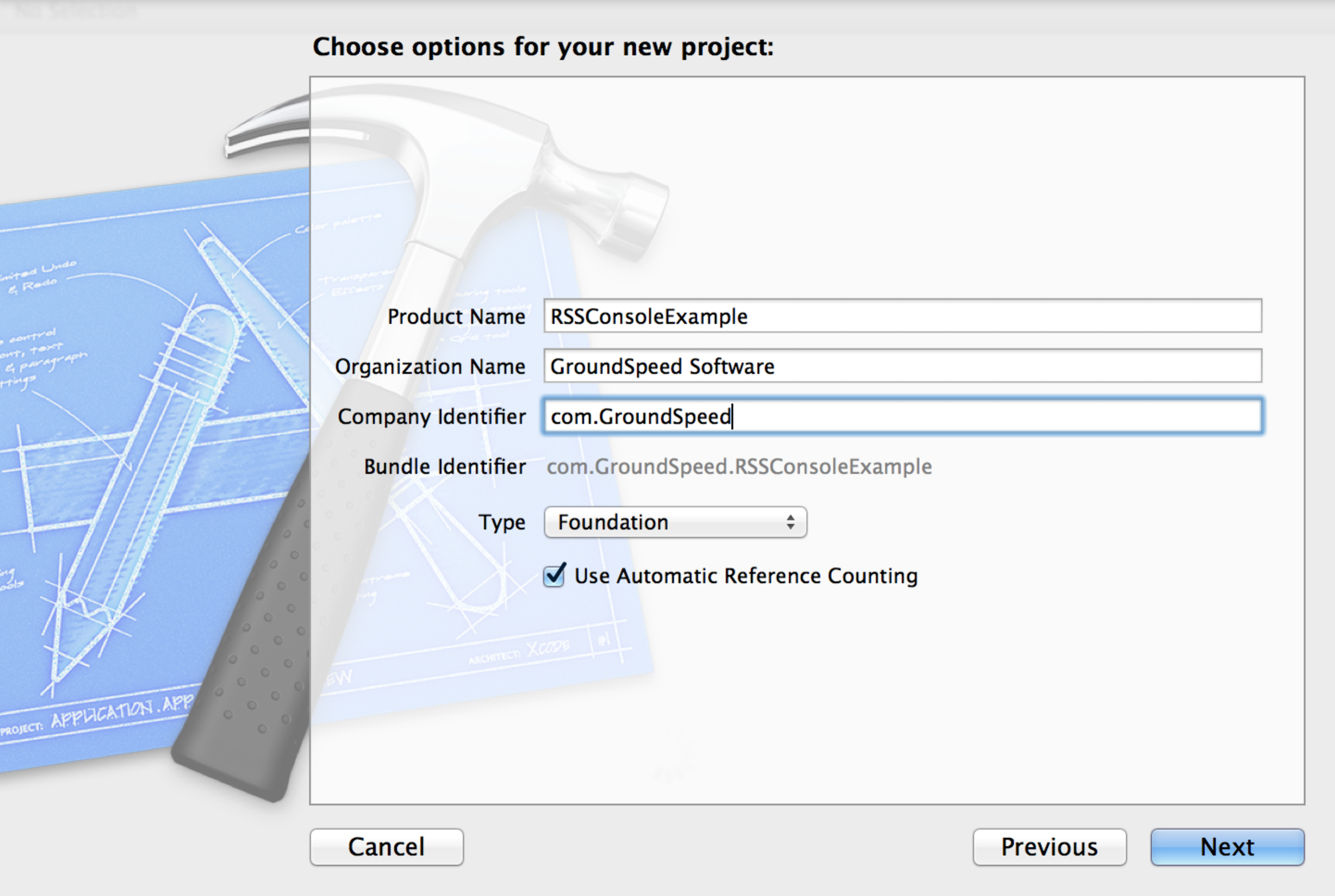
Parsing is a pretty straightforward process; however, it takes time to understand the needed classes and datatypes to handle the parsing of the XML file. In this first example, we will create a console application that reads an RSS feed and displays the information needed.

* NSXMLParser datatype overview
* NSURL datatype overview

***Step 1:*** Create a Console Application



***Step 2:*** Name the project, for this example we will use, “RSSConsoleExample”. Make sure to keep the default type as a Foundation program.



***Step 3A:*** Add an Objective-C Class to handle the parsing.

|  |  |
| --- | --- |
|  |  |

***Step 3B:*** Name the new class Parser and make it a subclass of NSObject.

|  |  |
| --- | --- |
|  |  |

***Step 4:*** Add the header information for Parser.h

#import <Foundation/Foundation.h>

@interface Parser : NSObject <NSXMLParserDelegate>

{

NSXMLParser \*rssParser;

NSMutableArray \*movies;

NSMutableDictionary \*item;

NSString \*currentElement;

NSMutableString \*currentTitle, \*currentDate;

NSMutableString \*currentSummary, \*currentLink;

}

-(void) parseXMLFileAtURL:(NSString \*)URL;

@end

***Step 5:*** Add the init method to write a string to show us that the Parser class is being called. [Parser.m]

- (id)init

{

self = [super init];

if (self)

{

NSLog(@"Found file and started parsing.");

}

return self;

}

***Step 6:*** Create parseXMLFileAtURL method to populate the movies array. [Parser.m]

- (void)parseXMLFileAtURL:(NSString \*)URL

{

movies = [[NSMutableArray alloc] init];

NSURL \*xmlURL = [NSURL URLWithString:URL];

rssParser = [[NSXMLParser alloc] initWithContentsOfURL:xmlURL];

[rssParser setDelegate:self]; //Receive callbacks

[rssParser parse];

}

***Step 7:*** Create parser override method to handle potential errors with the RSS feed. [Parser.m]

- (void)parser:(NSXMLParser \*)parser

parseErrorOccurred:(NSError \*)parseError

{

NSString \*errorString = [NSString stringWithFormat:@"Unable to download movie feed from web site (Error code %li)", [parseError code]];

NSLog(@"Error parsing XML: %@", errorString);

}

***Step 8:*** Create a parser method the handles the XML element and initializes the item dictionary.

- (void)parser:(NSXMLParser \*)parser

didStartElement:(NSString \*)elementName

namespaceURI:(NSString \*)namespaceURI

qualifiedName:(NSString \*)qName

attributes:(NSDictionary \*)attributeDict

{

NSLog(@"Found this element: %@", elementName);

currentElement = [elementName copy];

if ([elementName isEqualToString:@"item"])

{

item = [[NSMutableDictionary alloc] init];

currentTitle = [[NSMutableString alloc] init];

currentSummary = [[NSMutableString alloc] init];

currentLink = [[NSMutableString alloc] init];

}

}

***Step 9:*** Create a parser class that handles setting the elements in the item dictionary. This class allows for us to look inside the item element to parse out the title, link, and summary. Once we have that information, we will add it to the movies array.

- (void)parser:(NSXMLParser \*)parser

didEndElement:(NSString \*)elementName

namespaceURI:(NSString \*)namespaceURI

qualifiedName:(NSString \*)qName

{

NSLog(@"Ended element: %@", elementName);

if ([elementName isEqualToString:@"item"])

{

// save values to an item, then store into array...

[item setObject:currentTitle forKey:@"title"];

[item setObject:currentLink forKey:@"link"];

[item setObject:currentSummary forKey:@"summary"];

[movies addObject:[item copy]];

NSLog(@"Adding movie: %@", currentTitle);

// NSLog(@"Summary: %@", currentSummary);

// NSLog(@"Link: %@", currentLink);

}

}

***Step 10:*** This is the real brain of the operation. We are using this parser class to filter through the elements that we are most interested in; which is the title, link, and description. Since we initialized these variables in Step 8, we can now look at the current element and add to our strings depending on the element that we find.

- (void)parser:(NSXMLParser \*)parser foundCharacters:(NSString \*)string

{

NSLog(@"Found characters: %@", string);

// save the characters for the current item...

if ([currentElement isEqualToString:@"title"])

{

[currentTitle appendString:string];

}

else if ([currentElement isEqualToString:@"link"])

{

[currentLink appendString:string];

}

else if ([currentElement isEqualToString:@"description"])

{

[currentSummary appendString:string];

}

}

***Step 11:*** Let’s create the class to see if we are all done.

- (void)parserDidEndDocument:(NSXMLParser \*)parser

{

NSLog(@"All done!");

NSLog(@"Movies array has %li items", [movies count]);

}

***Now call new class from main.m:*** We now have an object-oriented class that we will be able to use outside of our console program in the future. However, we will modify our main class to call this class within the console program. Open the main.m class and make the following changes:

#import <Foundation/Foundation.h>

#import "Parser.h"

int main(int argc, const char \* argv[])

{

@autoreleasepool

{

NSString \*path = @"http://rss.netflix.com/Top100RSS";

// Initialize Parser class and run parseXMLFileAtURL

Parser \*myParser = [[Parser alloc] init];

[myParser parseXMLFileAtURL:path];

return 0;

}

return 0;

}

3.0: Use our Parser Class to Populate a UITableView

Now that we have created a parser class and executed it in the console app, we can now copy the parser class into a new project to display a UITableView and Detail screen.

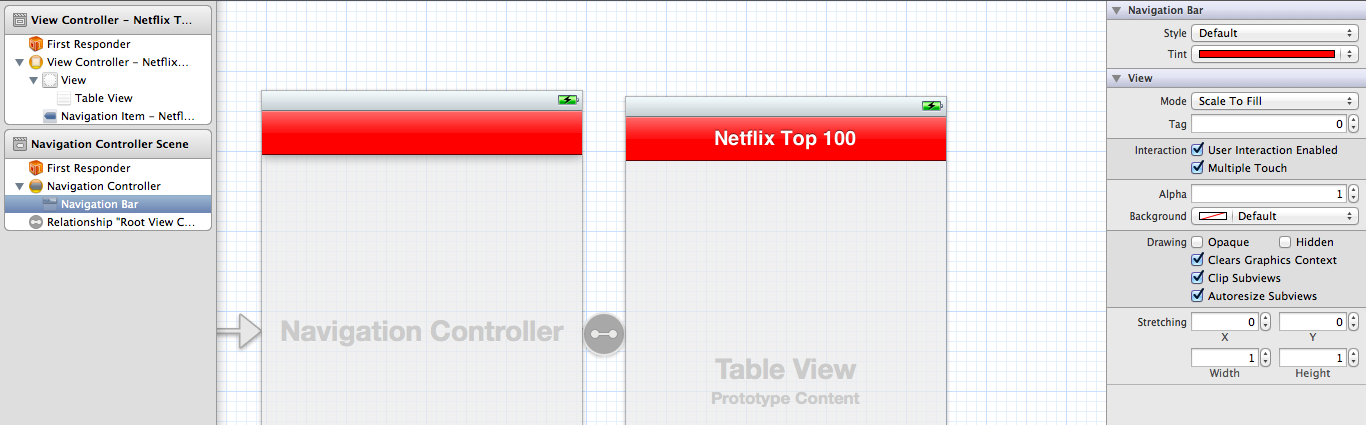
***Step 1:*** Create a new Single View Application

|  |  |
| --- | --- |
|  |  |

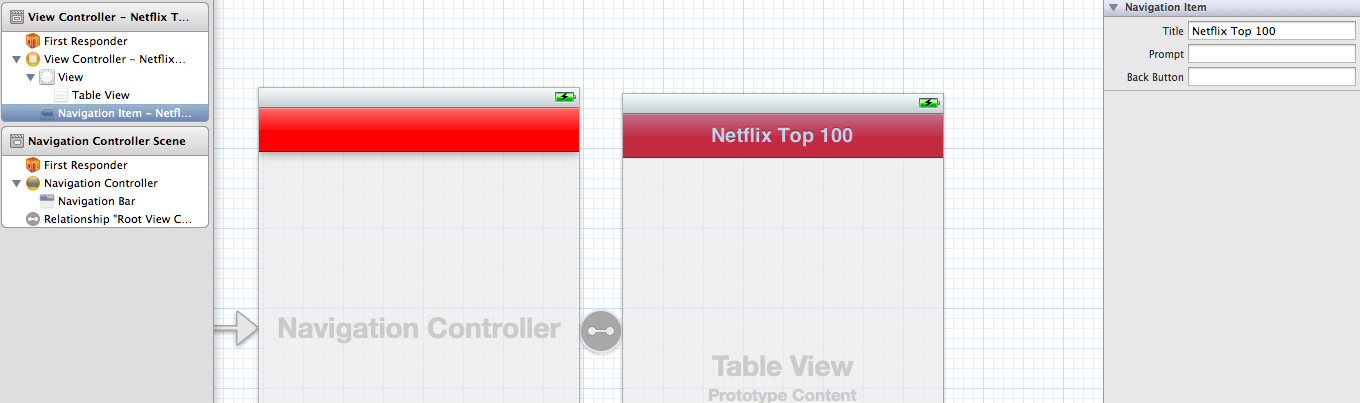
***Step 2:*** Navigate to the MainStoryboard.storyboard file. Select the view. Then select Editor -> Embed-In -> Navigation Controller. Then drag a table view to the new controller.

|  |  |
| --- | --- |
|  |  |

***Step 3:*** Change the Navigation Bar tint to [Red] to match the Netflix theme colors.



***Step 4:*** Change the title to the Navigation Item to read “Netflix Top 100”. This is done on the Navigation item in the View Controller.



***Step 5:*** Change the ViewController.h file to include the [items] array

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController<UITableViewDelegate, UITableViewDataSource>

{

NSMutableArray \*items;

}

@property (retain, nonatomic) NSMutableArray \*items;

***Step 6:*** Within ViewController.m, implement the numberOfRowsInSection method by adding the number of rows in the table by modifying the following method’s return.

- (NSInteger)tableView:(UITableView \*)tableView

numberOfRowsInSection:(NSInteger)section

{

return [items count];

}

***Step 7:*** Add the Parser Class that we created in the console app to your project under the Supporting Files group. Make sure that the [Copy items] box is checked.

|  |  |
| --- | --- |
|  |  |

***Step 8a:*** Let’s modify the Parser class to make the movies array accessible. This is a problem with the original creation in the class and “in the real world”, we would modify the original class before including in this new class. This is also a good example on why you would use a reference of a class in a corporate or consolidated development environment.

@interface Parser : NSObject <NSXMLParserDelegate>

{

NSXMLParser \*rssParser;

NSMutableArray \*movies;

NSMutableDictionary \*item;

NSString \*currentElement;

NSMutableString \*currentTitle, \*currentDate, \*currentSummary, \*currentLink;

}

@property (nonatomic, retain) NSMutableArray \*movies;

-(void) parseXMLFileAtURL:(NSString \*)URL;

***Step 8b:*** We will synthesize the object in the implementation file [Parser.m]

@implementation Parser

@synthesize movies;

***Step 9:*** We’re still not done. Go back to the ViewController.m and synthesize the items object we created in step 5. We also want to import the Parser.h class to have access to it. [ViewController.m]

#import "ViewController.h"

#import "Parser.h"

@implementation ViewController

@synthesize items;

***Step 10A:*** Make sure that we have the loadData added to the ViewController.h header file. We will use this to load the data into the items array.

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController <UITableViewDelegate, UITableViewDataSource>

{

NSMutableArray \*items;

}

@property (retain, nonatomic) NSMutableArray \*items;

-(void) loadData;

@end

***Step 10B:*** Inside the ViewController.m, let’s add the loadData method and receivedItems handler.

- (void)loadData

{

if (items == nil)

{

Parser \*rssParser = [[Parser alloc] init];

[rssParser parseXMLFileAtURL:@"http://rss.netflix.com/Top100RSS"];

items = [rssParser movies];

}

}

- (void)receivedItems:(NSMutableArray \*)theItems

{

items = theItems;

}

- (void)viewDidLoad

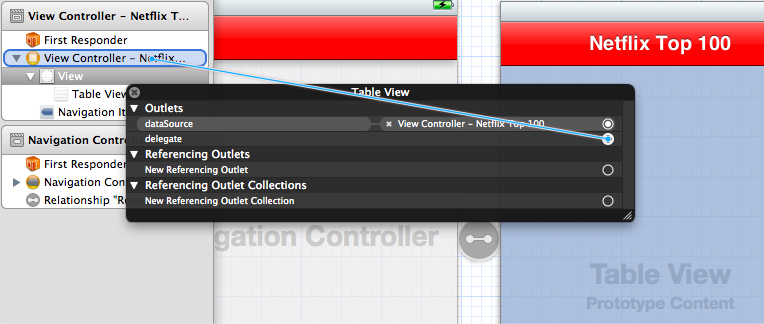
{

[super viewDidLoad];

[self loadData];

}

***Step 10c:*** In order to get the data to push to the table view, we have go to the Interface Builder to link the ViewController to the delegate and datasource outlets.



***Step 11:*** Inside the ViewController.m, let’s implement the tablview cellForRowAtIndexPath method and configure the cell’s label by adding the lines below.

- (UITableViewCell \*)tableView:(UITableView \*)tableView

cellForRowAtIndexPath:(NSIndexPath \*)indexPath

{

static NSString \*CellIdentifier = @"Cell";

UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

if (cell == nil) {

cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];

}

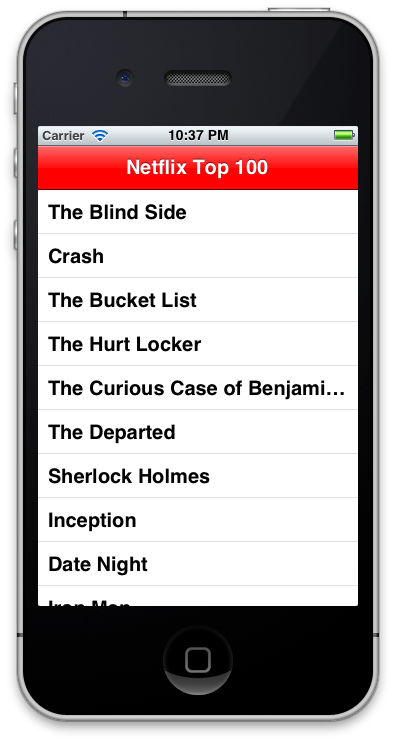
// Configure the cell.

cell.textLabel.text = [[items objectAtIndex:indexPath.row] objectForKey:@"title"];

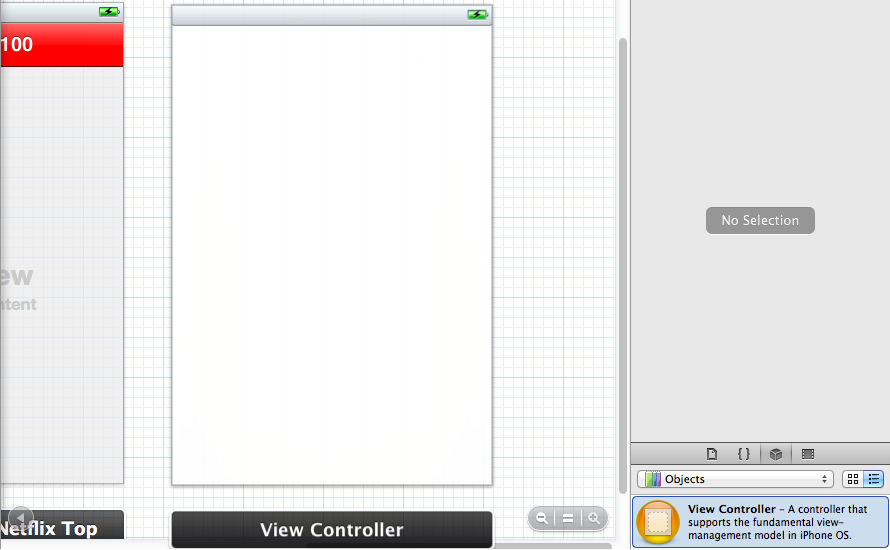
return cell;

}

At this point we should be able to see the initial table view populated.



***Step 12A:*** Our next step is to add the detail view screen. This will be a simple screen with a label and webview that displays the movie when selected. Search for a View Controller in the bottom right, and drag it to the Storyboard as shown below.



***Step 12B:*** Now let’s add the controller classes to the project. Right click on the first group (with our view controller) and add a file as shown below.

|  |  |
| --- | --- |
|  |  |

***Step 12B:*** Make sure to select UIViewController as the subclass and name the controller “DetailViewController”. We will also leave the xib box unchecked because we handled it in the storyboard.

|  |  |
| --- | --- |
|  |  |

***Step 13:*** Inside the DetailViewController.h file, we will need to change the header to setup the blueprint for controller. This consists of our two outlets and the dictionary that we are working with.

@interface DetailViewController : UIViewController

{

NSDictionary \*item;

IBOutlet UILabel \*itemTitle;

IBOutlet UIWebView \*itemSummary;

}

@property (nonatomic, retain) NSDictionary \*item;

@property (nonatomic, retain) IBOutlet UILabel \*itemTitle;

@property (nonatomic, retain) IBOutlet UIWebView \*itemSummary;

@end

***Step 14***: Let’s finish up the DetailViewController.m file by adding the following code.

@synthesize item, itemTitle, itemSummary;

- (void)viewDidLoad

{

[super viewDidLoad];

self.itemTitle.text = [item objectForKey:@"title"];

[self.itemSummary loadHTMLString:[item objectForKey:@"summary"] baseURL:nil];

}

- (void)viewDidUnload

{

[super viewDidUnload];

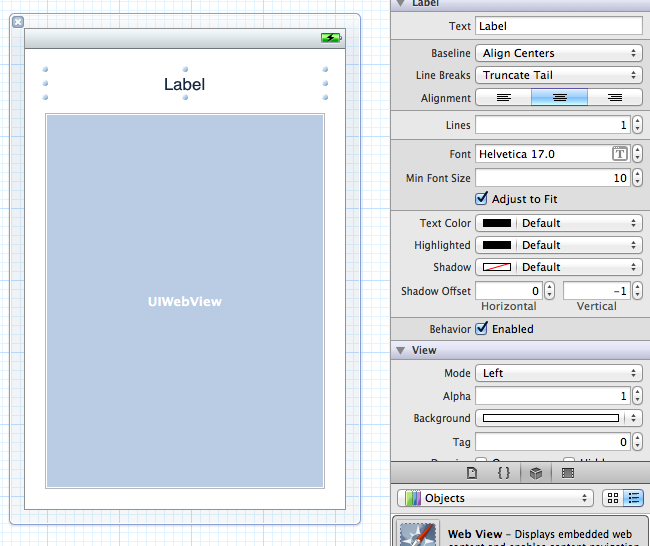
self.item = nil;

self.itemTitle = nil;

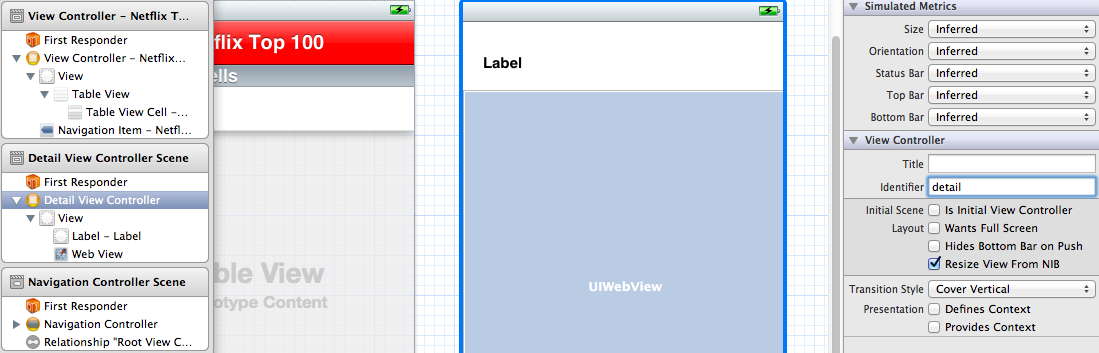
self.itemSummary = nil;

}

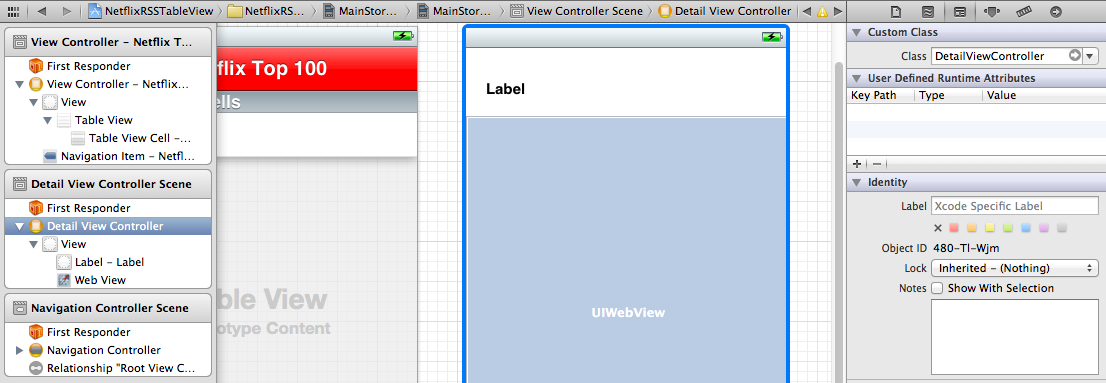
***Step 16:*** Add the Label and the UIWebView to the new DetailViewController.



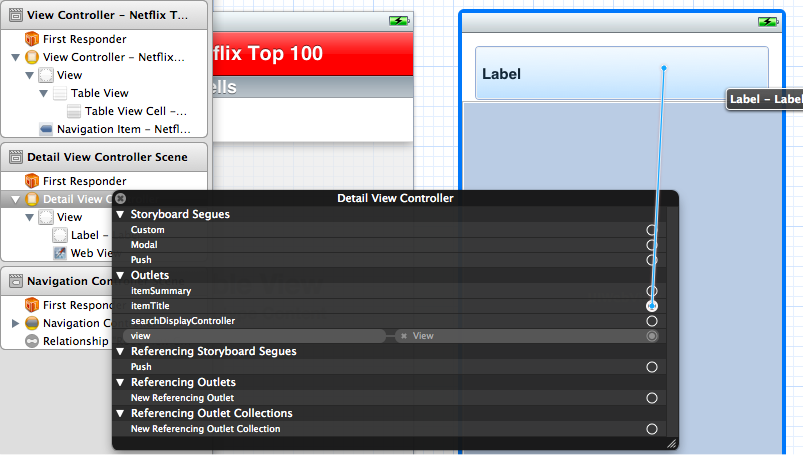
***Step 17a:*** Let’s link up your DetailViewController in the Interface Builder. First, change the identifier on your Detail View Controller.



***Step 17b:*** Associate your DetailViewController class with the Detail View Controller. This is found in the identity inspector on the right hand side (3rd icon over).



***Step 17c:*** Right click your Detail View Controller within the “Scene” and associate your IBOutlets to the label and webview.

******

***Step 18:*** Now lets finish up the ViewController.m file by adding the following code to the outstanding methods.

#import "DetailViewController.h"

- (void)tableView:(UITableView \*)tableView didSelectRowAtIndexPath:(NSIndexPath \*)indexPath

{

DetailViewController \*detail = [self.storyboard instantiateViewControllerWithIdentifier:@"detail"];

detail.item = [items objectAtIndex:indexPath.row];

[self.navigationController pushViewController:detail animated:YES];

}