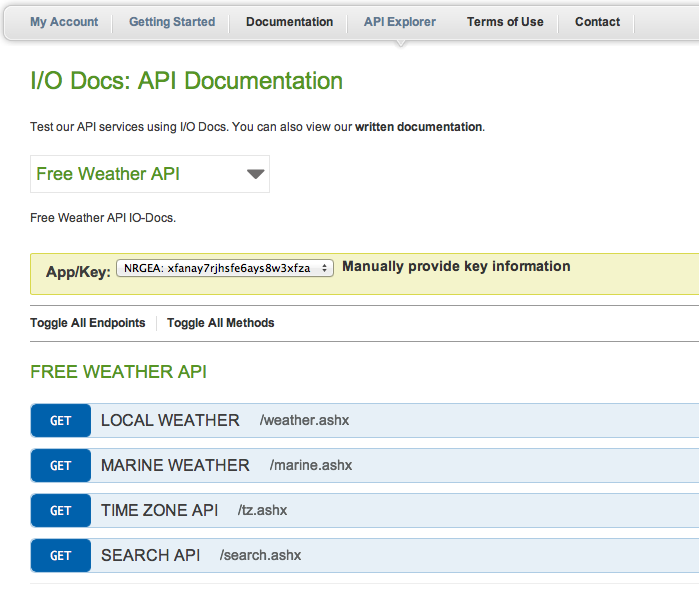
MOCK DATA – Pro Free Web Service Weather App

Before jumping into creating a web service, we will take a refreshment course using our previous project, the flickr based API calls. We will basically modify them to look into another webservice. This time we will quickly look at a more complex response. This will set us up for the meat of this tutorial, creating our own web service and interacting with it.

The basics remain the same throughout most web services. After all that is the purpose of a web service. Let’s try a weather service from WorldWeather.com who offer both a free and paid service. I used the free service for this example.

You can sign up here for a free API account: <http://www.worldweatheronline.com/free-weather-feed.aspx>

Once logged into your account, go to the API Explorer. Select the FREE API from the dropdown list and let’s GET the local weather. Here is a screenshot of where you should be at:



If you click on the blue GET button to the left of Local Weather, it will expand to show you the different options your GET HTTP Request can have. Leave the values as is and click on the Try it! Button at the bottom. You should get this:



This is basically telling us what the API request should be, in other words, the Request URI (Universal Resource Identifier). So let’s copy that link and past it into a browser window and see what we get. You should get a bunch of text that starts like this:

{ "data": { "current\_condition"

This is basically returning a JSON Dictionary, which you must now parse. So now let’s take this URI and put it into our previous flickr app, where the flickr URL used to be. Your code will look like this:

- (void)viewDidLoad{

[super viewDidLoad];

NSURL \*myURL = [NSURL URLWithString:@"http://api.worldweatheronline.com/free/v1/weather.ashx?q=London&format=json&num\_of\_days=5&key=xfanay7rjhsfe6ays8w3xfza"];

NSURLRequest \*myRequest = [NSURLRequest requestWithURL:myURL];

NSURLConnection \*myConnection = [NSURLConnection connectionWithRequest:myRequest delegate:self];

}

Run your app again. Your app will crash because we are still trying to parse our NEW data with our old schema or structure. If you had an NSLog for your flickrDictionary in your connection:didReceiveResponse method, add a breakpoint to that line or just after it. This will stop the program execution at that line and let you see the NSLog before the app crashes. (Breakpoints must be activated for the execution to stop at each breakpoint). You can also simply let it crash and look in the console. The received response will still be logged there.

The idea is to see what the NEW response dictionary looks like in order to re-structure our tableview methods to handle the new schema. Now, if you look at the line that logs the string itself, it looks like unreadable code. However, if you log the dictionary itself, you can see the structure itself in a clearer way. The new schema looks something like this:

1. data
   1. current condition
      1. a bunch of parameters such as cloudcover
   2. request
   3. weather
      1. date for 5 days since that’s the value we chose.

Ok now comes some brain racking thought processing. We have a data dictionary with 3 entries. We want some data from the first entry called “current\_condition”, specifically we want its “weatherDesc” key. So its “data.current\_condition.weatherDesc” the branch we wish to reach. Let me run through my way of doing it. This will introduce you to a method called isKindOfClass. We know our flickrDictionary is an NSDictionary because I basically told you to use that object. But let’s really test it. Add these lines below your NSLog flickrDictionary line:

//test

if ([flickrDictionary isKindOfClass:[NSDictionary class]]) {

NSLog(@"Yup, its a dictionary alright!");

}

Now run your app and check the console for that text. One quick and easy way to do it, especially when you have lots of things logging in the console, is to use the Find feature in your console. You must click inside the console to make sure the Find search bar opens up FOR the console and not for the editor window on top. So click in the console (after the app crashed or stopped at the breakpoint) and fo Cmd+F to open the search bar on the top right. Simply copy and paste the text.

Ok great, so you know it’s a dictionary. Big whoop, right?! If flickrDictionary is a dictionary, it must contain entries. One of the entries you see right at the beginning is “data” so let’s get that key’s object & log it! So let’s replace your //test code block with this:

//test

if ([flickrDictionary isKindOfClass:[NSDictionary class]]) {

id object = [flickrDictionary valueForKey:@"data"];

if ([object isKindOfClass:[NSDictionary class]]) {

NSLog(@"Again!");

}

}

What we did was take the dictionary’s data entry (which we could see in the console) and put it into an id type object. This means it’s an object of unknown type. We then test if THAT object is a dictionary. Run the app and find the logged text. Now let’s look for the next entry, “current\_condition”:

//test

if ([flickrDictionary isKindOfClass:[NSDictionary class]]) {

id object = [flickrDictionary valueForKey:@"data"];

if ([object isKindOfClass:[NSDictionary class]]) {

id anotherObject = [object valueForKey:@"current\_condition"];

if ([anotherObject isKindOfClass:[NSDictionary class]]) {

NSLog(@"OMG!");

}

}

}

Great so run it again! Hmmm…we were getting carried away for a moment there. Couldn’t find the “OMG!” text could you? That’s because that next object is not a dictionary. Go ahead and replace the NSDictionary class for NSArray class in that last if test and check the console again.

This is a very handy way of testing for unknown objects. Web services are notorious for returning json strings, which are not very human readable. Not as much as JSON claims to be anyway. So its always good to test. So now we have an array inside the “current\_condition” entry. One way to tell them apart is that dictionaries start with a “{“ curly brace whereas arrays start with a “(“ or “[“.

So this object inside “current\_condition” is an array, which has objects at certain indices instead of values or objects for keys. So to get our first object from that array we will now do:

if ([flickrDictionary isKindOfClass:[NSDictionary class]]) {

id object = [flickrDictionary valueForKey:@"data"];

if ([object isKindOfClass:[NSDictionary class]]) {

id anotherObject = [object valueForKey:@"current\_condition"];

if ([anotherObject isKindOfClass:[NSArray class]]) {

id firstArrayObject = [anotherObject objectAtIndex:0];

if ([firstArrayObject isKindOfClass:[NSDictionary class]]) {

NSLog(@"I cant take this anymore");

}

}

}

}

Run it now and find your text. Ok so now we are starting to get a feel for what a meaty server response looks like. We could go on like this but I think you get the point. Basically we would like to get the value for weatherDesc, which is an entry inside this new dictionary. But weatherDesc’s value is another array with a dictionary at its index:0 which itself has a dictionary consisting of only 1 entry called “value”…phew! That’s the one we want, which means we get it like this:

//test

if ([flickrDictionary isKindOfClass:[NSDictionary class]]) {

id object = [flickrDictionary valueForKey:@"data"];

if ([object isKindOfClass:[NSDictionary class]]) {

id anotherObject = [object valueForKey:@"current\_condition"];

if ([anotherObject isKindOfClass:[NSArray class]]) {

id firstArrayObject = [anotherObject objectAtIndex:0];

if ([firstArrayObject isKindOfClass:[NSDictionary class]]) {

NSLog(@"value is %@", **[[[firstArrayObject objectForKey:@"weatherDesc"] objectAtIndex:0]** objectForKey:@"value"]);

}

}

}

}

So if we were to get the value in one fell swoop, it would look something like:

NSLog(@"value is %@", [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"current\_condition"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"]);

Quite a mouthful! Ok, so we now have the value for today’s (current condition) weather, “Clear”. It would have been a lot easier to just open the window huh?

Ok so let’s create a mutable array to put that value into it. If you are using the old project as a starting point, you will have an NSMutableArray called cFRAIPArray already in there as an ivar. So let’s take our “Clear” value and put it inside:

NSString \*today = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"current\_condition"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:today];

Ok now let’s get the other days. If you look at the logged flickrDictionary in the console you will see that at the same hierarchical level as current\_condition, you have a dictionary called “weather”. Its contents reside in an array with many objects, each corresponding to a different day. So translated to code, the next value we would need is called:

NSString \*tomorrow = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"weather"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

NSLog(@"tomorrow will be %@", tomorrow);

Run and find tomorrow in the console. Cool, so now we can add this to the our cFRAIPArray and get the next 3 days. Remember our URI called for 5 days; day 1 is current\_condition. So our complete method would look like:

- (void)connection:(NSURLConnection \*)connection didReceiveData:(NSData \*)data{

NSLog(@"data is %@", data);

NSString \*myString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];

NSLog(@"string is %@", myString);

NSError \*e = nil;

flickrDictionary = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableContainers error:&e];

NSLog(@"dictionary is %@", flickrDictionary);

//Init array

cFRAIPArray = [[NSMutableArray alloc] initWithCapacity:6];

//DAY1

NSString \*today = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"current\_condition"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:today];

//DAY2

NSString \*tomorrow = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"weather"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:tomorrow];

//DAY3

NSString \*afterTomorrow = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"weather"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:afterTomorrow];

//DAY4

NSString \*next = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"weather"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:next];

//DAY5

NSString \*afterThat = [[[[[[flickrDictionary valueForKey:@"data"] objectForKey:@"weather"] objectAtIndex:0] objectForKey:@"weatherDesc"] objectAtIndex:0] objectForKey:@"value"];

[cFRAIPArray addObject:afterThat];

NSLog(@"cFRAIPArray is %@", cFRAIPArray);

}

This produces a nice, clean array with the weather condition for the next 5 days at this location. Great so now let’s just remove some old code we had in connectionDidFinishLoading and leave it clean, like this:

- (void)connectionDidFinishLoading:(NSURLConnection \*)connection {

// do something with the data

// receivedData is declared as a method instance elsewhere

NSLog(@"Succeeded!");

[self.tableView reloadData];

}

We can do this because we have all of our array loading code in the connection:didReceiveData. As this loads and finally reaches the connectionDidFinishLoading, the tableview’s reloadData method is called to refresh the view with the finalized data.

Ok, now that we have more experience with web services and parsing server results, let’s take a look at creating our own service.