Not for marks.

This document outlines two examples which allow you to:

* integrate JavaScript with Swift.
* implement asynchronous calls for remote JSON from Swift.

Example Using Swift to Interact with JavaScript

This example shows how you can run JavaScript from your iOS application. Also, it demonstrates how your JavaScript can call the native iOS code. Add the following file to your project.

**main.js**

|  |
| --- |
| function callNativeApp () {  try {  webkit.messageHandlers.callbackHandler.postMessage(  "Sent from JavaScript");  } catch(err) {  console.log('error');  }  }  function redHeader() {  document.querySelector('h1').style.color = "red";  callNativeApp()  } |

Here is the HTML which loads the JavaScript.

**Index.html**

|  |
| --- |
| <html>  <head>  <title>Test</title>  <meta charset="UTF-8">  </head>  <body>  <br/><br/>  <h1>Formatted with JavaScript</h1>  <script type="text/javascript" src="main.js"></script>  </body>  </html> |

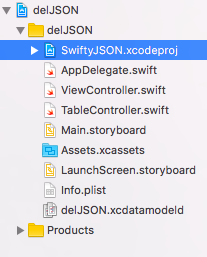
Here is the ViewController code which generates a *WkWebView* in the *loadView()* method. The *loadView()* method is a built-in iOS function which gets called before *viewDidLoad().* The code inside *loadView()* tells iOS to run the *redHeader()* function in JavaScript after the JavaScript file loads. The code in *loadView()* also sets up a callback so you can call the swift code from your JavaScript code. The *userContentController()* function in swift serves as the callback handler.

|  |
| --- |
| import UIKit  import WebKit  class ViewController: UIViewController, WKScriptMessageHandler {  // Declare the WKWebView object.  // Note this does not use the IBOutlet.  var myWebView: WKWebView!    // Load WKWebView dynamically. Called before viewDidLoad()  override func loadView() {  super.loadView()    // Set up bridge between JavaScript and Swift.  let contentController = WKUserContentController();    // Call JavaScript to set header color during load.  let userScript = WKUserScript(  source: "redHeader()",  injectionTime: WKUserScriptInjectionTime  .AtDocumentEnd,  forMainFrameOnly: true  )  contentController.addUserScript(userScript)  // Enable call back from JavaScript to swift.  contentController.addScriptMessageHandler(  self,  name: "callbackHandler")    let config = WKWebViewConfiguration()  config.userContentController = contentController  self.myWebView = WKWebView(frame: self.view.frame,  configuration: config)  self.view = self.myWebView!  }    override func viewDidLoad() {  super.viewDidLoad()    // Load HTML in the view generated earlier in  // loadView().  let url  = NSURL(fileURLWithPath: NSBundle.mainBundle()  .pathForResource("index", ofType: "html")!)  let req = NSURLRequest(URL: url)  self.myWebView!.loadRequest(req)  }    // This is the callback handler for messages from JavaScript.  func userContentController(  userContentController: WKUserContentController,  didReceiveScriptMessage message: WKScriptMessage) {  if(message.name == "callbackHandler") {  print("JavaScript is sending a message: ")  print(message.body)  }  }  override func didReceiveMemoryWarning() {  super.didReceiveMemoryWarning()  }  } |

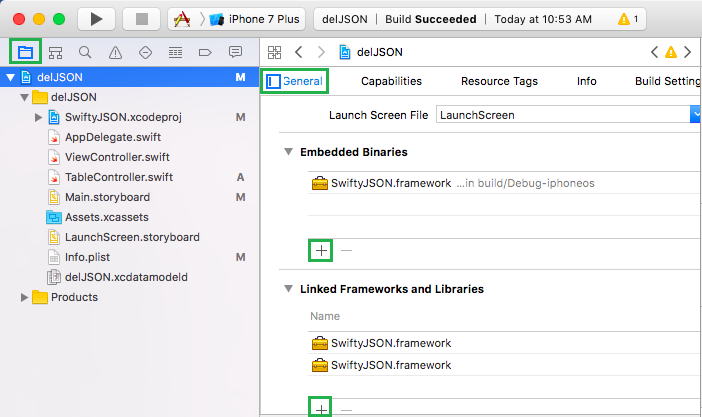
Example 2: Calling JSON from Swift

You would think that working with JSON would be easier – however this is much better than with Swift 2 so things are improving. This example shows how to retrieve JSON asynchronously.

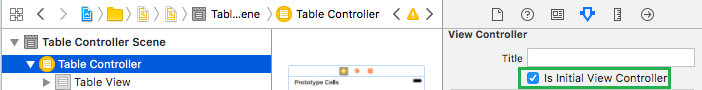
To get started, create a new single view application. Next, you will want to add SwiftyJSON to help with parsing JSON. (Parsing JSON can otherwise be incredibly difficult in Swift). Download the SwiftJSON project from <https://github.com/SwiftyJSON/SwiftyJSON>. Using finder, copy the downloaded files into a sub directory inside your iOS application. Then, drag the SwiftyJSON.xcodeproj file into your project.



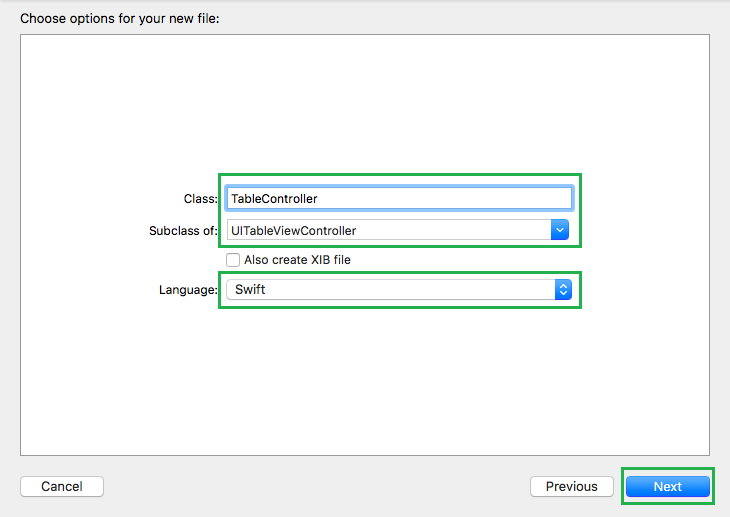
After, from the General tab in your project you will needed to reference the iOS version of SwiftyJSON inside your Embedded Binaries and Linked Frameworks and Libraries area.



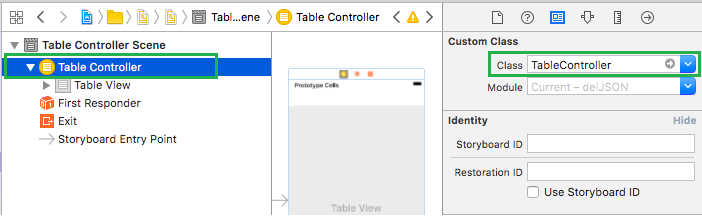
This example will use the TableViewController so remove the original ViewController and add a Table View controller. When the Table View Controller has been added select “Is Initial View Controller”.



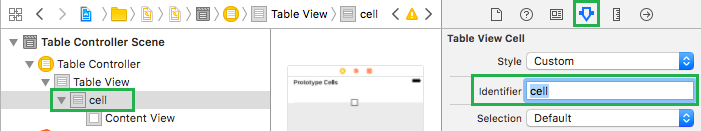
Next, add a UITableViewController class to support the new screen.



Remember to associate the class with the screen.



To allow a reference to the row cell of the table view, name it in the attributes inspector.



Next, make the following changes to the swift file:

|  |
| --- |
| import UIKit  // Use the SwiftyJSON library – parsing JSON is really difficult without it.  // <https://github.com/SwiftyJSON/SwiftyJSON>  import SwiftyJSON  class TableController: UITableViewController {  var tableRowsArray = [String]()  override func viewDidLoad() {  super.viewDidLoad()  tableView.refreshControl = refreshControl  }  override func didReceiveMemoryWarning() {  super.didReceiveMemoryWarning()  // Dispose of any resources that can be recreated.  }  override func numberOfSections(in tableView: UITableView) -> Int {  return 1  }  override func tableView(\_ tableView: UITableView,  numberOfRowsInSection section: Int) -> Int {  return tableRowsArray.count  }    override func tableView(\_ tableView: UITableView,  cellForRowAt indexPath: IndexPath) -> UITableViewCell {  let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)  cell.textLabel?.text = tableRowsArray[indexPath.row]  return cell  }  } |

Then, add these two new functions to help with retrieving and parsing the data:

|  |
| --- |
| // Use 'Swifty' to parse JSON. Otherwise it is crazy difficult.  func extractWithSwifty(dataStr:String) {  if let dataFromString = dataStr.data(  using: String.Encoding(rawValue: String.Encoding.utf8.rawValue),  allowLossyConversion: false) {  let json = JSON(data: dataFromString)    // Initialize table data.  tableRowsArray = [String](repeating: "", count: json.count)  for var i in (0..<json.count) {  print(json[i]["num"])  tableRowsArray[i] = String(describing: json[i]["num"])  }  }  else {  print("An error occurred while parsing the JSON.")  }    // Reload TableView once data is back.  DispatchQueue.main.async {  self.tableView.reloadData()  }  }    func get\_data\_from\_url(url:String) {  let myUrl = NSURL(string: "http://ssdprogram.ca/testJSON.php");  let request = NSMutableURLRequest(url:myUrl! as URL);  request.httpMethod = "GET"    // Excute HTTP Request  let task = URLSession.shared.dataTask(with: request as URLRequest) {  data, response, error in    // Check for error  if error != nil {  print("error=\(error)")  return  }    // Store data as string.  let dataStr  = NSString(data: data!,  encoding: String.Encoding.utf8.rawValue)    // Show raw data.  print(dataStr!)    // Parse data.  self.extractWithSwifty(dataStr: dataStr as! String)  }  task.resume()  } |

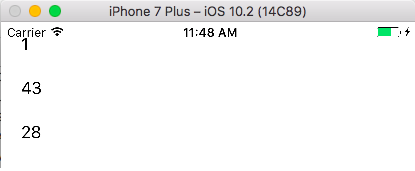
Finally, add the following call instruction to the viewDidLoad() function.

|  |
| --- |
| get\_data\_from\_url(url: "http://ssdprogram.ca/testJSON.php") |

To allow remote calls for JSON, the application must be given permission. To do this, open the info.plist file and add these tags the end of the file just before the last dict closing tag.

|  |
| --- |
| <key>NSAppTransportSecurity</key>  <dict>  <!--Include to allow all connections (DANGER)-->  <key>NSAllowsArbitraryLoads</key>  <true/>  </dict> |

When running the application the JSON content is displayed.



The debug window shows what the actual structure looks like as well.

