AllJoyn Programming Guide for the Objective-C Language

Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

The information contained in this document is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License; provided, that (i) any source code incorporated in this document is licensed under the Apache License version 2.0 **AND (ii) THIS DOCUMENT AND ALL INFORMATION CONTAINED HEREIN ARE PROVIDED ON AN "AS-IS" BASIS WITHOUT WARRANTY OF ANY KIND.**
Creative Commons Attribution-ShareAlike 3.0 Unported License

This document will introduce you to AllJoyn by walking through the steps to create a simple application. By presenting new concepts one at a time in a stepwise fashion, you will gain a better understanding of how AllJoyn operates and how to use the Objective-C language API to add functionality in your own apps.

AllJoyn From 10,000 ft

AllJoyn, simply put, provides applications with a mechanism to discover and communicate with each other across disparate mobile and desktop platforms. AllJoyn applications interact with each other via clients and services. A service is essentially a server, with one or more interfaces defining what operations the service can perform for any given client. On an AllJoyn network, or bus, each service has a unique name distinguishing it from other services on the bus. You can think of the AllJoyn bus as a logical network superimposed upon the underlying "real" network. A service name looks similar to an Internet host name, as seen in this example:

org.alljoyn.Bus.samples.chat.bob

AllJoyn clients and services communicate through messages sent and received over the bus. AllJoyn supports multiple types of physical networks to transport an AllJoyn message. The most commonly used transport to shuttle AllJoyn messages from clients to services over the bus is a wireless local area network and TCP/IP sockets.

When a client wishes to communicate with a service, a series of steps is usually followed. First, the client connects to the AllJoyn bus and attempts to discover any services also connected to the AllJoyn bus that have a given unique name. Once a service matching the name is discovered, the client can connect to the service, establish a session and begin using the interfaces that the service exposes. Each AllJoyn-enabled application can, and typically does, implement both client(s) and service(s) to perform useful tasks on behalf of the user. After connecting to a service and establishing a session, the client may then consume any member of the service's interfaces. An interface contains members, of which there are three types:

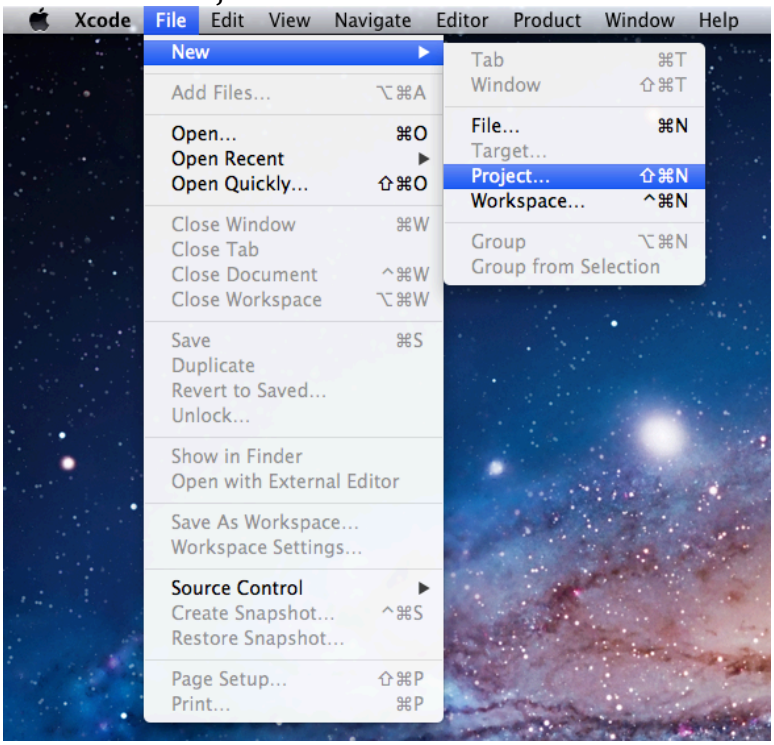
- Properties. A value stored in an object that may be read-only, read-write or write-only. Behind the scenes, properties are simply AllJoyn method calls, with a method defined for a property getter and a method defined for a property setter.
- Methods. A function call that typically takes a set of inputs then performs some processing using the inputs and returns one or more outputs reflecting the results of the processing operation.
- Signals. Events that are raised by a service and broadcast to clients in a session, or broadcast globally to all clients on the bus.

As a client of a service, you will use a proxy bus object to gain access to those interfaces and their members exposed by a service. The proxy object allows your code to interact with the remote service as though it were an object local to your app.

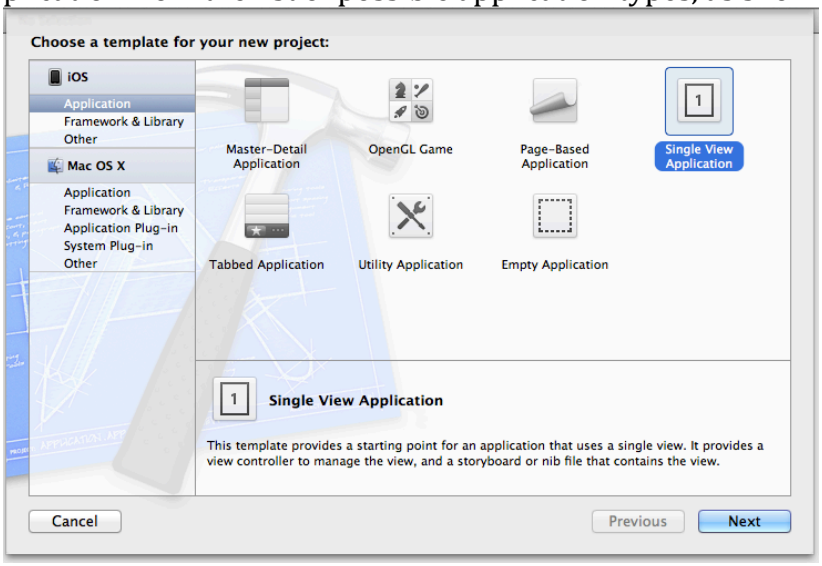
If you are implementing a service, you must create an implementation of the object that your clients will access through their proxy objects. In the object implementation, you add the code to support each interface and each member of each interface.

Step One. Create a new Xcode iOS project

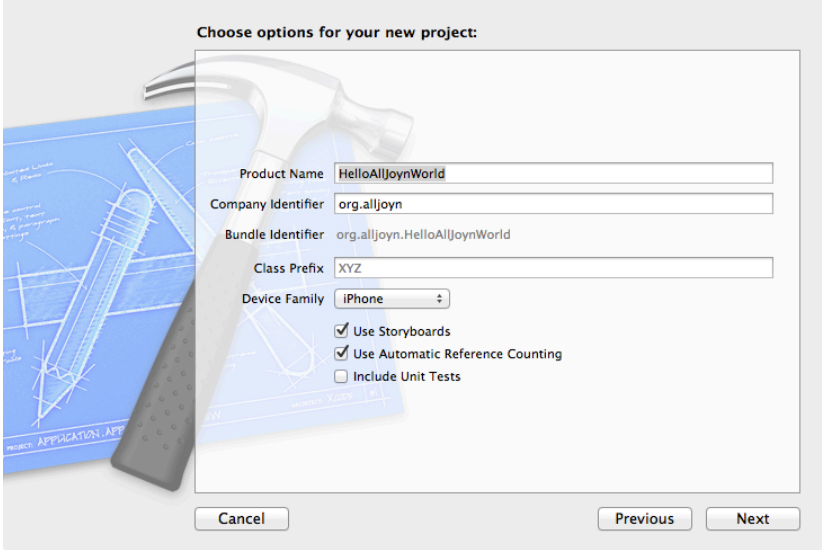
1. Open Xcode by selecting it from the Applications folder in the Dock at the bottom of the screen.
2. Select File -> New -> Project…



3. In the left hand menu, under iOS, select Application and then select Single View Application from the list of possible application types, as shown below:



4. Click the Next button and fill out the name of the new project, select iPhone only, check the Use Automatic Reference Counting and Use Storyboards options.



5. Click the Next button, select the parent folder for your new project and then select the Create button.
6. You now should see your HelloAllJoynWorld project loaded in Xcode. You are ready to begin your journey into AllJoyn development!

Step Two. Define the AllJoyn Object Model.

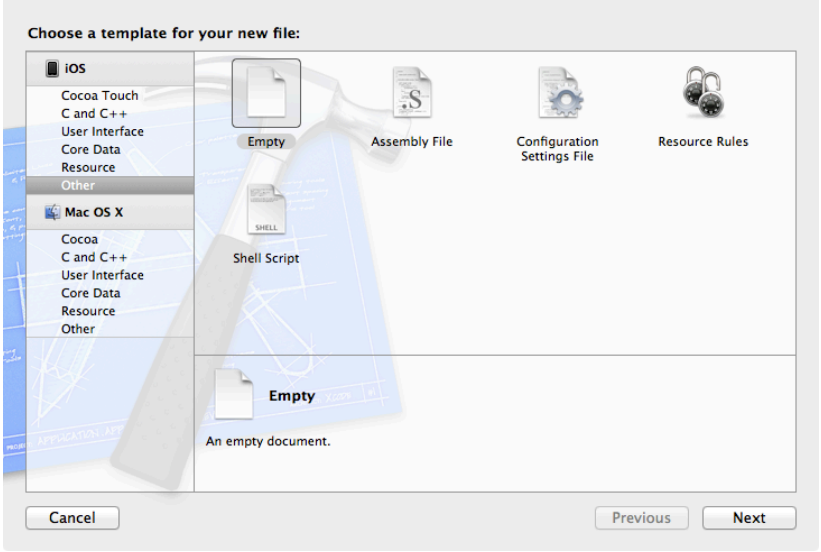
As mentioned in the AllJoyn From 10,000ft section, services are composed of interfaces, each of which has one or more members. To offer a service that does some useful work for a client, you must define and implement the service's

interface. For this example, we will create a simple interface with a single method that will concatenate two strings together and return the result. The AllJoyn for iOS SDK comes with a utility that will generate most of the boilerplate code for your service objects. All you will need to do is provide the Objective-C code to fill in the implementation of each member exposed by the interfaces of your service's bus object. Before we can run the code generator, though, we need to compose a representation of the service, or bus object, in XML, which is the format the code generator expects. As a side note, the format of the XML conforms to the standard used by D-Bus. AllJoyn implements the D-Bus specification, and hence is compatible with other D-Bus-enabled applications. For more information on D-Bus, see the following web site:

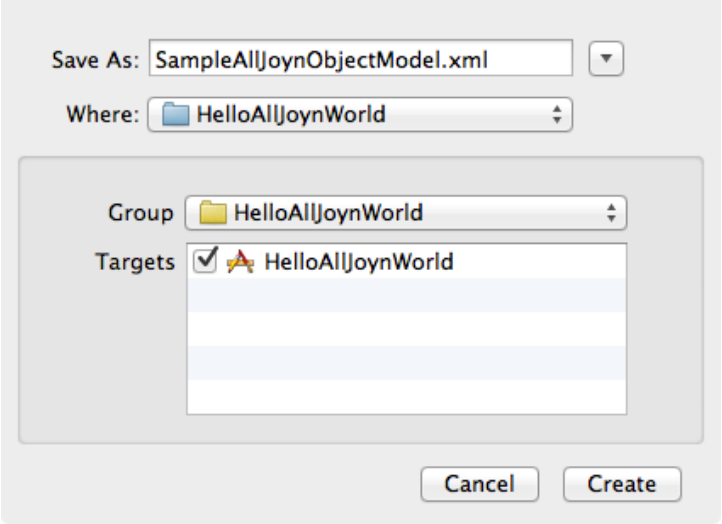
http://www.freedesktop.org/wiki/Software/dbus

In your HelloAllJoynWorld Xcode project, we are going to add a new file called SimpleAllJoynObjectModel.xml that will contain a description of the interface your service will support.

1. Right click on the HelloAllJoynWorld group folder in the project Navigator on the right side of the Xcode view.
2. Select New File... from the menu to bring up the template chooser, as shown below.



3. Under iOS, select the Other template group and then select Empty in the list of available templates. Click the Next button to proceed.
4. Type in the name of your new file, SampleAllJoynObjectModel.xml, as shown below:



5. Click the Create button to create your new XML file and add it to the Xcode project.
6. Select the SampleAllJoynObjectModel.xml file in the project Navigator, and add the following XML to it:

```
<xml>
    <node name="org/alljoyn/Bus/sample">
        <annotation name="org.alljoyn.lang.objc" value="SampleObject"/>
        <interface name="org.alljoyn.bus.sample">
            <annotation name="org.alljoyn.lang.objc" value="SampleObjectDelegate"/>
            <method name="Concatentate">
                <arg name="str1" type="s" direction="in">
                    <annotation name="org.alljoyn.lang.objc" value="concatenateString:"/>
                </arg>
                <arg name="str2" type="s" direction="in">
                    <annotation name="org.alljoyn.lang.objc" value="withString:"/>
                </arg>
                <arg name="outStr" type="s" direction="out"/>
            </method>
        </interface>
    </node>
</xml>
```

You now have a description of the bus object that your service will expose to its clients. Let's go line by line through the XML to build an understanding of the format and how it describes our bus object.

```
<xml>
```

The first line, shown above, is standard fare for all XML documents.

```
<node name="org/alljoyn/Bus/sample">
```

The second line declares an element named "node" which corresponds with the bus object our service will expose. A name attribute on the node element defines the object path. In the D-Bus XML format, Node elements contain interface elements that can contain method, property and signal elements. Node elements may also contain other child node elements, but let's keep it simple for now.

```
<annotation name="org.alljoyn.lang.objc" value="SampleObject"/>
```

The next line declares an annotation element. Annotations are used to store metadata about a node, interface, method, signal or property element. Annotations are name-value pairs that can contain virtually any data. In the context of AllJoyn for iOS, annotations named “org.alljoyn.lang.objc” are used to give the code generator hints as far as naming is concerned. In this instance, the annotation element gives the code generator a hint on what to name the object at path /org/alljoyn/Bus/sample. The annotation tells the code generator to give the object a friendly name of “SampleObject” in any Objective-C code emitted.

```
<interface name="org.alljoyn.bus.sample">
       <annotation name="org.alljoyn.lang.objc" value="SampleObjectDelegate"/>
```

On the above lines, an interface element named org.alljoyn.bus.sample is created. Interface elements contain method, signal and property elements, as well as annotation elements. In this case, the annotation tells the code generator to create an Objective-C protocol for the interface and name it SampleObjectDelegate. All interfaces on bus objects are implemented in Objective-C as protocols.

```
<method name="Concatentate">
       <arg name="str1" type="s" direction="in">
         <annotation name="org.alljoyn.lang.objc" value="concatenateString:"/>
       </arg>
       <arg name="str2" type="s" direction="in">
         <annotation name="org.alljoyn.lang.objc" value="withString:"/>
       </arg>
       <arg name="outStr" type="s" direction="out"/>
</method>
```

The final lines in the XML file shown above describe a method called Concatenate that takes two strings as arguments and returns a string. A method element can contain 0 to n arg child elements. Each arg element has three attributes:

1. name. The name of the argument.
2. type. The type of the element. Per the D-Bus specification, data types are expressed as a string of one or more letters, called a signature. The letter “s” signifies a string data type.
3. direction. Allowed values are “in” or “out” corresponding to input and output parameters respectively.

The D-Bus interface description format favors a C language style for declaring methods, with a name for a method followed by several input and output parameters, each with a parameter name. Objective-C, however, does not use this syntax when declaring a method, or more accurately in Objective-C parlance, a message. The message takes the form of a selector, with the complete name of the message interspersed with its parameters. Note the difference below:

void ConcatenateString(in String str1, in String str2, out String outStr);

NSString *concatenateString:(NSString *)str1 withString:(NSString *)str2;

Annotations help the code generator create message declarations that appeal to our Objective-C sense of good coding style. The arg elements contain an annotation element that specifies part of the selector associated with that argument. By processing all the arg elements and their child annotations, the code generator can emit the complete selector for the message.

Step Three. Build and Configure the Code Generator

Now that you have familiarized yourself with the D-Bus XML format, it is time to generate the code that will allow us to create an Objective-C object that you can use in your application. The AllJoynCodeGenerator project is located at the following path:

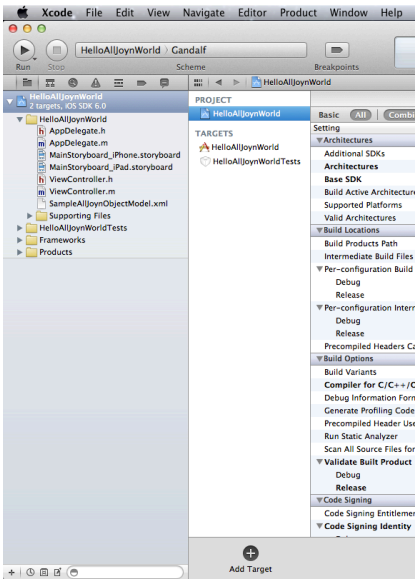
<AllJoyn SDK Root>/alljoyn_objc/AllJoynCodeGenerator

Navigate the to above directory in Finder and double click the AllJoynCodeGenerator.xcodeproj file to launch Xcode and load up the project. In Xcode, select Product -> Build to build the AllJoyn code generator executable. Your new executable is now ready for use, and is located in the following directory:

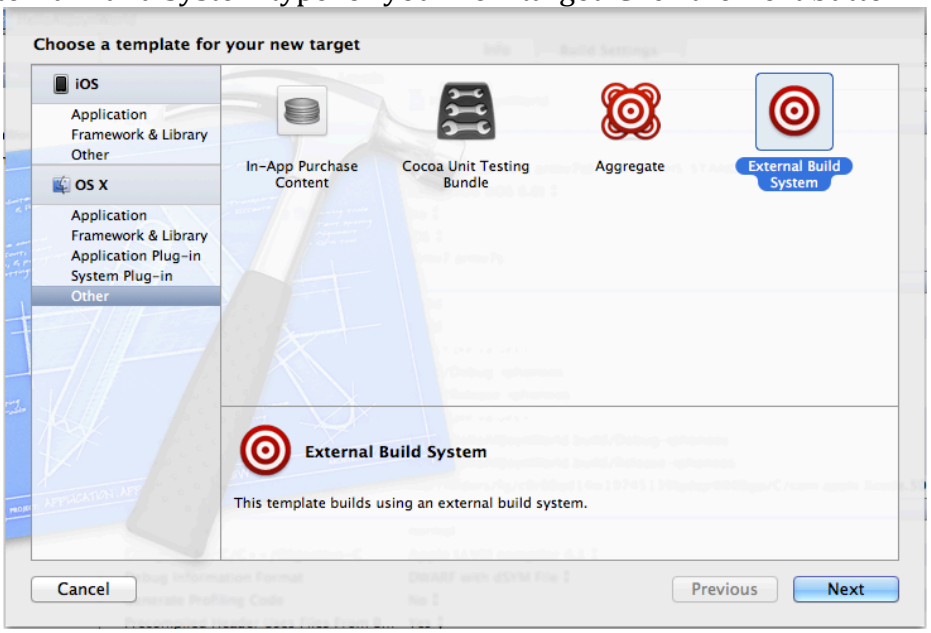
<AllJoyn SDK Root>/alljoyn_objc/bin

Now return to your HelloAllJoynWorld Xcode project, so that you may configure a target and a scheme in Xcode that will launch the code generator and pass the SampleAllJoynObjectModel.xml file to it.

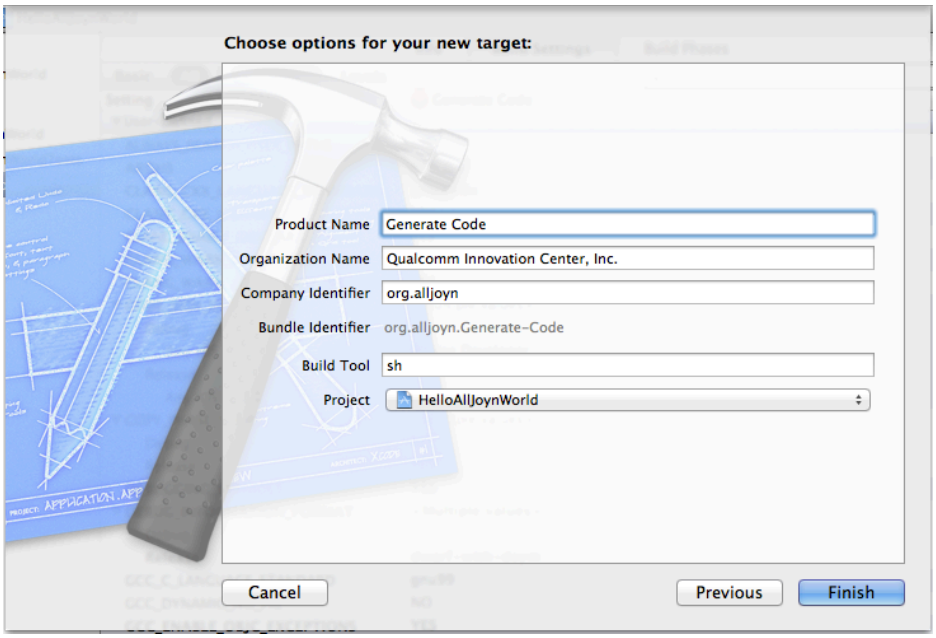
1. Select the HelloAllJoynWorld root node in the Project Navigator view, the left hand pane in Xcode, then click the Add Target button located at the bottom of the middle pane in Xcode.



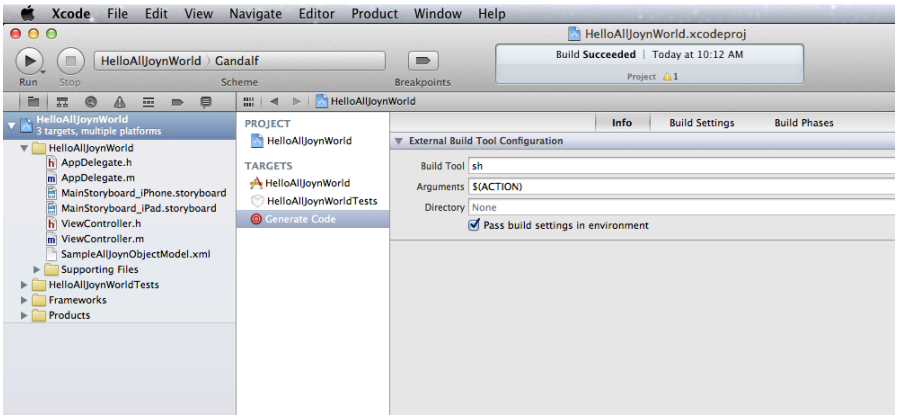
2. Select “Other” in the OS X list on the left side of the dialog and choose the External Build System type for your new target. Click the Next button.



3. Type in Generate Code into the Product Name text field and click the Finish button to create your new target and its accompanying scheme.



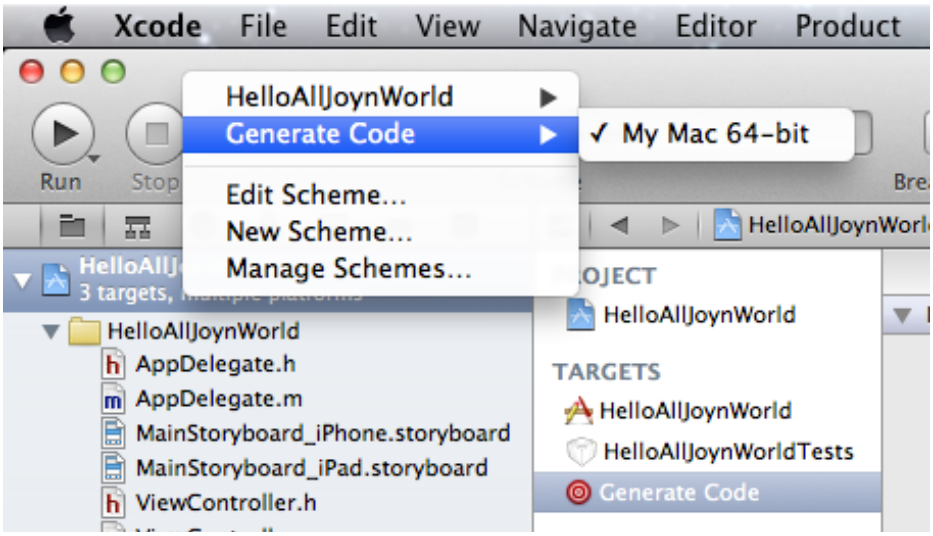
4. Now select the Generate Code target in the list of targets as shown below, and click the Info tab at the top of the middle pane in Xcode.



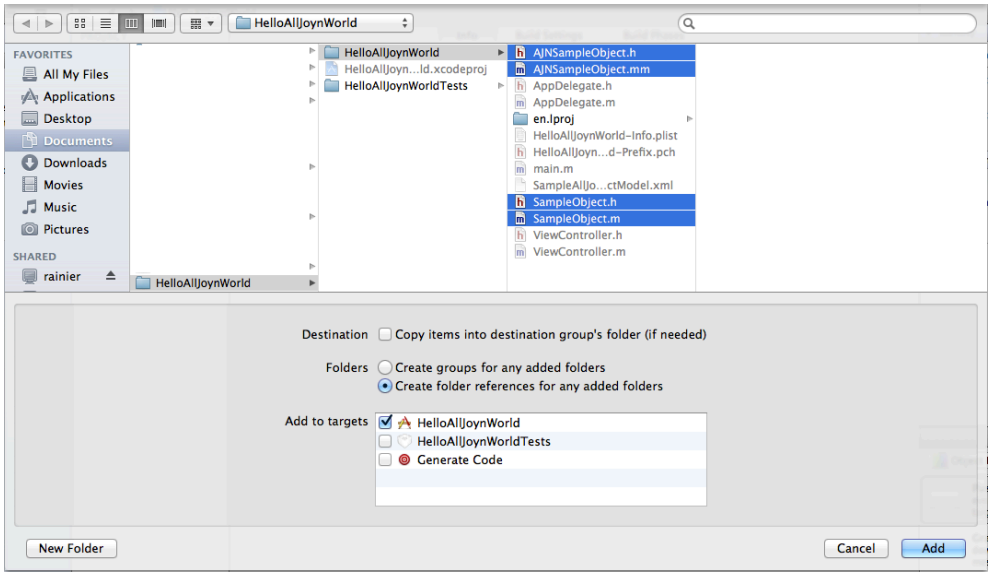
5. In the Build Tool text field for the "Generate Code" target, enter the full path to your AllJoynCodeGenerator binary, which should be located in your <ALLJOYN_SDK_ROOT>/alljoyn_objc/bin folder.
6. In the Arguments text field for the "Generate Code" target, enter the full path to your SampleAllJoynObjectModel.xml file followed by a space and then SampleObject, as follows:

```
$(SOURCE_ROOT)/HelloAllJoynWorld/SampleAllJoynObjectModel.xml SampleObject
```

7. Now select the Generate Code scheme and set it to your active scheme as shown below:



8. Click the Product -> Build and the code generator should successfully generate your SampleObject code. Add the new files to your project by right clicking on the HelloAllJoynWorld group and selecting Add Files To HelloAllJoynWorld..



9. Select the following files and click the Add button:
    a. AJNSampleObject.h
    b. AJNSampleObject.mm
    c. SampleObject.h
    d. SampleObject.m
10. Congratulations! You now have a skeleton Objective-C implementation of your sample AllJoyn bus object.

Take a look at the generated code files. As you can see, the code generator takes care of the implementation of most of the boilerplate you would normally need to create by hand to work with the AllJoyn C++ API. Also note that the code in AJNSampleObject.mm includes C++ code that interoperates with the Objective-C code contained in SampleObject.h/.m. By declaring and implementing the C++ classes only within the AJNSampleObject.mm file and not referring to any C++ classes in the header file, the code generator insulates your app's code from the AllJoyn C++ API. In this manner, the rest of your app can remain as pure Objective-C, rather than forcing you to move your entire project's code to Objective-C++.

Your implementation of the logic for SampleObject::concatenateString:withString: should reside in the SampleObject.m. You should not normally need to change the code in the AJN*.h/.mm files in order to implement your application.

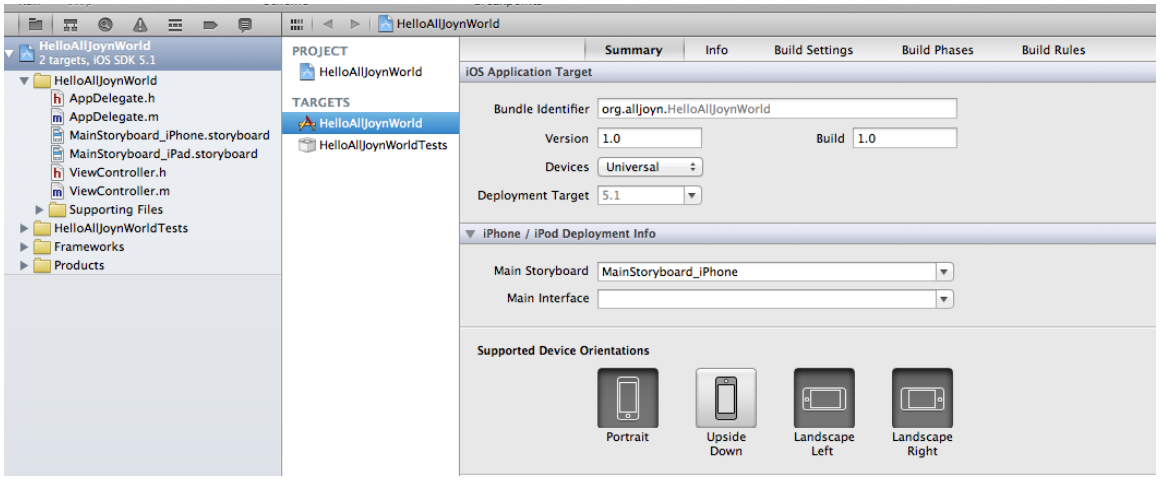
Step Four. Configure the Build Settings

Now you need to configure the Xcode project to successfully compile and link your app.

1. Make sure you know the location of the AllJoyn SDK folder. The AllJoyn SDK folder contains your alljoyn_core, common and alljoyn_objc folders.

2. Follow the directions in the README file in the AllJoyn SDK folder to compile openssl for iOS using Xcode.

3. Open Xcode, open your project and select the root of the tree in Project Navigator as shown below. Then select the App's target under Targets.



4. Select the Build Settings tab for the App target. Click on the "All" option at the top of the list as shown below.

5. At the top of the Build Settings list, click the Architectures setting and then select Other…

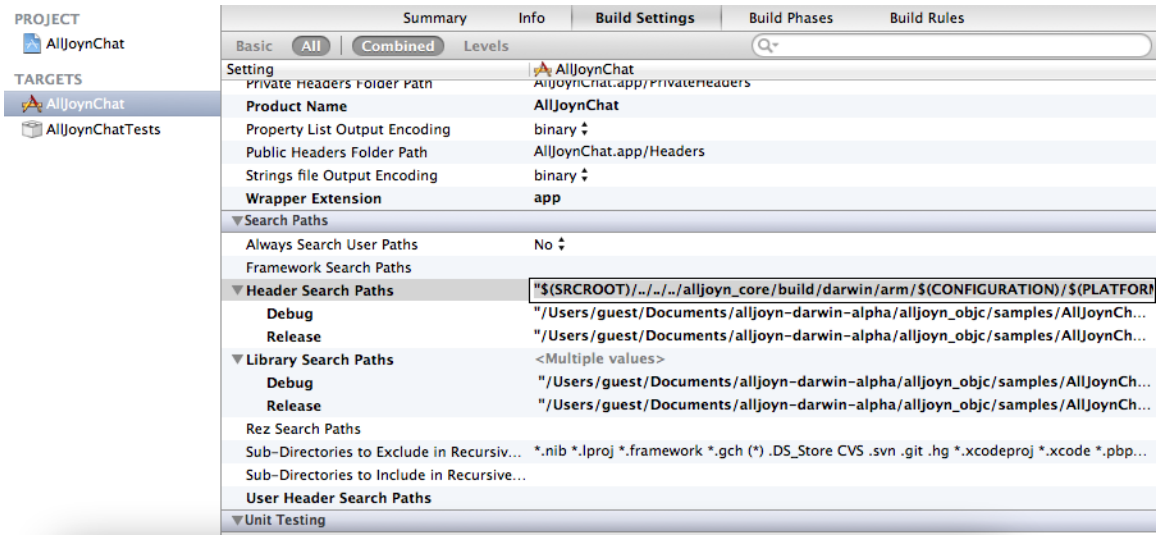
6. Click the + sign in the window that appears and add armv7, then close the window.

7. Set Build Active Architecture Only to Yes.

8. Scroll down to the Linking section, and set Other Linker Flags to the following:

   -lalljoyn -lajdaemon -lBundledDaemon.o -lssl -lcrypto

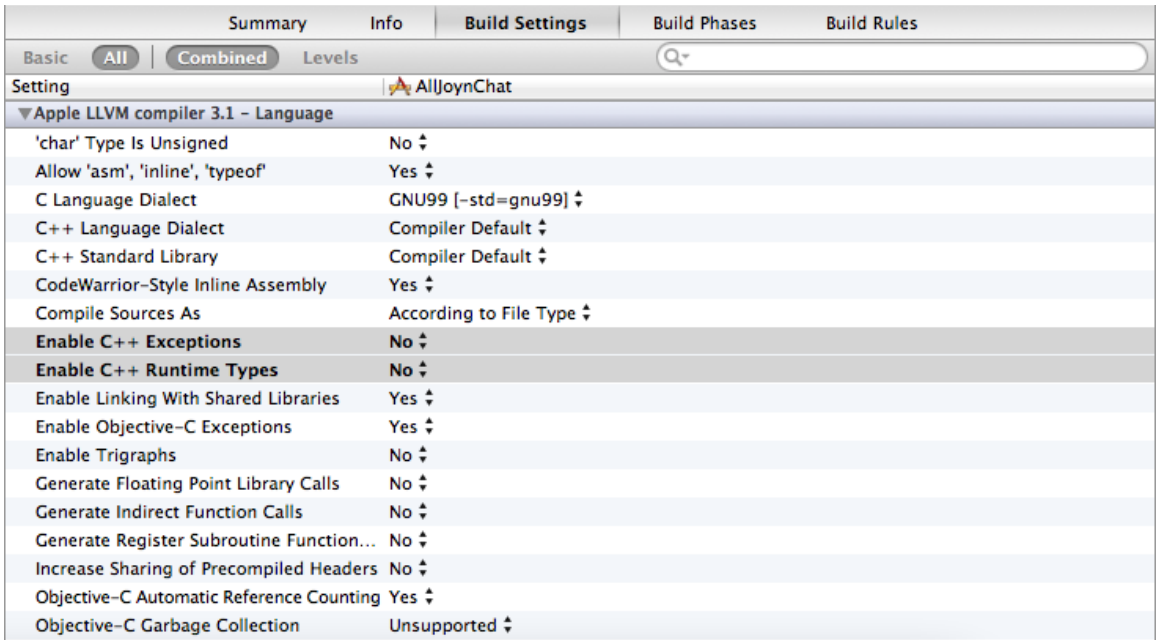
9. Scroll down the list of settings until you see the Search Paths group.



10. Double click the **Header Search Paths** text field and enter the following:
    ```
    "$(SRCROOT)/../alljoyn-sdk/alljoyn_core/build/darwin/arm/$(PLATFORM_NAME)/$(CONFIGURATION)/dist/inc" "$(SRCROOT)/../alljoyn-sdk/alljoyn_core/build/darwin/arm/$(PLATFORM_NAME)/$(CONFIGURATION)/dist/inc/alljoyn"
    ```

11. Double click the **Library Search** paths text field and enter the following:
    ```
    $(inherited) "$(SRCROOT)/../alljoyn-sdk/alljoyn_core/build/darwin/arm/$(PLATFORM_NAME)/$(CONFIGURATION)/dist/lib" "$(SRCROOT)/../alljoyn-sdk/common/crypto/openssl/openssl-1.01/build/$(CONFIGURATION)-$(PLATFORM_NAME)"
    ```

12. Scroll down in the **Build Settings** table until you see the **Apple LLVM compiler 3.1 – Language** group.



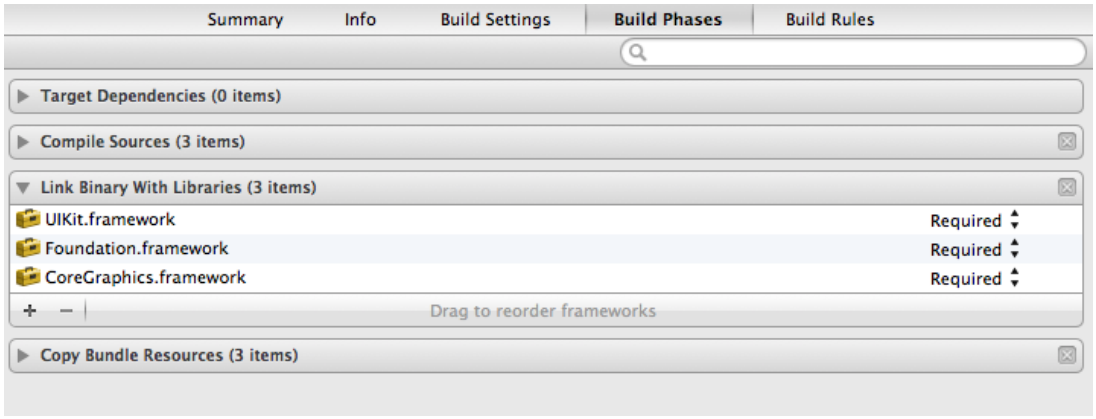
13. Set **Enable C++ Exceptions** to "No."

14. Set **Enable C++ Runtime Types** to "No."

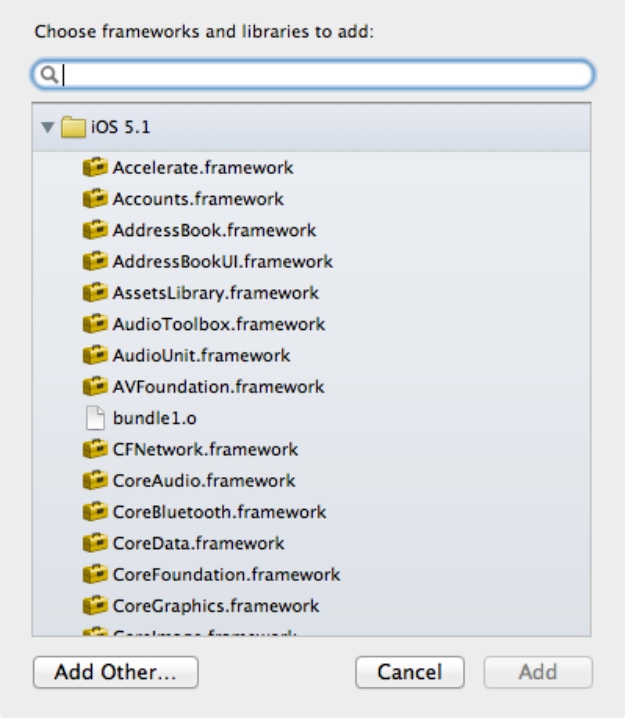
15. Set the **Other C Flags** text field for **Debug** to the following:
    `-DQCC_OS_GROUP_POSIX -DQCC_OS_DARWIN`

16. Set the **Other C Flags** text field for **Release** to the following:
    `-DNS_BLOCK_ASSERTIONS=1 -DQCC_OS_GROUP_POSIX -DQCC_OS_DARWIN`

17. Select the **Build Phases** tab as shown below:

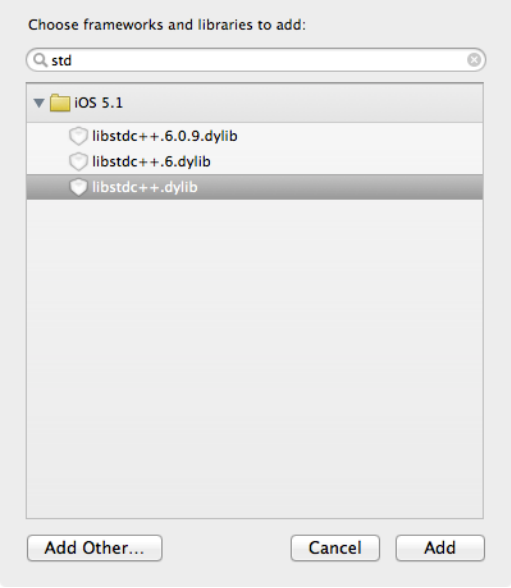


18. Expand the **Link Binary With Libraries** group and click the **+** sign at the lower left-hand corner. A dialog will appear as show below:



19. Select the SystemConfiguration.framework file.

20. Now click the **+** button again and add one last library to link against, libstdc++.dylib if it is not already included.



21. Type in "**std**" into the search text field to view only the standard template library binaries. Select the following file from the list:
`libstdc++.dylib`

22. Create a group to hold the AllJoyn framework by right clicking the HelloAllJoynWorld group in the Project Navigator tree and selecting New Group from the menu.

23. Type in “AllJoynFramework” to give your new group a pertinent name.

24. Select the newly created “AllJoynFramework” group, and choose Add Files…

25. Navigate to the following folder:
    a. <ALLJOYN_SDK_ROOT>/alljoyn_objc/AllJoynFramework/AllJoynFramework

26. Now select all the .h/.m* files in the directory and be sure to uncheck “Copy items into destination group’s folder” and make sure your HelloAllJoynWorld target is checked in the Add to targets list.

27. Click the Add button to add the AllJoyn Objective-c framework to your AllJoynFramework group in the project.

28. Select **Product -> Build** from the Xcode main menu and your project should build successfully! Congratulations!

Note that there is a template project located in the following folder that has the above configuration preloaded for you. Check it out at:

<ALLJOYN_SDK_ROOT>/alljoyn_objc/samples/iOS/AllJoyn iOS Project Template

This is a good starting point for any apps you may wish to build. Open the Xcode project for the above template and examine the source files within. Spend some time to examine the README file included with this project, as it contains information on the files included in the project.