LDAP and the iPhone

Part 1, Harnessing UNIX libraries for iOS development

Skill Level: Intermediate

David M. Syzdek (syzdek@bindlebinaries.com)
Software developer
Bindle Binaries

22 Feb 2011

A multitude of libraries have been written for UNIX® systems. Many of those libraries have been released using open source licenses that allow a library's source code to be reused in new projects. By porting an existing library to a new platform, a developer may be able to save the time it would take to duplicate the development work to achieve the same functionality on the new platform. This is the first of a two-part article series on porting the OpenLDAP client libraries to the iOS. Part 1 walks the reader through the steps of importing the OpenLDAP source code into Xcode and building two static libraries for the iOS. Using the Xcode project created in Part 1, Part 2 will guide the reader through the creation a simple iOS application that executes basic queries to an LDAP server using the OpenLDAP libraries.

Introduction

The introduction of the iPhone onto the cell phone market brought along a deluge of mobile applications in the iTunes store. With a great number of applications duplicating core functionality, it is easy for a new application to become buried among its competitors. For an application to stick out to a consumer, it is becoming increasingly necessary to add features not supported by competing applications; however, writing new functionality from scratch can be costly in terms of development time and labor. Understanding how to port libraries originally written for UNIX® systems may allow a developer to cut costs and release an application to market faster.

iOS, the operating system used by the iPhone, is built upon a UNIX core and

applications for iOS are compiled using GCC, the same compiler used by the majority of UNIX systems. This means that most libraries written for UNIX systems can be ported to the iPhone with some time and effort.

This article walks you through the steps of porting a library to the iPhone that is normally built in an environment with Autoconf and Make. Although not an extensive explanation of porting from packages built with GNU tools, this article should provide the needed bread crumbs for developers who have some knowledge of GNU tools to port a package into Xcode and to iOS platforms.

Getting started

The examples used in this article were tested using iOS SDK 4.1 on Mac OS X 10.6.4 and OpenLDAP 2.4.22. Although this article was written using the iOS SDK 4.1, it should work for future versions of the iOS SDK with little if any changes. The source files and project files for the examples are available in a zip file in the Download section.

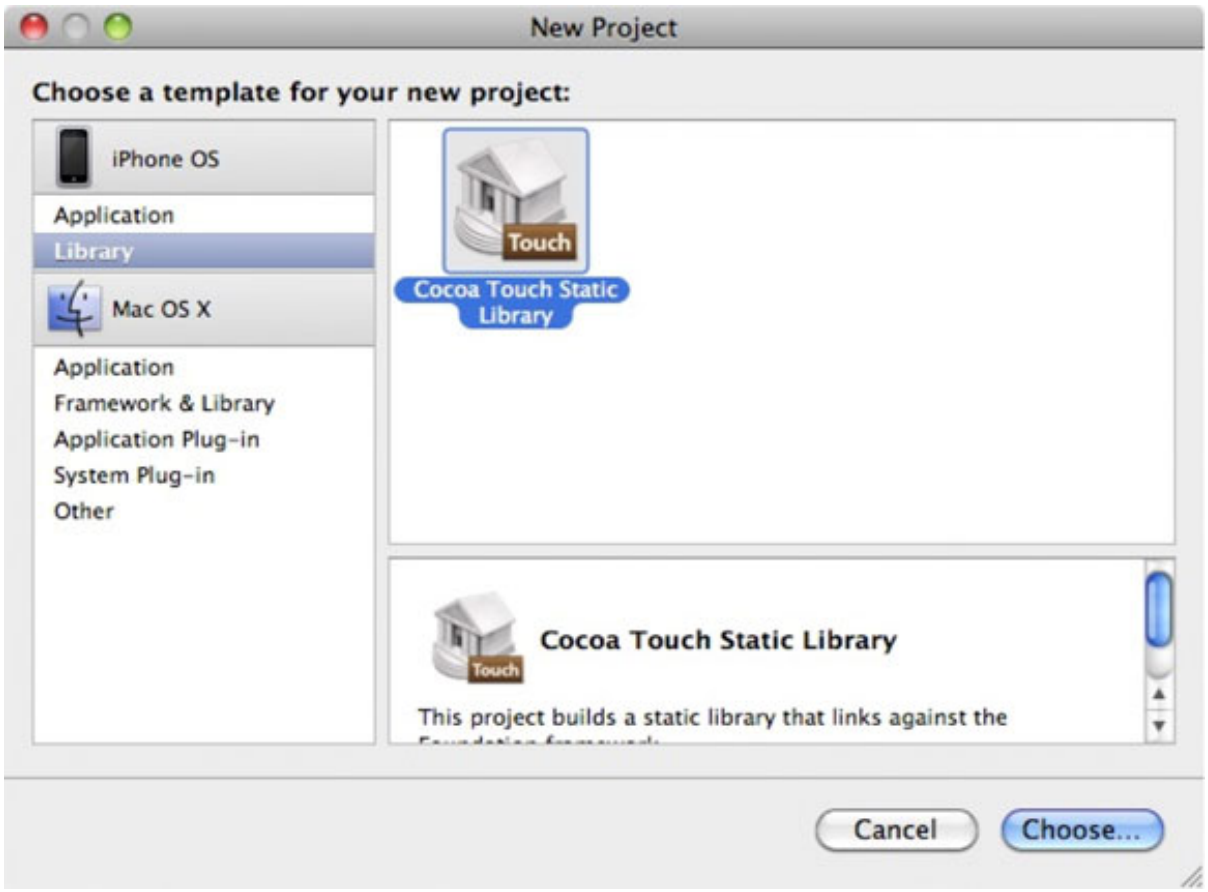
The iPhone SDK is used to compile mobile applications for the iPhone OS platform. The SDK provides documentation, an IDE, and simulator for testing mobile applications (see Resources for download information).

OpenLDAP is a free and open source implementation of the Lightweight Directory Access Protocol. The implementation includes libraries that provide access to X.500 directory services using LDAP over TCP (see Resources for download information).

Preparing OpenLDAP source code

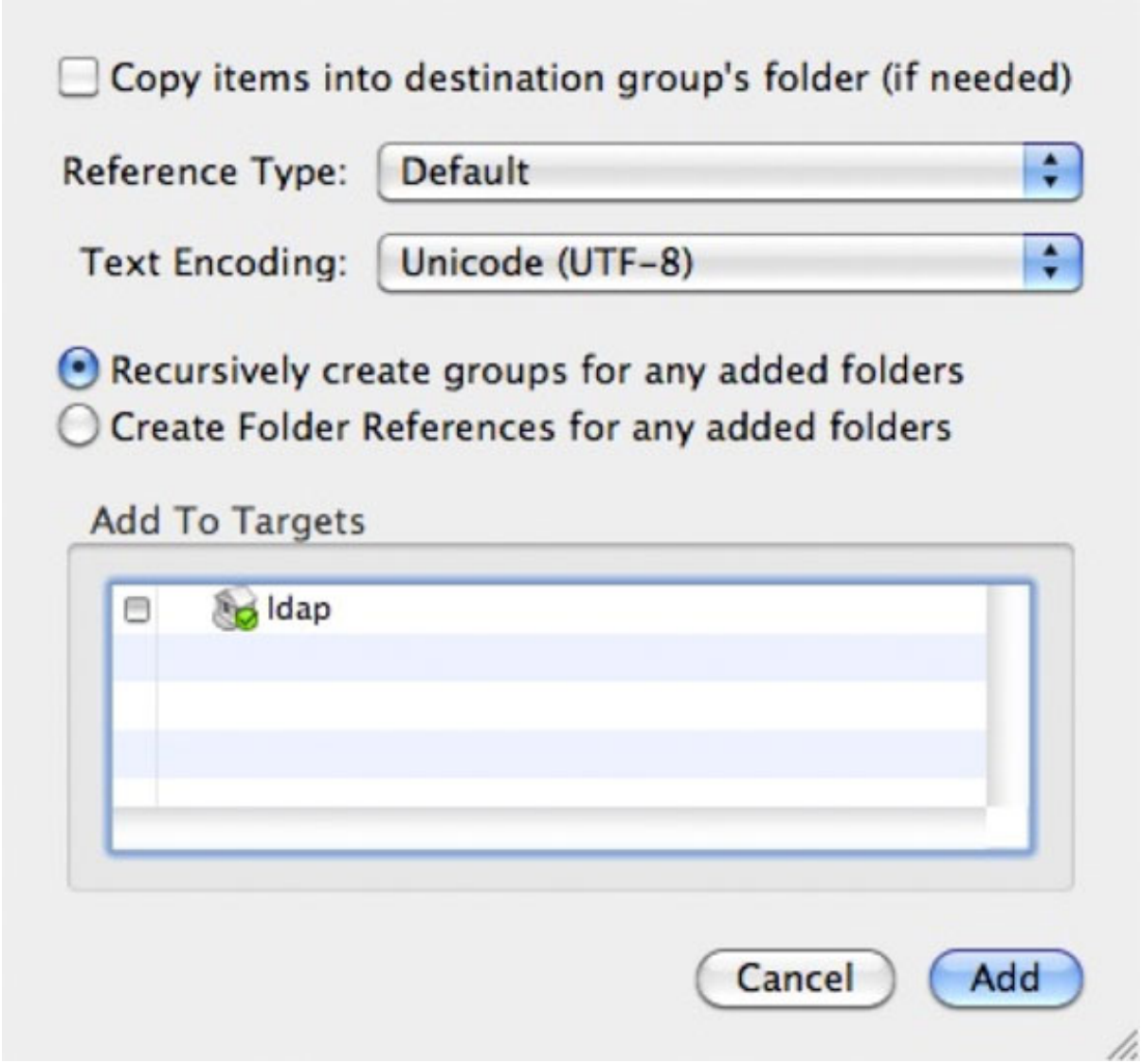
Xcode is the IDE that is included with the iPhone SDK. It contains templates and documentation for building applications and libraries for the iOS platform. To make navigating the source tree for OpenLDAP easier, we will first import the source code into an Xcode project. To create a new Xcode project for the iOS platform, open Xcode and select **New Project** from the File menu on the menu bar. This should open a new dialog that has three panels (see Figure 1). Since the project will be used to compile the OpenLDAP client library, the project should be created with the static library template. On the left panel, select **Library** under the iPhone OS heading. From the top right panel, select **Cocoa Touch Static Library**. Click **Choose** to continue.

Figure 1. New Project dialog in Xcode



The dialog in Figure 2 is used to set the name for the Xcode project.

Figure 2. Dialog for adding existing files to Xcode project



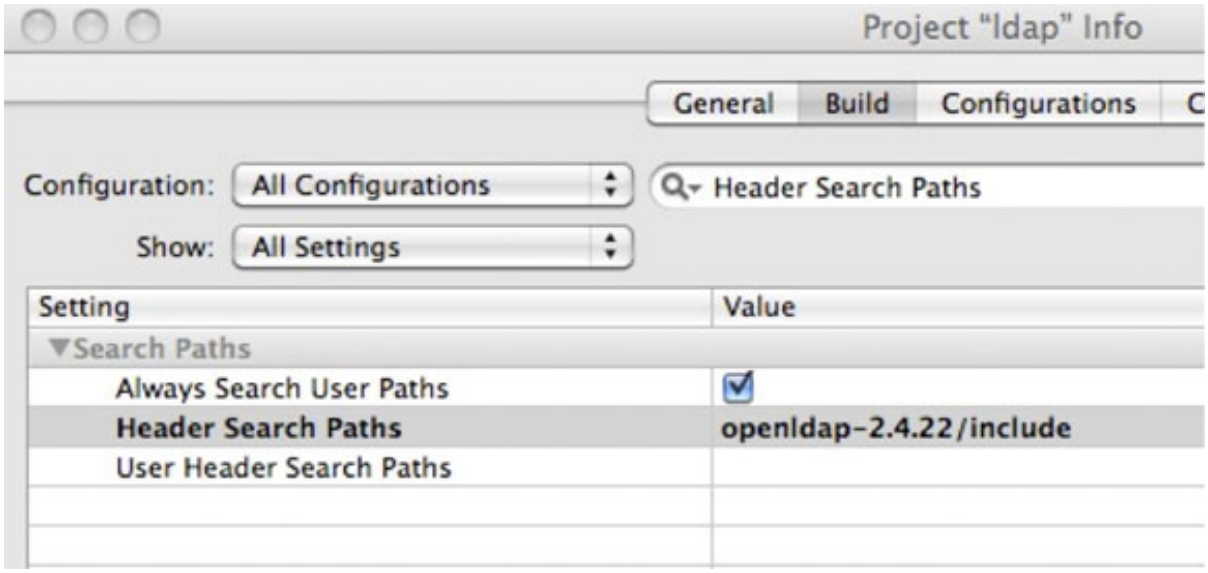
Xcode will create the library's name by pre-pending lib and appending .a to the project name. So to create a library with the name of libldap.a, the project name needs to be set to ldap. Type a name into the Save As box and click **Save**. This will create a directory that contains the initial files for the project.

Now that the project is created, the source code from OpenLDAP can be imported. Download the source code from the OpenLDAP project page and extract the tar file it into the Xcode project folder. From the Project menu on the menu bar select **Add** to Project. From the file browser that opens, select the folder within the Xcode project, which contains the source code for OpenLDAP, and click the **Add** button. A new dialog box should appear. Make sure the option "Recursively create groups for any added folders" is selected and unselect ldap from the list of targets. Click the **Add** button to finish adding the source code to the project. A new group with the name of the OpenLDAP directory should now be in the project's Groups & Files

panel of Xcode's main project window.

The OpenLDAP source tree contains a directory that has header files that will be included by the source files used by the libraries. Xcode needs to be configured to tell the preprocessor to search this directory when processing #include directives. Double-click on the project name in the Groups & Files panel of Xcode's main project window. This should open the project info window (see Figure 3). Click in the "Search in Build Settings" search box and type in `Header Search Paths`. Double-click in the Value field to add a new search path and add `openldap-2.4.22/include`.

Figure 3. Project info dialog



Preparing header files

OpenLDAP has been ported to many platforms. The OpenLDAP developers use C header files to define system information at compile time. Normally these header files are created by modifying template files with scripts generated from AutoConf. Since Xcode does not use Autoconf, these files must be modified manually. The files that must be modified are lber_types.hin, ldap_config.hin, ldap_features.hin, and portable.hin. These files are located within the include directory within the OpenLDAP source tree.

lber_types.hin

The file lber_types.hin contains the template for defining variable types. The template file needs to be renamed lber_types.h. This can be accomplished by right-clicking on the file name from within Xcode and selecting rename from the menu. This will update Xcode's meta data for the file to treat it as a C header file. Open the file for editing, and find the following lines shown in Listing 1.

Listing 1. lber_types.h

```
27 /* LBER boolean, enum, integers (32 bits or larger) */
28 #undef LBER_INT_T
29
30 /* LBER tags (32 bits or larger) */
31 #undef LBER_TAG_T
32
33 /* LBER socket descriptor */
34 #undef LBER_SOCKET_T
35
36 /* LBER lengths (32 bits or larger) */
37 #undef LBER_LEN_T
```

These lines will declare the variable types used by the library. Since the iPhone is a 32-bit platform, type int is a suitable value. Change the #undef macros to #define and set the variable types to int (see Listing 2).

Listing 2. Adjusting for the 32-bit platform

```
27 /* LBER boolean, enum, integers (32 bits or larger) */
28 #define LBER_INT_T int
29
30 /* LBER tags (32 bits or larger) */
31 #define LBER_TAG_T int
32
33 /* LBER socket descriptor */
34 #define LBER_SOCKET_T int
35
36 /* LBER lengths (32 bits or larger) */
37 #define LBER_LEN_T int
```

The remainder of the file declares custom variable types used by the library and do not need to be modified: `ldap_config.hin`.

The file ldap_config.hin contains the template for defining where the library should search for LDAP client configuration files. Since the iOS does not allow the user to create arbitrary files within the device's file system, the contents of the file doesn't need to be modified. The files does need to be renamed to ldap_config.h so that the source files do not generate errors when being compiled due to a failed include directive. The file can be renamed using the Xcode interface in the same manner as lber_types.hin was renamed to lber_types.h.

ldap_features.hin

ldap_features.hin contains information about which features are required by the iOS platform and information about the current version of OpenLDAP. Re-name ldap_features.hin to ldap_features.h and find the following lines shown in Listing 3.

Listing 3. ldap_features.h

```
23 /* OpenLDAP API version macros */
24 #undef LDAP_VENDOR_VERSION
24 #undef LDAP_VENDOR_VERSION_MAJOR
24 #undef LDAP_VENDOR_VERSION_MINOR
24 #undef LDAP_VENDOR_VERSION_PATCH
```

OpenLDAP uses a three part version number for each release. The version is notated in the format of X.Y.Z where X is the major vendor version number of the release, Y is the minor vendor version number of the release, and Z is the vendor patch revision of the release. The value of `LDAP_VENDOR_VERSION` is calculated with the formula of `((X*10,000)+(Y*100)+(Z))`. For example, the `LDAP_VENDOR_VERSION` for OpenLDAP 2.4.22 would be calculated using the values shown in Listing 4.

Listing 4. Calculating the LDAP_VENDOR_VERSION

```
LDAP_VENDOR_VERSION_MAJOR = X = 2
LDAP_VENDOR_VERSION_MINOR = Y = 4
LDAP_VENDOR_VERSION_PATCH = Z = 22

LDAP_VENDOR_VERSION = ((X*10000)+(Y*100)+(Z))
LDAP_VENDOR_VERSION = ((2*10000)+(4*100)+(22))
LDAP_VENDOR_VERSION = (20000+400+22)
LDAP_VENDOR_VERSION = (20422)
```

ldap_features.h needs to be updated to reflect the version of the OpenLDAP source code. Replace the #undef macros with #define and insert the version information. For example, OpenLDAP 2.4.22 would result in the following modifications shown in Listing 5.

Listing 5. Modifications from OpenLDAP 2.4.22

```
23 /* OpenLDAP API version macros */
24 #define LDAP_VENDOR_VERSION  20422
24 #define LDAP_VENDOR_VERSION_MAJOR 2
24 #define LDAP_VENDOR_VERSION_MINOR 4
24 #define LDAP_VENDOR_VERSION_PATCH 22
```

The rest of the file does not need to be modified.

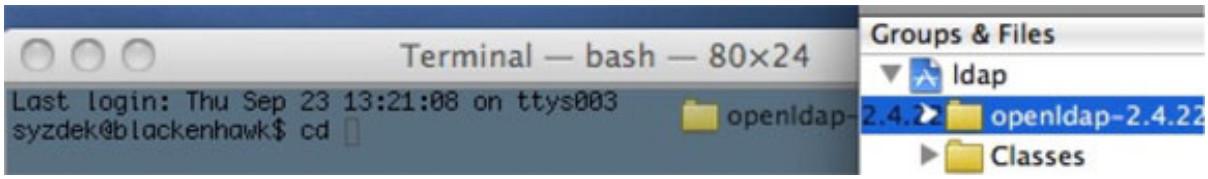
portable.hin

Information regarding the library functions and header files that are available in the iPhone SDK need to be set in a file named portable.hin. However, this file is very extensive and requires digging to set correctly. Luckily, OpenLDAP uses autoconf to perform the required tests of multiple platforms. The autoconf scripts can be used to generate the values for portable.hin.

To use the OpenLDAP autoconf scripts, open the Terminal.app found in /Applications/Utilities/Terminal.app. Type in the command `cd` and a space. Then

drag the folder within Xcode that contains the OpenLDAP source tree into the terminal window (see Figure 4).

Figure 4. Showing the process of dragging the Xcode group to the Terminal.app to obtain the path to the folder



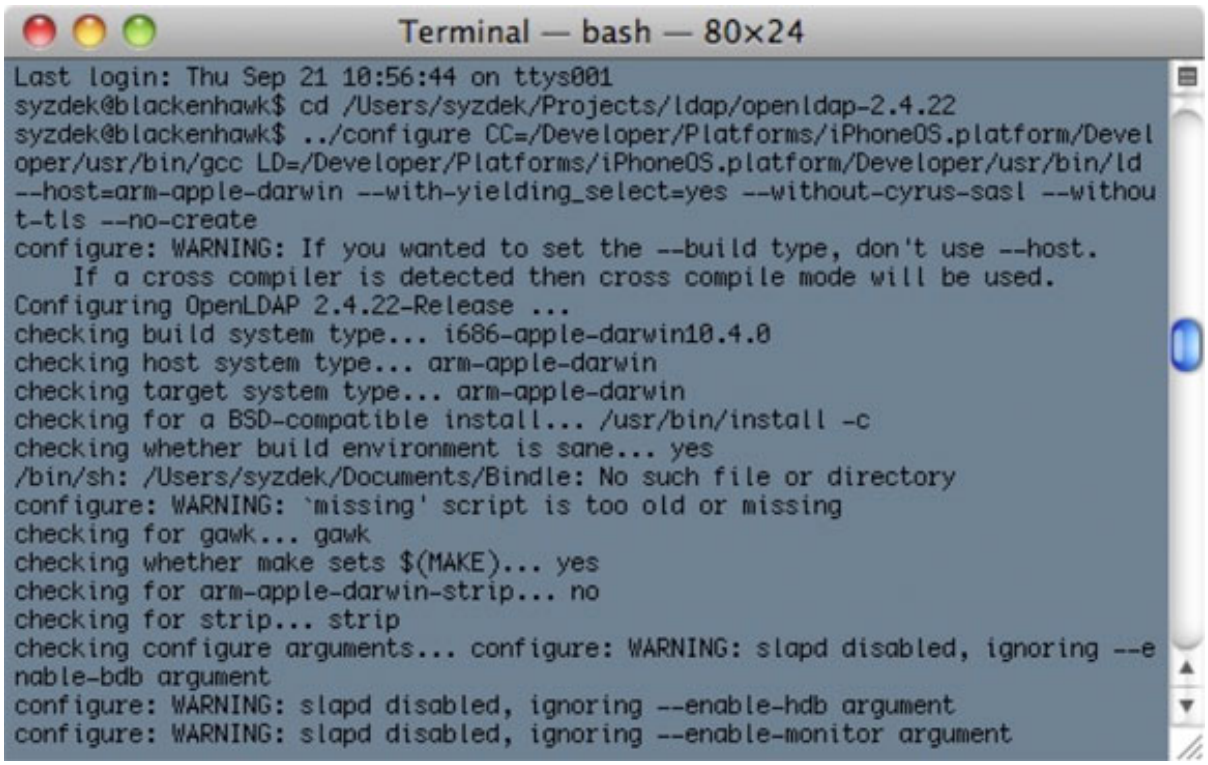
Press Enter on the keyboard to change directories within the Terminal.app to the location of the OpenLDAP source tree. Run the configure script with the following flags shown in Listing 6.

Listing 6. Running the configure script

```
./configure CC=/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/gcc \
   LD=/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/ld \
   --host=arm-apple-darwin --disable-slapd --without-cyrus-sasl \
   --without-tls --no-create
```

The flag `CC=/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/gcc` specifies the location of the compiler used by the iPhone SDK and the `LD=/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/ld` specifies the location of the linker. The flag `--host=arm-apple-darwin` informs autoconf on which platform the compiled code will be used. The flag `--disable-slapd` disables checks required to build the LDAP server, which ships with OpenLDAP. The iPhone SDK does not include the Cyrus SASL library, the OpenSSL library, or the GNU SSL/TLS library, so the flags `--without-cyrus-sasl` and `--without-tls` disable checks for the functions contained within these libraries. Finally, the flag `--no-create` prevents autoconf from creating Makefiles and the header files. The output should look similar to the Figure 5.

Figure 5. First few lines of the configure script running

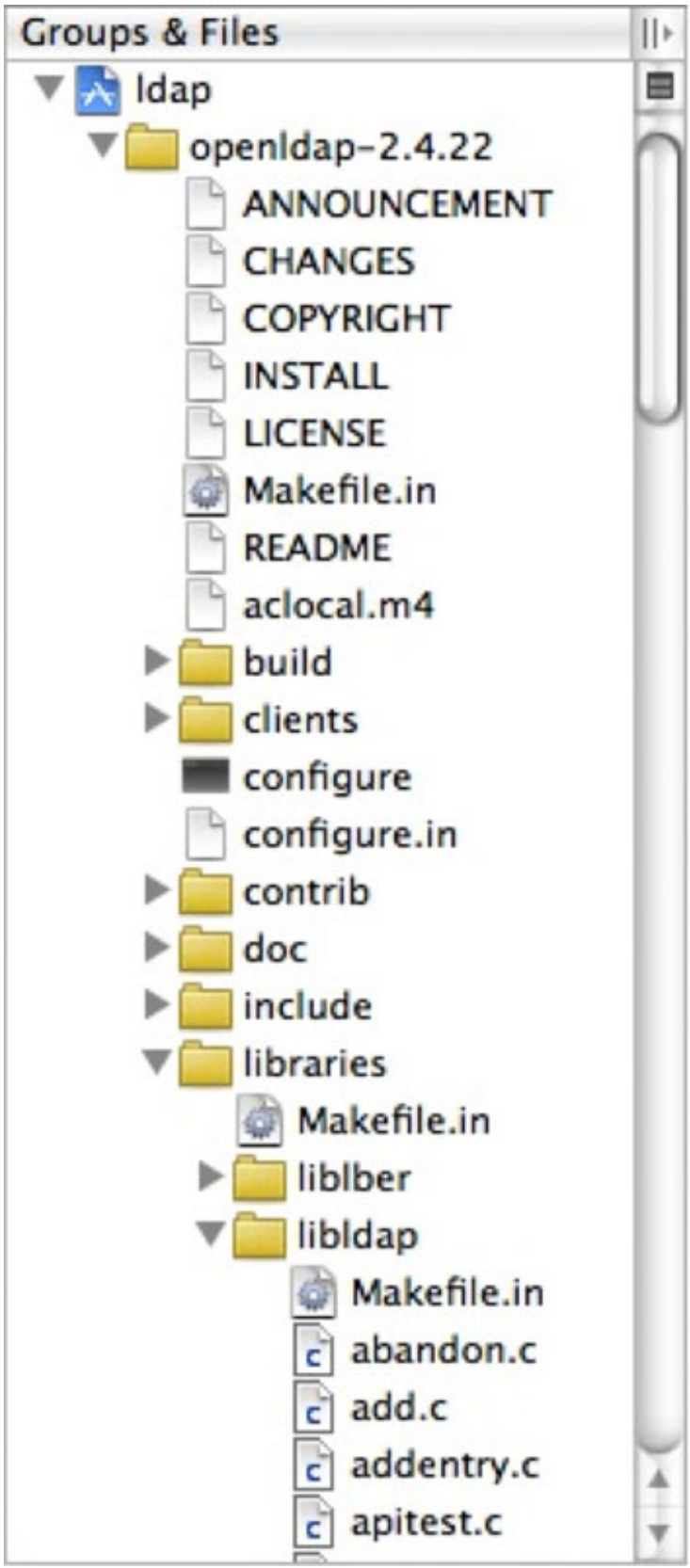


Run the following to have autoconf generate portable.h from the template file portable.hin: `./config.status --header=include/portable.h:include/portable.hin`.

Creating libldap.a

Now that the source code and the configuration files have been prepared, it is time to tell Xcode which source files will be used by the LDAP library. Using the Xcode interface, navigate the OpenLDAP source tree to the directory libldap (see Figure 6).

Figure 6. Location of the libldap source files within the Groups & Files panel of Xcode



In the libldap directory, there is a file called Makefile.in that contains a list of the files used to build libldap.a. Open Makefile.in and find the lines shown in Listing 7.

Listing 7. Makefile.in

```
20 SRCS = bind.c open.c result.c error.c compare.c search.c \
21         controls.c messages.c references.c extended.c cyrus.c \
22         modify.c add.c modrdn.c delete.c abandon.c \
23         sasl.c gssapi.c sbind.c unbind.c cancel.c  \
24         filter.c free.c sort.c passwd.c whoami.c \
25         getdn.c getentry.c getattr.c getvalues.c addentry.c \
26         request.c os-ip.c url.c pagectrl.c sortctrl.c vlvctrl.c \
27         init.c options.c print.c string.c util-int.c schema.c \
28         charray.c os-local.c dnssrv.c utf-8.c utf-8-conv.c \
29         tls2.c tls_o.c tls_g.c tls_m.c \
30         turn.c ppolicy.c dds.c txn.c ldap_sync.c stctrl.c \
31         assertion.c deref.c
```

The files in this list need to be updated to be included in the ldap target. To do this, right-click on a file from the list and select item **Get Info** from the menu that appears. In the new window, select the Targets tab and click the ldap target from the Target Memberships panel. Repeat this for the entire list of files.

Creating liblber.a

liblber.a is a library required by libldap.a when compiling an application. The library is included with the OpenLDAP distribution. To build the library, an Xcode target for the library must first be created. To create the new target, right-click on the project icon in the Groups & Files panel of Xcode. Select **Add** from the menu and then select **New Target** from the secondary menu. A New Target wizard should appear. Select **Cocoa Touch** from the panel on the left and then select **Static Library** from the panel on the right. Click the **Next** button to continue. In the Target Name: text box, type `lber` and click **Finish**. This should create the new library target.

liblber.a depends upon the Foundation framework. To configure the Foundation framework, scroll to the Targets section in the Groups & Files panel of Xcode. Expand the Targets section and double-click on the target lber. This should open the target info window for lber. Click on the General tab and then click on the + button below the Linked Libraries panel. From the pop-up window, select Foundation.framework from the Device - iPhone OS 4.1 SDK section. Click the **Add** button to finish adding the framework to the library.

Finally, Xcode needs to be configured with the source files to compile for liblber.a. Using the Xcode interface, navigate the OpenLDAP source tree to the directory liblber. In the liblber directory, there is a file called Makefile.in which contains a list of the files used to build liblber.a.

Open Makefile.in and find the lines shown in Listing 8.

Listing 8. Lines from Makefile.in

```
21 UNIX_SRCS = stdio.c

26 SRCS= assert.c decode.c encode.c io.c bprint.c debug.c \
27         memory.c options.c sockbuf.c $(@PLAT@_SRCS)
```

The files in these lists need to be updated to be included in the lber target. To do this, right-click on a file from the list and select item **Get Info** from the menu that appears. In the new window, select the **Targets** tab and click the lber target from the Target Memberships panel. Repeat this for all the files in the lists.

The library can be built by clicking the **Build** button in the Xcode toolbar. To switch between building for the simulator or a device, choose the desired SDK from the drop-down box labeled **Overview**.

Conclusion

As with most methods of software development, there are other ways to compile ported libraries for the iOS. This article uses Xcode to perform the compiling to allow the libraries to be easily integrated into other packages using Xcode's dependency tracking and to allow the libraries to be easily updated with future versions of the iPhone SDK.

Part 2 of this series will show how to create a second Xcode project adds the Xcode project created in this article as a dependency and uses the static libraries to create an LDAP client for the iPhone.

Downloads

Description	Name	Size	Download method
Zip file	ldap.zip	836KB	HTTP

Information about download methods

Resources

Learn

- Android and iPhone browser wars, Part 1: WebKit to the rescue (Frank Ableson, developerWorks, December 2009): See how you can leverage the features of mobile browsers in this developerWorks article.
- Introduction to LDAP: Part 1, Installation and simple Java LDAP programming (Jeng Yoong Tan, developerWorks, June 2010): Get a general overview of LDAP (Lightweight Directory Access Protocol).
- Start your learning with Open Source: Read this blog entry from Chris Walden about the wealth of open source packages available for almost any job.
- For an article series that will teach you how to program in bash, see Bash by example, Part 1: Fundamental programming in the Bourne again shell (bash) (Daniel Robbins, developerWorks, March 2000), Bash by example, Part 2: More bash programming fundamentals (Daniel Robbins, developerWorks, April 2000), and Bash by example, Part 3: Exploring the ebuild system (Daniel Robbins, developerWorks, May 2000).
- Making UNIX and Linux work together (Martin Brown, developerWorks, April 2006) is a guide to getting traditional UNIX distributions and Linux® working together.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Follow developerWorks on Twitter..
- Access the Xcode Port of OpenLDAP.
- To listen to interesting interviews and discussions for software developers, check out developerWorks podcasts.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Get the iPhone SDK. It can be downloaded by registered ADC members.
- Check out GitHub project for iOS ports. It automates the building of OpenSSL, Cyrus-SASL, and OpenLDAP and has been tested with SASL Auth, LDAPS, and LDAP TLS.
- Download OpenLDAP from the project's web site.
- Get the Xcode Port of OpenLDAP.

- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

David M. Syzdek

David M. Syzdek has ten years experience developing UNIX software for telecommunications companies. When the iPhone SDK was released in 2007, he started developing applications for mobile devices and was among the few developers to release apps during the iTunes App Store initial launch. David currently works as an independent developer and releases applications under the name of Bindle Binaries.