SUNGARD 金仕达

Futures Trading API Special Instruction

Document Description

Document Name	KSFT_API Special Instruction
ID	KS/IRDG-KSFT-05-2014
Version	<V3.1.1>
Status	Updated

Document Revision History

version	date of change	memo
V3.1.0	<2014-05-30>	Creating this document, contents added: 1. introduction to compiling environment of the API; 2. programming guide for Mac; 3. some tips.
V3.1.1	2014-06-13	Add programming guide for iOS App
V3.1.1	2014-06-24	Detail programming guide for iOS App
V3.1.1	2014-07-03	1. Add CPU architectures that the API supports. 2. Add others: the initialization parameters only take affect while creating an API instance at the first time.

Contents

Programming Guide for Mac App

This part is used to guide development of Mac App with KSFT_API. If no special comments, the version of Xcode mentioned in this part is 5.0.2.

1. Environment for Compiling KSFT_API

- OS: Mac OS X 10.9.3
- Compiler: clang 500.2.79
- C++ standard library used: libc++
- Optimization level: O2

2. Guide to build App

2.1. Include Header Files

Please add header files of KSFT_API to the project.

2.2. Link to KSFT_API

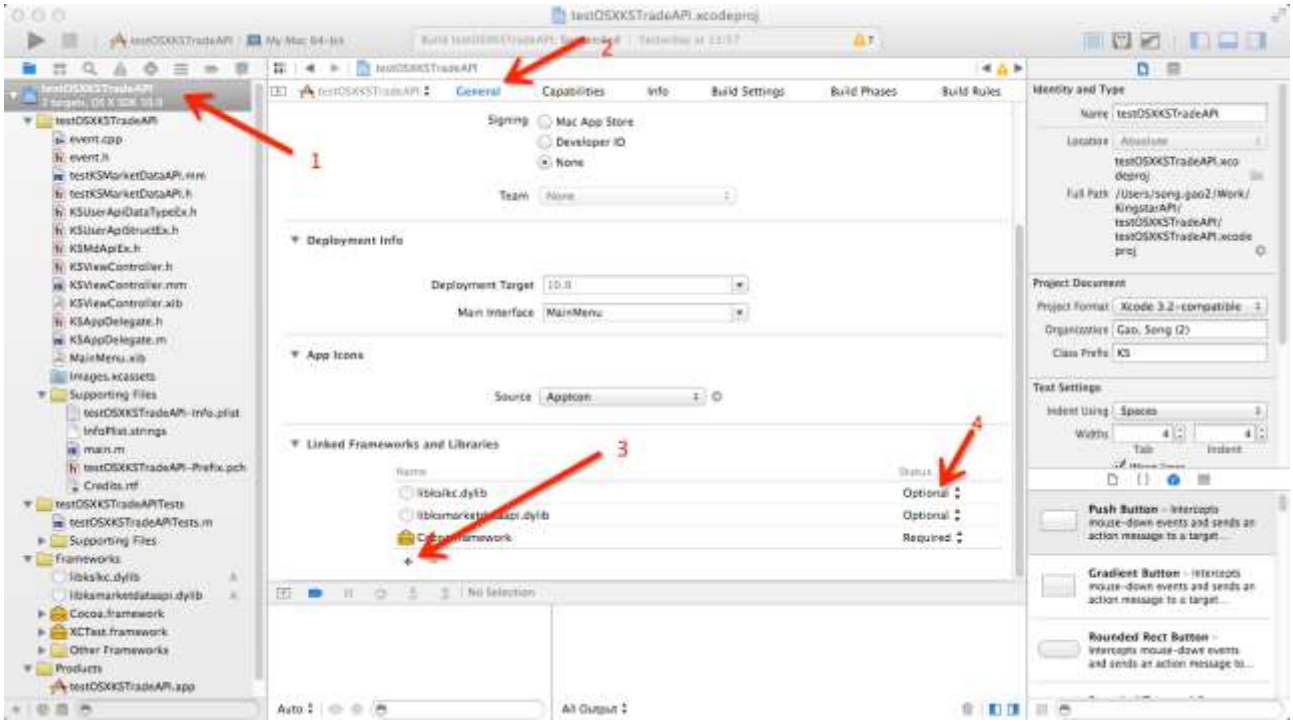

Figure 1 Linking dynamic libraries of the API

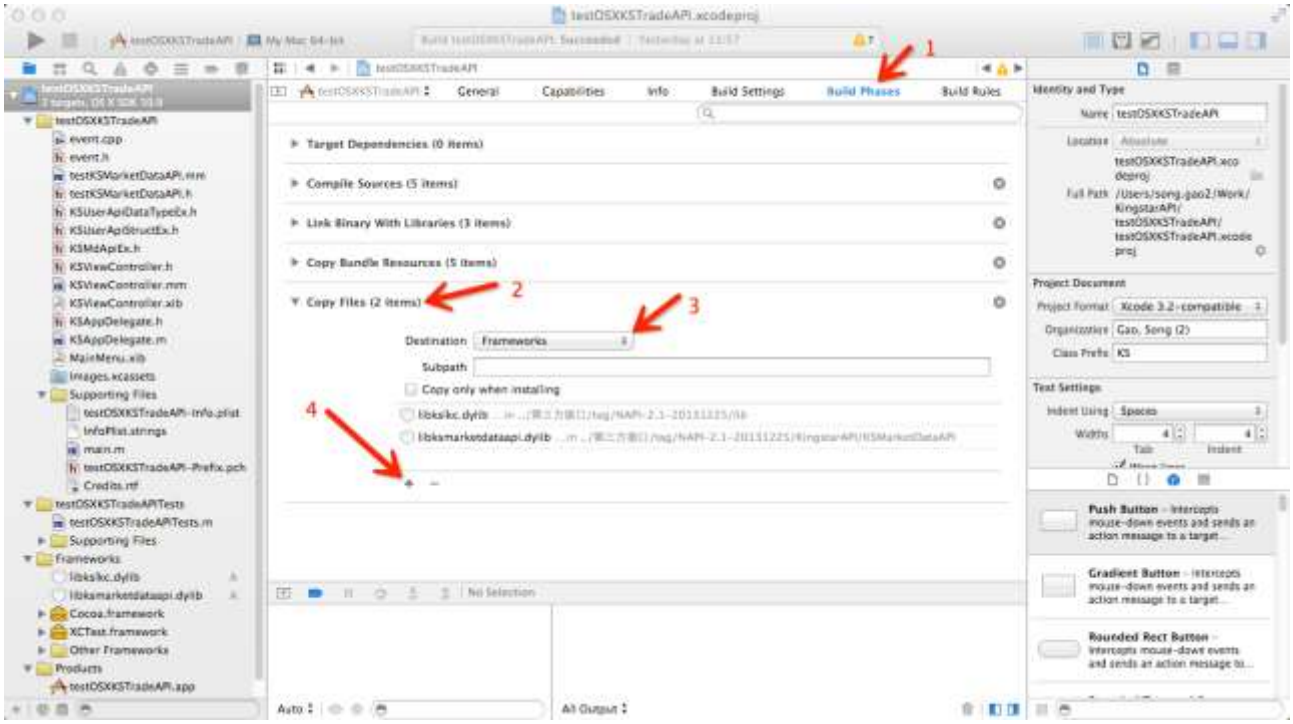

Figure 2 Copy dynamic libraries to the bundle

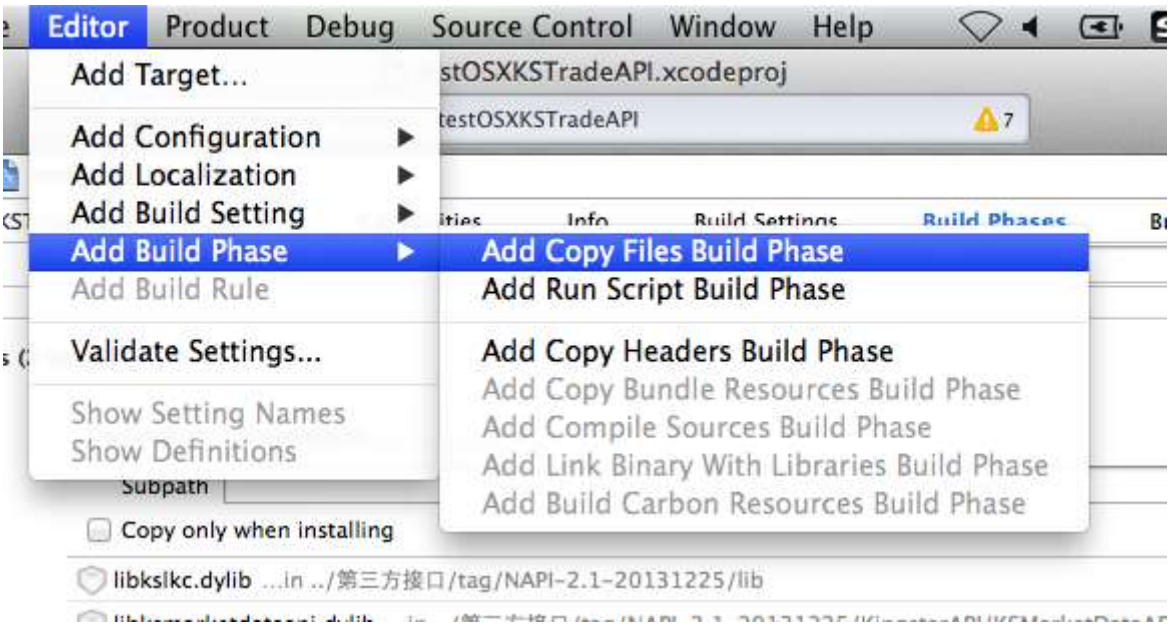

Figure 3 Add “Copy Files” Build Phase

Steps:

1. Select project file in Xcode and switch to “General” Tab (as shown in Figure 1, Tag 1 and 2).
2. Find “Linked Frameworks and Libraries” group， and add dynamic libraries to the project. Change Status in the right column to “Optional” (as shown in Figure 1, Tag 3 and 4).
3. Switch to “Building Phases” Tab (as shown in Figure 2, Tag 1) and add “Copy Files” Phase, as shown in Figure 3.
4. Set Destination to “Frameworks” and click “+” button to add dynamic libraries of the API.

2.3. Set up Runpath

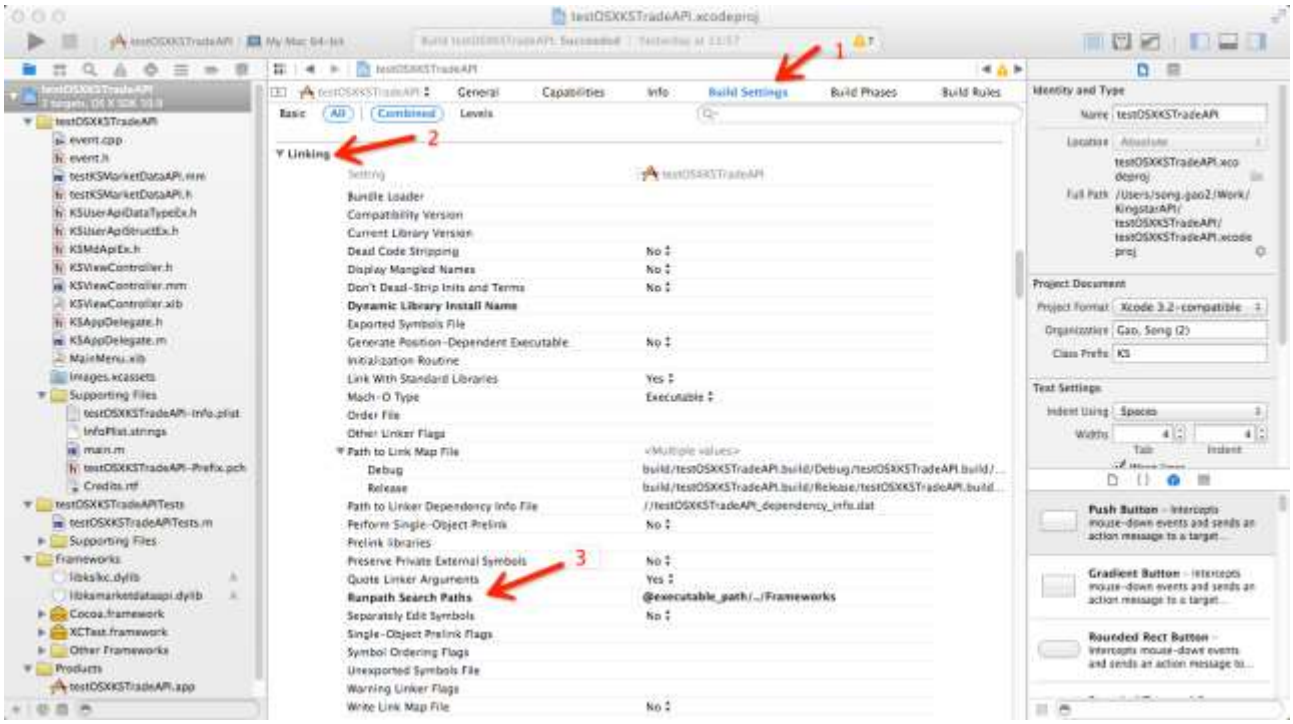

Figure 4 Set up rpath

Steps：

1. Switch to “Build Setting” Tab and find “Linking” group (as shown in Figure 4, Tag 1 and 2).
2. Set up “Runpath Search Paths” with “@executable_path/../Frameworks” (as shown in Figure 4, Tag 3).

2.4. Copy License File “KSInterB2C.lkc” to Bundle

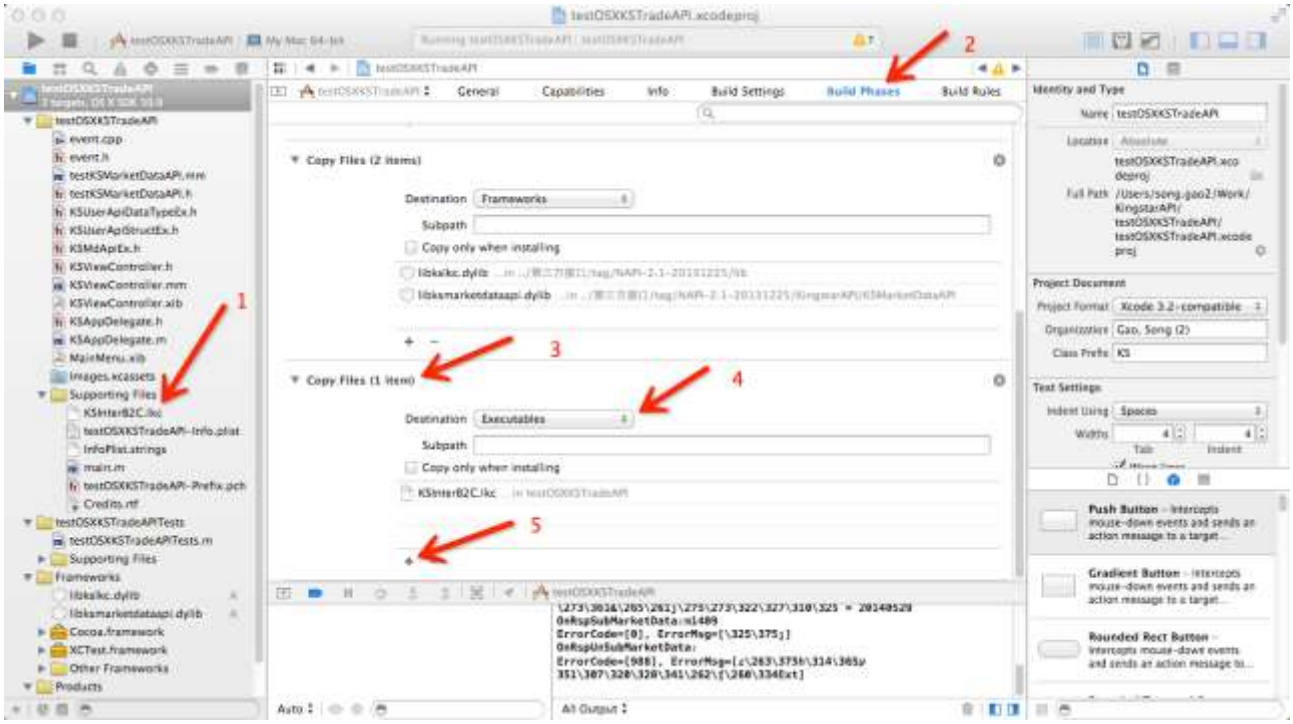

Figure 5 Copy License File

Steps：

1. Add license file “KSInterB2C.lkc” to project (as shown in Figure 5, Tag 1).
2. Switch to “Build Phases” Tab and add “Copy Files” phase, as shown in Figure 3.
3. Set Destination to “Executable” and click “+” button to add the license file (as shown in Figure 5, Tag 4 and 5).

2.5. Others

KSFT_API is written with C++, and needs corporations with Objective-C++. Please make sure the “Objective-C++” source files have suffix “.mm”. Otherwise, compiling or linking errors may occur.

Programming Guide for iOS App

This part is used to guide development of iOS App with KSFT_API. If no special comments, the version of Xcode mentioned in this part is 5.1.

1. Environment for Compiling KSFT_API

- OS: Mac OS X 10.9.3
- Compiler: clang 503.0.38
- C++ standard library used: libc++
- Optimization level: O2
- iOS SDK: 7.1
- Minimum iOS version: 7.0
- Supported CPU architectures: KS_API for Hardware supports armv7 and arm64, and that for simulator supports i386 and x86_64.

2. Guide to build App

2.1. Include Header Files

Please add header files of KSFT_API to the project.

2.2. Link to KSFT_API

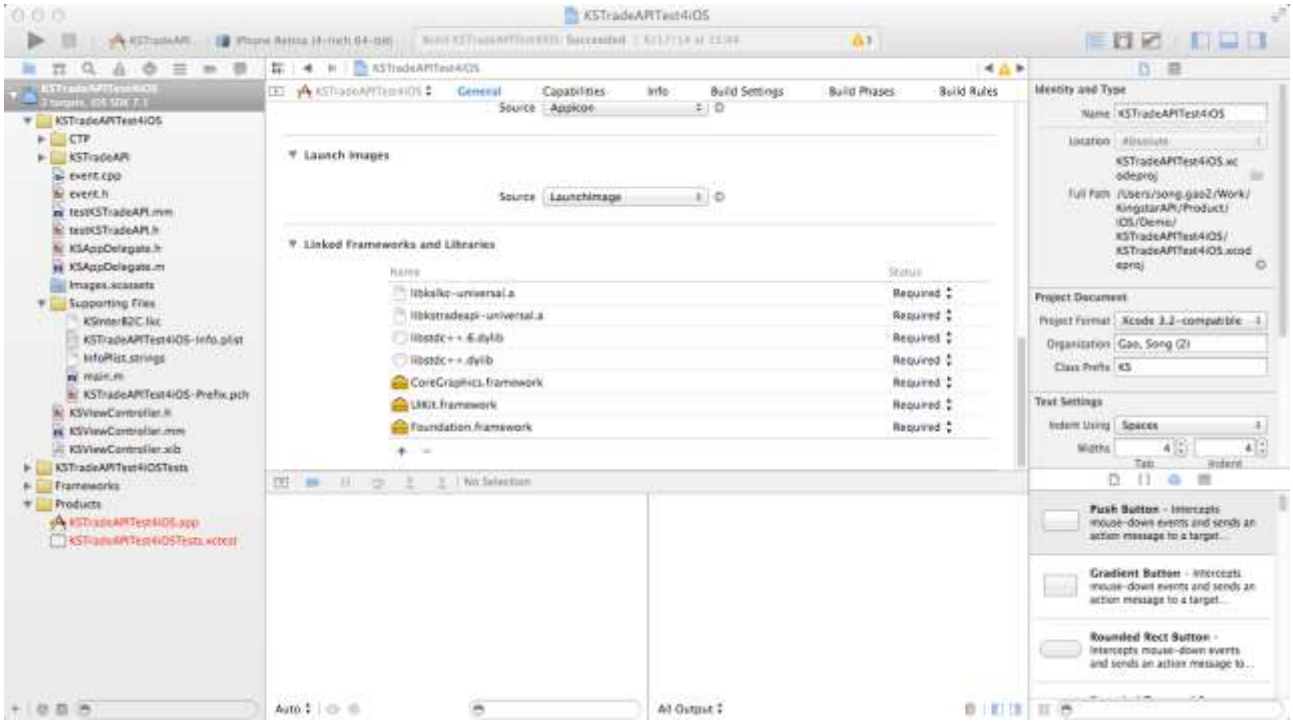

Figure 6 Link to KSFT_API static libraries

Steps:

1. Select project file in Xcode and switch to “General” Tab.
2. Find “Linked Frameworks and Libraries” group， and add API static libraries (“libkslkc.a” and “libksft.a”) to the project. And please make sure “libstdc++.6.dylib” and “libstdc++.dylib” are added too.

2.3. Add License File “KSInterB2C.lkc” to Bundle

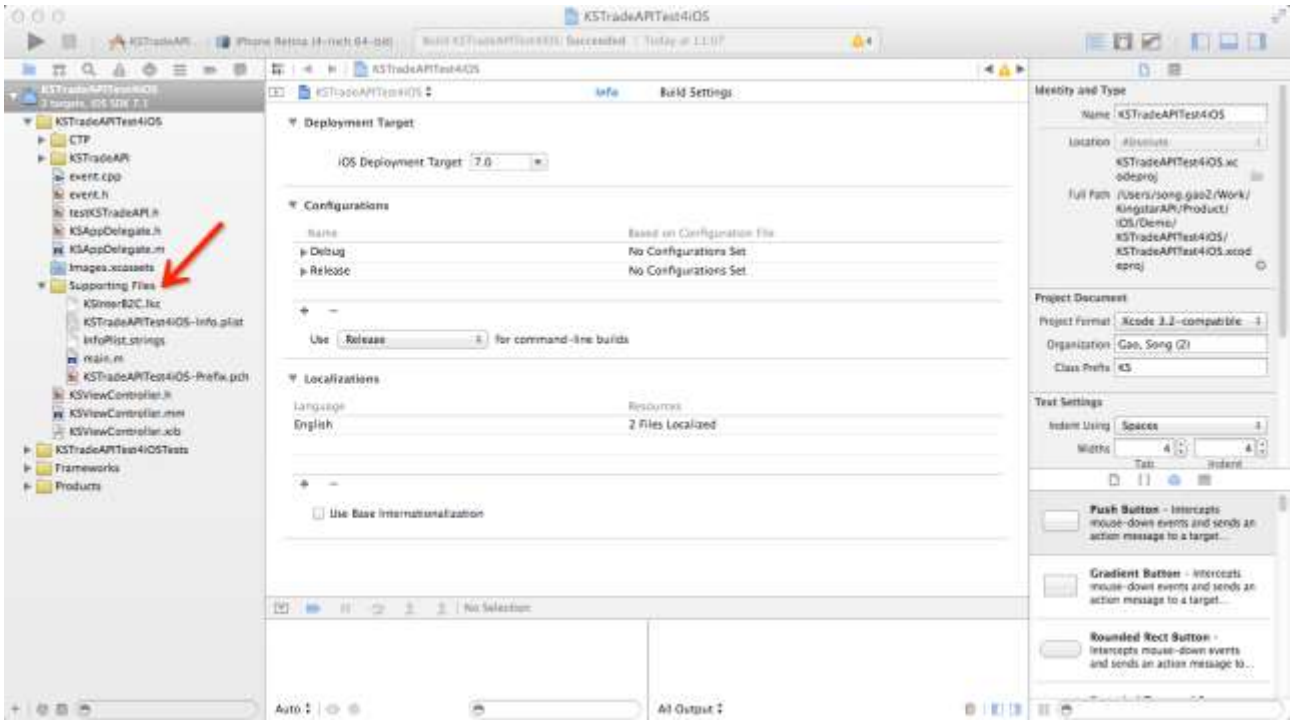

Figure 7 Add license file

Select “Supporting Files” group, and add license file “KSInterB2C.lkc” to it.

2.4. Others

1. KSFT_API is written with C++, and needs corporations with Objective-C++. Please make sure the “Objective-C++” source files have suffix “.mm”. Otherwise, compiling or linking errors may occur.
2. Character encoding. Chinese characters in this API and server-side are encoded with GBK.
3. The default log path is Document directory in App.
4. Trade and market API working together, the initialization parameters only take affect while creating an API instance at the first time.

Tips

1. Check dependencies of a dynamic library or an executable file.

   $ otool –L <dylib or executable>

   Example：

```
AP-CHN-LP140007:KSTradeAPI song.gao2$ otool -L libkslkc.dylib
libkslkc.dylib:
        @rpath/libkslkc.dylib (compatibility version 0.0.0, current version 0.0.0)
        /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.0.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1197.1.1)
```

2. Check “install name” of a dynamic library.

   $ otool –D <dylib>

   Example：

```
AP-CHN-LP140007:KSTradeAPI song.gao2$ otool -D libkstradeapi.dylib
libkstradeapi.dylib:
@rpath/libkstradeapi.dylib
```

3. Change the “install name” of a dynamic library.

   $ install_name_tool –id <install name> <dylib>

   Example：

```
AP-CHN-LP140007:KSTradeAPI song.gao2$ install_name_tool -id libkstradeapi.dylib libkstradeapi.dylib
AP-CHN-LP140007:KSTradeAPI song.gao2$ otool -D libkstradeapi.dylib
libkstradeapi.dylib:
libkstradeapi.dylib
```

4. Check CPU architectures supported by a library.

   $ lipo –info <library>

   Example：

```
AP-CHN-LP140007:KSTradeAPI song.gao2$ lipo -info libkstradeapi.dylib
Architectures in the fat file: libkstradeapi.dylib are: i386 x86_64
```

5. Extract a library supporting specified CPU architecture from an universal library.

   $ lipo -extract <arch_type> -o <output> <universal dylib>

   Example：

```
AP-CHN-LP140007:KSTradeAPI song.gao2$ lipo -info libkstradeapi.dylib
Architectures in the fat file: libkstradeapi.dylib are: i386 x86_64
AP-CHN-LP140007:KSTradeAPI song.gao2$ lipo -extract i386 -o libkstradeapi_i386.dylib libkstradeapi.dylib

AP-CHN-LP140007:KSTradeAPI song.gao2$ ls
KSTradeAPI.h            libkslkc.dylib          libkstradeapi.dylib     libkstradeapi_i386.dylib
AP-CHN-LP140007:KSTradeAPI song.gao2$ lipo -info libkstradeapi.dylib
Architectures in the fat file: libkstradeapi.dylib are: i386 x86_64
AP-CHN-LP140007:KSTradeAPI song.gao2$ lipo -info libkstradeapi_i386.dylib
Architectures in the fat file: libkstradeapi_i386.dylib are: i386
```

6. Create an universal library.

   $ lipo –create <i386 dylib> < x86_64 dylib> -o <dylib>

   Example：

```
AP-CHN-LP140007:KS_API song.gao2$ ls
libksmarketdataapi_i386.dylib   libkstradeapi_i386.dylib
libksmarketdataapi_x86_64.dylib libkstradeapi_x86_64.dylib
AP-CHN-LP140007:KS_API song.gao2$ lipo -create libkstradeapi_i386.dylib libkstradeapi_x86_64.dylib -o lib
kstradeapi.dylib
AP-CHN-LP140007:KS_API song.gao2$ lipo -info libkstradeapi_i386.dylib
Non-fat file: libkstradeapi_i386.dylib is architecture: i386
AP-CHN-LP140007:KS_API song.gao2$ lipo -info libkstradeapi_x86_64.dylib
Non-fat file: libkstradeapi_x86_64.dylib is architecture: x86_64
AP-CHN-LP140007:KS_API song.gao2$ lipo -info libkstradeapi.dylib
Architectures in the fat file: libkstradeapi.dylib are: i386 x86_64
```

7. Pass “rpath” to a linker, when compiling.

   $ clang++ -Wl,-rpath –Wl,<run path> …

   Refer to the makefile for test program for the example.