# Building Ice Touch from Source

This page describes the Ice Touch source distribution, including information about compiler requirements, third-party dependencies, and instructions for building and testing the distribution. If you prefer, you can download a binary package that contains pre-compiled libraries, executables, and everything else necessary to build Ice Touch applications on OS X.

On this page:



## Build Requirements

Ice Touch was extensively tested using the operating systems and compiler versions listed on our platforms page.

### Third-party Libraries

Ice Touch depends on mcpp 2.7.2 (with patches). Binary and source distributions of mcpp are available on our download page.

### Xcode Plug-in

Our Xcode Plug-in is required for building the Cocoa and iPhone sample projects included in this distribution. We strongly recommended that you develop Ice for C++ and Ice Touch applications with Xcode, as it greatly simplifies the build process.

Back to Top ^

## Compiling and Testing Ice Touch

Unpack the Ice Touch source archive. The source files are contained in the `IceTouch-1.3.3` subdirectory.

### Building the Slice Compilers

The mandatory first step is to build the Slice-to-C++ and Slice-to-Objective-C compilers (`slice2cpp` and `slice2objc`, respectively).

Change the working directory to `IceTouch-1.3.3/cpp`:

```
$ cd IceTouch-1.3.3/cpp
```

Set `MCPP_HOME` to point to the installation directory of your MCPP binaries, for example:

```
$ export MCPP_HOME=$HOME/IceTouch-1.3.3-ThirdParty
```

Set `OPTIMIZE_SPEED` to `yes` to create an optimized build:

```
$ export OPTIMIZE_SPEED=yes
```

Then run make to build the compilers:

```
$ make
```

Back to Top ^

### Building the C++ SDKs

You can build a C++ SDK for each of three different targets: Cocoa, iPhone, and iPhone Simulator. Change to the `IceTouch-1.3.3/cpp` directory and run one of the following commands to build an SDK:

```
$ cd IceTouch-1.3.3/cpp
$ make COMPILE_FOR_COCOA=yes
or
$ make COMPILE_FOR_IPHONE=yes
or
$ make COMPILE_FOR_IPHONE_SIMULATOR=yes
```

This will build static libraries and a test suite for the specified target. The build configuration options can be found in `config/Make.rules`.

When building for iPhone in debug mode, make will by default build only for `armv7`. You can specify different architecture(s) with `CXXARCHFLAGS`, for example:

```
$ make COMPILE_FOR_IPHONE=yes CXXARCHFLAGS="-arch arm64"
```

Ice Touch supports `armv7`, `armv7s` and `arm64`.

OS X and iOS provide two C++ run time libraries, the older `libstd++` and newer `libc++`. By default, Xcode 6 links with `libc++` when targeting OS X 10.9 or iOS >= 7, and with `libstd++` when targeting earlier versions.

The Ice Touch build system targets iOS 5.1.1 and OS X 10.7, and as a result creates libraries linked with `libstd++`. If you want to create libraries linked with `libc++`, set `CPP11` to `yes`, for example:

```
$ make COMPILE_FOR_IPHONE=yes CXXARCHFLAGS="-arch arm64" CPP11=yes
```

The resulting libraries will carry a `-libc++` suffix, such as `libIceCpp-libc++.a` or `libIceGridCpp-libc++.a`

These `-libc++` libraries can be used by C++ applications that use C++11 features (`--std=c++11`), and also by C++ applications that do not (no `--std=c++11`).

If you built for iPhone or the iPhone simulator, you should now use Xcode to build the demos and test suite GUI located in the following subdirectories:

- `demo/iPhone/hello`
- `test/iPhone/container`

After a successful build, you can run the test suite. For a Cocoa build, run:

```
$ python allTests.py
```

If everything works, you should see lots of "ok" messages. In case of a failure, the tests abort with "failed".

For the iPhone or iPhone simulator, the test suite runs within an iPhone application named "Test Suite". You can run it from the Xcode project located in the `cpp/test/iPhone/container` directory. Note that the C++ tests must be run from Xcode; they won't run if you launch the Test Suite application from the device directly.

The container project has two targets:

- container, for default builds linked with `libstdc++`
- container c++11, for `CPP11=yes` builds linked with `libc++`

The architecture used to build the "Test Suite" application or the hello demo is specified with the "Architectures" build setting in the Xcode project. It must match the architecture used to build Ice Touch.

To build for a different target, you first need to clean the current build using the `clean` command. For example, if you built for Cocoa, run

```
$ make COMPILE_FOR_COCOA=yes clean
```

You can now start a new build for the desired target.

Back to Top ^

## Building the Objective-C SDKs

You can build an Objective-C SDK for each of four different targets: OS X (command line), Cocoa, iPhone, and iPhone Simulator. Change to the `IceTouch-1.3.3/objc` directory and run one of the following commands to build an SDK:

```
$ cd IceTouch-1.3.3/objc
$ make
or
$ make COMPILE_FOR_COCOA=yes
or
$ make COMPILE_FOR_IPHONE=yes
or
$ make COMPILE_FOR_IPHONE_SIMULATOR=yes
```

The first `make` command (without any arguments) builds the OS X SDK with dynamic libraries, the test suite, and the demos. The remaining `make` commands generate static libraries and build the test suite. The build configuration options can be found in `config/Make.rules`.

When building for iPhone in debug mode, make will by default build only for `armv7`. You can specify different architecture(s) with `CXXARCHFLAGS`, for example:

```
$ make COMPILE_FOR_IPHONE=yes CXXARCHFLAGS="-arch arm64"
```

Ice Touch supports `armv7`, `armv7s` and `arm64`.

The Ice Touch Objective-C SDK relies on C++ code. OS X and iOS provide two C++ run time libraries, the older `libstd++` and newer `libc++`. By default, Xcode 6 links with `libc++` when targeting OS X 10.9 or iOS >= 7, and with `libstd++` when targeting earlier versions.

The Ice Touch build system targets iOS 5.1.1 and OS X 10.7, and as a result creates libraries linked with `libstd++`. If you want to create libraries linked with `libc++`, set `CPP11` to `yes`, for example:

```
$ make COMPILE_FOR_IPHONE=yes CXXARCHFLAGS="-arch arm64" CPP11=yes
```

The resulting libraries will carry a `-libc++` suffix, such as `libIceObjC-libc++.a` or `libIceGridObjC-libc++.a`

If you built for Cocoa, you should now use Xcode to build the demos located in the following subdirectories:

- `demo/Cocoa/chat`
- `demo/Cocoa/library`

If you are building for iPhone or the iPhone simulator, you should now use Xcode to build the demos and test suite GUI located in the following subdirectories:

- `demo/iPhone/chat`
- `demo/iPhone/library`
- `demo/iPhone/hello`
- `demo/iPhone/router`
- `demo/iPhone/voip`
- `test/iPhone/container`

To compile the demos, you can also open the Xcode workspace `IceTouch-1.3.3/demos.xcworkspace`. The workspace includes the Cocoa and iPhone demo projects.

Make sure you build the appropriate SDK before opening the corresponding demo or test projects in Xcode.

After a successful build, you can run the test suite. For an OS X or Cocoa build, run:

```
$ python allTests.py
```

If everything works, you should see lots of "ok" messages. In case of a failure, the tests abort with "failed".

For the iPhone or iPhone simulator, the test suite runs within an iPhone application named "Test Suite". You can run it from the Xcode project located in the `test/iPhone/container` directory.

Back to Top ^

# Installing Ice Touch

For the OS X build, simply run `make install` to install Ice Touch in the directories specified by the `prefix` variables in `cpp/config/Make.rules` and `objc/config/Make.rules`.

For OS X builds of Ice Touch, the install name of each shared library is prefixed with `@rpath`. Link your executables with `-Wl,-rpath,/Library/Developer/IceTouch-1.3/lib` to allow the dynamic linker to locate the libraries in the Ice Touch installation directory.

If you choose not to embed a runpath into executables at build time, you will need to add the `/Library/Developer/IceTouch-1.3/lib` directory to your `DYLD_LIBRARY_PATH`.

Back to Top ^