Ice Touch 1.1.1 Release Notes

Ice Touch offers an Objective-C language mapping, an Ice run time for iOS and Mac OS X, and an Xcode SDK for Cocoa and iOS applications.

The Ice Touch distribution does not include any Ice services, but its support for the complete Ice protocol means that your Ice Touch applications can work seamlessly with existing Ice servers as well as Ice services such as IceGrid, Glacier2, and IceStorm.

On this page:

Back to Top ^

New features in Ice Touch 1.1

This section outlines changes and improvements in this release that may affect the operation of your applications or have an impact on your source code.

For a detailed list of the changes in this release, please refer to the CHANGES file included in your Ice Touch distribution.

Changes and fixes in Ice Touch 1.1.1

- Added support for Xcode 3.2, Xcode 4.0, and Xcode 4.1 to the Ice Touch Xcode plug-in.
- Updated Ice Touch to support iOS 4.3.
- Updated Ice Touch to support OS X 10.7 (Lion) and Xcode 4.1.
- Updated build system to create fat binaries with both armv6 and armv7 architectures when targeting the iOS SDK.
- Updated build system to support Xcode installed in a user-specified location.
- Added option to install the Ice Touch iOS and Cocoa SDKs in a user-specified location; this way, you can install Xcode 3.2, Xcode 4.0, and Ice Touch for both versions of Xcode on the same system.

Back to Top ^

Changes and fixes in IceTouch 1.1.0

- Replaced the AMI mapping. Applications that use the AMI mapping from Ice Touch 1.0 must be updated. Refer to the Ice manual for more information on the new mapping.
- SSL now works with the iOS simulator.
- The iOS simulator builds now use static linking.
- The Xcode plug-in now adds `-all_load` to the linker flags. See QA1490 for more information.
- The demos and tests were updated to support the iPad screen resolution.
- Ice threads are now registered with the garbage collector.

- The `ICECallbackOnMainThread` helper has been removed. The new Ice dispatcher facility should be used instead. See below for more information.
- Unmarshaling now supports size checks to prevent memory over allocations by malicious clients or servers.
- The Xcode plug-in now supports both 32- and 64-bit architectures.

Back to Top ^

Features added in Ice Touch 1.1.0

Accessory transport

Ice Touch supports a new accessory transport to enable Ice Touch applications to communicate with accessories connected to an iOS device (via USB or Bluetooth). The accessory must run an Ice server with a custom transport to allow the communication.

Back to Top ^

New AMI mapping

This release features a completely new AMI facility that allows you to structure your code with the same flexibility as the new AMI mapping introduced in Ice 3.4. Callbacks are now specified using Objective-C blocks to provide much greater flexibility.

Back to Top ^

New Dispatcher facility

In previous releases, the developer of a graphical Ice application would need to take precautions to make sure that updates to the user interface were performed in the proper thread. For example, graphical applications typically use AMI because it does not block the calling thread, but AMI callbacks are invoked from an Ice run time thread. Since the callback cannot update the user interface directly from such a thread, it is forced to schedule an update instead. With Ice Touch 1.0.0, this was facilitated with the `ICECallbackOnMainThread` helper.

Ice Touch 1.1.0 introduces the same dispatcher facility that was added in Ice 3.4. The dispatcher lets you control the thread in which servant methods and AMI callbacks are invoked. It is especially useful for a graphical application, in which you can easily install a custom dispatcher to guarantee that all of your servant and callback invocations are made in a thread that can safely update the user
interface.

For example, to execute all Ice servant methods and AMI callbacks on the main GUI thread, you just need to set up the following dispatcher when initializing a communicator:

Objective-C

```
initData.dispatcher = ^(id<ICEDispatcherCall> call, id<ICEConnection> con)
{
    dispatch_sync(dispatch_get_main_queue(), ^ { [call run]; });
};
```

This technique is demonstrated in the GUI demo applications located in the following directories:

- `demo/Cocoa`
- `demo/iPhone`

Back to Top ^

New Slice syntax for default values

It is now possible to specify in Slice the default values for data members of classes, structures, and exceptions. The semantics are the same as for Slice constants in that you can only specify default values for a data member whose type is a primitive or enumeration. For example:

Slice

```
enum Color { red, green, blue };

struct Point
{
    int x = -1;
    int y = -1;
    Color c = blue;
};
```

Back to Top ^

Support for background applications

IceTouch 1.1.0 adds support for VoIP applications as documented in Executing Code in the Background.

Follow these steps to ensure your application is correctly configured:

- Add the `voip` flag to the `UIBackgroundModes` key of your application's `Info.plist`.
- Set the configuration property `Ice.Voip` to `1` in your communicator configuration properties. As described in the Configuring Sockets for VoIP Usage, this causes the Ice run time to set the `kCFStreamNetworkServiceType` property to `kCFStreamNetworkServiceTypeVoIP` for all sockets.
- Before moving your application to the background, call the `setKeepAliveTimeout:handler:` method to specify the frequency at which your application must be woken to maintain your service.

The new sample application `demo/iPhone/voip` provides a demonstration of these steps.

Back to Top ^

Corresponding Ice release

The Slice definitions included in Ice Touch 1.1.1 are the same as the Slice definitions included in Ice 3.4.2. In particular, the Glacier 2 client library included in this Ice Touch release (libGlacier2ObjC.a, libGlacier2ObjC.11.dylib) uses the Ice 3.4.2 Glacier2 definitions. If the Glacier2 service in a future Ice release adds new APIs (such as a new operation, or a new interface, you will need to rebuild this Glacier2ObjC library using the newer Glacier2 Slice definitions to be able to use these APIs.

Upgrading your application from Ice Touch 1.1.0

Ice Touch 1.1.1 maintains binary compatibility with Ice Touch 1.1.0, therefore you may upgrade your application from Ice Touch 1.1.0 to Ice Touch 1.1.1 without recompiling your Slice files or your program code. However, if your application is linking with libIceObjC.a (a static library), you will need to re-link your application. Ice Touch supports only static-linking when building for iOS devices, the iOS simulator or Cocoa on OS X.

If you are building Ice Touch 1.1.1 from sources, you need to manually remove the Xcode plug-in installed by Ice Touch 1.1.0 prior to the Ice Touch 1.1.1 installation. To do so, remove the directory containing this plug-in, which can be either `/Developer/Library/Xcode/Plug-ins/slice2objcplugin.pbplugin` or `~/Library/Application Support/Developer/Shared/Xcode/Plug-ins/slice2objcplugin.pbplugin`.

Back to Top ^

Upgrading your application from Ice Touch 1.0.0

Xcode project settings

For Xcode iOS and Cocoa applications, you need to update the project property `ADDITIONAL_SDKS` to match the location of the new Ice Touch SDK installation. If you installed the Ice Touch 1.1.1 SDK in the default location, you would use `/Developer/SDKs/IceTouch-1.1/iphoneos.sdk` instead of `/Developer/SDKs/IceTouch-1.0/iphoneos.sdk`.

Source installation

If you are building Ice Touch 1.1.1 from sources, you need to manually remove the Xcode plug-in installed by Ice Touch 1.0.0 prior to the Ice Touch 1.1.1 installation. To do so, remove the directory containing this plug-in, which can be either `/Developer/Library/Xcode/Plug-ins/slice2objcplugin.pbplugin` or `~/Library/Application Support/Developer/Shared/Xcode/Plug-ins/slice2objcplugin.pbplugin`.

Ice Touch feature set

Ice Touch supports the following features:

- Objective-C mapping

This topic is described in the manual.

Ice Touch currently lacks support for the following Ice features:

- Asynchronous method dispatch (AMD)
- Collocation optimization

- Servant locators
- Implicit contexts
- Protocol plug-ins
- Local interfaces
- `Ice::Application` and `Ice::Service` helper classes
- `Ice::Stats` interface

Ice Touch has limited support for:

- SSL
  See below for more information.

- UDP
  On the iPhone, UDP requests do not transparently establish a 3G/Edge connection.

Back to Top ^

SSL support

Ice Touch for Mac OS X and Cocoa uses the Ice for C++ SSL protocol plug-in.

For iOS devices, Ice Touch SSL provides only a subset of this functionality. Due to limitations in iOS SSL support, the following restriction applies:

- Ice Touch servers cannot authenticate SSL clients.

Furthermore, the semantics of some IceSSL configuration properties have changed, and new properties have been added. The IceSSL property reference provides complete details.

Back to Top ^

Logger notes

Custom loggers must be installed via `ICEInitializationData`. You cannot use any of the Ice for C++ logger properties, such as `Ice.UseSyslog`, and you cannot install a custom logger with a plug-in (`Ice.Plugin.*`).

Back to Top ^

Garbage collection requirements

For Mac OS X and Cocoa, the Ice Touch run time is built with garbage collection support.

For iOS devices and the simulator, you must use explicit `retain` and `release` because garbage collection is not supported. Consequently, the Ice Touch run time for iOS cannot garbage collect graphs of objects containing cycles. Consider this Slice definition:

Slice

```
class Foo
{
Foo next;
};
```

Suppose we use these definitions as follows:

Objective-C

```
Foo a = [[Foo alloc] init];
Foo b = [[Foo alloc] init];
a.next = b;
b.next = a;
```

If you send this graph over the wire, your application will leak memory unless you somehow retain the graph and manually break the cycle.

Back to Top ^

Auto release pool

The Ice run time creates an `NSAutoReleasePool` object before each server-side dispatched invocation and client-side AMI callback. The pool is released once the dispatch is complete.

Back to Top ^

Xcode project settings

For Cocoa and iOS applications that use the Xcode SDK, you must add the following to `Additional SDKs`:

`/Developer/SDKs/IceTouch-1.1/$(PLATFORM_NAME).sdk`

In addition, when creating a new Xcode project for iOS applications, you must set the `Code Signing Resource Rules Path` to:

`$(SDKROOT)/ResourceRules.plist`

To set the path, select `Target`, press `Info`, select the `Build` tab, and press `Select`.

If you do not do this, you will receive an error when signing your application.

You must also add to the Frameworks folder:

- CFNetwork.framework
- Security.framework
- Foundation.framework
- ExternalAccessory.framework

Back to Top ^

Accessory transport

The accessory transport enables Ice Touch clients running on an iOS device to communicate with accessories connected either via Bluetooth or USB.

This transport is built on top of the iPhone OS External Accessory framework. In order to use it, proxies must use the following endpoint syntax:

`accessory [-p` *`PROTOCOL`*`] [-n` *`NAME`*`] [-m` *`MANUFACTURER`*`] [-o` *`MODELNUMBER`*`]`

For example, to invoke on a proxy for the `hello` object running on an accessory that implements the `com.zeroc.helloWorld` protocol, use the following stringified proxy:

`hello:accessory -p com.zeroc.helloWorld`

If the `-p` option is omitted, the transport will look up an accessory that supports the `com.zeroc.ice` protocol by default. This protocol string is application-defined and must match the protocol string that the accessory advertises.

In order to enable the accessory transport, Ice Touch applications must add properties to the Ice communicator initialization property set using the `ICEConfigureAccessoryTransport` method.

For example:

Objective-C

```
ICEInitializationData* initData = [ICEInitializationData initializationData];
initData.properties = [ICEUtil createProperties];
ICEConfigureAccessoryTransport(initData.properties);
communicator = [[ICEUtil createCommunicator:initData] retain];
```

This call will add the required properties and ensure that the accessory transport is linked in with your application. The project for your application will need to include the `ExternalAccessory` framework as well.

To specify that your application supports a given accessory protocol, you need to set `UISupportedExternalAccessoryProtocols` in your project `Info.plist` file as demonstrated below:

XML

```
<key>UISupportedExternalAccessoryProtocols</key>
<array>
<string>com.zeroc.helloWorld</string>
</array>
```

The iPhone hello world demo from the `demo/iPhone/hello` directory demonstrates how to configure the accessory transport.

In order to use the Ice Touch accessory transport, your accessory must also support running an Ice server that is capable of receiving Ice requests from Ice Touch over USB or Bluetooth. We can assist you with the implementation of the Ice server-side transport for your accessory. For more information on this, please contact us at info@zeroc.com.

Back to Top ^

Running iPhone tests with Xcode 3.2 & iOS 4.2

Running the iPhone test applications with Xcode 3.2 and iOS 4.2 requires that you update the "Base SDK" and "Deployment target" of the test project application; please refer to the INSTALL file included in your Ice Touch distribution for details.

Known Problems in Ice Touch 1.1.1

This section describes known problems in this Ice Touch release.

`test/Ice/operations` with Xcode 3.2 on Mac OS X

The AMI portion of the Ice operations test fails on Mac OS X when built with Xcode 3.2 with debug information for x64. While the exact cause is unknown, the error suggests a compiler or run time library issue. The same test runs successfully with Xcode 4.0 (on Mac OS X 10.6) and Xcode 4.1 (on Mac OS X 10.7).