Using Plugins with Static Libraries

When a plug-in is packaged in a static library, you need to link your application with the plug-in's static library and call a `register` function from your application to ensure the corresponding code gets correctly included.

Ice provides the following plug-in registration functions for C++ and Objective-C applications:

C++

C++

```
namespace Ice
{
    void registerIceUDP(bool loadOnInitialize = true);
    void registerIceIAP(bool loadOnInitialize = true);
    void registerIceBT(bool loadOnInitialize = true);
    void registerIceDiscovery(bool loadOnInitialize = true);
    void registerIceLocatorDiscovery(bool loadOnInitialize = true);
    void registerIceSSL(bool loadOnInitialize = true);
    void registerIceWS(bool loadOnInitialize = true);
    void registerIceStringConverter(bool loadOnInitialize = true);
}
```

Objective-C

Objective-C

```
void ICEregisterIceUDP(BOOL loadOnInitialize);
void ICEregisterIceIAP(BOOL loadOnInitialize);
void ICEregisterIceDiscovery(BOOL loadOnInitialize);
void ICEregisterIceLocatorDiscovery(BOOL loadOnInitialize);
void ICEregisterIceSSL(BOOL loadOnInitialize);
void ICEregisterIceWS(BOOL loadOnInitialize);
```

For example, if you are using a static IceSSL plug-in, link your application with `libIceSSL.a` and call `Ice::registerIceSSL` from your application. The `UDP`, `WS`, and `StringConverter` plug-ins are in the main Ice library (`libIce.a`).

When the bool parameter is `true` (the default in C++), the plug-in is always loaded (created) during communicator initialization, even if `Ice.Plugin.`*`name`* is not set. When `false`, the plug-in is loaded (created) during communication initialization only if `Ice.Plugin.`*`name`* is set to a non-empty value (e.g., `Ice.Plugin.IceSSL=1`).

`registerIceIAP` is currently available only on iOS.

The table below summarizes the available registration functions and the static libraries where they are defined:

C++

Plug-in Name	Functionality	Function	Library
IceUDP	`udp` transport	`Ice::registerIceUDP`	`libIce.a`
IceWS	`ws` and `wss` transports	`Ice::registerIceWS`	`libIce.a`
IceStringConverter	convert native string encoding to/from UTF-8	`Ice::registerIceStringConverter`	`libIce.a`
IceIAP	`iap` and `iaps` transports	`Ice::registerIceIAP`	`libIceIAP.a`
IceBT	`bt` and `bts` transports	`Ice::registerIceBT`	`libIceBT.a`
IceSSL	`icessl` transport	`Ice::registerIceSSL`	`libIceSSL.a`
IceDiscovery	Location service implemented using UDP multicast	`Ice::registerIceDiscovery`	`libIceDiscovery.a`
IceLocatorDiscovery	Location service implemented using UDP multicast	`Ice::registerIceLocatorDiscovery`	`libIceLocatorDiscovery.a`

Objective-C

Plug-in Name	Functionality	Function	Library
IceUDP	`udp` transport	`ICEregisterIceUDP`	`libIce.a`
IceWS	`ws` and `wss` transports	`ICEregisterIceWS`	`libIce.a`
IceIAP	`iap` and `iaps` transports	`ICEregisterIceIAP`	`libIceIAP.a`
IceSSL	`icessl` transport	`ICEregisterIceSSL`	`libIceSSL.a`

IceDiscovery	Location service implemented using UDP multicast	`ICEregisterIceDiscovery`	`libIceDiscovery.a`
IceLocatorDiscovery	Location service implemented using UDP multicast	`ICEregisterIceLocatorDiscovery`	`libIceLocatorDiscovery.a`

If you are developing your own plug-in, you will also need to register your plug-in factory function using `Ice::registerPluginFactory`. Refer to the Plug-in API for more information on this function.

Back to Top ^