Plug-in API

On this page:

- The Plugin Interface
- C++ Plug-in Factory
- Java Plug-in Factory
- C# Plug-in Factory

The Plugin Interface

The plug-in facility defines a local Slice interface that all plug-ins must implement:

```
Slice
module Ice {
local interface Plugin {
    void initialize();
    void destroy();
};
};
```

The lifecycle of an Ice plug-in is structured to accommodate dependencies between plug-ins, such as when a logger plug-in needs to use IceSSL for its logging activities. Consequently, a plug-in object's lifecycle consists of four phases:

Construction

The lce run time uses a language-specific factory API for instantiating plug-ins. During construction, a plug-in can acquire resources but must not spawn new threads or perform activities that depend on other plug-ins.

Initialization

After all plug-ins have been constructed, the lce run time invokes initialize on each plug-in. The order in which plug-ins are initialized is undefined by default but can be customized using a configuration property. If a plug-in has a dependency on another plug-in, you must configure the lce run time so that initialization occurs in the proper order. In this phase it is safe for a plug-in to spawn new threads; it is also safe for a plug-in to interact with other plug-ins and use their services, as long as those plug-ins have already been initialized. If initialize raises an exception, the lce run time invokes destroy on all plug-ins that were successfully initialized (in the reverse order of initialization) and raises the original exception to the application.

Active

The active phase spans the time between initialization and destruction. Plug-ins must be designed to operate safely in the context of multiple threads.

• Destruction The Ice run time invokes destroy on each plug-in in the reverse order of initialization.

This lifecycle is repeated for each new communicator that an application creates and destroys.

C++ Plug-in Factory

In C++, the plug-in factory is an exported function with C linkage having the following signature:

C++
<pre>extern "C" { [[CE_DECLSPEC_EXPORT Ice::Plugin* functionName(const Ice::CommunicatorPtr& communicator,</pre>

You can define the function with any name you wish. We recommend that you use the ICE_DECLSPEC_EXPORT macro to ensure that the function is exported correctly on all platforms. Since the function uses C linkage, it must return the plug-in object as a regular C++ pointer and not as an Ice smart pointer. Furthermore, the function must not raise C++ exceptions; if an error occurs, the function must return zero.

The arguments to the function consist of the communicator that is in the process of being initialized, the name assigned to the plug-in, and any arguments that were specified in the plug-in's configuration.

Java Plug-in Factory

In Java, a plug-in factory must implement the Ice.PluginFactory interface:

Java
package Ice;
<pre>public interface PluginFactory { Plugin create(Communicator communicator, String name, String[] args); }</pre>

The arguments to the create method consist of the communicator that is in the process of being initialized, the name assigned to the plug-in, and any arguments that were specified in the plug-in's configuration.

The create method can return null to indicate that a general error occurred, or it can raise PluginInitializationException to provide more detailed information. If any other exception is raised, the lce run time wraps it inside an instance of PluginInitializationException.

C# Plug-in Factory

In .NET, a plug-in factory must implement the Ice.PluginFactory interface:

```
C#
namespace Ice {
   public interface PluginFactory
   {
      Plugin create(Communicator communicator, string name, string[] args);
   }
}
```

The arguments to the create method consist of the communicator that is in the process of being initialized, the name assigned to the plug-in, and any arguments that were specified in the plug-in's configuration.

The create method can return null to indicate that a general error occurred, or it can raise PluginInitializationException to provide more detailed information. If any other exception is raised, the lce run time wraps it inside an instance of PluginInitializationException.

See Also

- Plug-in Configuration
- Advanced Plug-in Topics