

Ice Plug-In Properties

On this page:

- [Ice.InitPlugins](#)
- [Ice.Plugin.name.cpp](#)
- [Ice.Plugin.name.java](#)
- [Ice.Plugin.name.clr](#)
- [Ice.Plugin.name](#)
- [Ice.PluginLoadOrder](#)

Ice.InitPlugins

Synopsis

```
Ice.InitPlugins=num
```

Description

If *num* is a value greater than zero, the Ice run time automatically initializes the plug-ins it has loaded. The order in which plug-ins are loaded and initialized is determined by [Ice.PluginLoadOrder](#). An application may need to set this property to zero in order to [interact directly with a plug-in](#) after it has been loaded but before it is initialized. In this case, the application must invoke `initializePlugins` on the plug-in manager to complete the initialization process. If not defined, the default value is 1.

Ice.Plugin.name.cpp

Synopsis

```
Ice.Plugin.name.cpp=path[,version]:function [args]
```

Description

Defines a C++ [plug-in](#) to be installed during communicator initialization. The *path* and optional *version* components are used to construct the path name of a DLL or shared library. If no version is supplied, the Ice version is used. The *function* component is the name of a function with C linkage. For example, the entry point `MyPlugin,35:create` would imply a shared library name of `libMyPlugin.so.35` on Unix and `MyPlugin35.dll` on Windows. Furthermore, if Ice is built on Windows with debugging, a `d` is automatically appended to the version (for example, `MyPlugin35d.dll`).

The function must be declared with external linkage and have the following signature:

C++

```
<Plugin>* function(const Ice::CommunicatorPtr& communicator,
                  const std::string& name,
                  const Ice::StringSeq& args);
```

Note that the function must return a pointer and not a smart pointer. The Ice run time deallocates the object when it unloads the library.

Any arguments that follow the entry point are passed to the entry point function. For example:

```
Ice.Plugin.MyPlugin.cpp=MyFactory,35:create arg1 arg2
```

Whitespace separates the arguments, and any arguments that contain whitespace must be enclosed in quotes.

The *path* component may optionally contain a relative or absolute path name, indicated by the presence of a path separator (`/` or `\`). In this case, the last component of the path is used to construct the version-specific name of the shared library or DLL. Consider this example:

```
Ice.Plugin.MyPlugin.cpp=./MyFactory,35:create arg1 arg2
```

The use of a relative path means the Ice run time will look in the current working directory for `libMyPlugin.so.35` on Unix or `MyPlugin35.dll` on Windows.

If the *path* component contains spaces, the entire entry point must be enclosed in quotes:

```
Ice.Plugin.MyPlugin.cpp="C:\Program Files\MyPlugin\MyFactory,35:create" arg1 arg2
```

If the *path* component does not include a leading path name, Ice delegates to the operating system to locate the shared library or DLL, which typically means that the plug-in can reside in any of the directories in your shared library or DLL search path.

Ice.Plugin.name.java

Synopsis

```
Ice.Plugin.name.java=[path:]class [args]
```

Description

Defines a Java [plug-in](#) to be installed during communicator initialization. The specified class must implement the `Ice.PluginFactory` interface. Any arguments that follow the class name are passed to the `create` method. For example:

```
Ice.Plugin.MyPlugin.java=MyFactory arg1 arg2
```

Whitespace separates the arguments, and any arguments that contain whitespace must be enclosed in quotes.

If *path* is specified, it may be the path name of a JAR file or class directory, as shown below:

```
Ice.Plugin.MyPlugin.java=MyFactory.jar:MyFactory
Ice.Plugin.MyOtherPlugin.java=/classes:MyOtherFactory
```

If *path* contains spaces, it must be enclosed in quotes:

```
Ice.Plugin.MyPlugin.java="factory classes.jar":MyFactory
```

If *class* is specified without a path, Ice attempts to load the class using [class loaders](#) in a well-defined order.

Ice.Plugin.name.clr

Synopsis

```
Ice.Plugin.name.clr=assembly:class [args]
```

Description

Defines a .NET [plug-in](#) to be installed during communicator initialization. The *assembly* component can be a partially or fully qualified assembly name, such as `myplugin,Version=0.0.0.0,Culture=neutral`, or an assembly DLL name such as `myplugin.dll` that may optionally include a leading relative or absolute path name.

The locations that are searched for the assembly varies depending on how you define the *assembly* component:

Value for <i>assembly</i>	Example	Semantics
Fully-qualified assembly name (strong-named assembly)	<code>myplugin,Version=...,Culture=neutral,publicKeyToken=...</code>	<ol style="list-style-type: none"> 1. Checks assemblies that have already been loaded 2. Searches the Global Assembly Cache (GAC) 3. Searches the directory containing the <code>iceboxnet</code> executable

Partially-qualified assembly name	myplugin	<ol style="list-style-type: none"> 1. Checks assemblies that have already been loaded 2. Searches the directory containing the <code>iceboxnet</code> executable
Relative path name	plugins\MyPlugin.dll	Path name is relative to the application's current working directory. Be sure to include the <code>.dll</code> extension in the path name.
Absolute path name	C:\plugins\MyPlugin.dll	Assembly must reside at the specified path name. Be sure to include the <code>.dll</code> extension in the path name.

See MSDN for more information on [how the CLR locates assemblies](#).

The specified `class` must implement the `Ice.PluginFactory` interface. Any arguments that follow the class name are passed to the factory's `create` method. For example:

```
Ice.Plugin.MyPlugin.clr=MyFactory,Version=1.2.3.4,Culture=neutral:MyFactory arg1 arg2
```

Whitespace separates the arguments, and any arguments that contain whitespace must be enclosed in quotes.

If you specify a relative path name in the entry point, the assembly is located relative to the program's current working directory:

```
Ice.Plugin.MyPlugin.clr=..\MyFactory.dll:MyFactory arg1 arg2
```

Enclose the assembly's path name in quotes if it contains spaces:

```
Ice.Plugin.MyPlugin.clr="C:\Program Files\MyPlugin\MyFactory.dll:MyFactory" arg1 arg2
```

Ice.Plugin.name

Synopsis

```
Ice.Plugin.name=entry_point [args]
```

Description

Defines a [plug-in](#) to be installed during communicator initialization. The format of *entry_point* varies by Ice implementation language, therefore this property cannot be defined in a configuration file that is shared by programs in different languages. Ice provides an alternate syntax that facilitates such sharing:

- `Ice.Plugin.name.cpp` for C++
- `Ice.Plugin.name.java` for Java
- `Ice.Plugin.name.clr` for the .NET Common Language Runtime

Refer to the relevant property for your language mapping for details on the entry point syntax.

Ice.PluginLoadOrder

Synopsis

```
Ice.PluginLoadOrder=names
```

Description

Determines the order in which [plug-ins](#) are loaded. The Ice run time loads the plug-ins in the order they appear in *names*, where each plug-in name is separated by a comma or white space. Any plug-ins not mentioned in *names* are loaded afterward, in an undefined order.