Ice Plug-In Properties

On this page:

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=`*`basename`*`[,`*`version`*`]:`*`function`* `[`*`args`*`]`

Description

Defines a C++ plug-in to be installed during communicator initialization. The *`basename`* and optional *`version`* components are used to construct the 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,34:create would imply a shared library name of `libMyPlugin.so.34` on Unix and `MyPlugin34.dll` on Windows. Furthermore, if Ice is built on Windows with debugging, a `d` is automatically appended to the version (for example, `MyPlugin34d.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 `create` method. For example:

```
Ice.Plugin.MyPlugin=MyFactory,34:create arg1 arg2
```

Ice.Plugin.*name*.java

Synopsis

`Ice.Plugin.`*`name`*`.java=`*`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=MyFactory arg1 arg2
```

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 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`.

> You *must* use a fully qualified assembly name to load a plug-in from an assembly in the Global Assembly Cache.

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=MyFactory,Version=1.2.3.4,Culture=neutral: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.