

Custom String Converter Plug-ins with C++98



If the default [string converter plug-in](#) does not satisfy your requirements, you can install your own string converters with a plug-in, for example:

C++

```
class MyStringConverterPlugin : public Ice::Plugin
{
public:

    MyStringConverterPlugin(const Ice::StringConverterPtr& stringConverter,
                           const Ice::WstringConverterPtr& wstringConverter = 0)
    {
        setProcessStringConverter(stringConverter);
        setProcessWstringConverter(wstringConverter);
    }

    virtual void initialize() {}
    virtual void destroy() {}
};
```

Like in this example, you should install the string converters in your plug-in's constructor. Do not install the string converters in `initialize`.

In order to create such a plug-in, you must do the following:

- Define and export a [factory function](#) that returns an instance of your plug-in class.
- Implement the string converter(s) that you will pass to your plug-in's constructor, or use the ones [included with Ice](#).
- Package your code into a shared library or DLL.

To install your plug-in, use a [configuration property](#) like the one shown below:

```
Ice.Plugin.MyConverterPlugin=myconverter:createConverter ...
```

The first component of the property value represents the plug-in's [entry point](#), which includes the abbreviated name of the shared library or DLL (`myconverter`) and the name of a factory function (`createConverter`).

If the configuration file containing this property is shared by programs in multiple implementation languages, you can use an alternate syntax that is loaded only by the Ice for C++ run time:

```
Ice.Plugin.MyConverterPlugin.cpp=myconverter:createConverter ...
```

[Back to Top ^](#)

See Also

- [The C++98 Ice String Converter Plug-in](#)
- [Installing String Converters with C++98](#)
- [Plug-in API](#)
- [Ice.Plugin.*](#)
- [Ice.InitPlugins](#)
- [Ice.PluginLoadOrder](#)

