Custom String Converter Plug-ins with C++11

Previous Next

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

C++

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

    MyStringConverterPlugin(const shared_ptr<Ice::StringConverter>& stringConverter,
                            const shared_ptr<Ice::WstringConverter>& wstringConverter = nullptr)
    {
        setProcessStringConverter(stringConverter);
        setProcessWstringConverter(wstringConverter);
    }

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

Like in this example, you should install the string converters in your plugin'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 plugin class
- Implement the string converter(s) that you will pass to your plugin'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++11 Ice String Converter Plug-in
- Installing String Converters with C++11
- Plug-in API
- Ice.Plugin.*
- Ice.InitPlugins
- Ice.PluginLoadOrder

Previous Next