Custom String Converter Plug-ins

If the default string converter plug-in does not satisfy your requirements, you can implement your own solution with help from the `StringConverterPlugin` class:

C++

```
namespace Ice {
class StringConverterPlugin : public Ice::Plugin {
public:

    StringConverterPlugin(const CommunicatorPtr& communicator,
                          const StringConverterPtr&,
                          const WstringConverterPtr& = 0);

    virtual void initialize();

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

The converters are installed by the `StringConverterPlugin` constructor (you can supply an argument of `0` for either converter if you do not wish to install it). The `initialize` and `destroy` methods are empty, but you can subclass `StringConverterPlugin` and override these methods if necessary.

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

- Define and export a factory function that returns an instance of `StringConverterPlugin`.
- Implement the converter(s) that you will pass to the `StringConverterPlugin` 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 ...
```

See Also

- The Ice String Converter Plug-in
- Plug-in API
- Ice Plug-In Properties