Installing String Converters with C++11

Previous Next

The default behavior of the run time can be changed by providing application-specific string converters. If you install such converters, all Slice strings will be passed to the appropriate converter when they are marshaled and unmarshaled. Therefore, the string converters allow you to convert all strings transparently into their native representation without having to insert explicit conversion calls whenever a string crosses a Slice interface boundary.

You can install string converters by calling `setProcessStringConverter` for the narrow string converter, and `setProcessWstringConverter` for the wide string converter. Any strings that use the default (`std::string`) mapping are passed through the specified narrow string converter and any strings that use the wide (`std::wstring`) mapping are passed through the specified wide string converter. Passing `null` to one of these set functions resets the corresponding narrow or wide string converter to its default.

You can retrieve the previously installed string converters (or default string converters) with `getProcessStringConverter` and `getProcessWstringConverter`. The default narrow string converter is null, meaning all `std::string`s use the UTF-8 encoding. The default wide-string converter converts from UTF-16 or UTF-32 (depending on the size and endianness of the native wchar_t) to UTF-8.

The string converters are defined as follows:

C++

```
namespace Ice
{
    class UTF8Buffer
    {
    public:
        virtual Byte* getMoreBytes(size_t howMany, Byte* firstUnused) = 0;
        virtual ~UTF8Buffer() = default;
    };

    template<typename charT>
    class BasicStringConverter
    {
    public:
        virtual Byte* toUTF8(const charT* sourceStart, const charT* sourceEnd, UTF8Buffer&) const = 0;

        virtual void fromUTF8(const Byte* sourceStart,
                              const Byte* sourceEnd,
                              std::basic_string<charT>& target) const = 0;

        virtual ~BasicStringConverter() = default;
    };

    typedef BasicStringConverter<char> StringConverter;
    typedef BasicStringConverter<wchar_t> WstringConverter;

    std::shared_ptr<StringConverter> getProcessStringConverter();
    void setProcessStringConverter(const std::shared_ptr<StringConverter>&);

    std::shared_ptr<WstringConverter> getProcessWstringConverter();
    void setProcessWstringConverter(const std::shared_ptr<WstringConverter>&);
}
```

As you can see, both narrow and wide string converters are simply templates with either a narrow or a wide character (`char` or `wchar_t`) as the template parameter.

Each communicator caches the narrow string converter and wide string converter installed when this communicator is initialized.

You should always install your string converters before creating your communicator(s). When using a plugin to set your string converters, you need to set the string converters in the constructor of your plugin class (which is executed when the plugin is loaded) and not in the initialization function of the plugin class (which is executed after the communicator has read and cached the process-wide string converters).

Back to Top ^

See Also

- Communicator Initialization

- C++11 Mapping for Built-In Types

Previous Next