

Preface

My Uncle John is what my parents' generation would call "a man's man." He's tough, rugged, a bit scary, with more than a little of the cowboy in him, and he would admit to fear about as readily as I could render modest defeat. So when he described to me that the challenge in doing your second parachute jump is overcoming the fear of the known, I took note. Having now written two books, I can certainly attest to this same fear. Starting a second when you know how much suffering awaits is not something done lightly. So the question arises, why have I done so?

The reason, elucidated in the Prologue, amounts to an attempt to answer the following seemingly simple dichotomy.

- C++ is too complex.
- C++ is the only language sufficiently powerful for my needs.

One area in which this dichotomy is most pronounced is in using and, particularly, in extending the Standard Template Library (STL). This book (and its sibling, Volume 2), distills the knowledge and experience I have accumulated in tackling this challenging subject over the last decade or so.

Aims

This book describes one good way to use and extend the STL. It defines the following:

- The *collection* concept and how it differs from the *container* concept
- The *element reference category* concept, including why it's important, how it's defined, how it's detected, and the compromises it imposes on the design of STL extension collections and iterators
- The phenomenon of *external iterator invalidation* and the implications of its surprising behavior on the design of STL-compatible collections
- A mechanism for detecting features of arbitrary collections that may or may not provide mutating operations

It explains several issues:

- Why a transforming iterator adaptor must return elements by value
- Why a filtering iterator must always be given a pair of iterators to manipulate
- What to do if the underlying collection changes during iteration
- Why you should proscribe meaningless syntax for your output iterator classes and how to do so using the *Dereference Proxy* pattern

It demonstrates how to:

- Adapt *elements-en-bloc APIs* to the STL collection concept
- Adapt *element-at-a-time APIs* to the STL collection concept
- Share enumeration state in order to properly fulfill the requirements of the *input iterator* concept
- Enumerate potentially infinite collections
- Specialize standard algorithms for specific iterator types to optimize performance
- Define a safe, platform-independent STL extension for the system environment, implemented in terms of a global variable
- Adapt a collection whose iterator instances' copyability is determined at runtime
- Provide access to a reversible collection that is not repeatable
- Write into a character buffer using an iterator

Extended STL addresses these issues and more. It also looks at how general-purpose, STL-compliant libraries may be built without sacrificing robustness, flexibility, and, especially, performance. *Extended STL* teaches you how to have your abstraction cake, with efficiency cream, and eat it.

You should read this book if you want to:

- Learn specific principles and techniques for STL extension
- Learn more about the STL, by looking *inside* the implementation of STL extensions
- Learn general techniques for implementing wrappers over operating system APIs and technology-specific libraries
- Learn how to write iterator adaptors and understand the reasons behind the restrictions on their implementations and use
- Pick up techniques for optimizing the performance of general-purpose libraries
- Use proven software components for STL extension

Subject Matter

I believe that you must write about what you know. Because the main purpose of this book is to impart understanding of the process and issues of STL extension, much of the material discussed derives from my work with my own (open-source) **STLSoft** libraries. Since I've implemented just about everything in **STLSoft** from scratch, it's therefore the best material to enable me to speak authoritatively. This is especially important when discussing design errors; publicly documenting other people's design errors in detail is unlikely to achieve many positive outcomes.

But this does not mean that reading *Extended STL* obliges you to use **STLSoft** or that followers of other libraries cannot learn anything appropriate to their practice here. Indeed, rather than proselytizing the use of any particular library, the material presented gives you an inside-out look at STL extension, focusing on STL principles and extension practices, rather than relying on extant

knowledge of **STLSoft** or any other library. If, when you've read this book, you don't use the **STLSoft** libraries, I won't be troubled, so long as you've taken away useful knowledge on how to implement and use other STL extensions.

I make no pretension that the methods of STL extension I shall demonstrate in any way represent the *only* way. C++ is a very powerful language that, sometimes to its detriment, supports a variety of styles and techniques. For example, many, though not all, collections are best implemented as STL collections, while others are better represented as stand-alone iterators. There's a fair amount of overlap, about which much equivocation abides.

For most STL extensions discussed, I take the reader (and the author, in some cases!) on a journey from the raw APIs being wrapped up through intermediate, and often flawed, implementations before reaching an optimum, or at least optimal, version. I do not shy away from the implementation and/or conceptual complexities. Indeed, some of the techniques required to mold external APIs into STL form necessarily involve a degree of technical cunning. I'm not, for the sake of a simple tale, going to pretend that these things don't exist or leave them unexplained in the implementation. I will cover these things, and in so doing I hope to be able to (1) debunk some of their complexity and (2) explain why they are necessary.

One of the best ways to understand STL is to learn how STL components are implemented, and the best way to do that is by implementing them. If you don't have the time (or the inclination) to do that, I recommend that you avail yourself of the *second* best way, which is to read this book.

Structure

This book is divided into three main parts.

Part One: Foundations

This collection of small chapters provides grounding for the material discussed in Parts II and III. It begins with a brief recap of the main features of the STL, followed by a discussion of concepts and principles pertinent to STL extension, including the introduction of a new concept, the *element reference category*. The next few chapters consider foundational concepts, mechanisms, paradigms, and principles: conformance, constraints, contracts, *DRY SPOT*, RAII, and shims. The remaining chapters cover template tools and techniques, including traits and *inferred interface adaptation*, and several essential components used in the implementations described in Parts II and III.

Part Two: Collections

This represents the bulk of the book. Each chapter covers one or more related real-world collection and its adaptation into an STL extension collection component along with suitable iterator types. The subject matter tracks adaptations of subjects as diverse as file system enumeration, COM enumerators, non-STL containers, Scatter/Gather I/O, and even collections whose elements are subject to external change. The issues covered include concepts of iterator category selection and element reference categories, state sharing, mutability, and external iterator invalidation.

Part Three: Iterators

While the material in Part II includes the definition of iterator types associated with collections, Part III is devoted to stand-alone iterator types. The subjects covered range from custom output iterator types, including a discussion of simple extension of the functionality of `std::ostream_iterator`, to sophisticated iterator adaptors that can filter and transform the types and/or values of the underlying ranges to which they're applied.

Volume 2

Volume 2 is not yet complete and its structure not finalized, but it will contain material on *functions, algorithms, adaptors, allocators*, and the STL extension concepts *range* and *view*.

Supplementary Material

CD-ROM

The accompanying CD-ROM contains various free software libraries (including all those covered in the text), test programs, tools, and other useful software. Also included are three full but unedited chapters that didn't make it into print—either to save space or to avoid too much compiler specificity—and numerous notes and subsections from other chapters.

Online Resources

Supplementary material will also be available online at <http://extendedstl.com/>.