

Effective STL Programming

Duration

2 days

Instructor

Scott Meyers

Class Limit

20 students

Prerequisite

Participants should already know the basic features of the STL (e.g., containers, iterators, algorithms, and function objects), but expertise is not expected.

Price

On-site:

Please contact SPC for pricing (contact information on page 2)

Public Training:

\$995 (2 days)

*Discount available for early registration

Materials Provided

- Student manual containing the course slides
- Student handouts with class exercises and class studies

C++'s Standard Template Library is revolutionary, but learning to use it really well has, until now, been a significant challenge. This seminar, based on Scott's book, *Effective STL*, reveals the critical rules of thumb employed by the experts—the things they almost always do or almost always avoid doing—to get the most out of the library.

Intended Audience

Systems designers, programmers, and technical managers involved in the design, implementation, and maintenance of production libraries and applications using the STL. Participants should already know the basic features of the STL (e.g., containers, iterators, algorithms, and function objects), but expertise is not expected. People who have learned the STL recently (e.g., from Scott's Introductory STL course), as well as people who have been programming with it for some time, will come away from this intensive seminar with useful, practical, proven information.

Difficulty Level: Primarily intermediate, with some more elementary and some more advanced material.

Instructor

Scott Meyers is one of the world's foremost experts on C++ software development. He is the author of the best-selling *Effective C++*, *More Effective C++*, and *Effective STL*; author and designer of the acclaimed *Effective C++ CD*; a former columnist for *C++ Report*; a frequent contributor to *C/C++ Users Journal*; and a member of the Advisory Boards of several software start-ups. In addition to nearly three decades of programming experience, he has an M.S. in Computer Science from Stanford University and a Ph.D. in Computer Science from Brown University.

Scott's consulting work has spanned a wide range of industries, including CAD/CAE applications, video games, newspaper layout, medical diagnostics, and nuclear simulations.

TRAINING

Effective STL Programming

Outline

The seminar is divided into the following modules:

- *Preliminaries:* The true behavior of remove and remove-like algorithms and how this interacts with containers of pointers; the importance of equivalence, and how it differs from equality.
- *Containers:* The many technical issues that determine which container is best for the job; the folly of trying to write container-independent code; avoiding resource leaks when storing pointers; why containers should never hold auto_ptr; choosing the correct erasing option; what degree of thread safety should be expected from STL containers.
- *Vector and String:* Why vector and string are preferable to dynamically allocated arrays; the wide variety of string implementations; using vector and string with legacy APIs.

For more information on this or other SPC Springboard courses, please visit www.spcspringboard.com or e-mail SPC at info@spc.ca

Software Productivity Center
Suite 460—1122 Mainland Street
Vancouver, BC V8M 4T8
www.spc.ca

Toll Free:	Fax:
1.877.548.1948	604.689.0141
Vancouver:	Toronto:
604.662.8181	416.885.0512

- *Associative Containers:* Why comparison functions should always return false for equal values; issues involved in in-place modification of set and multiset elements.
- *Algorithms:* Choosing among sort, stable_sort, partition, stable_partition, and nth_element; understanding the algorithms that work with sorted ranges; implementing copy_if.
- *Functors, Functor Classes, Functions, etc.:* Why functor classes should be designed for pass-by-value; why predicates should be pure functions; why and how to make functor classes adaptable; why less<T> should always mean operator<.
- *Programming with the STL:* Why algorithm calls are usually preferable to handwritten loops; choosing among count, find, binary_search, lower_bound, upper_bound, and equal_range; #include-related portability headaches.
- *Allocators:* What allocators do; what they are and aren't; conventions and limitations governing their use; differences between allocators and operators new and delete; the whys and hows of rebinding; when custom allocators may make sense; using allocators to support shared memory; two designs for a complete sample allocator.

