STL: The C++ Standard Template Library

Algorithms, Containers, Iterators, Functors, Adaptors, Allocators, Performance, Internals

STL is an amazing masterpiece of software engineering. In addition to learning about a rich object collection system, developers will profit from studying STL deeply as they will learn how to put together their own modern frameworks to comprehensively tackle their specific needs.

The goal of this course is to bring C++ developers up to speed with all aspects of STL programming. We start with a tour of what STL has to offer and how it builds on some of the latest ideas in modern C++. We explore all the technical constructs defined by STL.

Iterators are used to flexibly define sequences of elements and manage the navigation among elements, without exposing the internal arrangement of elements. Algorithms are the operations we wish to perform on sequence of elements (independently of how those elements are stored). Containers (both sequence and associative) are how storage of groups of elements is managed. Adaptors provide more specialist access to containers. Allocators are used for memory layout. We also examine the very interesting STL architecture and discover how some of its design ideas used internally may be applied in designing our own class libraries.

	Contents of One-Day Training Course
Target Audience C++ developers who wish to learn about the power of STL and see how to use it aggressively in their own applications **Prerequisites** Good knowledge of the fundamentals of C++ programming, especially templates and memory management.	**STL Overview** Tour of STL capabilities Generic programming Visiting with iterators Generic containers Container-independent algorithms Application programming using STL Highly efficient and flexible solutions **C++ Review** Review of aspects of modern C++ (e.g. templates, memory) that STL leverages **Iterators** A specialist pointer to an element Category Element type & distance type Reverse/stream/insertion iterators **Algorithms** Operates on sequences Passing in sequences using begin-/end- Template-based functions to perform ops Review of available algorithms Functors as algorithm predicates **Sequence Containers** Containers are collections Ordered collection of elements deque, list, vector Random access vs. sequential access **Associative Containers** Elements and their associated keys map, set, bitset **Adaptors** Adapting a container to a specialist need e.g. look at stack/[priority|]queue interface **Allocators** Use default allocators at first, expand later Various strategies for managing blocks of memory using <memory> header **STL & Shared Libraries** Issues when passing STL collections across DLL/.so shared library boundaries Need for using same binary layout **Deploying STL** Optimizing STL use in your own projects Container selection – different containers have differing performance capabilities and differing feature sets Algorithm selection – being aware of large range of algorithms available is important **Custom** Building custom: - containers - algorithms - iterators - adaptors - allocators **Design Ideas** Review of architecture of STL Incorporating ideas from STL into your own framework designs **Internals** STL is delivered as a set of header files Exploring how it is put together More specialist functionality **Project** Building a C++ project that uses STL extensively