Username: Pralay Patoria **Book:** The C++ Standard Library: A Tutorial and Reference, Second Edition. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

6.13. Extending the STL

The STL is designed as a framework that may be extended in almost any direction.

6.13.1. Integrating Additional Types

You can supply your own containers, iterators, algorithms, or function objects, provided that they meet certain requirements. In fact, the C++ standard library lacks some useful extensions. This happened because at some point, the committee had to stop introducing new features and concentrate on perfecting the existing parts; otherwise, the job would never have been completed. That was the reason, for example, that hash tables were not part of C++98.

Useful extensions can be iterators (see Section 9.6, page 471, or Section 14.3, page 726), containers (see Section 7.10, page 385), and algorithms (for example, see Section 7.6.2, page 308, or Section 9.5.1, page 468). Note that all these extensions follow the principle of generic programming:

- Anything that *behaves* like a container *is* a container.
- Anything that *behaves* like an iterator *is* an iterator.

Thus, whenever you have a container-like class, you can integrate it into the framework of the STL by providing the corresponding interface (`begin()` , `end()` , some type definitions, etc.). If you can't add members to such a class, you can still provide a wrapper class that provides corresponding iterators.

Note, however, that some container-like objects do not fit into the concept of the STL. For example, the fact that STL containers have a begin and an end makes it hard for circular container types, such as a ring buffer, to fit in the STL framework.

Section 7.1.2, page 254, lists all common container operations and marks those that are required for STL containers. Note, however, that this doesn't mean that you have to fit in the STL framework

only if you meet *all* these requirements. It might often be enough to fulfill requirements only partially so that some but not all behavior might work. Even some standard STL containers violate STL container requirements. For example, `forward_list` s do not provide `size()` , and `array` s do not fulfill the general requirement that an STL container initialized with the default constructor is empty.

6.13.2. Deriving from STL Types

Another question is whether you can extend the behavior of STL types by deriving from them and adding behavior. However, usually that's not possible. For performance reasons, all the STL classes have no virtual functions and are therefore not equipped for polymorphism through public inheritance. To add new behavior for containers, you should define a new class that internally uses STL classes or derives privately from them.