```
        int& objId = widget.objectId(PassKey<Odb>());  // cannot call out of Odb
           // Error, PassKey<T>::PassKey() [with T = Odb] is private within
           // this context.
    }
```

Notice that use of the extended **friend** syntax to befriend a template parameter and thereby enable the `PassKey` idiom here improved the granularity with which we effectively grant privileged access to an individually named type but didn't fundamentally alter the testability issues that result when private access to specific C++ types is allowed to extend across physical boundaries; again, see *Potential Pitfalls — Long-distance friendship* below.

Curiously recurring template pattern

Befriending a template parameter via extended **friend** declarations can be helpful when implementing the curiously recurring template pattern (CRTP). For use-case examples and more information on the pattern itself, see *Appendix — Curiously Recurring Template Pattern Use Cases* on page 1042.

Potential Pitfalls

Long-distance friendship

Since before C++ was standardized, granting private access via a **friend** declaration across physical boundaries, known as long-distance friendship, was observed[6,7] to potentially lead to designs that are qualitatively more difficult to understand, test, and maintain. When a user-defined type, `X`, befriends some other specific type, `Y`, in a separate, higher-level translation unit, testing `X` thoroughly without also testing `Y` is no longer possible. The effect is a test-induced cyclic dependency between `X` and `Y`. Now imagine that `Y` depends on a sequence of other types, `C1`, `C2`, …, `CN-2`, each defined in its own physical component, `CI`, where `CN-2` depends on `X`. The result is a physical design cycle of size *N*. As *N* increases, the ability to manage complexity quickly becomes intractable. Accordingly, the two design imperatives that were most instrumental in shaping the C++20 modules feature were (1) to have no cyclic module dependencies and (2) to avoid intermodule friendships.

See Also

- "**using** Aliases" (§1.1, p. 133) describes a means to create type aliases and alias templates which can be befriended via the extended friend declarations.

[6]**lakos96**, section 3.6.1, """Long-Distance Friendship and Implied Dependency," pp. 141–144
[7]**lakos20**, section 2.6, "Component Design Rules," pp. 342–370, specifically p. 367 and p. 362