```
// client.cpp:
#include <abc_header1.h>
#include <abc_header2.h>

int function()
{
    if (abc::smart() < 0) { return -1; }  // uses smart() from abc_header1.h
    return abc::z() + abc::i() + abc::a() + abc::h() + abc::c();  // Oops!
        // Bug, silently uses the abc::i() defined in abc_header1.h
}
```

In trying to cede control to the client as to whether the declared or imported `abc::i()` function is to be used, we have, in effect, invited the defect illustrated in the above example whereby the client was expecting the `abc::i()` from `abc_header2.h` and yet picked up the one from `abc_header1.h` by default. Had the `nested` namespace in `abc_header2.h` been declared `inline`, the qualified name `abc::i()` would have automatically been rendered *ambiguous* in namespace `abc`, the translation would have failed *safely*, and the defect would have been exposed at compile time. The downside, however, is that no method would be available to recover nominal access to the `abc::i()` defined in `abc_header1.h` once `abc_header2.h` is included, even though the two functions (e.g., including their mangled names at the ABI level) remain distinct.

Potential Pitfalls

`inline`-namespace–based versioning doesn't scale

The problem with using `inline` namespaces for ABI link safety is that the protection they offer is only partial; in a few major places, critical problems can linger until run time instead of being caught at compile time.

Controlling which namespace is `inline` using macros, such as was done in the `my::VersionedThing` example in *Use Cases — Link-safe ABI versioning* on page 1067, will result in code that directly uses the unversioned name, `my::VersionedThing` being bound directly to the versioned name `my::v1::VersionedThing` or `my::v2::VersionedThing`, along with the class layout of that particular entity. Sometimes details of using the `inline` namespace member are not resolved by the linker, such as the object layout when we use types from that namespace as data members in other objects:

```
// my_thingaggregate.h:

// ...
#include <my_versionedthing.h>
// ...

namespace my
{
```