

## User-Defined Literals

## Chapter 2 Conditionally Safe Features

```

namespace ns1 // namespace containing types returned by UDL operators
{
    struct Type1 { };
    bool check(const Type1&);

    namespace literals // nested namespace for UDL operators returning ns1 types
    {
        Type1 operator""_t1(const char*);
    }

    using namespace literals; // Make literals available in namespace ns1.
}

void test1() // file scope: finds UDL operator via using directive
{
    using namespace ns1::literals; // OK, imports only the inner UDL operators
    check(123_t1); // OK, finds ns1::check via ADL
}

```

To use the `_t1` UDL suffix above, `test1` must somehow be able to find the declaration of its corresponding UDL operator locally, which is accomplished by placing the operator in a nested namespace and importing the entire namespace via a **using directive**. We could have avoided the nested namespace and, instead, required each needed operator to be imported individually:

```

namespace ns2 // namespace defining types returned by non-nested UDL operators
{
    struct Type2 { };
    bool check(const Type2&);

    Type2 operator""_t2(const char*); // BAD IDEA: not nested
}

void test2() // file scope: finds UDL operator via using declaration
{
    using ns2::operator""_t2; // OK, imports just the needed UDL operator
    check(123_t2); // OK, finds ns2::check via ADL
}

```

When multiple UDL operators are provided for a collection of types, however, the idiom of placing just the UDL operators in a nested namespace (typically incorporating the name “literals”) obviates most of the commonly cited ill effects (e.g., accidental unwanted name collisions) attributed to more general use of **using directive**. In the interest of brevity, we will freely omit the nested-literal namespaces in expository-only examples.

Finally, despite its use in the Standard for this specific purpose, there is never a need for a namespace comprising only UDLs to be declared **inline** and doing so is contraindicated; see Section 3.1. “**inline namespace**” on page 1055.