```
        return { /* ... (decode UUID expressed in canonical format) */ };
    }
}

using namespace uuid_literals;
constexpr UUIDv4 buildId = "eeec1114-8078-49c5-93ca-fea6fbd6a280"_uuid;
```

Unit conversions and dimensional units

UDLs can be convenient for specifying a unit name on a numeric literal, providing a concise way both to convert the number to a normalized unit and to annotate a value's unit within the code. For example, the standard trigonometric functions all operate on **double** values where angles are expressed in radians. However, many people are more comfortable with degrees than radians, especially when expressing the value directly as a handwritten number:

```
#include <cmath>  // std::sin, std::cos

constexpr double pi = ~~3.14159265358979311599796346854;~~

double s1 = std::sin(30.0);    // Bug, intended sin(30 deg) but got sin(30 rad)
double s2 = std::sin(pi / 6);  // OK, returns sin(30 deg)
```

The normalized unit in this case is a radian, expressed as a **double**, but radians are generally fractions of π and are thus inconvenient to write. UDLs can provide convenient normalization from degrees or gradians to radians:

```
namespace trig_literals {

constexpr double operator""_rad(long double r)  { return r; }
constexpr double operator""_deg(long double d)  { return pi * d / 180.0; }
constexpr double operator""_grad(long double d) { return pi * d / 200.0; }

}

using namespace trig_literals;
double s3 = std::sin(30.0_deg);     // OK, returns sin(30 deg)
double s4 = std::sin(4.7124_rad);   // OK, returns approx -1.0
double s5 = std::cos(50.0_grad);    // OK, returns cos(50 grad) == cos(45 deg)
```

Unfortunately, the applicability of the above approach to unit normalization is limited. First, the conversion is one way — e.g., the expression `std::cout << 30.0_deg` will print out `0.524`, not `30.0`, necessitating a call to a radians-to-degrees conversion function when a human-readable value is desired. Second, a **double** does not encode any information about the units that it holds, so **double** `inputAngle` doesn't tell the reader (or program) whether the angle is expected to be input in degrees or radians.