rules.sonarsource.com

C++ static code analysis: Scoped enumerations should be used

2-3 minutes

There are two kinds of enumeration:

- The unscoped `enum` inherited from C
- The scoped enumeration `enum class` or `enum struct` added in C++ 11

  Unscoped enumerations have two major drawbacks that are fixed by scoped enumerations:

- `enum` elements are visible from their enclosing scope, instead of requiring the scope resolution operator (ex: `Red` instead of `Color::Red`)
- `enum` elements convert implicitly to `int`, so that heterogeneous comparisons such as `Red == Big` don't result in compile errors.

This rule raises an issue when an unscoped enumeration is used.

Noncompliant Code Example

```
enum Color { // Noncompliant; replace this "enum" with "enum
class".
  Red   = 0xff0000,
  Green = 0x00ff00,
  Blue  = 0x0000ff
};

enum ProductType { // Noncompliant; replace this "enum" with
"enum class".
  Small   = 1,
  Big     = 2
};

void printColor(int color);
void printInt(int value);

void report() {
  printColor(Red); // correct
  printColor(Big); // clearly buggy
  printInt(Red);   // conversion is implicit
}
```

Compliant Solution

```
enum class Color { // declared using "enum class"
  Red   = 0xff0000,
  Green = 0x00ff00,
  Blue  = 0x0000ff
};

enum class ProductType { // declared using "enum class"
  Small   = 1,
  Big     = 2
};

void printColor(Color color); // requires "Color" instead of "int"
void printInt(int value);

void report() {
  printColor(Color::Red);       // correct
  // printColor(ProductType::Big); => Compilation error, no known
conversion from 'ProductType' to 'Color'
  printInt(static_cast<int>(Color::Red)); // conversion never occurs
implicitly and must be explicit
}
```

Exceptions

When the enum is a private member of a class, its use is encapsulated by the class and the drawbacks of unscoped enums can be avoided. Therefore, no issue will be raised in that case.

See

- C++ Core Guidelines Enum.3 - Prefer class enums over “plain” enums