D1771r1 - [[nodiscard]] for constructors

Peter Sommerlad

2019-07-18

Document Number:	D1771r1
Date:	2019-07-18
Project:	Programming Language C++
	Programming Language Vulnerabilities C++
Audience:	EWGI/EWG/CWG

1 Introduction

The paper p0189 that introduced the `[[nodiscard]]` attribute did not consider constructors. However, gcc for example implements the checking for constructors, even so it warns about putting `[[nodiscard]]` on a constructor definition. Here I propose to allow `[[nodiscard]]` also on constructors (which it implicitly is allowed by the current wording) and suggest checking it for cast expressions so that we can put it on things like `scoped_lock` etc.

The need is more obvious in C++ 17 and later, where CTAD allows for fewer factory functions and thus the easy to make mistake by just typing the type and constructor arguments instead of defining a local variable.

Since this change is editorial only, it might be considered to be applied for the current working paper.

R1 of this paper extends the example to demonstrate the added cases and integrates feedback by CWG in Cologne 2019.

Reviewers, please note that a constructor declaration is a function declaration.

Thanks to EWG, CWG and Alisdar Meredith for helping with this paper and giving feedback.

2 Wording

The following changes are relative to n4820.

Change section [dcl.attr.nodiscard] as follows.

2.0.1   Nodiscard attribute   [dcl.attr.nodiscard]

1   The *attribute-token* `nodiscard` may be applied to the *declarator-id* in a function declaration or to
    the declaration of a class or enumeration. It shall appear at most once in each *attribute-list* and no
    attribute-argument-clause shall be present.

2   [*Note*: A *nodiscard type* is a possibly cv-qualified class or enumeration type marked `nodiscard`.
    A *nodiscard call* is a function call expression (7.6.1.2 [expr.call]) that calls a function previously
    declared `nodiscard`, or whose return type is a ~~possibly cv-qualified class or enumeration type~~
    ~~marked nodiscard~~ *nodiscard type*. Also, a *nodiscard call* is an explicit type conversion (7.6.1.8
    [expr.static.cast], 7.6.3 [expr.cast], 7.6.1.3 [expr.type.conv]) that constructs an object through a
    constructor previously declared `nodiscard`, or that initializes an object of a *nodiscard type*.

    Appearance of a nodiscard call as a potentially-evaluated discarded-value expression (7.2)is discour-
    aged unless explicitly cast to `void`. Implementations should issue a warning in such cases. This is
    typically because discarding the return value of a nodiscard call has surprising consequences. — *end
    note*]

3   [*Example*:

```
struct [[nodiscard]] my_scopeguard { /* ... */ };
struct my_unique {
  my_unique() = default;  // does not acquire resource
  [[nodiscard]] my_unique(int fd) { /* ... */ } // acquires resource
  ~my_unique() noexcept { /* ... */ } // releases resource, if any
  /* ... */
};

struct [[nodiscard]] error_info { /* ... */ };
error_info enable_missile_safety_mode();
void launch_missiles();
void test_missiles() {
  my_scopeguard();                  // warning encouraged
  void(my_scopeguard()),            // warning not encouraged, cast to void
    launch_missiles();              // comma operator, statement continues
  my_unique(42);                    // warning encouraged
  my_unique();                      // warning not encouraged
  enable_missile_safety_mode(); // warning encouraged
  launch_missiles();
}
error_info &foo();
void f() { foo(); }               // warning not encouraged: not a nodiscard call, because neither
                                  // the (reference) return type nor the function is declared nodiscard
```

    — *end example*]