

Doc Number: X3J16/92-0085  
 WG21/N0162  
 Date: August 4, 1992  
 Project: Programming Language C++  
 Ref Docs: X3J16/92-0060, 92-0064  
 Reply to: Samuel C. Kendall  
 CenterLine Software, Inc.  
 kendall@centerline.com

## Qualifiers in Overriding Return Types of Virtual Functions

*Samuel C. Kendall*

### 1. Introduction

We propose first an editorial change, then a small extension. The extension subsumes the editorial change.

The extension is to allow the return type of the derived virtual member function to omit qualifiers that are in the return type of the base virtual member function. This extension is type-safe, simple to implement, and useful.

### 2. Editorial Change Proposal

The current WP (working paper), 92-0060, does not allow the following example:

```
class A {
public:
    virtual const A* ro_clone();
};
class B : public A {
public:
    const B* ro_clone(); // error according to 10.2
};
```

This should be allowed. The problem is that qualifiers are not discussed in §10.2.

The fix: In §10.2p1, CHANGE the sentence:

It is an error for the return type ....

TO (new words are in boldface):

It is an error for the return type of an overriding function to differ from the return type of the overridden function unless the return type of the overridden function is a pointer or reference to a **possibly-qualified** class B, and the return type of the overriding function is a pointer or reference (respectively) to an **identically-qualified** class that is either B or publicly derived (directly or indirectly) from B.

This change is similar to editorial changes I proposed (on x3j16-edit) in §§4.6p1, 4.7p1, 13.4p6, 13.4.6p1, and 13.4.7p1.

## 3. Extension Proposal

### 3.1 Introduction

The extension is to allow the return type of the derived virtual member function to omit qualifiers that are in the return type of the base virtual member function. For example, the extension would permit the following code:

```
class A;
class B {
public:
    virtual const A* f();
    virtual volatile A* g();
};
class D : public B {
    A* f();
    A* g();
};
```

### 3.2 Rationale

#### 3.2.1 Implementation

The code generation and run-time aspects of this extension are trivial to implement, since the representation of an A\* is identical to that of a const A\* or a volatile A\* on most (probably all) systems.

Of course, in the front end there is a slight additional complication (in order to implement the revised §10.2p1). I think this cost is negligible.

#### 3.2.2 Type-Safety

The extension is type-safe. We argue this first for the const qualifier, then for the volatile qualifier. For these cases consider the code above along with the following function calls:

```
B* bp;
const A* cap = bp->f();
volatile A* vap = bp->g();
```

For the const qualifier: the caller expects a const A\* but D::f returns an A\*. The const qualifier means that the caller is (more or less) promising not to modify the pointed-to object. D::f returns a pointer to an object that can be modified, but there is no conflict; the caller will not modify the object even though it is allowed to.

For the volatile qualifier: the argument is similar to that for const. The caller expects a volatile A\* but D::g returns an A\*. The 'volatile' qualifier means that the caller is (more or less) promising to avoid certain optimizations in accessing the pointed-to object. D::g returns a pointer to an object that can be accessed with those optimizations, but there is no conflict; the caller will not perform the optimizations even though it is allowed to.

More pedantically, we note that A is a subtype of const A. (A subtype, but not a subclass.) The same can be argued of A and volatile A, though I omit the argument here.

#### 3.2.3 Utility

All the examples in 92-0064 are of clone-like functionality. Here I introduce a different idiom: that of a hierarchy of interface classes. A derived interface class offers more access than the base interface class (this is a form of progressive disclosure). One kind of "more access" is to allow writing.

For example:

```

class car;
class engine;
class normal_interface {
public:
    normal_interface(car*);
    const engine* get_engine();
    ...;
};
class mechanic_interface : public normal_interface {
public:
    mechanic_interface(car*);
    engine* get_engine();
    ...;
};

```

`normal_interface` and `mechanic_interface` are interfaces to a car. A normal interface allows inspection, but not modification, of the engine. A mechanic interface allows modification as well.

The cast for `const`, then, is straightforward.

The case for `volatile` is less compelling, simply because `volatile` is not as commonly used. But it should be allowed for symmetry.

### 3.3 Proposed Wording for the Working Paper

In §10.2p1, CHANGE the sentence

It is an error for the return type ....

TO (new words are in **boldface**):

It is an error for the return type of an overriding function to differ from the return type of the overridden function unless the return type of the overridden function is a pointer or reference to a **possibly-qualified** class B, and the return type of the overriding function is a pointer or reference (respectively) to an **identically-qualified or less-qualified** class **that is either B or publicly derived (directly or indirectly) from B.**

The term *less-qualified* must be defined somewhere. A type T is less qualified than a type U if T's qualifiers are a proper subset of U's qualifiers.

## 7. Meeting Schedule

Date	Location	National Standards Bodies	Corporate Sponsors
Nov 1-6, 1992	Boston, MA	ANSI	OSF
Mar 7-12, 1993	Portland, Oregon	ANSI	Mentor Graphics Sequent Computer, and Tektronix
Jul 11-16, 1993	Munich, Germany	DIN	Siemens Nixdorf
Nov 7-12, 1993	Asilomar, CA or Palo Alto, CA	ANSI	Apple or Taligent
Mar 6-11, 1994	San Diego, CA	ANSI	Taumatic

## 8. Issues for Discussion

### 8.1 Standards Development Procedures

WG21 feelings are strongly against approval of "*Procedures for accelerating the development of Standards*", JTC1/SC22/N118 and WG21 has endorsed the convener's comments, WG21/N0127 also known as SC22/N1161. In addition, WG21 has requested that the convener identify this issue as a point for discussion before SC22.

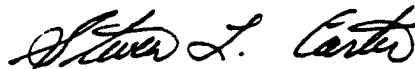
### 8.2 Language Independent Standards Conformance

WG21 is of the opinion that conformance to language independent standards needs to be defined. This could entail answering the following questions:

- a. What does conformance mean? Is it providing one or more appendices describing differences and similarities between the language standard and the several language independent standards? Is it implementing the specifications of the several language independent standards and, if so, which ones or all, and are they of the language development group's choosing or designated by SC22?
- b. When must language independent standards conformance be completed by language development working groups?
- c. Which languages must conform to language independent standards? Is it up to the language development working groups or will the languages be designated by SC22?

## 9. Acknowledgements

I offer special thanks to Dan Saks, WG21 and X3J16 Secretary, for providing superb minutes, without which, I could not have provided this summary.



Steven L. Carter  
Convener  
WG21, C++ Working Group