1 Introduction

```c
int a;
int &volatile x = a;
int &const y = a;
```

The paper X3J16/93-0135 = WG21/N0342 (section 7) proposes to ban the `volatile` qualifier on reference types, arguing that it would be difficult to make sense of the resulting semantics. Further discussion in San Jose revealed a similar sentiment for `const`-qualified reference types. I argue that both should be allowed (as per the current WP).

2 `volatile`-qualified references

2.1 The definition of `volatile`

```c
union {
    void *p;
    int &volatile r;
}
```

My best argument for allowing the `volatile` qualifier on reference types is the following definition, from the WP (3.6.3 CV-qualifiers)

> There are two cv-qualifiers, `const` and `volatile`. When applied to an object, `const` means the program may not change the object, and `volatile` has an implementation-defined meaning.

Why should we restrict the contexts in which implementations may choose to provide semantics for `volatile`?

2.2 When `volatile`-qualified references might make sense

Consider the following example:

```c
union {
    void *p;
    int &volatile r;
}
```

The `volatile` qualifier is used here to give this implementation a hint that it shouldn't cache the address of the referenced object. Why not allow an implementation to support this?

2.3 Orthogonality

Shouldn't `const` and `volatile` be either both allowed or both banned?
3 const-qualified references

3.1 Innocent introduction of const-qualified references

(1) The following example shows that templates can sometimes cause const-qualified references to be introduced. Disallowing const-qualified references in the language forces the template class to be rewritten just to support the use of reference types.

```cpp
template<class T> class C {
    const T a;
};
typedef int &intr;
C<intr> x;
```

(2) Notice that “C<intr>::a” is of type “int & const”, so this whole program would be ill-formed if const-qualified references were disallowed. Where did the poor user go wrong?

(3) Nowhere, I claim.

3.2 Possible extension

(1) Why should a user want to const-qualify a reference? After all, the reference can only be initialized once anyway; specifying const is therefore redundant.

(2) Allowing const-qualified references certainly harms nothing. But it proves meaningful if we (or some implementation) extends the language to support re-binding references:

```cpp
int x, y;
int &r = x; // bind r to x
r := y; // rebind r to y
int & const s = x; // bind s to x
s := y; // error: can’t rebind a const reference
```

(3) I’m certainly not proposing this extension. But the idea of this extension shows that allowing const-qualified references makes the language more orthogonal, while costing nothing.