

C++ Tip 0001 — Defining Static Data Members

© 2009 Rex Jaeschke. All rights reserved.

Issue: The syntax for declaring a static data member inside a class, and defining it is different; one *needs* the keyword `static` while the other must *not* have it. This is confusing.

Response: Let's consider the case in which a `Message` class maintains a count of the messages created, and uses each value of that count as a unique message ID. The relevant part of the class definition (in `Message.h`, for example) is, as follows:

```
class Message
{
    string messageText;
    unsigned long int messageID;

    static unsigned long int lastIssuedID;
    static unsigned long int getNextID() { return ++lastIssuedID; }
    // ...
};
```

And the corresponding definition of `lastIssuedID` (in `Message.cpp`, for example) is, as follows:

```
unsigned long int Message::lastIssuedID = 0;
```

The declaration of `lastIssuedID` in the class *must* contain the keyword `static`, as that is what makes the member a property of the class rather than a property of any instance of that class.

In C/C++, an external definition can contain the keyword `static`; however, the meaning of that is completely unrelated to the use of `static` in the class context. According to C (of which C++ is, for the most part, a proper superset), an identifier declared at file scope using `static` has internal linkage. That is, such a name is visible only from within its own source file, and would not be exported by the linker. On the other hand, the application of `static` on a class member implicitly requires that member's name to be global; that is, as having external linkage. Therefore, `static` is *required* on the declaration in the class but is *not permitted* in the definition.

Note that the explicit initialization to zero is redundant, as that is the default initial value for a global integer.