In a C++ class template specialization, how can I call the unspecialized version of a method?

devblogs.microsoft.com/oldnewthing/20250116-00

January 16, 2025

Suppose you have some class

```
template<typename T>
struct Producer
{
    T Produce();
};
```

and suppose you want to specialize it for one particular type, but you want your specialization to call the unspecialized version too.

```
template<>
struct Producer<void>
{
    void Produce()
    {
        Log("Producing void!");
        // I want to call Producer<T>::Produce()
        // as if this specialization didn't exist
        Producer::Produce(); // this doesn't work
    }
};
```

The idea is that you want the specialized `Producer<void>::Produce()` method to behave the same as an unspecialized `Producer<void>::Produce()`, but do a little extra logging.

What is the magic syntax for calling the unspecialized version of a template method?

This is a trick question. There is no magic syntax for this. There is no way to talk about a template that doesn't exist but "would exist" if a specialization were not present. Template specialization is not derivation. Specializing a class template means "For this case, I want the class to look like this." It is not an override or overload or derivation; it is a definition. The

definition of `Producer<void>` is the class you defined. There is no syntax for saying, “Give me `Producer<void>` as it would have been in the absence of this specialization,” and your redefinition does not implicitly derive from anything.

There are number of possible workarounds to this.

Probably the most common one is to move everything to a base class, have the primary template derive from the base class, and then override the method in the specialization.

```
template<typename T>
struct ProducerBase
{
    T Produce();
};

template<typename T>
struct Producer : ProducerBase<T>
{
};

template<>
struct Producer<void> : ProducerBase<void>
{
    using ProducerBase = typename Producer::ProducerBase;

    void Produce()
    {
        Log("Producing void!");
        ProducerBase::Produce();
    }
};
```

You might be in a case where the `Producer` class comes from a library you do not control. In that case, you can still follow the pattern, but using the library class as the base class.

```
template<typename T>
struct Producer
{
    T Produce();
};

template<typename T>
struct MyProducer : Producer<T>
{
};

template<>
struct MyProducer<void> : Producer<void>
{
    using Producer = typename MyProducer::Producer;

    void Produce()
    {
        Log("Producing void!");
        Producer::Produce();
    }
};

// and your code uses MyProducer everywhere
```