Virtual Destructors for Interfaces

- author: Mike Griese **migrie**
- created on: 2019-Feb-20

As you look through the code, you may come across patterns that look like the following:

```
class IRenderData
{
public:
    virtual ~IRenderData() = 0;
    // methods
};

inline IRenderData::~IRenderData() {}
```

You may ask yourself, why is the destructor deleted, then later defined to the default destructor? This is a good question, because it seems both unintuitive and unnecessary. However, if you don't define your interfaces exactly like this, then sometimes on object destruction, the interface's dtor will be called instead of the destructor for the base class. There is other strangeness that can occur as well, the details of which escape my memory from when @austdi and I first investigated this early 2018.

The end result of not defining your interfaces exactly like this will be that occasionally, when destructing objects, you'll get a segfault.

To check that this behavior works, I direct your attention to the VtIoTests. There are a bunch of tests in that module that create objects, then delete them, to make sure that they won't ever crash.