When APIs Break, What Can You Do

Billy Baker
billy.baker@flightsafety.com
FlightSafety International Simulation Systems

May 13, 2015

The setup (protecting the names of the innocent)

```
typedef char X;
template<typename T>
struct bar {
   typedef T value_type;
};
struct foo {
   typedef X value_type;
   typedef bar<value_type> Y;

   operator Y();
   bar<char> bar();
};
```

The setup (protecting the names of the innocent)

```
typedef char X;
template<typename T>
struct bar {

    typedef T value_type;
};

struct foo {

    typedef X value_type;

    typedef bar<value_type> Y;

    operator Y();
    bar<char> bar();
};
```

```
typedef wchar_t X;
template<typename T>
struct bar {

    typedef T value_type;
};

struct foo {

    typedef X value_type;

    typedef bar<value_type> Y;

    operator Y();
    bar<char> bar();
};
```

What happens

- `void g(const bar<char>& b);`
- `typedef char X;`
  - `g(foo());` // compiles and uses operator Y
- `typedef wchar_t X;`
  - `g(foo());` // doesn't compile

What are we really talking about

- `typedef char X;`
  - Filesystem V2 path
    - string() and operator string_type() return same type but different formatted values
- `typedef wchar_t X;`
  - Filesystem TS/V3 path
    - string() and operator string_type() return different types but same formatted values

SFINAE to the rescue(?)

```
template<typename T>
inline typename std::enable_if<
         std::is_same<typename T::value_type, wchar_t>::value,
         std::string>::type
 to_string(const T& from) {
         return narrow(std::wstring(from).c_str());
}

template<typename T>
inline typename std::enable_if<
         std::is_same<typename T::value_type, char>::value,
         std::string>::type
 to_string(const T& from) {
         return from;
}
```

A little more on path API

	V2 (VS2012, VS2013, older Boost)	TS/V3 (VS2015, newer Boost)	
path	template	non-template	
path::filename()	Returns string_type	Returns path	to_string(p.filename())
path::basename()	Valid	Does not exist	use path::stem instead
path::stem()	Returns string_type	Returns path	to_string(p.stem())
path::file_string()	Returns string_type in native format	Does not exist	to_string(p)
path::operator string_type	Returns string_type (std::string) in native format	Returns sting_type (std::wstring) in native format	to_string(p)
path::string()	Returns std::string in pathname grammar	Returns std::string in native format	to_string(p)

where p is a path and to_string always returns a std::string in native format

Just add a compatibility layer

- Danny Dig, Stas Negara, Vibhu Mohindra, and Ralph Johnson. 2008. *ReBA*: *re*factoring-aware *b*inary *a*daptation of evolving libraries. In *Proceedings of the 30th international conference on Software engineering* (ICSE '08). ACM, New York, NY, USA, 441-450.