Library In A Week – Day 3

Revamping C++ I/O

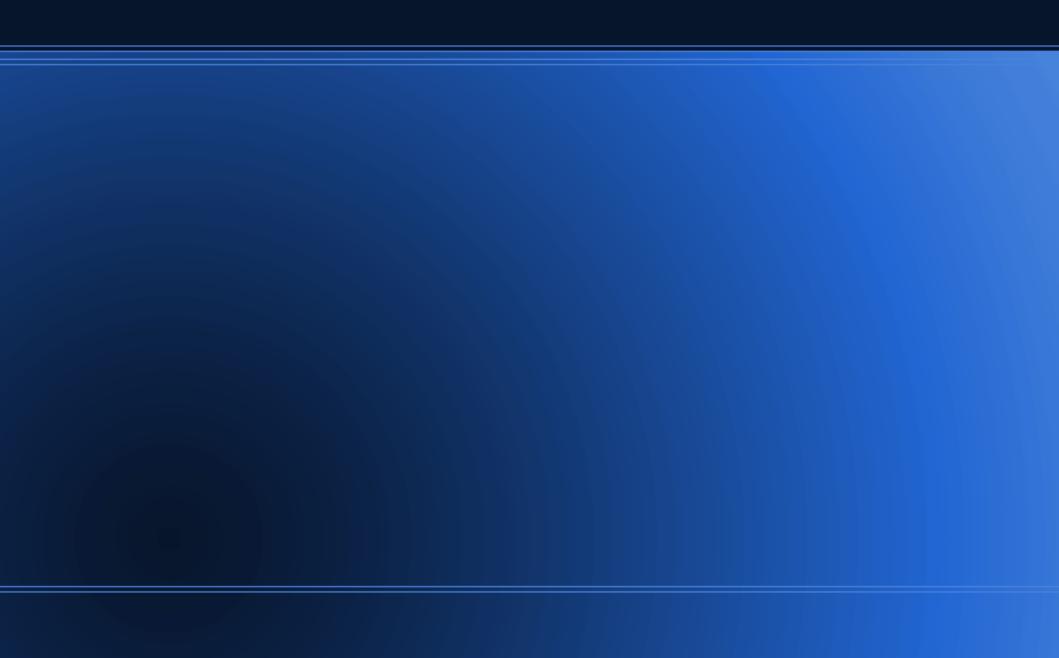
Jeff Garland

jeff@crystalclearsoftware.com

ReCap / Agenda

- Split into 2 groups
 - Starting with iostream / asio
 - Functional approach
- Solutions
 - Krishna Experience re-writing
 - Sebastian iochain
 - Jeff More on boost::iostreams
- Discussion

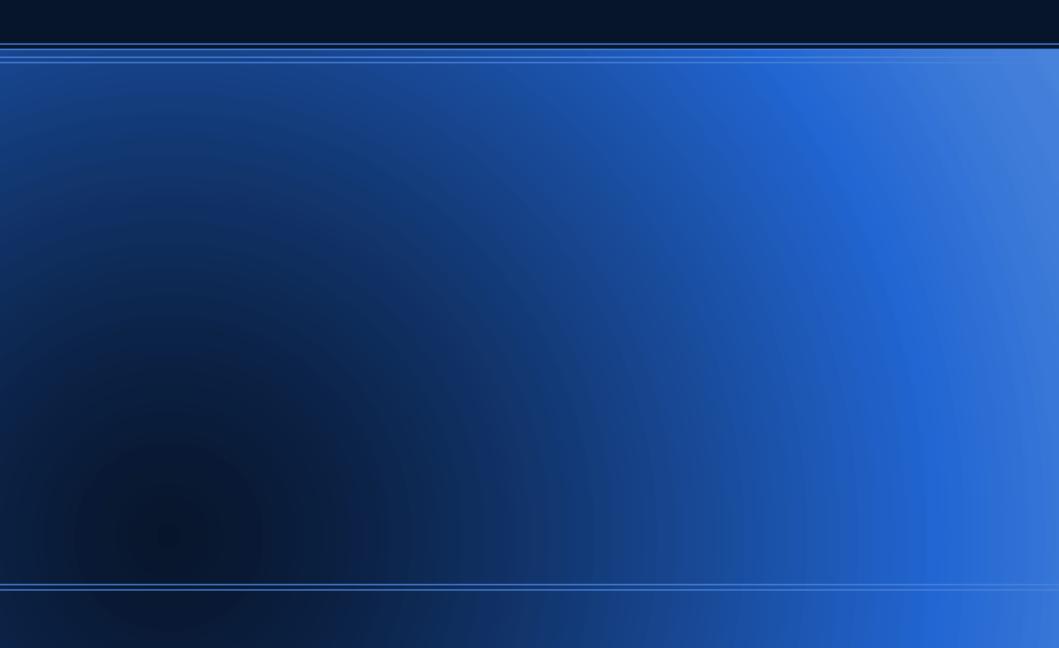
More boost::iostream examples



boost::iostreams – repimlement base of stream?

- The template stream derives from a specialization of std::basic_istream, std::basic_ostream or std::basic_iostream, depending on whether the underlying Device models Source, Sink or both..
- Couldn't we just redo the interface of stream to support needed ios capabilities?

chaining



Compatibility Layer

- What are the requirements?
 - Ideally users won't have to rewrite all code
 - Needing to modify code ideally compatible with new and old
- What are the approaches?
 - Can't put things in name space std, so will need to rename

Compatibility Example

```
struct foo
foo(int field1, double field2) :
 f1(field1),
     f2(field2)
{}
int f1;
double f2;
};
```

Compatibility Example

```
std::ostream&
                                template<StreamType>
operator<<(std::ostream& os, StreamType&
                                operator<<(StreamType& os,
  const foo& f)
                                const foo& f)
   os << f.f1
                                   os << f.f1
      << "I" "I"
                                      << "I" "I"
      << f.f2;
                                      << f.f2;
   return os;
                                   return os;
```

A 'Real' Extraction Operator

```
template <class CharT, class TraitsT>
inline std::basic ostream<CharT, TraitsT>&
operator<<(std::basic ostream<CharT, TraitsT>& os, const boost::gregorian::date& d) {
  boost::io::ios flags saver iflags(os);
  typedef boost::date_time::date_facet<date, CharT> custom_date_facet;
  std::ostreambuf iterator<CharT> output itr(os);
  if (std::has_facet<custom_date_facet>(os.getloc()))
    std::use facet<custom date facet>(os.getloc()).put(output itr, os, os.fill(), d);
  else {
    custom date facet* f = new custom date facet();
    std::locale 1 = std::locale(os.getloc(), f);
    os.imbue(1);
    f->put(output itr, os, os.fill(), d);
  return os;
```

Compatibility Approaches

- Real need is to support existing paradigm for user code
- Ignore basic_classes, provide only equivalents for *stringstream, *fstream,
- What about the 'global objects'
 - Perhaps better approach is to export the global buffer interface for cout, clog, etc.