Writing applications in modern C++ and Qt

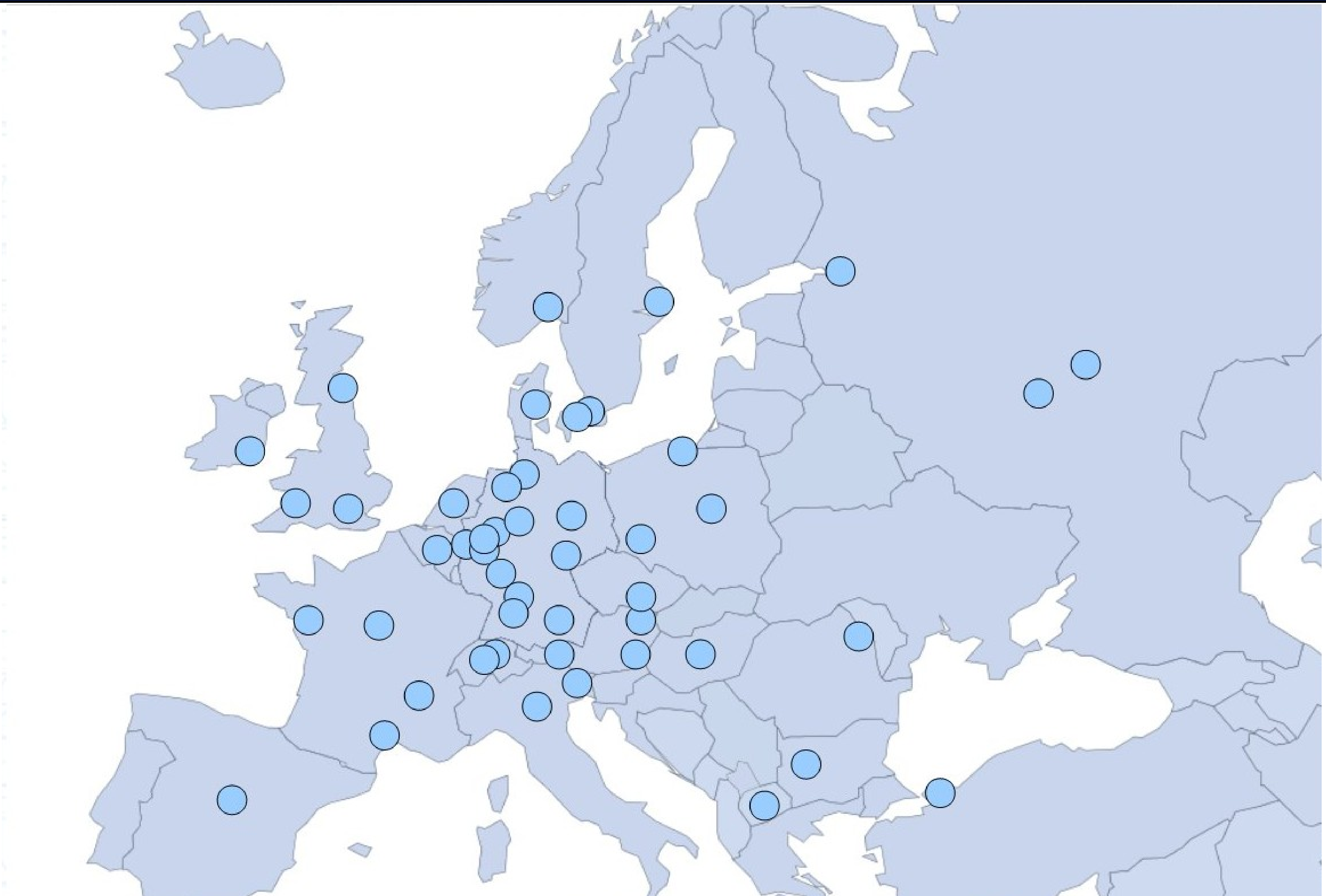
Jens Weller

CppCon 2016
@meetingcpp
info@meetingcpp.com

Introduction

- * '81
- C++ since '98
- was a C++ Freelancer
- Meeting C++
- C++ Evangelist
  - Supporting C++ UGs
  - Social Media
  - global Network for C++

User Groups in Europe



Lets get started!

- What I did
  - started writing a CMS
    - in C++
    - for static websites
  - other goals
    - use Modern C++
    - combine it with Qt
    - templates
    - reusability

- This Talk
  - isn't about
    - writing a CMS
    - Qt introduction
  - Focus
    - Modern C++
    - patterns in Qt
    - combining both

Modern C++

- What is it exactly?
  - a[n old] book?
  - a new book?
  - C++11/14/17?
  - boost?
  - buzz word?

- IMHO
  - depends largely
    - your audience
    - your own background
- Using C++ to its fullest
  - right tools for the right job

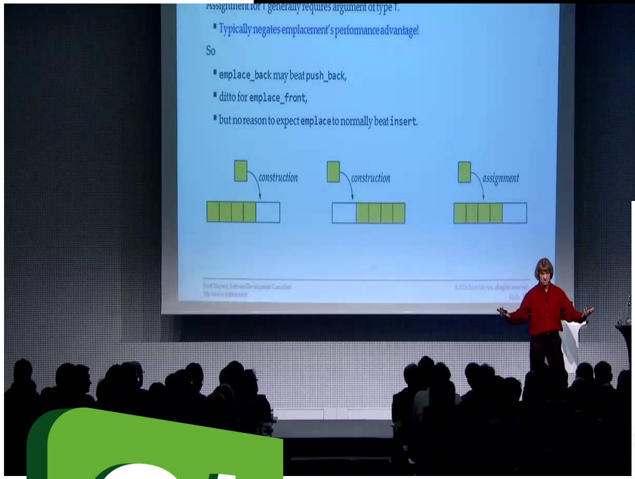
Modern C++

- As defined by Andrei Alexandrescu

  My understanding is that the book "Modern C++ Design" coined the term "modern C++".

  The term refers to a template-intensive, generic style of writing code.

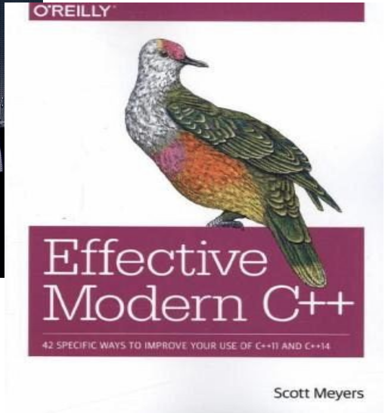
Goals of good C++



Typically negates emplacement's performance advantage!

So

- emplace_back may beat push_back,
- ditto for emplace_front,
- but no reason to expect emplace to normally beat insert.



- less Code
- yes to templates
  - generic code
  - re-usability
- static polymorphism
  - variant
- C++ Standard
  - old AND new

Qt design principles

- Lars Knolls Keynote

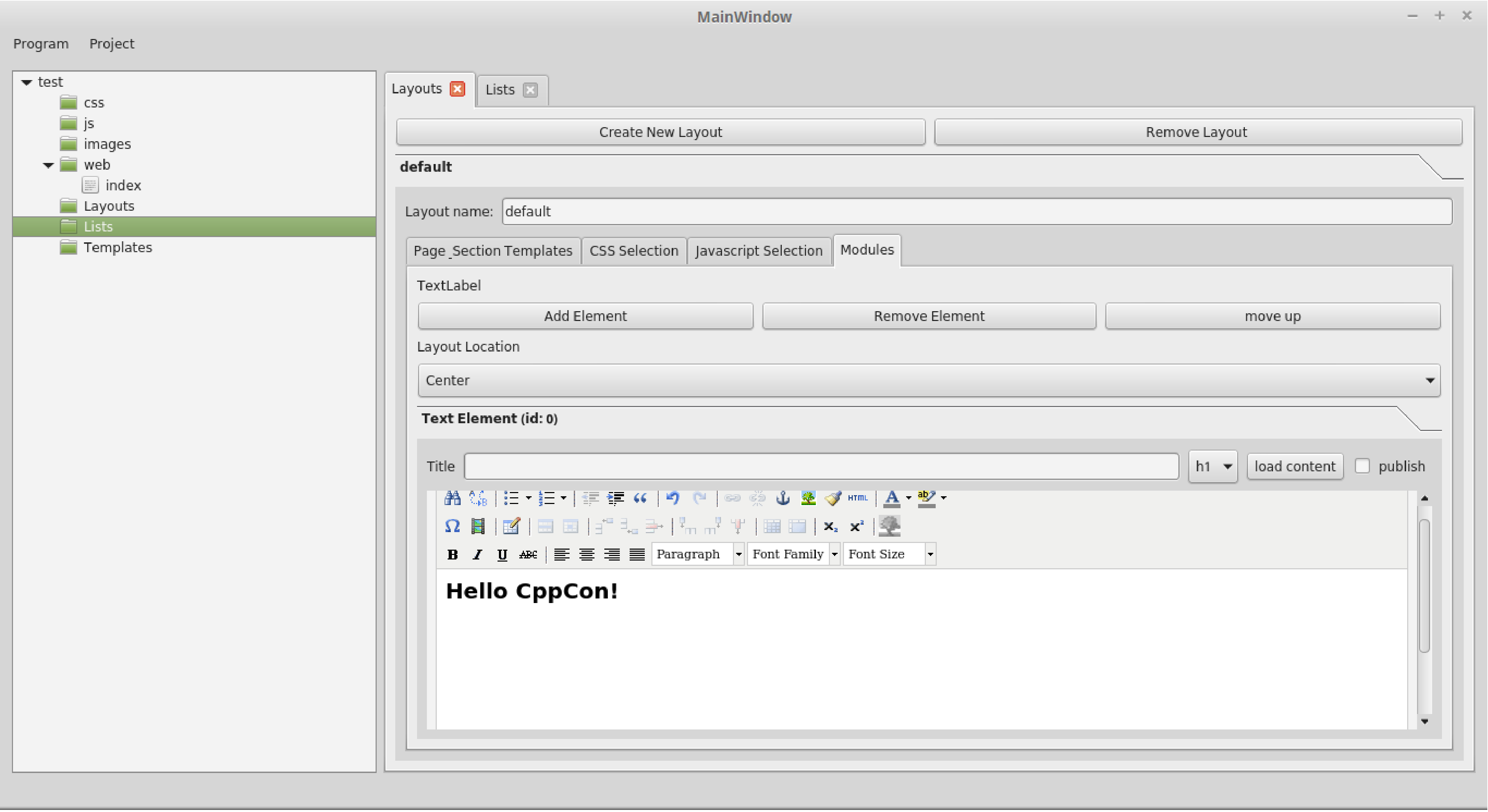
Creating intuitive APIs

Meeting C++
5.12.2015

Lars Knoll
CTO, Qt Chief maintain

- “Qt is the JDK of C++”
- Heavy focus on
  - public / private APIs
  - OOP
  - Signal & Slots
- public API
  - usability
  - easy to learn and access
  - Subset of C++

CMS Screenshot



default

Layout name: default

Layout Location: Center

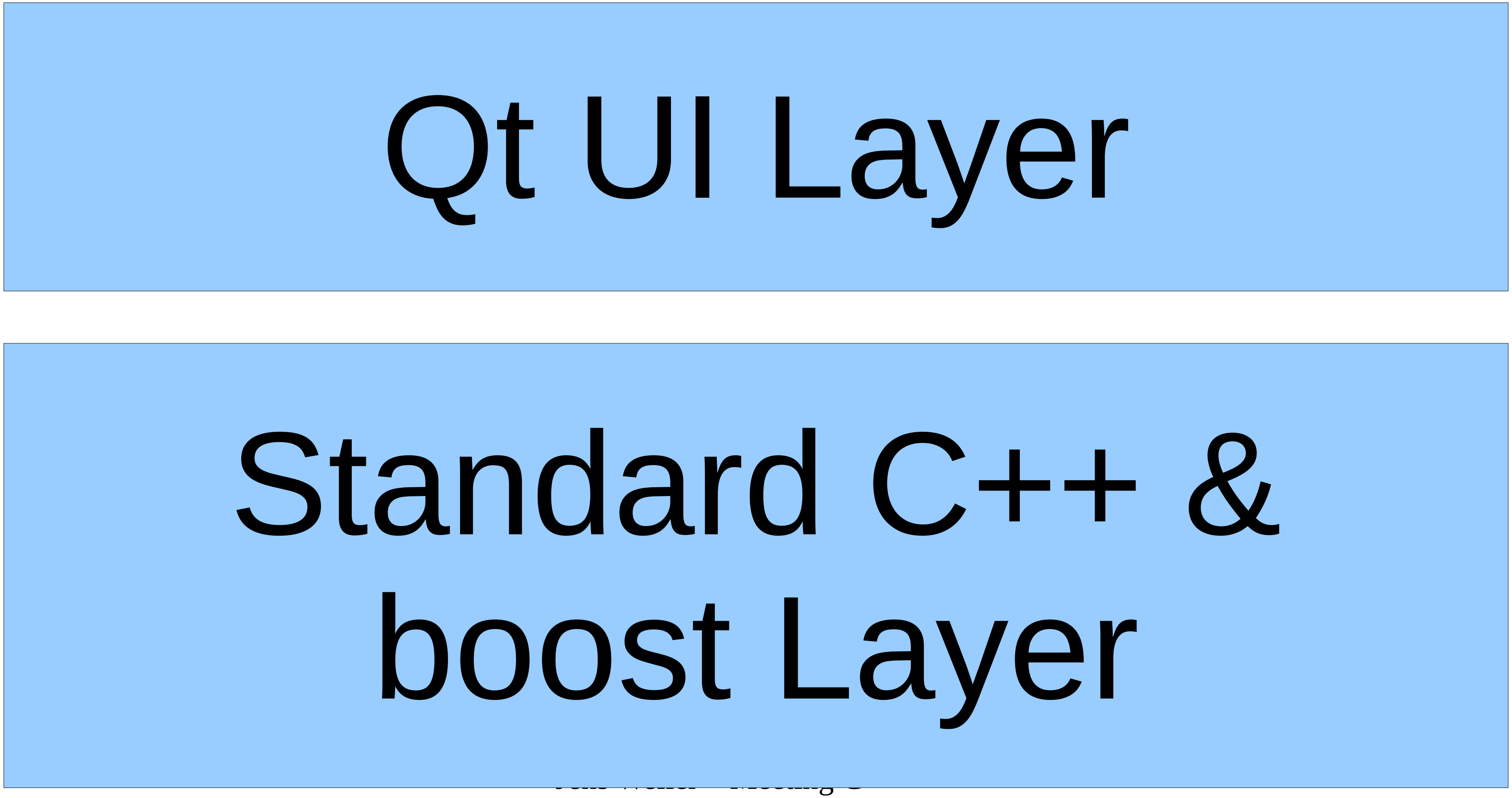
Text Element (id: 0)

Title: h1

☐ publish

Hello CppCon!

Overview



Qt UI Layer

Standard C++ & boost Layer

Boost vs. Standard C++

- Boost is still useful
  - C++17 features
    - Optional
    - Any
    - Variant
    - …
  - flat_map/set
  - asio
- Boost depends on boost
  - Boost::function

Boost vs. Standard C++

- Standard
  - New implementation
  - No build requirements
    - Often headeronly
  - Needs compiler support

Implemented Features

- Tree
- Factory
- Context Menu
- QWidgets and data
- Integration of an HTML Texteditor
- Filesystem access

Implemented Features

- Page Tree

```
template class ...types>
class TreeItem : public std::enable_shared_from_this< TreeItem<types... > >
{
    using self = TreeItem;
    using const_item_t = std::shared_ptr< const self >;
    using weak_item_t = std::weak_ptr< self >;
    variant node;
    std::vector<item_t> children;
    weak_item_t parent;
public:
    using variant = boost::variant< types...>;
    using item_t = std::shared_ptr< self >;
```

Implemented Features

- Page Tree

```
template<class ...types>
class TreeItem : public std::enable_shared_from_this< TreeItem< types... > >
{
    using self = TreeItem;
    using const_item_t = std::shared_ptr< const self >;
    using weak_item_t = std::weak_ptr< self >;
     variant node;
     std::vector<item_t> children;
     weak_item_t parent;
public:
    using variant = boost::variant< types...>;
    using item_t = std::shared_ptr< self >;
```

Implemented Features

- Factories

```
template<class AbstractClass,class IdType = size_t,
          class MakeType = boost::function<AbstractClass*()> >
struct Factory
{
    boost::container::flat_map<IdType,MakeType> factory_map;
public:
    void register_factory(IdType type_id,const MakeType& make)
...
    template<class ...args>
    abstract_type* create(IdType id, args&&... a)const
...
```

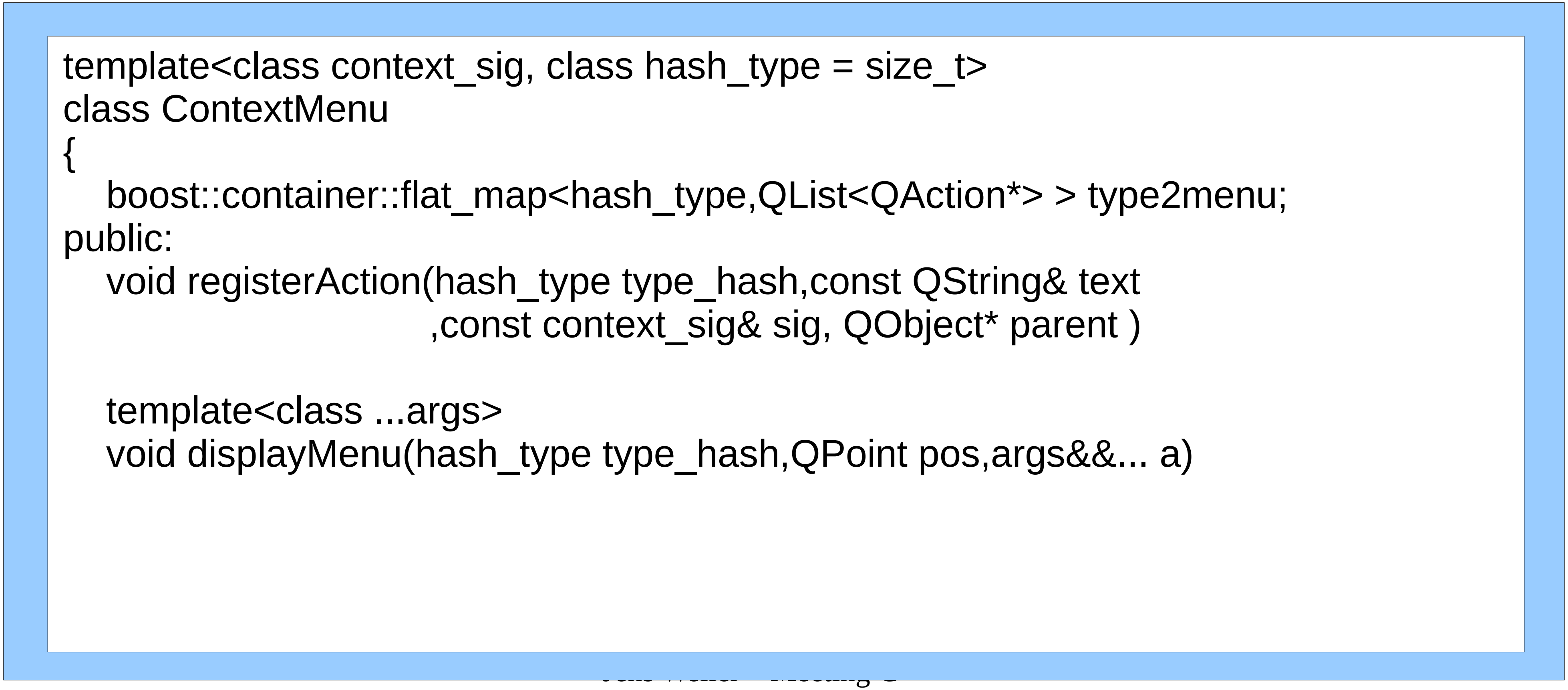
Implemented Features

- Factories

```
template<class AbstractClass,class IdType = size_t,
          class MakeType = boost::function<AbstractClass*()> >
struct Factory
{
    boost::container::flat_map<IdType,MakeType> factory_map;
public:
    void register_factory(IdType type_id,const MakeType& make)
...
    template<class ...args>
    abstract_type* create(IdType id, args&&... a)const
…
factory.registerType(dir_typeid,boost::bind(boost::factory<DirPanel*>(),_1,_2));
```

Implemented Features

- Generic Context Menus



```
template<class context_sig, class hash_type = size_t>
class ContextMenu
{
    boost::container::flat_map<hash_type,QList<QAction*> > type2menu;
public:
    void registerAction(hash_type type_hash,const QString& text
                                      ,const context_sig& sig, QObject* parent )

    template<class ...args>
    void displayMenu(hash_type type_hash,QPoint pos,args&&... a)
```

Implemented Features

- Generic Context Menus

```
template<class context_sig, class hash_type = size_t>
class ContextMenu
{
    boost::container::flat_map<hash_type,QList<QAction*> > type2menu;
public:
    template<class ...args>
    void displayMenu(hash_type type_hash,QPoint pos,args&&... a)
    {
        auto action = QMenu::exec(type2menu[type_hash],pos);
        if(action)
            action->data(). template value< context_sig >()(std::forward<args>(a)...);
    }
```

Implemented Features

- QWidgets and data...

```
template<class control>
std::string getText(QObject* obj)
{
    control* c = qobject_cast<control*>(obj);
    return c->text().toStdString();
}

std::string getCurrentText(QObject* obj)
std::string getPlainText(QObject* obj)
bool getCheck(QObject* obj)
unsigned int getTimestamp(QObject* obj)
R getValue(QObject* obj)
```

Implemented Features

- QWidgets and data...

```
template<class control>
std::string getText(QObject* obj)
{
    control* c = qobject_cast<control*>(obj);
    return c->text().toStdString();
}

std::string getCurrentText(QObject* obj)
std::string getPlainText(QObject* obj)
bool getCheck(QObject* obj)
unsigned int getTimestamp(QObject* obj)
R getValue(QObject* obj)
```

Implemented Features

- QWidgets and data...

```
template<class SetType>
class Filter
{
    using sig = std::function<void(const SetType&)>;
    using qsig = std::function<SetType(QObject*)>;
public:
    Filter(sig setter, qsig getter,QEvent::Type type  = Qevent::FocusOut):...
    bool operator()(QObject* obj,QEvent* e)
    {
        if(e->type() == eventtype)
            setter(getter(obj));
        return true;
    }
};
```

Implemented Features

- QWidgets and data...

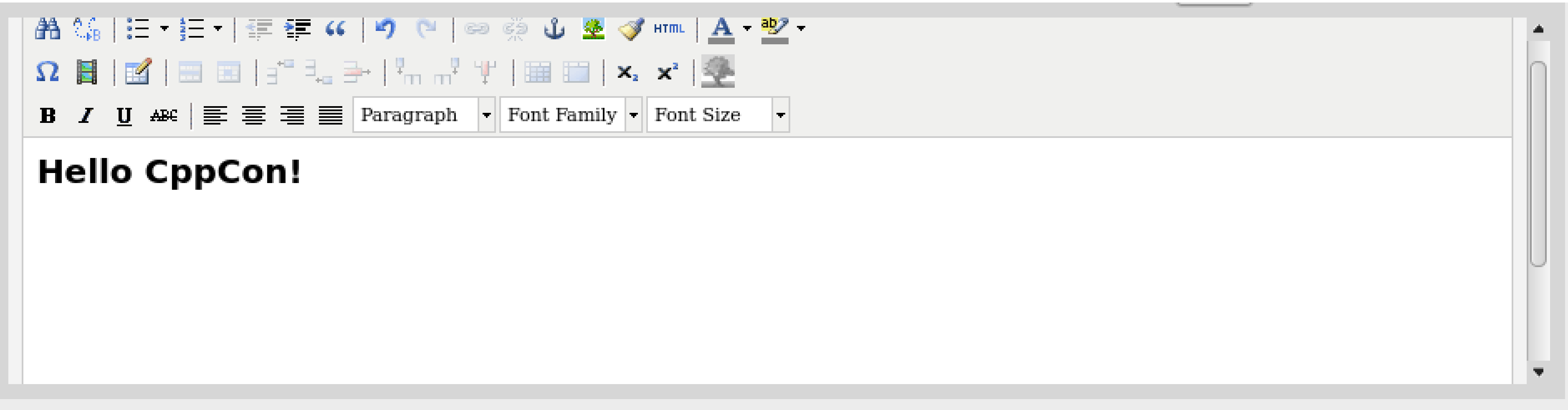
```
class EventFilter : public QObject
{
    Q_OBJECT
public:
    using eventfilter_sig = std::function<bool(QObject*,QEvent*)>;
    explicit EventFilter(eventfilter_sig filter, QObject *parent = 0);
...
protected:
    bool eventFilter(QObject *obj, QEvent *event)override
    {
        return filter(obj,event) && QObject::eventFilter(obj,event);
    }
    eventfilter_sig filter;
```

Implemented Features

- HTML Text Editor
- Integrating TinyMCE3 into my Qt Application
  - QWebView + QWebkit
- Issues
  - Qt Webkit can't render some blog posts
  - QWebEngine
    - Pure async API
    - Porting not so easy

HTML Editor

- Working HTML Editor
- Some fine tuning still needed
- Some parts will always be a hack
- TinyMCE 4.x didn't run in QWebView



Hello CppCon!

HTML Editor 2016

- QWebKit deprecated
- QWebEngine in Qt 5.7
  - MinGW not supported
  - VS 2015 Build Tools + QtCreator
  - Modern Qt API
  - Async only
  - Blink based

Implemented Features

- boost::filesystem

```
boost::container::flat_set<std::string> load_dir_recursive(const fs::path& path)
{
    boost::container::flat_set<std::string> set;
    std::string::size_type pathsize = path.generic_string().size()+1;
    for(fs::directory_entry& entry: fs::recursive_directory_iterator(path))
        set.insert(entry.path().generic_string().substr(pathsize));
    return set;
}
```

Implemented Features

- boost::filesystem

```
namespace fs = boost::filesystem;
boost::container::flat_set<std::string> load_dir_recursive(const fs::path& path)
{
    boost::container::flat_set<std::string> set;
    std::string::size_type pathsize = path.generic_string().size()+1;
    for(fs::directory_entry& entry: fs::recursive_directory_iterator(path))
        set.insert(entry.path().generic_string().substr(pathsize));
    return set;
}
```

Implemented Features

- boost::filesystem

```
namespace fs = boost::filesystem;

//create directories for a new project
fs::path p = basepath +"/"+ name;
fs::create_directories(p / "web" / "css");
fs::create_directory(p / "web" / "img");

//when loading document, check for existing archive
bool load_web = fs::exists(basepath + "/" + name +"/"+ "data.dat");
```

boostache

- Goal
  - Support any type
  - Stable mustache C++ implementation
- Its pretty close to that...

Boostache progress 2016

- Any types you say?
  - std::vector?
    - Yes!
  - What name?
    - ?
  - vector<string>
    - oh.

Boostache currently

- Dev branch can handle sequences
  - vector<string>
- My branch compiles with c++11
  - No sequence support though
  - Waiting for merge
- VS 2015
  - Build error

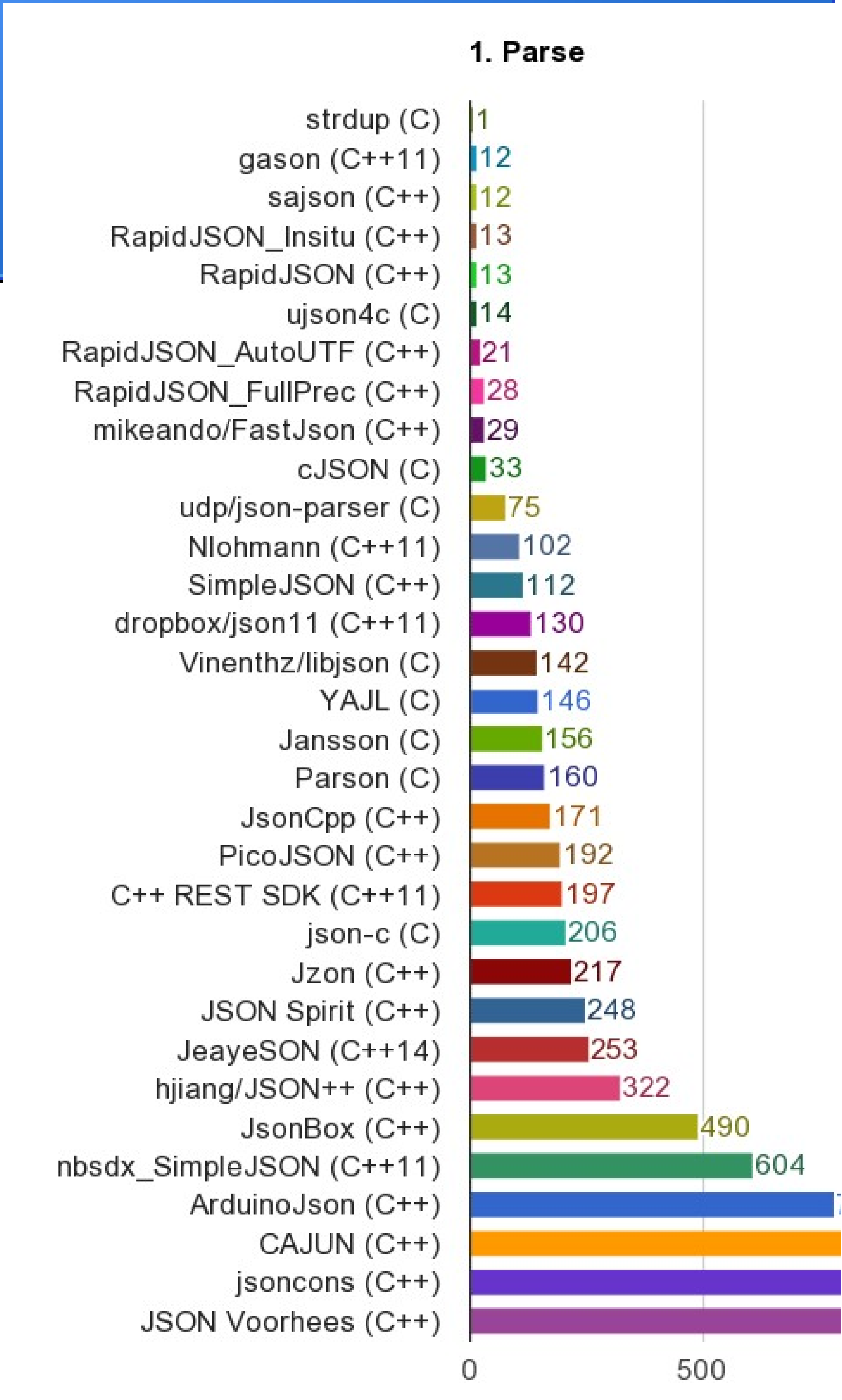
DataStore & Import

- DataStore
  - “JSON Table”

- Import
  - DB dump
  - “JSON Table”

Generic JSON

- Lots of JSON Libraries
- Which one to pick?
- Native JSON Benchmark



Best JSON Lib...

- Depends on your use case
  - API & usability
  - Speed
- generic_json
  - One API as a frontend

Generic JSON

- Should be for JSON, what arabica is for XML:
  - One interface library for other libraries
- Switch between JSON Libraries
- JSON <> generic types
  - Fusion / Hana

JSON Import Example

- Mapping
  - JSON → Member name
- Traverse JSON data
  - Import the mapped fields
- Reflect assignment
  - setField(Type,Name,Value)
  - Boost::fusion

Behind the scenes...

```
BOOST_FUSION_ADAPT_ADT(
        ListEntry,
        (std::string,const std::string, obj.getText(),obj.setText(val))
        (std::string,const std::string, obj.getTitle(),obj.setTitle(val))
)
#define ADT_MEMBER_NAME(CLASS, INDEX, MEMBER)      \
template <> struct struct_member_name<CLASS, INDEX>
{typedef char const *type; static type call() { return
#MEMBER; } };

namespace boost { namespace fusion { namespace extension {
        ADT_MEMBER_NAME(ListEntry, 0, text)
        ADT_MEMBER_NAME(ListEntry, 1, title)
} } }
```

Behind the scenes...

```
BOOST_FUSION_ADAPT_ADT(
        ListEntry,
        (std::string,const std::string, obj.getText(),obj.setText(val))
        (std::string,const std::string, obj.getTitle(),obj.setTitle(val))
)
#define ADT_MEMBER_NAME(CLASS, INDEX, MEMBER)      \
template <> struct struct_member_name<CLASS, INDEX> {typedef
char const *type; static type call() { return #MEMBER; } };

namespace boost { namespace fusion { namespace extension {
        ADT_MEMBER_NAME(ListEntry, 0, text)
        ADT_MEMBER_NAME(ListEntry, 1, title)
} } }
```

Behind the scenes...

```
BOOST_FUSION_ADAPT_ADT(
        ListEntry,
        (std::string,const std::string, obj.getText(),obj.setText(val))
        (std::string,const std::string, obj.getTitle(),obj.setTitle(val))
)
#define ADT_MEMBER_NAME(CLASS, INDEX, MEMBER)     \
template <> struct struct_member_name<CLASS, INDEX>
{typedef char const *type; static type call() { return
#MEMBER; } };

namespace boost { namespace fusion { namespace extension {
        ADT_MEMBER_NAME(ListEntry, 0, text)
        ADT_MEMBER_NAME(ListEntry, 1, title)
} } }
```

Behind the scenes...

```
BOOST_FUSION_ADAPT_ADT(
        ListEntry,
        (std::string,const std::string, obj.getText(),obj.setText(val))
        (std::string,const std::string, obj.getTitle(),obj.setTitle(val))
)
#define ADT_MEMBER_NAME(CLASS, INDEX, MEMBER)      \
template <> struct struct_member_name<CLASS, INDEX>
{typedef char const *type; static type call() { return #MEMBER; } };

namespace boost { namespace fusion { namespace extension
{
        ADT_MEMBER_NAME(ListEntry, 0, text)
        ADT_MEMBER_NAME(ListEntry, 1, title)
} } }
```

Set Value in Fusion

```
template <typename Seq, int I>
struct setvalue
{
    template<class value>
    static void call(Seq& s, const std::string& name, const value& v)
    {
        if(fusion::extension::struct_member_name<Seq,I>::call() ==
name )
            assign<I>(s,v);
        else
            setvalue<Seq,I - 1>::call(s,name,v);
    }
};
```

Set Value in Fusion

```
template <typename Seq, int I>
struct setvalue
{
    template<class value>
    static void call(Seq& s, const std::string& name, const value& v)
    {
        if(fusion::extension::struct_member_name<Seq,I>::call() ==
name )
            assign<I>(s,v);
        else
            setvalue<Seq,I - 1>::call(s,name,v);
    }
};
```

Set Value in Fusion

```
template <typename Seq, int I>
struct setvalue
{
    template<class value>
    static void call(Seq& s, const std::string& name, const value& v)
    {
        if(fusion::extension::struct_member_name<Seq,I>::call() ==
name )
            assign<I>(s,v);
        else
            setvalue<Seq,I - 1>::call(s,name,v);
    }
};
```

Generic json

- Now I just need a way to traverse json...

  … in a generic/visitable way
- Most JSON Libs
  - Nope
  - RapidJSON
    - Its own visitor

Unifying the Interface

- type()
  - Type()
  - type()
  - GetType()

Unifying the Interface

- type()
  - Type()
  - type()
  - GetType()

- Boost TTI
  - has_member_trait
  - enable_if

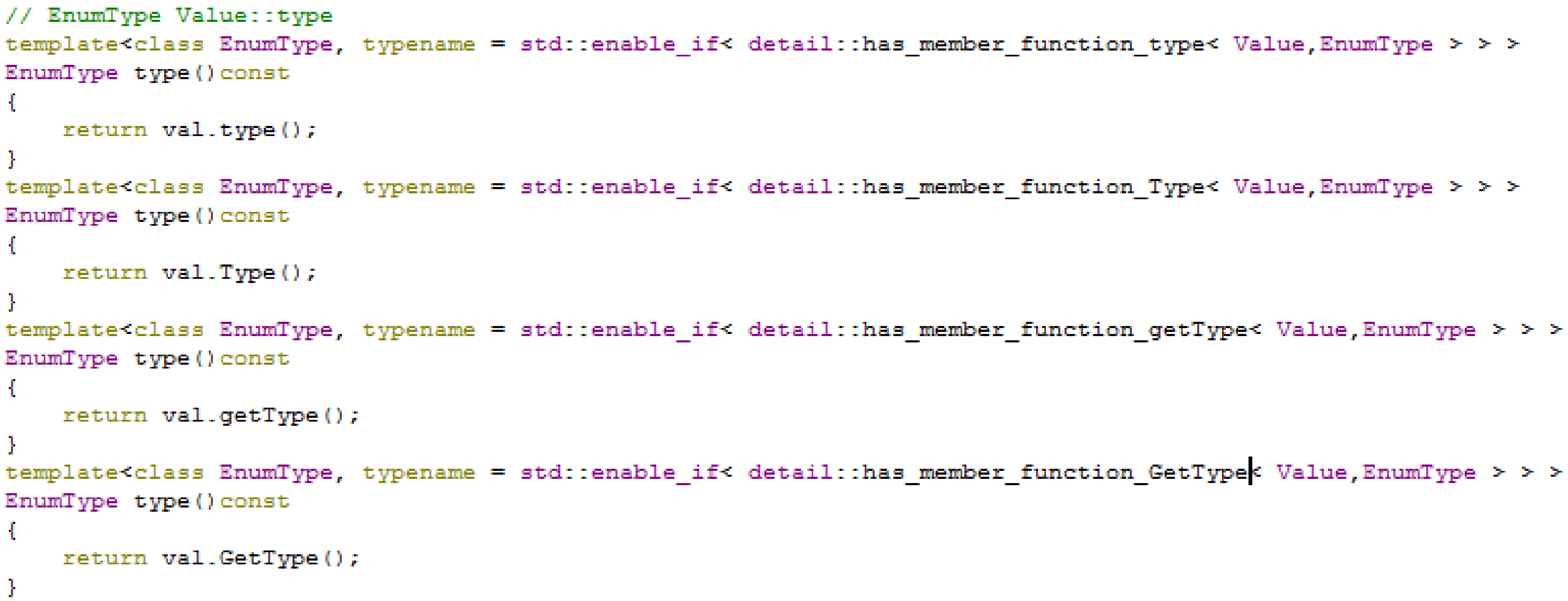
Unifying the Interface

- type()
  - Type()
  - type()
  - GetType()

- Boost TTI
  - has_member_trait
  - enable_if

- auto f()->decltype(t.Type(),void)

Unifying the Interface



```
// EnumType Value::type
template<class EnumType, typename = std::enable_if< detail::has_member_function_type< Value,EnumType > > >
EnumType type()const
{
    return val.type();
}
template<class EnumType, typename = std::enable_if< detail::has_member_function_Type< Value,EnumType > > >
EnumType type()const
{
    return val.Type();
}
template<class EnumType, typename = std::enable_if< detail::has_member_function_getType< Value,EnumType > > >
EnumType type()const
{
    return val.getType();
}
template<class EnumType, typename = std::enable_if< detail::has_member_function_GetType< Value,EnumType > > >
EnumType type()const
{
    return val.GetType();
}
```

JSON Interfaces

Intentionally left blank

generic_json

- JsonValue
- JsonTraverser
  - As only RapidJson provides this
- JsonVisitor
- fromString(string&, JsonValue&)

Status

- Prototype
- Refactor from my CMS into a Library
- Feedback from C++Now
  - Not sure its the right way to do it
- Lack of use case, it works for me now
- Lack of time currently

Conclusion & Goals

- Combining Modern C++ and OOP Style
  - Possible
  - Might require boilerplate code
- Provide an overview on techniques
  - Using modern C++
  - Using boost

C++ changes

- C++11
  - Tuple
  - Function
  - Lambdas
- C++17
  - Apply
  - Any
  - Variant

- Not C++ with classes
  - Static polymorphism
- Compile time
  - Boost.Hana
- Variant Story
  - Pattern matching
- C++Next
  - Concepts
  - Modules

Questions?

?