Practical Type Erasure

A boost::any Based Configuration Framework

Code: https://github.com/cheinan/any_config

Tag: cppnow2014

Cheinan Marks

Outline

- What is type erasure?
- How does it work?
- boost::any
- **Practical** type erasure
- Conclusion

Lies and Damn Lies

- C++11 code is now used

Type Erasure

Scott Meyers: http://www.artima.com/cppsource/top_cpp_aha_moments.html

```
namespace boost {

  template<class T> class shared_ptr {

    public:
      template<class Y> explicit shared_ptr(Y * p);
      template<class Y, class D> shared_ptr(Y * p, D d);
...
  }
}
```

Type Erasure

```
class C
{
public:
	template<typename TInject> C(TInject injectedInstance);

	void invoke();

};
```

Type Erasure: Implementation

```
class C
{
public:
        template<typename TInject> C(TInject injectedInstance)
                : m_internalBase(new CInjected<TInject>(injectedInstance)) {}
        void invoke() { m_internalBase->DoIt(); }
private:
        struct CInternalBase
        {
                virtual void DoIt() {}
        };

        template<typename TInjected> struct CInjected : public CInternalBase
        {
                CInjected(TInjected i) : m_injected(i) {}
                virtual void DoIt() { m_injected.Deploy(); }
        private:
                TInjected m_injected;
        };

        CInternalBase* m_internalBase;
};
```

Boost Any

```
#include <vector>
#include <string>
#include <iostream>
#include <boost/any.hpp>

int main()
{
	boost::any a = std::string("Anything?");
	std::vector<std::string> v = {"Anything!"};
	a = v;
	a = 5;

	std::cout << boost::any_cast<int>(a) << std::endl;

	return 0;
}
```

```
cheinan@cppnowdev:~/dev$ g++ -std=c++11 any.cpp
cheinan@cppnowdev:~/dev$ ./a.out
5
```

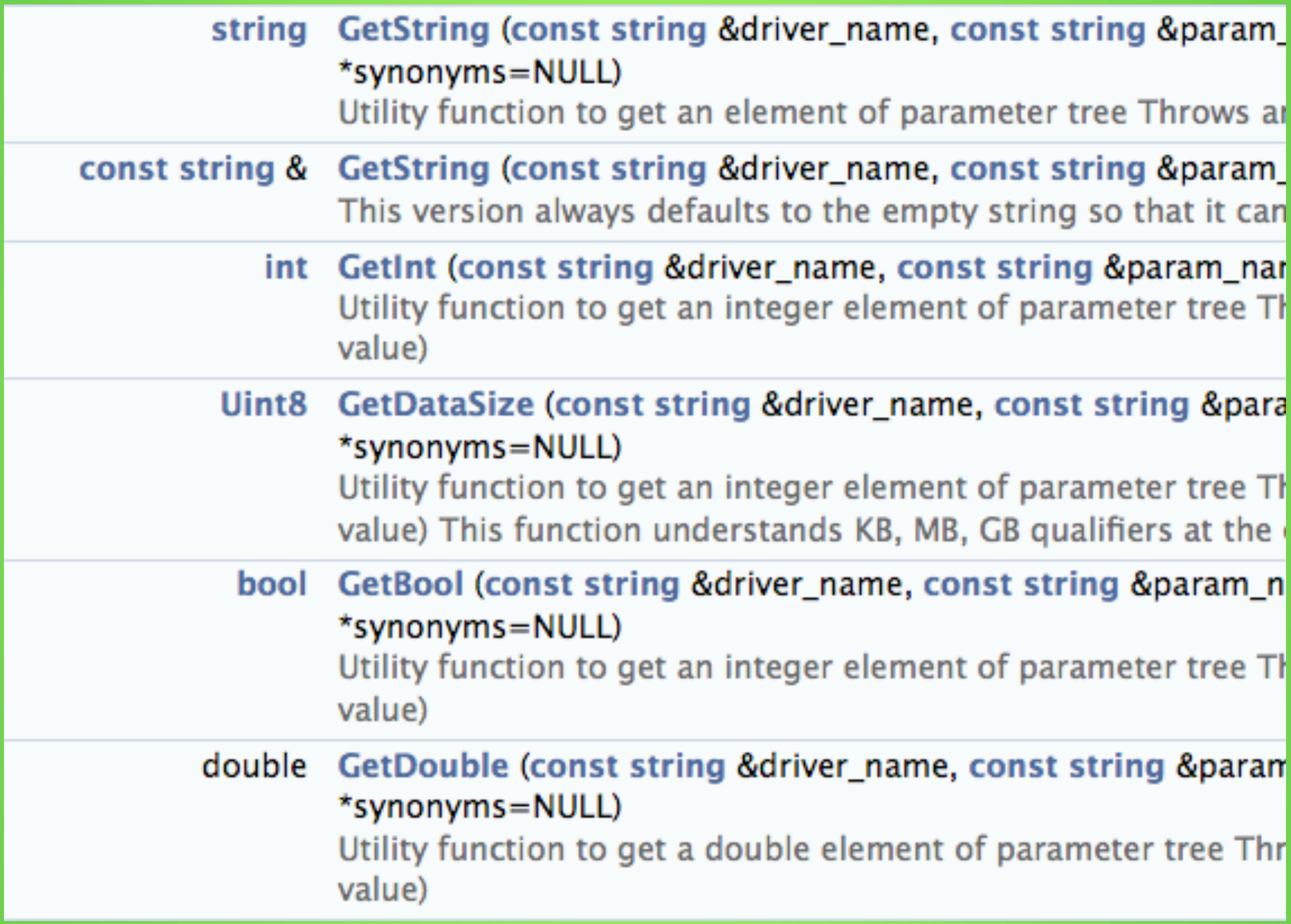
Practical Type Erasure

- Not a New Idea
- Smart Pointer Deletion
- Heterogeneous Containers
- any_iterator
- std::function

Practical Type Erasure

- Not Magic
- Someone Must Know Type
- Polymorphism Possible, but Ugly
- Use with Caution
  - Can produce unmaintainable mess

Has This Happened to You?



string	GetString (const string &driver_name, const string &param_ *synonyms=NULL) Utility function to get an element of parameter tree Throws a
const string &	GetString (const string &driver_name, const string &param_ This version always defaults to the empty string so that it can
int	GetInt (const string &driver_name, const string &param_nar Utility function to get an integer element of parameter tree Th value)
Uint8	GetDataSize (const string &driver_name, const string &para *synonyms=NULL) Utility function to get an integer element of parameter tree Th value) This function understands KB, MB, GB qualifiers at the
bool	GetBool (const string &driver_name, const string &param_n *synonyms=NULL) Utility function to get an integer element of parameter tree Th value)
double	GetDouble (const string &driver_name, const string &param *synonyms=NULL) Utility function to get a double element of parameter tree Thr value)

Configuration Framework

- Get and Set Properties
- Multiple Back Ends
  - .INI or Config File
  - Database
  - Environment
- Return More Than POD
- No Recompiling

Architecture

- Client-facing Front End Interface
  - Return any object or data by key
  - Client decides on backend(s) to use
  - Compile only used back ends
  - Client ignorant of implementation

Architecture

- Back end
  - Extendible
  - Supports specified types
  - Instances created and destroyed
  - Multiple instances supported

Architecture

- Generic Front End
- OO Back End

Architecture

- Generic Front End
- OO Back End
- Glue: Type Erasure
  - Where the rubber meets the road

http://www.artima.com/cppsource/type_erasure.html

Type Erasure [is] the Glue between OO and Generic Programming

– Thomas Becker

Client Facing Front End

```
class CAnyProperty
{
public:
    typedef     std::shared_ptr<CAnyHandlerBase>    THandlerPtr;

    template<typename T> T  Get( const std::string & key ) const

    template<typename T> void   Set( const std::string & key, const T & value )

    void    AddGetHandler( THandlerPtr handler_ptr );

    template<typename T> void    SetSetHandler( THandlerPtr handler_ptr )
private:
    typedef std::map<Loki::TypeInfo, std::vector<THandlerPtr> >    TGetHandlerMap;
    TGetHandlerMap  m_GetHandlerMap;

    typedef std::map<Loki::TypeInfo, THandlerPtr>    TSetHandlerMap;
    TSetHandlerMap  m_SetHandlerMap;
};
```

Back End Base Class

```
class CAnyHandlerBase
{
public:
    CAnyHandlerBase() {}
    virtual ~CAnyHandlerBase() {}

    virtual boost::any Get( const std::string & /*key*/ ) const
    {
        throw CAnyPropertyException(CAnyPropertyException::eNoGet);
        return boost::any();
    }

    virtual void Set( const std::string & key, const boost::any & /*value*/ )
    {
        throw CAnyPropertyException(CAnyPropertyException::eNoSet);
    }

    virtual std::string Name() const = 0; // For error reporting.

    virtual std::vector<Loki::TypeInfo> GetHandledTypes() const = 0;
};
```

Client Facing Front End

```
class CAnyProperty
{
public:
    template<typename T> T  Get( const std::string & key ) const
    {
        return  boost::any_cast<T>( x_GetAny( key, typeid( T ) ) );
    }

    template<typename T> void   Set( const std::string & key, const T & value )
    {
        x_SetAny( key, value );
    }
};
```

Glue Getter

```
boost::any
CAnyProperty::x_GetAny( const std::string & key,
                        const Loki::TypeInfo & value_type ) const
{
    if ( key.empty() ) throw CAnyPropertyException(CAnyPropertyException::eEmptyKey);

    TGetHandlerMap::const_iterator handler_list_iter = m_GetHandlerMap.find( value_type );
    if ( m_GetHandlerMap.end() == handler_list_iter ) {
        throw CAnyPropertyException( CAnyPropertyException::eNoReadHandler,
                    value_type.name() );
    }

    const TGetHandlerMap::mapped_type & handler_list = handler_list_iter->second;

    CQueryHandler   a_query_handler =
        for_each_if( handler_list.begin(), handler_list.end(), CQueryHandler( key ) );

    if ( a_query_handler.GetValue().empty() ) {
        throw CAnyPropertyNoKeyException( eKeyNotFound, key );
    }
    boost::any a = a_query_handler.GetValue();
    return a;
}
```

Glue Getter

```
template<typename InputIterator, typename Function>
    Function
    for_each_if(InputIterator first, InputIterator last, Function f)
    {
        for (; first != last; ++first)
               if ( f(*first) )    break;
        return f;
    }
```

Glue Getter Predicate

```
class   CQueryHandler : public std::unary_function<CAnyProperty::THandlerPtr, bool>
{
public:
    CQueryHandler( const std::string & key ) : m_Key( key ) {}
    boost::any  GetValue() const { return m_Value; }

    /// Execute the handler function and look for a return value.
    bool    operator() ( CAnyProperty::THandlerPtr handler_ptr )
    {
        assert(m_Value.empty());
        m_Value = handler_ptr->Get( m_Key );

        return  ! m_Value.empty();
    }

private:
    std::string m_Key;
    boost::any  m_Value;
};
```

Glue Setter

```
void    CAnyProperty::x_SetAny( const std::string & key,
                                const boost::any & value )
{
    if ( key.empty() ) {
        throw CAnyPropertyException( CAnyPropertyException::eEmptyKey);
    }

    Loki::TypeInfo  value_type( value.type() );
    TSetHandlerMap::iterator handler_iter = m_SetHandlerMap.find( value_type );
    if ( handler_iter == m_SetHandlerMap.end() ) {
        throw CAnyPropertyException( CAnyPropertyException::eNoWriteHandler,
            value_type.name() );
    }

    THandlerPtr handler_ptr = handler_iter->second;
    assert( handler_ptr );

    handler_ptr->Set( key, value );
}
```

Glue Handlers

```
inline void    CAnyProperty::AddGetHandler( CAnyProperty::THandlerPtr handler_ptr )
{
    std::vector<Loki::TypeInfo> handled_types = handler_ptr->GetHandledTypes();
    for( auto type_iter : handled_types ) {
        TGetHandlerMap::mapped_type &  handler_list = m_GetHandlerMap[type_iter];
        handler_list.push_back( handler_ptr );
    }
}
```

```
    template<typename T> void    SetSetHandler( THandlerPtr handler_ptr )
    {
        m_SetHandlerMap[Loki::TypeInfo( typeid( T ) )] = handler_ptr;
    }
```

Back End Base Class

```
class CAnyHandlerBase
{
public:
    CAnyHandlerBase() {}
    virtual ~CAnyHandlerBase() {}

    virtual boost::any Get( const std::string & /*key*/ ) const
    {
        throw CAnyPropertyException(CAnyPropertyException::eNoGet);
        return boost::any();
    }

    virtual void Set( const std::string & key, const boost::any & /*value*/ )
    {
        throw CAnyPropertyException(CAnyPropertyException::eNoSet);
    }

    virtual std::string Name() const = 0; // For error reporting.

    virtual std::vector<Loki::TypeInfo> GetHandledTypes() const = 0;
};
```

Back End Simple Handler

```
template <typename TValue>
class CAnyPropertyHandlerMemory : public CAnyHandlerBase
{
public:
    virtual boost::any  Get( const std::string & key ) const
    {
        boost::any  value;
        typename std::map<std::string, TValue>::const_iterator   it = m_Map.find( key );
        if ( it !=  m_Map.end() ) { value = it->second; }
        return  value;
    }
    virtual void  Set( const std::string & key, const boost::any & value )
    {
        m_Map[key] = boost::any_cast<TValue> ( value );
    }
    virtual std::vector<Loki::TypeInfo> GetHandledTypes() const
    {
        return  CreateTypeVector<TValue>()();
    }
private:
    std::map<std::string, TValue>    m_Map;
};
```

Backend Env Handler

```
boost::any  CAnyHandlerEnv::Get( const std::string & key ) const
{
    boost::any  value;
    char* env_value = ::getenv(key.c_str());
    if (env_value) {
        value = std::string(env_value);
    }
    return  value;
}
```

```
void CAnyHandlerEnv::Set( const std::string & key, const boost::any & value )
{
    std::string env_value(key + "=" + boost::any_cast<std::string> (value));
    int putenvReturn = ::putenv(const_cast<char*>(env_value.c_str()));
    if (putenvReturn) { throw CAnyPropertyException(...); }
}
```

```
std::vector<Loki::TypeInfo> CAnyHandlerEnv::GetHandledTypes() const
{
    return  CreateTypeVector<std::string>()();
}
```

Back End JSON Handler

```
{
    "firstName": "Homer",
    "lastName": "Simpson",
    "age": 38,
    "address": {
        "streetAddress": "742 Evergreen Terrace",
        "city": "Springfield",
        "state": "OR",
        "postalCode": "96522"
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "939 555-1234"
        },
        {
            "type": "fax",
            "number": "636 555-4567"
        }
    ],
}
```

Back End JSON Handler

```
class CAnyHandlerJSON : public CAnyHandlerBase
{
    ...
    std::map<std::string, boost::any> m_values;
};
```

```
CAnyHandlerJSON::CAnyHandlerJSON(const std::string& jsonFileName)
{
        ...
        m_values["firstName"] = topObject["firstName"].get_value<std::string>();
        m_values["lastName"] = topObject["lastName"].get_value<std::string>();
        m_values["age"] = topObject["age"].get_value<int>();

        std::map<std::string, std::string> addressMap;
        ...
        m_values["address"] = addressMap;

        std::vector<SPhoneNumber> phoneVector;
        ...
        m_values["phone"] = phoneVector;
}
```

Back End Real Life

```
class IConnection;
class CGPAttrHandlerBuildrunID : public CGPAttrHandlerBase
{
public:
    virtual boost::any  Get( const std::string & key ) const;

    virtual std::string Name() const;
    virtual std::vector<Loki::TypeInfo> GetHandledTypes() const;

    CGPAttrHandlerBuildrunID();
private:
    void    x_ConnectToDatabase();
    int     x_GetBuildID() const;
    std::string x_ConstructSQL() const;

    CGPipeProperty m_Environment;

    std::string m_Database;
    std::string m_Username;
    std::string m_Password;

    std::auto_ptr<IConnection> m_Connection;
};
```

Additional Applications

- Heterogeneous Factory
- Registry

Conclusions

- No Magic Bullet
  - Someone will have to cast
- Helps Expose Clean Interfaces
  - Even when internals are dirty
- Glues OO and Generic Code

Acknowledgements

- Mike Dicuccio
- Andrei Alexandrescu
- Kevlin Henney
- Scott Meyers
- Edvard Munch