The Reposit Project

An Improved Solution
For Autogenerating
QuantLibXL Source Code

Father Guido Sarducci's Five Minute University

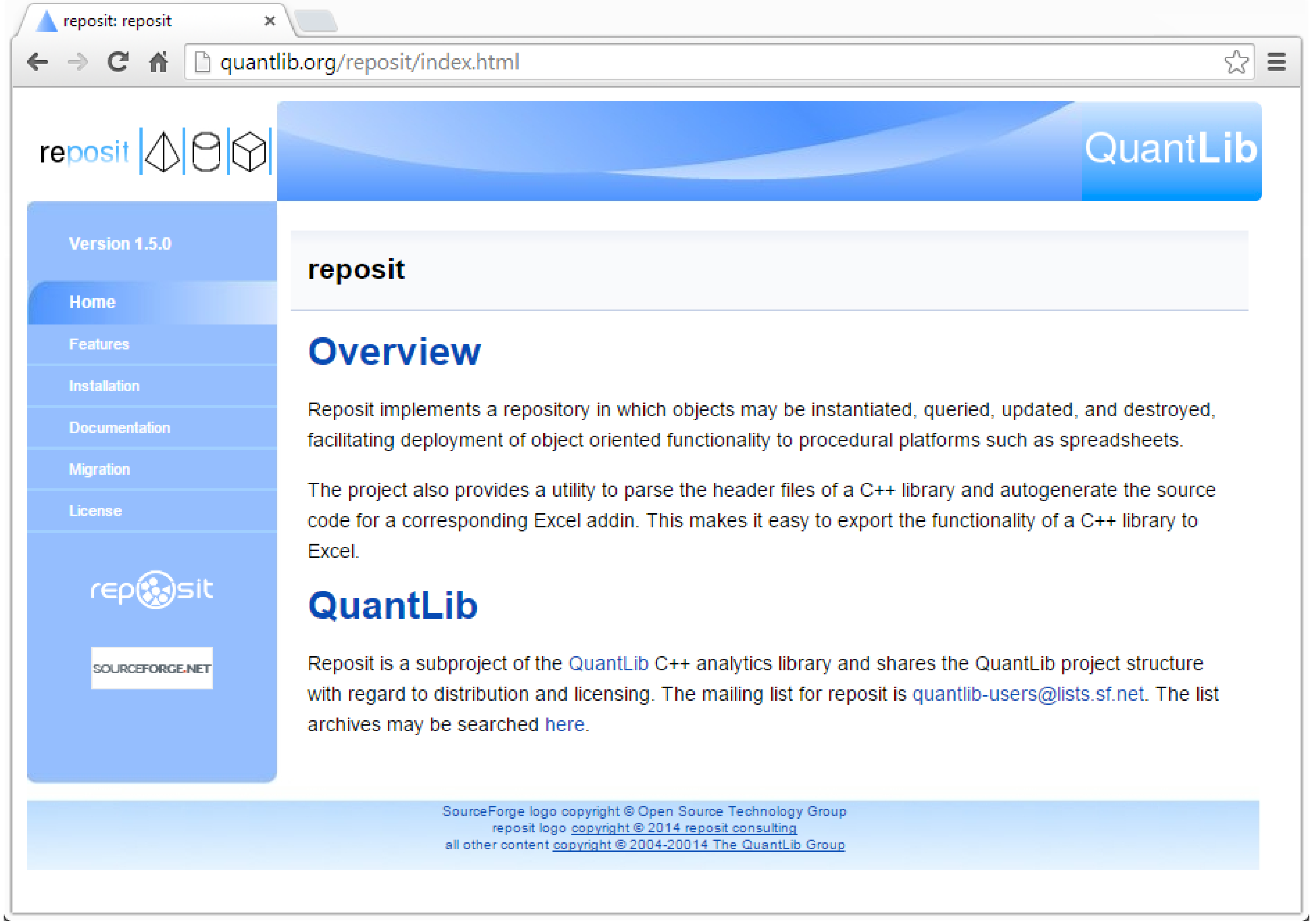
In five minutes, you learn
what the average college graduate remembers
five years after he or she is out of school.
https://www.youtube.com/watch?v=kO8x8eoU3L4

Reposit Project Five Second University:

- Replace the gensrc Python script with the reposit SWIG module
- QuantLibAddin object wrapper code autogenerated not handwritten
- Objective: Export all of QuantLib to Excel

Reposit Project Website

http://www.quantlib.org/reposit



reposit

Overview

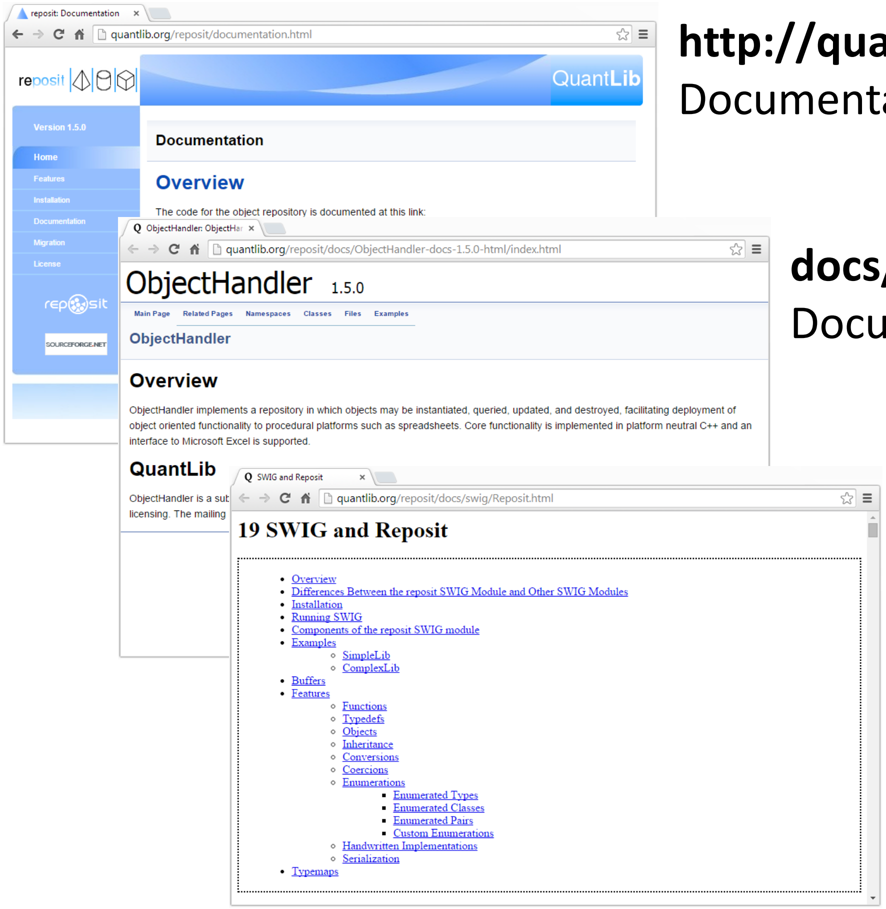
Reposit implements a repository in which objects may be instantiated, queried, updated, and destroyed, facilitating deployment of object oriented functionality to procedural platforms such as spreadsheets.

The project also provides a utility to parse the header files of a C++ library and autogenerate the source code for a corresponding Excel addin. This makes it easy to export the functionality of a C++ library to Excel.

QuantLib

Reposit is a subproject of the QuantLib C++ analytics library and shares the QuantLib project structure with regard to distribution and licensing. The mailing list for reposit is quantlib-users@lists.sf.net. The list archives may be searched here.

Documentation



quantlib.org/reposit/documentation.html

Documentation

Overview

The code for the object repository is documented at this link:

quantlib.org/reposit/docs/ObjectHandler-docs-1.5.0-html/index.html

ObjectHandler 1.5.0

ObjectHandler

Overview

ObjectHandler implements a repository in which objects may be instantiated, queried, updated, and destroyed, facilitating deployment of object oriented functionality to procedural platforms such as spreadsheets. Core functionality is implemented in platform neutral C++ and an interface to Microsoft Excel is supported.

QuantLib

ObjectHandler is a sub
licensing. The mailing

quantlib.org/reposit/docs/swig/Reposit.html

19 SWIG and Reposit

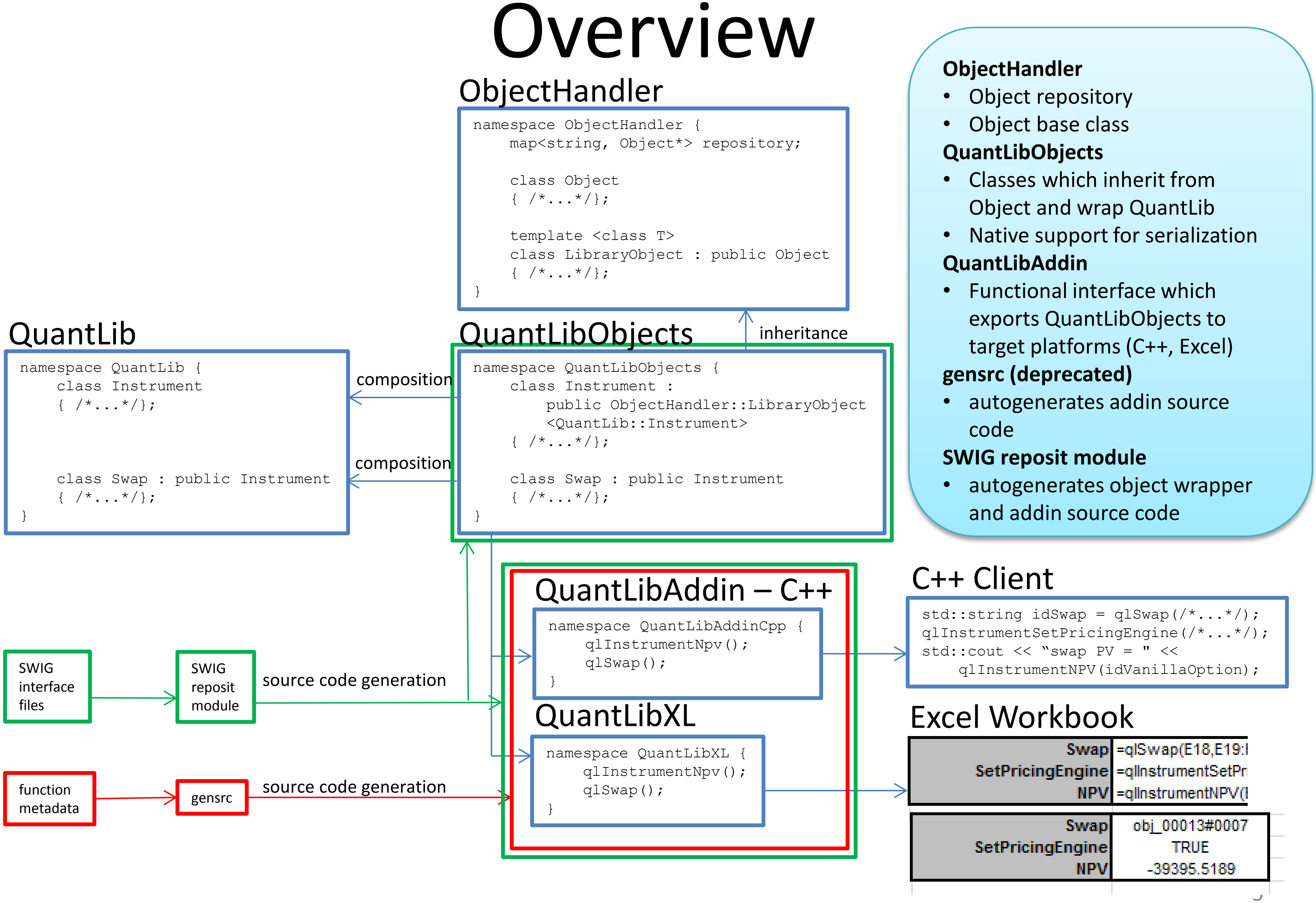
- Overview
- Differences Between the reposit SWIG Module and Other SWIG Modules
- Installation
- Running SWIG
- Components of the reposit SWIG module
- Examples
  - SimpleLib
  - ComplexLib
- Buffers
- Features
  - Functions
  - Typedefs
  - Objects
  - Inheritance
  - Conversions
  - Coercions
  - Enumerations
    - Enumerated Types
    - Enumerated Classes
    - Enumerated Pairs
    - Custom Enumerations
  - Handwritten Implementations
  - Serialization
- Typemaps

http://quantlib.org/reposit/documentation.html
Documentation for the Reposit project.

docs/ObjectHandler-docs-1.5.0-html/index.html
Documentation for the ObjectHandler repository.

docs/swig/Reposit.html
Documentation for the SWIG module.

Overview



ObjectHandler

```
namespace ObjectHandler {
    map<string, Object*> repository;

    class Object
    { /*...*/};

    template <class T>
    class LibraryObject : public Object
    { /*...*/};
}
```

QuantLib

```
namespace QuantLib {
    class Instrument
    { /*...*/};



    class Swap : public Instrument
    { /*...*/};
}
```

QuantLibObjects

```
namespace QuantLibObjects {
    class Instrument :
        public ObjectHandler::LibraryObject
        <QuantLib::Instrument>
    { /*...*/};

    class Swap : public Instrument
    { /*...*/};
}
```

QuantLibAddin – C++

```
namespace QuantLibAddinCpp {
    qlInstrumentNpv();
    qlSwap();
}
```

QuantLibXL

```
namespace QuantLibXL {
    qlInstrumentNpv();
    qlSwap();
}
```

C++ Client

```
std::string idSwap = qlSwap(/*...*/);
qlInstrumentSetPricingEngine(/*...*/);
std::cout << "swap PV = " <<
    qlInstrumentNPV(idVanillaOption);
```

Excel Workbook

Swap	=qlSwap(E18,E19:I
SetPricingEngine	=qlInstrumentSetPr
NPV	=qlInstrumentNPV(I

Swap	obj_00013#0007
SetPricingEngine	TRUE
NPV	-39395.5189

ObjectHandler
- Object repository
- Object base class

QuantLibObjects
- Classes which inherit from Object and wrap QuantLib
- Native support for serialization

QuantLibAddin
- Functional interface which exports QuantLibObjects to target platforms (C++, Excel)

gensrc (deprecated)
- autogenerates addin source code

SWIG reposit module
- autogenerates object wrapper and addin source code

Changes

This page provides an overview of how ObjectHandler, QuantLibAddin, and QuantLibXL will change after gensrc is replaced by the Reposit SWIG module.

Component	Changes
Source code generation	• The gensrc Python script is discontinued and is replaced by the Reposit SWIG module.
ObjectHandler	• Some ObjectHandler source code that was previously autogenerated by gensrc is now maintained manually. • Otherwise no changes to ObjectHandler code or functionality. • I might like to rename ObjectHandler to Reposit.
QuantLibAddin	• Object wrapper source code that was previously handwritten is now autogenerated • Some less important source code (e.g. enumerations) that was previously autogenerated is now maintained manually. • C++ Addin is now easier to use and its interface is now more similar both to QuantLib and to QuantLibXL. • Conversion/Coercion code completely rewritten, cleaned up, clarified, and commented. Many other minor improvements.
QuantLibXL	• Old design supports 1,000+ functions, new design currently supports only a dozen or so functions, enough to price an Equity Option. • It is hoped that the new design will be easier to use and will result in more QuantLib functionality being exported to Excel. • In principle, changing the method of autogenerating source code should not change the design of QuantLibXL. In practice, some things will change, e.g. function names.

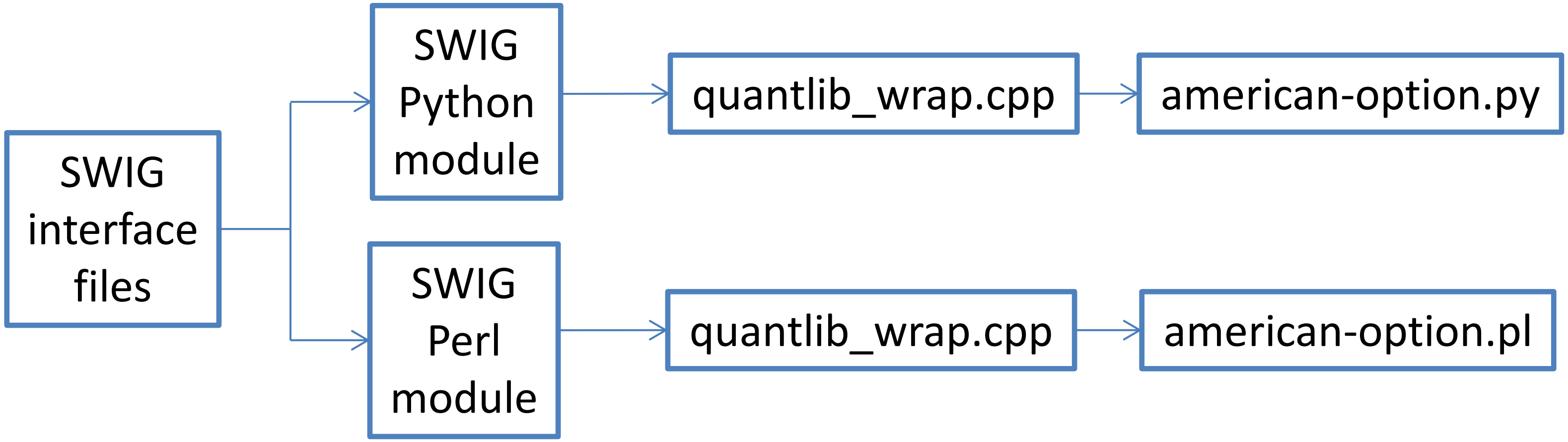
SWIG

Typical usage e.g. QuantLib-SWIG

Used in the normal way, SWIG performs two steps:

1) parse the SWIG interface files
2) generate a single source code file which can be compiled into an addin for the target platform.

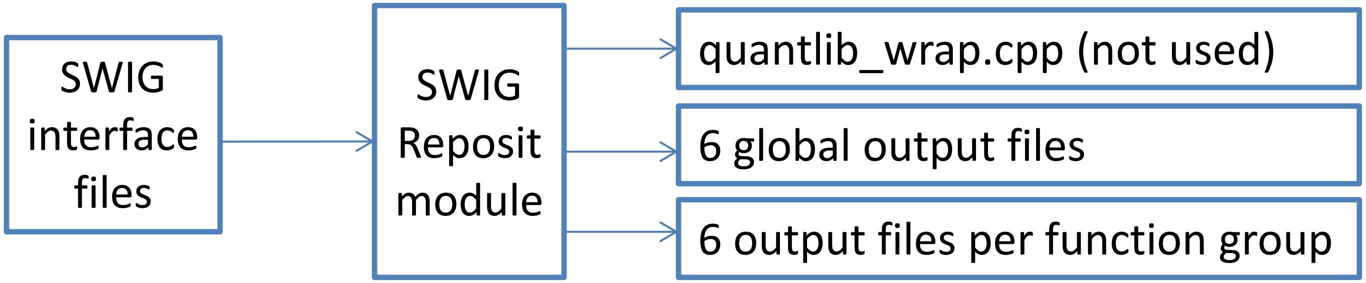
QuantLib-SWIG uses SWIG in the usual way:



SWIG

Custom usage by Reposit

Reposit relies on the core SWIG functionality to parse the interface files.
Reposit then does its own thing for code generation.
The standard SWIG output file is generated, but it is not used.
Instead Reposit generates a completely different set of output files.


SWIG interface files
SWIG Reposit module
quantlib_wrap.cpp (not used)
6 global output files
6 output files per function group

We will describe the Reposit output files in more detail.
But first let us answer The Most Frequently Asked Question...

SWIG Interface Files

How Come Reposit Doesn't Reuse QuantLib's SWIG Interface Files?

QuantLib

```
// plain option and engines

%{
using QuantLib::VanillaOption;
typedef boost::shared_ptr<Instrument> VanillaOptionPtr;
%}

%rename(VanillaOption) VanillaOptionPtr;
class VanillaOptionPtr : public boost::shared_ptr<Instrument> {
  public:
    %extend {
        VanillaOptionPtr(
                const boost::shared_ptr<Payoff>& payoff,
                const boost::shared_ptr<Exercise>& exercise) {
            boost::shared_ptr<StrikedTypePayoff> stPayoff =
                 boost::dynamic_pointer_cast<StrikedTypePayoff>(payoff);
            QL_REQUIRE(stPayoff, "wrong payoff given");
            return new VanillaOptionPtr(new VanillaOption(stPayoff,exercise));
        }
    }
};
```

Reposit

```
namespace QuantLib {
    class Instrument {
      public:
        //Instrument();
        void setPricingEngine(const boost::shared_ptr<QuantLib::PricingEngine>& engine);
        QuantLib::Real NPV();
    };
    class VanillaOption : public Instrument {
      public:
        VanillaOption(const boost::shared_ptr<QuantLib::StrikedTypePayoff>& payoff,
                      const boost::shared_ptr<QuantLib::Exercise>& exercise);
    };
}
```

Shown at left:

- the QuantLib SWIG interface file for an Option
- the Reposit SWIG interface file for an Option

The QuantLib SWIG files were written before SWIG introduced support for boost shared pointers. The file contains additional logic to hide the shared pointer.

Reposit's SWIG interface file is much more similar to the corresponding QuantLib C++ header file.

Output Files

Reposit generates six output files global to the Addin:

Path	Component
ComplexLibAddin/clo/obj_all.hpp	#include directives
ComplexLibAddin/clo/serialization/register_creators.cpp	register addin classes with the serialization layer
ComplexLibAddin/clo/serialization/create/create_all.hpp	#includes relating to creation of serializtion objects
ComplexLibAddin/clo/serialization/register/serialization_register.hpp	#includes relating to registration for serialization
ComplexLibAddin/clo/serialization/register/serialization_all.hpp	#includes relating to registration for serialization
ComplexLibAddin/AddinCpp/add_all.hpp	#includes for the C++ addin

Reposit generates six output files for each group of functions (instruments, term structures, etc:

Component	
ComplexLibAddin/clo/valueobjects/vo_xx.?pp	implementation of value objects in support of serialization
ComplexLibAddin/clo/serialization/create/create_xx.?pp	functions to create objects as they are deserialized
ComplexLibAddin/clo/serialization/register/serialization_xx.?pp	register addin classes with the serialization layer
ComplexLibAddin/clo/obj_xx.?pp	addin objects that wrap classes in the library
ComplexLibAddin/AddinCpp/add_xx.?pp	the functions in the C++ addin
ComplexLibXL/clxl/functions/function_xxx.cpp	The functions in the Excel addin

SimpleLib

Very nearly* the smallest Reposit project that it is possible to have.

1. Define your Library

```
namespace SimpleLib {

    std::string func();

    class Adder {
    private:
        long x_;
    public:
        Adder(long x);
        long add(long y);
    };

};
```

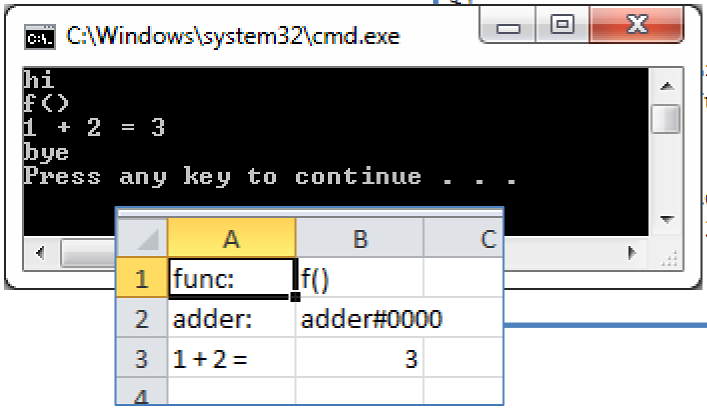
2. Create your SWIG interface file

```
%typemap(rp_tm_xll_cod) SimpleLib::Adder * "C";

%module(
    rp_obj_dir="../AddinObjects",
    rp_add_dir="../AddinCpp",
    rp_xll_dir="../AddinXl",
    rp_obj_inc="AddinObjects",
    rp_add_inc="AddinCpp",
    rp_xll_inc="AddinXl"
) SimpleLibAddin

%feature("rp:group", "adder");
%feature("rp:obj_include") %{
#include "Library/adder.hpp"
%}

                                    ib {
                                    unc();


                                    g x);
                                    long y);
```

3. Generate your Addins

```
// BEGIN typemap rp_tm_add_ret std::string
std::string
// END   typemap rp_tm_add_ret
slFunc(
    // BEGIN typemap rp_tm_add_prm
    // END   typemap rp_tm_add_prm
);

std::string slAdder(
    // BEGIN typemap rp_tm_add_prm
    std::string const & objectID,
    long x
    // END   typemap rp_tm_add_prm
);

// BEGIN typemap rp_tm_add_ret long
long
// END   typemap rp_tm_add_ret
slAdderAdd(
    // BEGIN typemap rp_tm_add_prm
    std::string const & objectID,
    long y
    // END   typemap rp_tm_add_prm
);
```

4. Run them ☺



```
hi
f()
1 + 2 = 3
bye
Press any key to continue . . .
```

	A	B	C
1	func:	f()	
2	adder:	adder#0000	
3	1 + 2 =	3	
4			

* you could make it smaller by dropping the class and keeping only the function...

ComplexLib

This example project supports a bucket list of all features supported by Reposit.

Feature	Description/Example
Functions	`std::string helloWorld();`
Typedefs	`typedef double Real;`
Objects	`class Foo { ... };`
Inheritance	`class Bar : public Foo { ... };`
Conversions	`void f(Real r);`
Coercions	`void setQuote(X x); // x could be a double or a string id of a Quote object`
Enumerated Types	`enum AccountType { Current, Savings };`
Enumerated Classes	`class TimeZoneUtc : public TimeZone { /* ... */ };`
Enumerated Pairs*	`template<type A, type B> class Foo { ... };`
Custom Enumerations*	Calendar factory – create new joint calendars on the fly as they are named.
Overrides	The developer may suppress autogeneration of selected source code files in order to provide handwritten code.
Serialization*	Serialization of objects, exactly as in the old build of ObjectHandler/QuantLibAddin/QuantLibXL.

* not yet supported

Inheritance

Example – Step 1 of 7 – Overview

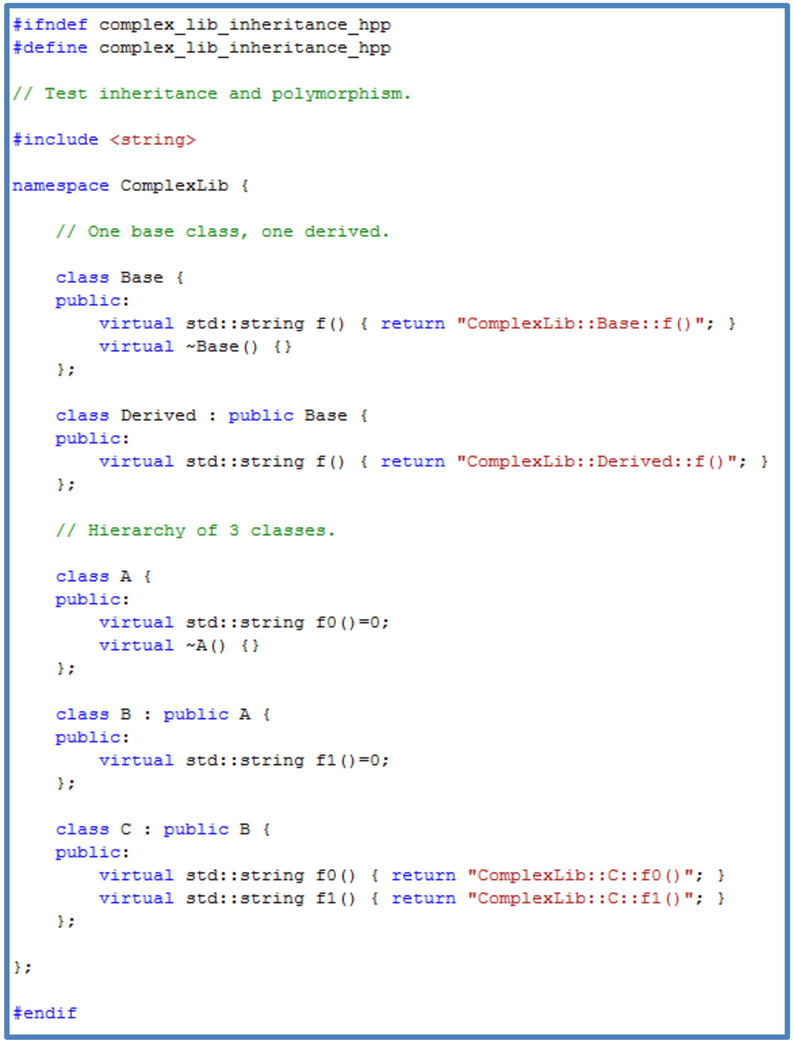
Here we take one of the features supported by Reposit – Inheritance – and work through the ComplexLib example step by step.

When your C++ library (e.g. QuantLib) contains inheritance relationships, the code to be autogenerated by Reposit for each class will differ depending upon whether the class has a parent and/or a constructor.

Parent?	Constructor?	Code	Description
No	Yes	full class inheriting LibraryObject	If the library class is a base class, and if it has a constructor, then reposit autogenerates a complete implementation of the wrapper class. For base class ComplexLib::Foo, you get a wrapper class ComplexLibAddin::Foo which inherits from helper class ObjectHandler::LibraryObject.
No	No	OH_LIB_CLASS	If the library class is a base class, and if it has no constructor, reposit still generates a wrapper class. But the wrapper is a skeleton and the entire implementation is provided by macro OH_LIB_CLASS.
Yes	Yes	full class inheriting Object	If the library class is a derived class, and if it has a constructor, then reposit autogenerates a complete implementation of the wrapper class. For base class ComplexLib::Bar deriving from ComplexLib::Foo, you get a wrapper class ComplexLibAddin::Bar deriving from ComplexLibAddin::Foo.
Yes	No	OH_OBJ_CLASS	If the library class is a derived class, and if it has no constructor, reposit still generates a wrapper class. But the wrapper is a skeleton and the entire implementation is provided by macro OH_OBJ_CLASS.

Inheritance

Example – Step 2 of 7 – Library Header File



```cpp
#ifndef complex_lib_inheritance_hpp
#define complex_lib_inheritance_hpp

// Test inheritance and polymorphism.

#include <string>

namespace ComplexLib {

    // One base class, one derived.

    class Base {
    public:
        virtual std::string f() { return "ComplexLib::Base::f()"; }
        virtual ~Base() {}
    };

    class Derived : public Base {
    public:
        virtual std::string f() { return "ComplexLib::Derived::f()"; }
    };

    // Hierarchy of 3 classes.

    class A {
    public:
        virtual std::string f0()=0;
        virtual ~A() {}
    };

    class B : public A {
    public:
        virtual std::string f1()=0;
    };

    class C : public B {
    public:
        virtual std::string f0() { return "ComplexLib::C::f0()"; }
        virtual std::string f1() { return "ComplexLib::C::f1()"; }
    };

};

#endif
```

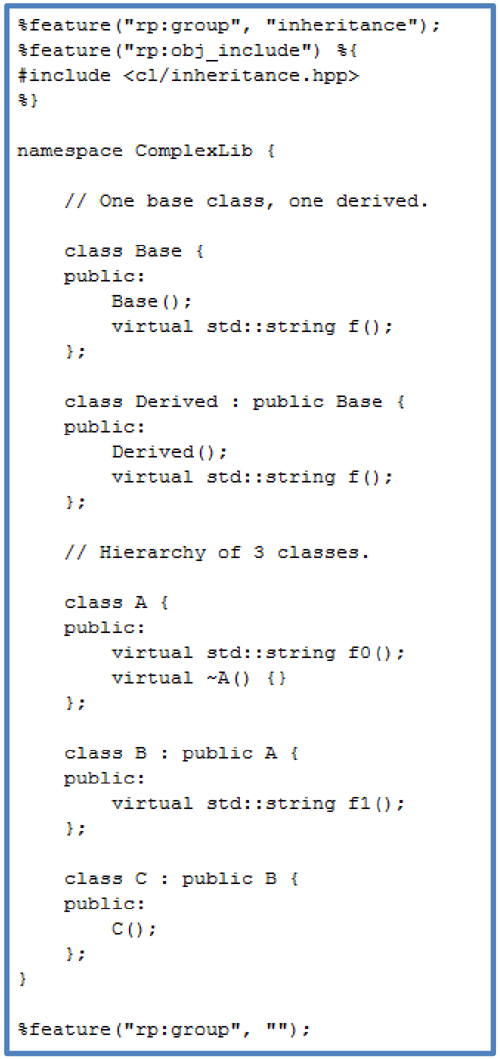
This is a C++ header file from the example ComplexLib application.

It defines a few inheritance relationships.

In the real world this would be a header file from QuantLib or some other library that you want to wrap.

Inheritance

Example – Step 3 of 7 – SWIG interface file



```
%feature("rp:group", "inheritance");
%feature("rp:obj_include") %{
#include <cl/inheritance.hpp>
%}

namespace ComplexLib {

    // One base class, one derived.

    class Base {
    public:
        Base();
        virtual std::string f();
    };

    class Derived : public Base {
    public:
        Derived();
        virtual std::string f();
    };

    // Hierarchy of 3 classes.

    class A {
    public:
        virtual std::string f0();
        virtual ~A() {}
    };

    class B : public A {
    public:
        virtual std::string f1();
    };

    class C : public B {
    public:
        C();
    };
}

%feature("rp:group", "");
```

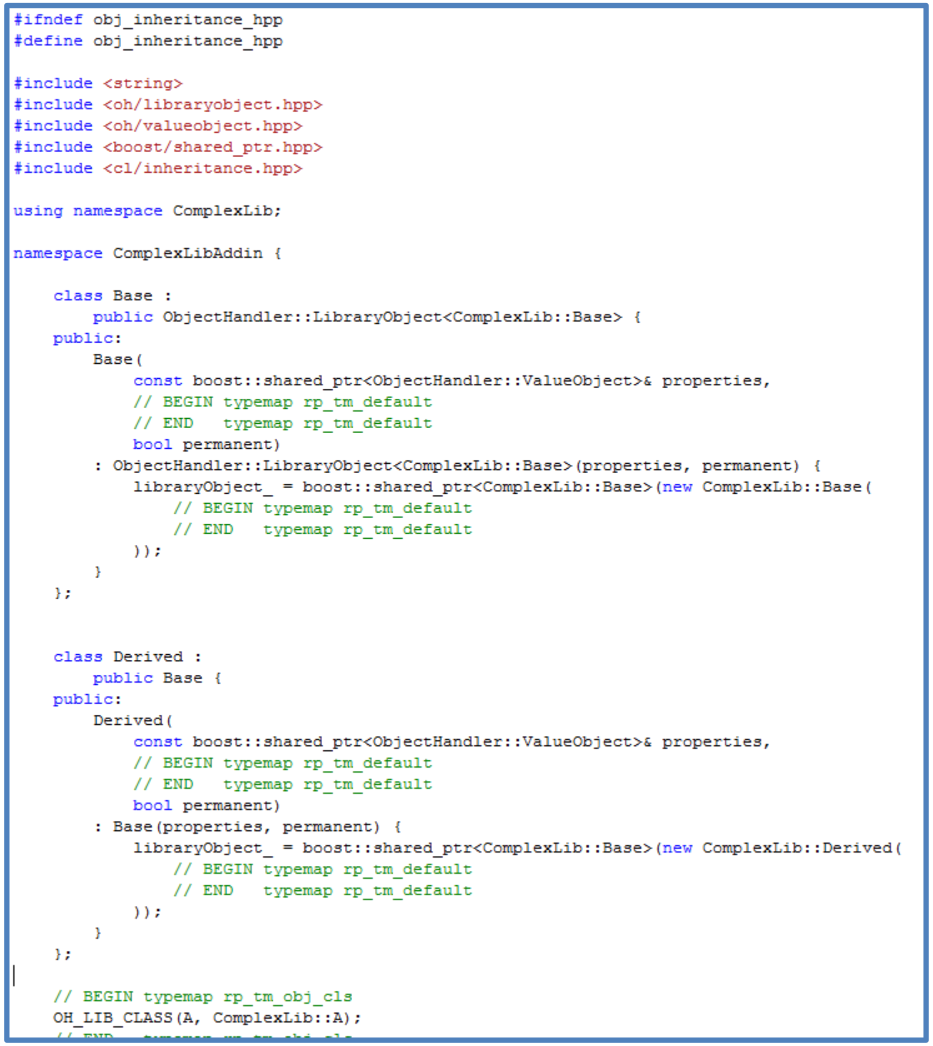
This is a SWIG interface file, written for consumption by the Reposit SWIG module.

This file defines the subset of the C++ header file that we want to export to our Addins (C++ and Excel).

This file is very similar in format to the corresponding C++ header file.

Inheritance

Example – Step 4 of 7 – Autogenerated Object Wrapper Code



```
#ifndef obj_inheritance_hpp
#define obj_inheritance_hpp

#include <string>
#include <oh/libraryobject.hpp>
#include <oh/valueobject.hpp>
#include <boost/shared_ptr.hpp>
#include <cl/inheritance.hpp>

using namespace ComplexLib;

namespace ComplexLibAddin {

    class Base :
        public ObjectHandler::LibraryObject<ComplexLib::Base> {
    public:
        Base(
            const boost::shared_ptr<ObjectHandler::ValueObject>& properties,
            // BEGIN typemap rp_tm_default
            // END   typemap rp_tm_default
            bool permanent)
        : ObjectHandler::LibraryObject<ComplexLib::Base>(properties, permanent) {
            libraryObject_ = boost::shared_ptr<ComplexLib::Base>(new ComplexLib::Base(
                // BEGIN typemap rp_tm_default
                // END   typemap rp_tm_default
            ));
        }
    };


    class Derived :
        public Base {
    public:
        Derived(
            const boost::shared_ptr<ObjectHandler::ValueObject>& properties,
            // BEGIN typemap rp_tm_default
            // END   typemap rp_tm_default
            bool permanent)
        : Base(properties, permanent) {
            libraryObject_ = boost::shared_ptr<ComplexLib::Base>(new ComplexLib::Derived(
                // BEGIN typemap rp_tm_default
                // END   typemap rp_tm_default
            ));
        }
    };

    // BEGIN typemap rp_tm_obj_cls
    OH_LIB_CLASS(A, ComplexLib::A);
    // END   typemap rp_tm_obj_cls
```

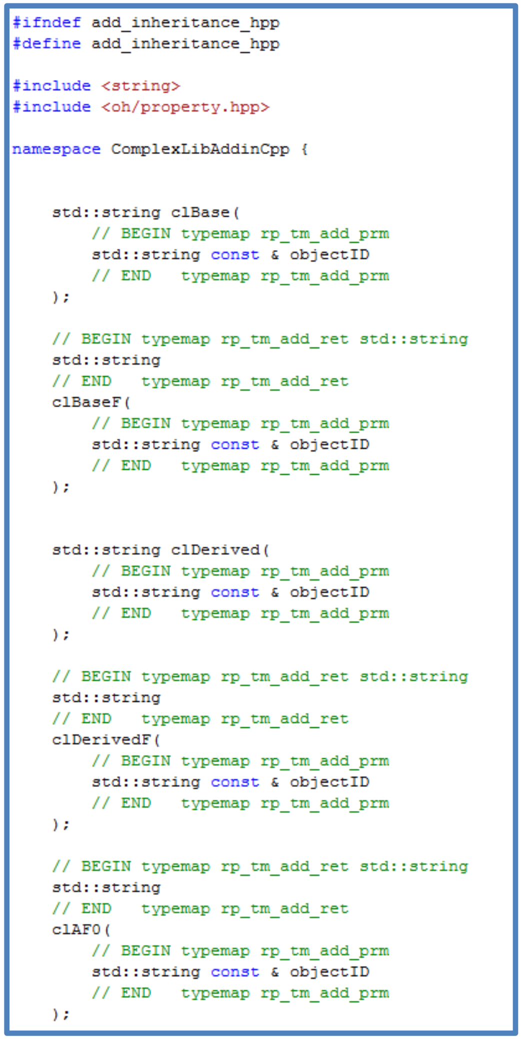
This is the autogenerated wrapper code.

In this example we call it ComplexLibAddin, in the real world this would be QuantLibAddin (QuantLibObjects).

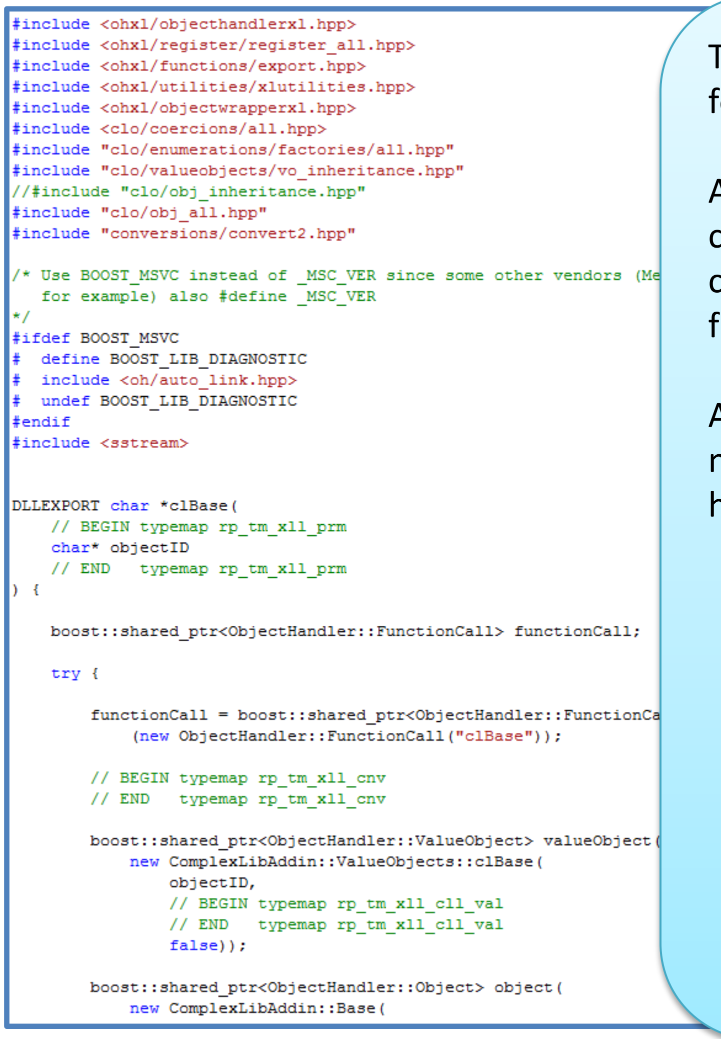
Each class here inherits from ObjectHandler::Object and holds a pointer to a ComplexLib object.

Inheritance

Example – Step 5 of 7 – Autogenerated Addin Code



```
#ifndef add_inheritance_hpp
#define add_inheritance_hpp

#include <string>
#include <oh/property.hpp>

namespace ComplexLibAddinCpp {


    std::string clBase(
        // BEGIN typemap rp_tm_add_prm
        std::string const & objectID
        // END   typemap rp_tm_add_prm
    );

    // BEGIN typemap rp_tm_add_ret std::string
    std::string
    // END   typemap rp_tm_add_ret
    clBaseF(
        // BEGIN typemap rp_tm_add_prm
        std::string const & objectID
        // END   typemap rp_tm_add_prm
    );


    std::string clDerived(
        // BEGIN typemap rp_tm_add_prm
        std::string const & objectID
        // END   typemap rp_tm_add_prm
    );

    // BEGIN typemap rp_tm_add_ret std::string
    std::string
    // END   typemap rp_tm_add_ret
    clDerivedF(
        // BEGIN typemap rp_tm_add_prm
        std::string const & objectID
        // END   typemap rp_tm_add_prm
    );

    // BEGIN typemap rp_tm_add_ret std::string
    std::string
    // END   typemap rp_tm_add_ret
    clAF0(
        // BEGIN typemap rp_tm_add_prm
        std::string const & objectID
        // END   typemap rp_tm_add_prm
    );
```



```
#include <ohxl/objecthandlerxl.hpp>
#include <ohxl/register/register_all.hpp>
#include <ohxl/functions/export.hpp>
#include <ohxl/utilities/xlutilities.hpp>
#include <ohxl/objectwrapperxl.hpp>
#include <clo/coercions/all.hpp>
#include "clo/enumerations/factories/all.hpp"
#include "clo/valueobjects/vo_inheritance.hpp"
//#include "clo/obj_inheritance.hpp"
#include "clo/obj_all.hpp"
#include "conversions/convert2.hpp"

/* Use BOOST_MSVC instead of _MSC_VER since some other vendors (Me
   for example) also #define _MSC_VER
*/
#ifdef BOOST_MSVC
#  define BOOST_LIB_DIAGNOSTIC
#  include <oh/auto_link.hpp>
#  undef BOOST_LIB_DIAGNOSTIC
#endif
#include <sstream>


DLLEXPORT char *clBase(
    // BEGIN typemap rp_tm_xll_prm
    char* objectID
    // END   typemap rp_tm_xll_prm
) {

    boost::shared_ptr<ObjectHandler::FunctionCall> functionCall;

    try {

        functionCall = boost::shared_ptr<ObjectHandler::FunctionCa
            (new ObjectHandler::FunctionCall("clBase"));

        // BEGIN typemap rp_tm_xll_cnv
        // END   typemap rp_tm_xll_cnv

        boost::shared_ptr<ObjectHandler::ValueObject> valueObject(
            new ComplexLibAddin::ValueObjects::clBase(
                objectID,
                // BEGIN typemap rp_tm_xll_cll_val
                // END   typemap rp_tm_xll_cll_val
                false));

        boost::shared_ptr<ObjectHandler::Object> object(
            new ComplexLibAddin::Base(
```

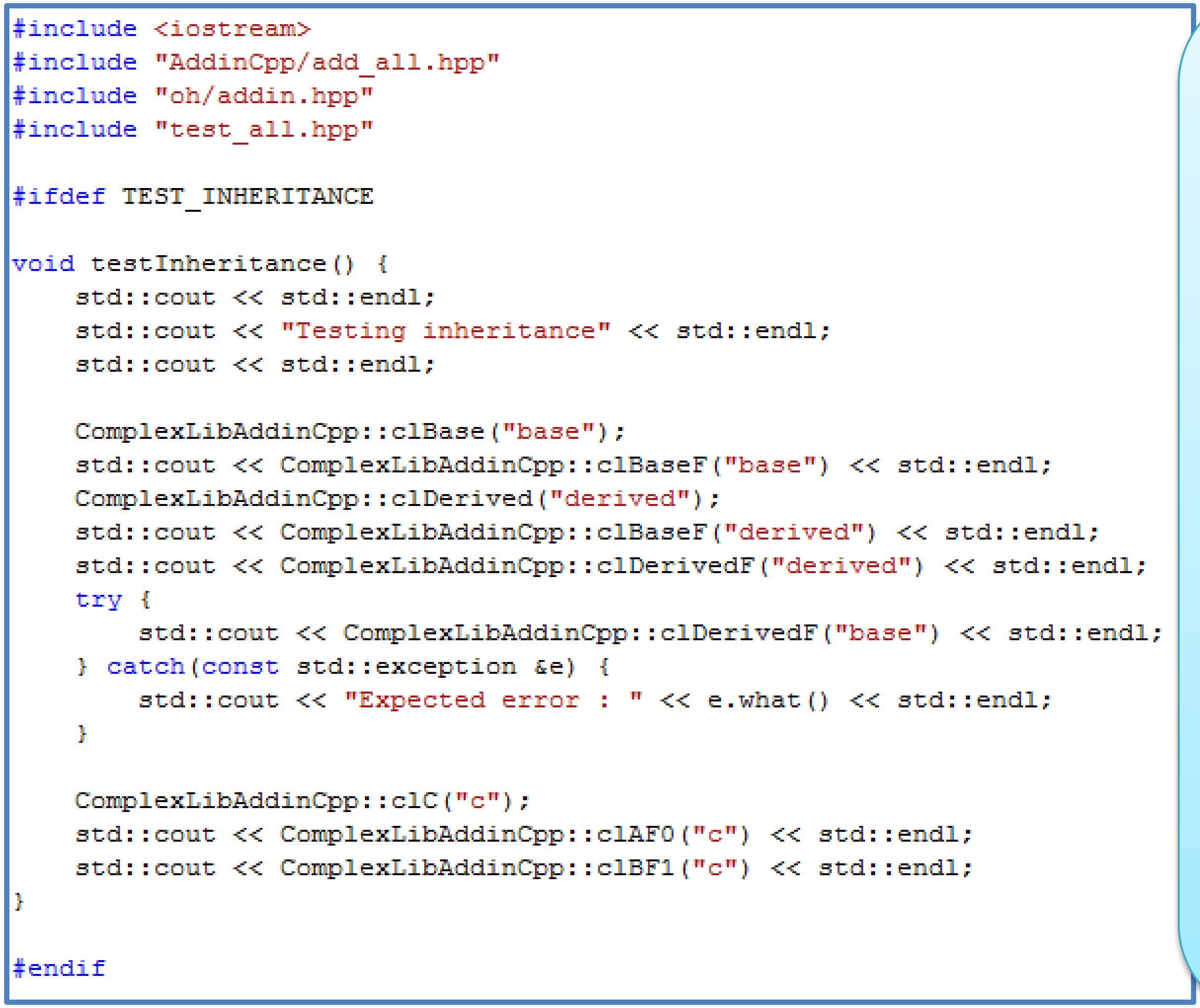
This is the autogenerated code for the C++ and Excel addins.

As Excel worksheet functions cannot directly handle C++ constructors, this code is functional, not object oriented.

All of the code required for the necessary dataype conversions has been autogenerated.

Inheritance

Example – Step 6 of 7 – Client Code



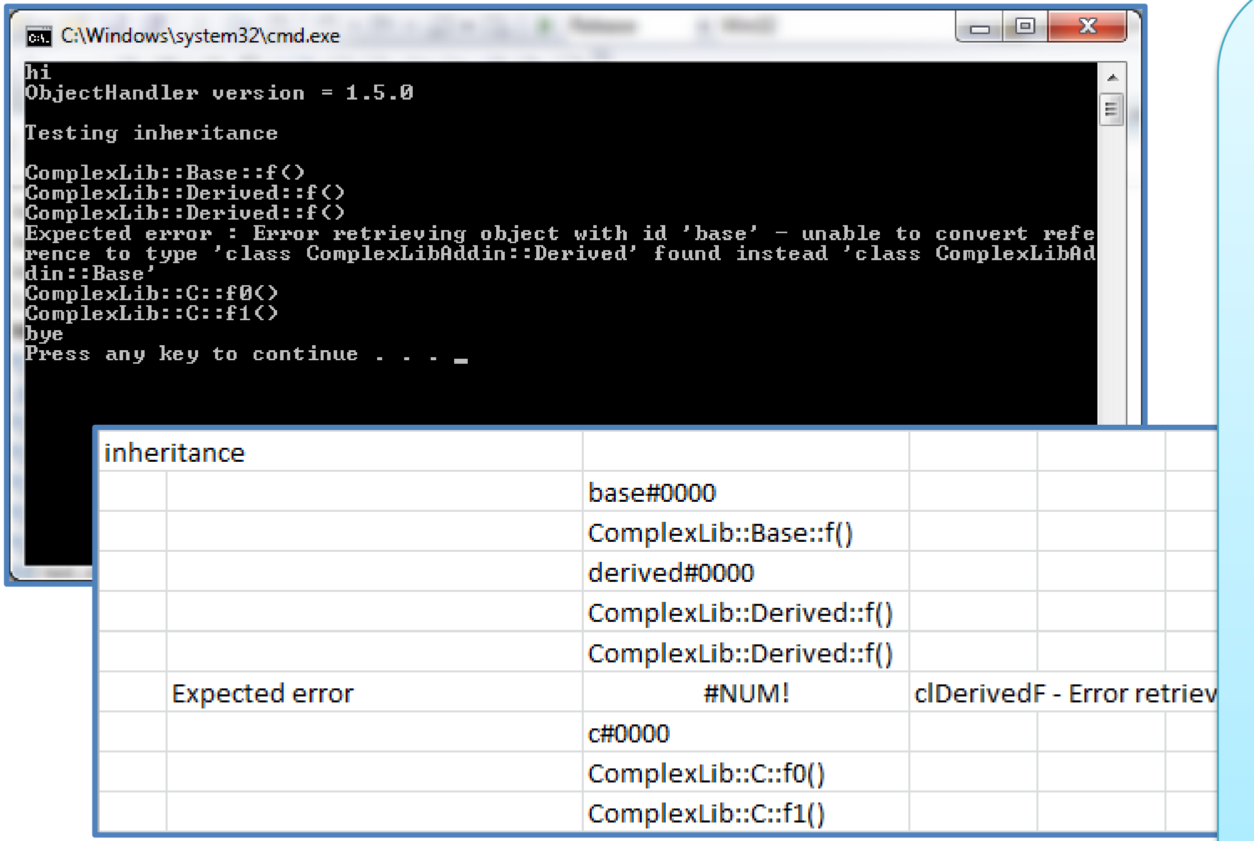
```cpp
#include <iostream>
#include "AddinCpp/add_all.hpp"
#include "oh/addin.hpp"
#include "test_all.hpp"

#ifdef TEST_INHERITANCE

void testInheritance() {
    std::cout << std::endl;
    std::cout << "Testing inheritance" << std::endl;
    std::cout << std::endl;

    ComplexLibAddinCpp::clBase("base");
    std::cout << ComplexLibAddinCpp::clBaseF("base") << std::endl;
    ComplexLibAddinCpp::clDerived("derived");
    std::cout << ComplexLibAddinCpp::clBaseF("derived") << std::endl;
    std::cout << ComplexLibAddinCpp::clDerivedF("derived") << std::endl;
    try {
        std::cout << ComplexLibAddinCpp::clDerivedF("base") << std::endl;
    } catch(const std::exception &e) {
        std::cout << "Expected error : " << e.what() << std::endl;
    }

    ComplexLibAddinCpp::clC("c");
    std::cout << ComplexLibAddinCpp::clAF0("c") << std::endl;
    std::cout << ComplexLibAddinCpp::clBF1("c") << std::endl;
}

#endif
```

For C++, we write by hand some code to test the Addin.

For Excel we enter the same formulas into a workbook (see below).

Inheritance

Example – Step 7 of 7 – Client Code / Spreadsheets



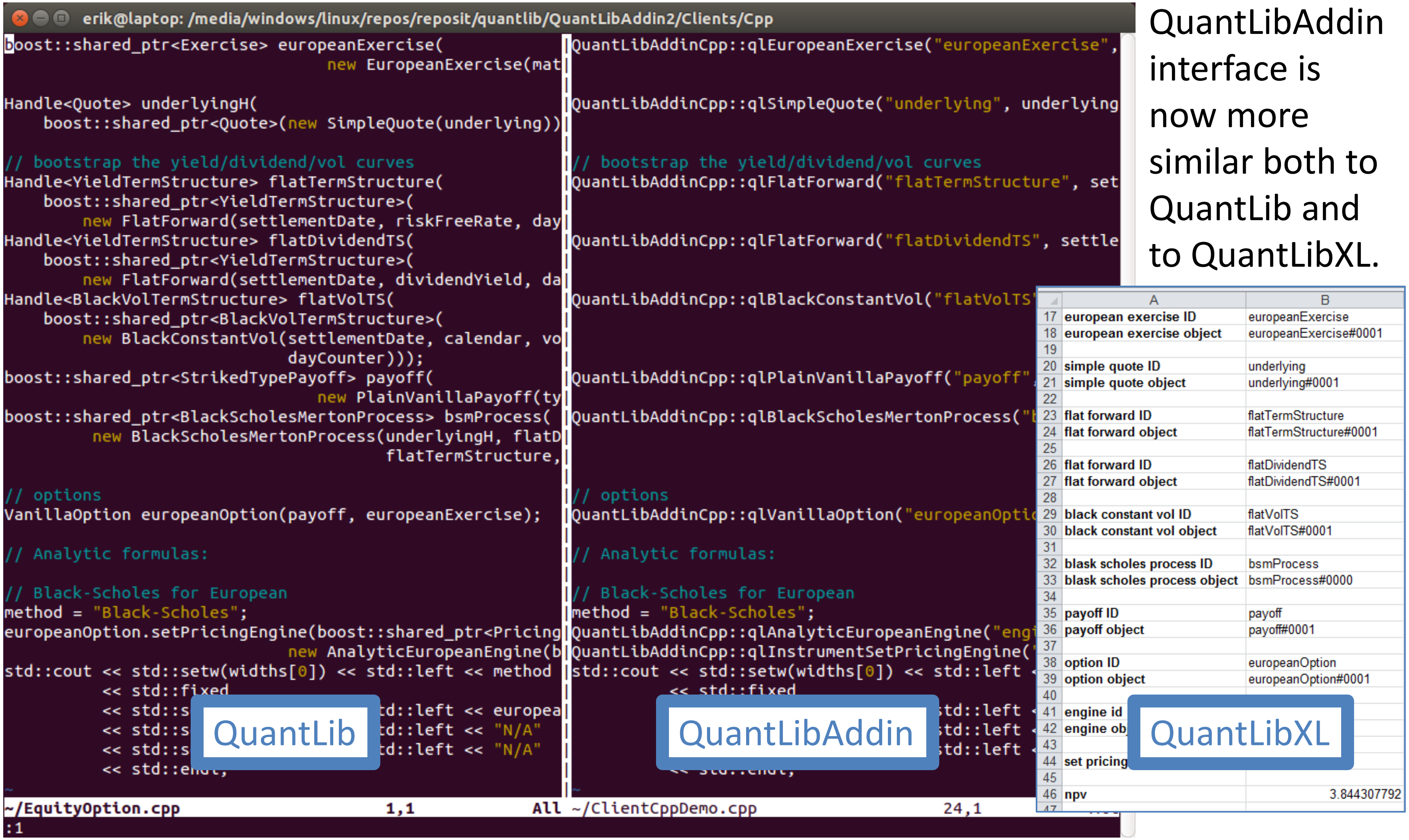
```
hi
ObjectHandler version = 1.5.0

Testing inheritance

ComplexLib::Base::f()
ComplexLib::Derived::f()
ComplexLib::Derived::f()
Expected error : Error retrieving object with id 'base' - unable to convert refe
rence to type 'class ComplexLibAddin::Derived' found instead 'class ComplexLibAd
din::Base'
ComplexLib::C::f0()
ComplexLib::C::f1()
bye
Press any key to continue . . .
```

inheritance					
		base#0000			
		ComplexLib::Base::f()			
		derived#0000			
		ComplexLib::Derived::f()			
		ComplexLib::Derived::f()			
	Expected error	#NUM!	clDerivedF - Error retriev		
		c#0000			
		ComplexLib::C::f0()			
		ComplexLib::C::f1()			

This is the output from the C++ client program, and from the corresponding test workbook.

On both platforms the interface and behavior is the same.

Improved C++ Addin



erik@laptop: /media/windows/linux/repos/reposit/quantlib/QuantLibAddin2/Clients/Cpp

```
boost::shared_ptr<Exercise> europeanExercise(
                                    new EuropeanExercise(mat

Handle<Quote> underlyingH(
    boost::shared_ptr<Quote>(new SimpleQuote(underlying))

// bootstrap the yield/dividend/vol curves
Handle<YieldTermStructure> flatTermStructure(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, riskFreeRate, day
Handle<YieldTermStructure> flatDividendTS(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, dividendYield, da
Handle<BlackVolTermStructure> flatVolTS(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, calendar, vo
                             dayCounter)));
boost::shared_ptr<StrikedTypePayoff> payoff(
                                   new PlainVanillaPayoff(ty
boost::shared_ptr<BlackScholesMertonProcess> bsmProcess(
         new BlackScholesMertonProcess(underlyingH, flatD
                                         flatTermStructure,

// options
VanillaOption europeanOption(payoff, europeanExercise);

// Analytic formulas:

// Black-Scholes for European
method = "Black-Scholes";
europeanOption.setPricingEngine(boost::shared_ptr<Pricing
                             new AnalyticEuropeanEngine(b
std::cout << std::setw(widths[0]) << std::left << method
          << std::fixed
          << std::s              td::left << europea
          << std::s              td::left << "N/A"
          << std::s              td::left << "N/A"
          << std::en
~
~/EquityOption.cpp                          1,1           All
:1
```

```
QuantLibAddinCpp::qlEuropeanExercise("europeanExercise",

QuantLibAddinCpp::qlSimpleQuote("underlying", underlying

// bootstrap the yield/dividend/vol curves
QuantLibAddinCpp::qlFlatForward("flatTermStructure", set

QuantLibAddinCpp::qlFlatForward("flatDividendTS", settle

QuantLibAddinCpp::qlBlackConstantVol("flatVolTS

QuantLibAddinCpp::qlPlainVanillaPayoff("payoff"

QuantLibAddinCpp::qlBlackScholesMertonProcess("

// options
QuantLibAddinCpp::qlVanillaOption("europeanOpti

// Analytic formulas:

// Black-Scholes for European
method = "Black-Scholes";
QuantLibAddinCpp::qlAnalyticEuropeanEngine("engi
QuantLibAddinCpp::qlInstrumentSetPricingEngine("
std::cout << std::setw(widths[0]) << std::left
          << std::fixed
                                 std::left
                                 std::left
                                 std::left
~
~/ClientCppDemo.cpp                         24,1
```

	A	B
17	european exercise ID	europeanExercise
18	european exercise object	europeanExercise#0001
19		
20	simple quote ID	underlying
21	simple quote object	underlying#0001
22		
23	flat forward ID	flatTermStructure
24	flat forward object	flatTermStructure#0001
25		
26	flat forward ID	flatDividendTS
27	flat forward object	flatDividendTS#0001
28		
29	black constant vol ID	flatVolTS
30	black constant vol object	flatVolTS#0001
31		
32	blask scholes process ID	bsmProcess
33	blask scholes process object	bsmProcess#0000
34		
35	payoff ID	payoff
36	payoff object	payoff#0001
37		
38	option ID	europeanOption
39	option object	europeanOption#0001
40		
41	engine id	
42	engine ob	
43		
44	set pricing	
45		
46	npv	3.844307792

QuantLibAddin interface is now more similar both to QuantLib and to QuantLibXL.

Development Environment

Reposit SWIG module

```
repos/reposit/swig/Source/Modules/reposit.cxx
```

Reposit SWIG interface file

```
repos/reposit/swig/Lib/reposit/reposit.swg
```

SimpleLib Example

```
repos/reposit/swig/Examples/reposit/simple
```

ComplexLib Example

```
repos/reposit/swig/Examples/reposit/complex
```

new QuantLibAddin

```
repos/reposit/quantlib/QuantLibAddin2
```

new QuantLibXL

```
repos/reposit/quantlib/QuantLibXL2
```

Typemaps

Reposit defines a series of typemaps. Each typemap is used to generate the required code at a specific point in a source code file.

Buffer	Typemap
rp_val_*	rp_tm_val_prm
rp_val_*	rp_tm_val_dcl
rp_val_*	rp_tm_val_ser
rp_val_*	rp_tm_val_nam
rp_val_*	rp_tm_val_ini
rp_val_*	rp_tm_val_cnv
rp_ser_*	rp_tm_cre_cnv
rp_obj_*	rp_tm_obj_ret
rp_obj_*	rp_tm_obj_rdc
rp_add_*	rp_tm_add_ret
rp_add_*	rp_tm_add_prm
rp_add_*	rp_tm_add_cnv
rp_add_*	rp_tm_add_cll
rp_add_*	rp_add_ret
rp_add_*	rp_tm_add_oh_get
rp_xll_*	rp_tm_xll_cod
rp_xll_*	rp_tm_xll_prm
rp_xll_*	rp_tm_xll_cnv
rp_xll_*	rp_tm_xll_cll_obj
rp_xll_*	rp_tm_xll_cll_val
rp_xll_*	rp_tm_xll_ret
rp_xll_*	rp_xll_get
rp_xll_*	rp_tm_xll_rdc

Normally SWIG typemaps are applied directly to native C++ types, e.g. bool, double, etc.

Reposit instead defines a few placeholders for C++ types. Each addin must map its own types to these placeholders.

rp_tp_double
rp_tp_cnv
rp_tp_crc
rp_tp_enm
rp_tp_enm_cls
rp_tp_add_obj

The application developer has to map the types defined in his library to the type placeholders defined by Reposit. This will be the most difficult step for exporting QuantLib to QuantLibXL.

```
%apply rp_tp_double { LongDouble };
%apply const rp_tp_double & { const LongDouble & };

%apply rp_tp_cnv { Grade };

%apply rp_tp_crc { Grade2 };

%apply rp_tp_enm { AccountType };
%apply rp_tp_enm { Account2::Type2 };
%apply rp_tp_enm_cls { boost::shared_ptr<TimeZone> };
```

Status

Done:

- Working prototype supporting an Equity Option, including addins for C++ and Excel.

To Do:

- Implement support for the rest of the QuantLib functionality – Yield curve bootstrap, price interest rate swap, everything else.
- Implement support for serialization
- For all addin functions, need to autogenerate the trigger/permanent/anonymous parameters
- LibreOffice Calc addin?