lcallenbach lcallenbach@web.de

20170123

LibreOffice Calc addin based on python (version 0.1)

License

Introduction

The OpenOffice / LibreOffice Calc addin exists for a longer time. Currently you can download an alternative implementation of addin code generation under the official QuantLib download path.

So: why another implementation?

I have used QuantLib at my employers for clients in the financial area for many years (audit and independent validation). It is c++ based and so not everyone understands it (especially the interaction of objects in QuantLib).

The other/old implementation for LibreOffice (and OpenOffice) is based on XML files describing the addin functions. In short the XML files describe the interface of the addin functions. For each QuantLib object a function has to be defined (manually) to define the mapping of the interface data to the QuantLib object in c++ code (lets call these files constructor definition files). This mapping can be quite complex or with a lot of predefined values (not known in the interface description). So it is difficult to identify which data are of which type and have what meaning. Additionally you do not have a predefined relation to the QuantLib object – e.g. you can change the order of the variables in the addin function or even omit them in the XML files (but you have to define them in the constructor definition files). Both things make it complicated to look at the underlying QuantLib object if you have a question concerning the parameters in the addin function (and you need a c++ aware programmer to find the relevant code in the QuantLib addin code and in the QuantLib code).

In the current implementation I have used another approach which is related to the QuantLib c++ code. It is similar to SWIG – but you need not redefine all of the classes and reproduce their interdependence (inheritance and so forth). The intention is to define a python tuple like

```
1   (“Constructor",
2   "ql/termstructures/yield/zerocurve.hpp",
3   "QuantLib::ZeroCurve(std::vector<QuantLib::Date> Dates, std::vector<QuantLib::Real> ZeroRates,
4         QuantLib::DayCounter DayCounter=QuantLib::Actual365Fixed())", "",
5   "qlZeroCurve")
```

It describes that a constructor should be defined (an object of a QuantLib class should be created and stored) with parameters given by the third and forth line (which are identical with the parameters in the QuantLib declaration header file), with an addin function named “qlZeroCurve” and with a class declaration in the file “ql/termstructures/yield/zerocurve.hpp”. This definition is enough for code generation and has the feature, that all variables are in the same order as in the c++ declaration file. For constructors one further variable will be added in the first place for the name of the object in the Calc addin. In short: the parameters are identical to the parameters in QuantLib header files for classes and all interface code will be generated automatically.

I have chosen a similar approach for member function of QuantLib classes, e.g.

```
1    ("MemberFunction",
2    "ql/termstructures/yieldtermstructure.hpp",
3    "QuantLib::YieldTermStructure()",
4    "QuantLib::DiscountFactor discount(QuantLib::Date Date)",
5    "qlYieldTSDiscount")
```

This is the definition of the addin function “qlYieldTSDiscount” calling the QuantLib function “discount” of the class “YieldTermStructure”. As for constructors a further argument will be added in the first position of the parameter list of the addin function for the name of the yield curve (for which the discount value should be calculated).

If you know the structure and interdependence of QuantLib classes and their functions/procedures it is simple to understand the creation of objects in the addin and the meaning of variables (in the help of each variable the QuantLib variable type will be displayed, e.g. “QuantLib::Date” for the variable “Date”).

I have structured the document in the following chapters: in the next chapter I give a short overview of how the code will be generated. In the next chapter I describe the current types of QuantLib addin objects that can be defined.

Overview of c++ addin code generation

The following picture presents a quick overview:

1.	*read configuration*	*class GensrcConfiguration*
2.	*read metadata*	*class ParseMetaData*
	read type conversions	
	read constructors, memberfunctions, functions, enumerated classes and enumerated types	
3.	*generate LibreOffice IDL file and addin code (storage of code in files) in addin.calc.py*	
	IDL and addin specific code	*addin.calc.py*
	code for enumerated types	*codegeneration.enumeratedobjects.py*
	code for the base classes	*codegeneration.enumeratedclasses.py*

Types are analysed for each variable by the class Parameter (parser.parameter.py). The information for constructors is analysed by the class Constructor (parser.constructor.py), for functions by the class MemberFunction (parser.memberfunction.py).

In the addin you have to decide whether a variable is an object of the repository, a native type (e.g. float, string) or should be created “on the fly” (e.g. enumerated class). The idea is the following: if a type is not listed in the conversions then it is a type of an object (and is stored with a name as key in the repository). Native types and enumerated types are defined in the conversions.