The grand idea

Map the compile-time C++ classes and respective run-time objects to User Interface compatible elements.
The grand idea (example)

- A **QuantLib developer** has just implemented a new class called "**Swaption**" that has a method called "**CalculateHedge**" which returns a vector of **Swap** instruments plus a vector of notionals.

- An **Excel user** who accesses QuantLib through Deriscope will see in the wizard a new type called "**Swaption**" that has the method "**CalculateHedge**", which produces two columns of data:

- One column containing objects of type **Swap** and one column containing plain numbers (*)

**C++ World**

**Excel World** (generated by Deriscope from the C++ world)
IoData: Bridge between C++ and UI

- C++ classes and data can be implemented in many different ways
- User Interfaces offer limited Data Representation
- Deriscope’s trick:
  - 1. Map from C++ to special class called IoData
  - 2. Map from IoData to each supported User Interface (*)

C++ object of type Swaption

- Expiry = 30.11.18
- Strike = 2%
- Notional = 1M
- Underlying = 

IoData

0010110111011000 1011111000010101 0101010001010000 0101100100100010 0001111111111110 10000010101

Excel UI

XLOPER

XML UI

Text
Deriscope consists of two main Dlls: **Kernel & Export**

The Kernel Dll contains all the analytics, including the QuantLib library.

The Export DLL contains classes Export<X>, where X is a Kernel class deriving from the root class Object (*)

Excel API → Zero code maintenance here!

XML API → Zero code maintenance here!
Export DLL (IoData): Key-Value

- Input/output data are mostly expressed as **Key-Value pairs**
- **Key** = Text label
- **Value** = scalar, array, object or set of key/value pairs
- **Key-Value pairs** may be dynamically declared as optional
- **Key-Value pairs** apply to all User Interfaces (*)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30/360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>01-Apr-00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>01-Jun-01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comma separated input format: 
=AccrualFraction(C5,C6,C7)
Note the lack of transparency with regard to the meaning of the input.

Whole range input format: 
=AccrualFraction(G5:H7)

DayCount= 30/360
Start= 01-Apr-00
End= 01-Jun-01
Export DLL: Advantages

- Similar User Interface across Products and across APIs.
- APIs protected from Kernel changes.
- UI Type Inheritance mimics the C++ Type Inheritance
- C++ debugging transferred to the GUI level
- Hierarchical classification of exported Functions
- Overloading of exported Functions
- Efficient pool management (*)
UI Benefits from C++-level Reflection

- List of available exported classes
- List of available exported functions within a given class
- Required input data for a given function
- Detailed description of any class
- Detailed description of any function
- Detailed description of any key-value pair input (*)
Drag & Drop across applications

- Any Deriscope object can be cast as an xml file and thus delivered to another application that can read such an XML file

- Effectively every object can be serialized in a recursive fashion so that all its dependencies are also serialized. The result is a stateless object in the form of a text file

- The reverse process is also possible. A properly formatted XML file can give rise to an object with a "state" in the pool of persistent objects maintained by the application that reads the XML file

- Drag & Drop is just a simple application of all this (*)
Kernel Architectural Aspects

- **C++** with great emphasis on **Object Oriented** principles.
- Proprietary **smart pointer**
- **Reflexion** capabilities through template-based **metaprogramming**
- Definition of Concepts in terms of a few **orthogonal** concepts
- Deriscope has a relatively large number of pure header files (*)
Parent type of any object that describes a tradable instrument

Examples of types deriving from **Tradable** are Bond, Stock, Swap etc

The Market Price of a **Tradable A** with respect to another **Tradable B** at some time t is defined as the number of units of the **Tradable B** that are exchanged for one unit of the **Tradable A**

The Theoretical Price of a **Tradable A** with respect to a **Tradable B** at some time t relies on additional objects of type Market and Model

The landmark identifier of the **Tradable** class is its "Price" function (*)

QuantLib User Meeting, Deriscope, Nov 30, 2017
Parent type of any object that describes a financial variable

Examples of types deriving from Quotable are Stock Price, FX Rate

The Value of a Quotable A at time t is a Measure

For t = 0, the Measure typically collapses to a single number

The Theoretical Value of a Quotable A with respect to a Quotable B at some time t relies on additional objects of type Market and Model

The landmark identifier of the Quotable class is its "Value" function
This type serves as a container of "Market Data", i.e. of the objective information that is available as of a given time \( t \).

A **Market** object is defined as a collection of "**Valuation**" objects.

A "**Valuation**" object is a pair of a Quotable and its associated Value.

It is required **input** to the **Price** and **Value** functions.

It is also the **output** of the **Price** and **Value** functions! (*)

---

QuantLib User Meeting, Deriscope, Nov 30, 2017
This type serves to hold everything that can be seen as "subjective", i.e. information that can be potentially disputed

- **Model** objects hold assumptions, such as an assumed interpolation of zero rates or the Gaussian dynamics of a stock price

- Every Kernel class X can have a corresponding class **Model<X>**

- Deriscope discourages the creation of **Model** classes M that are not of the form **Model<X>** (* )
This Type has a simple definition:

- It contains all those objects that do not already fall under any of Tradable, Quotable, Market, or Model (*)
Top Level Class Hierarchy Benefits

- Concept Orthogonalization makes the maintainance of a huge number of Classes possible

- Elimination of “orphan” classes
  - Class BlackScholes derives from Model<TradablePrice>

- Enhanced teamwork without overlap

- Single "Market" class
  - Additional inherited classes, such as YieldCurve, can be added for convenience but are not strictly necessary

- Automatic availability of "Model" classes: $T \rightarrow \text{Model}<T> \ (*)$
Run-time Composition of Dynamic Types

- Synthesize new types out of old ones during run time

- There are certain classes that make this possible. Two important such classes are described below:

- ** Tradable Price**
  - Manufactures a *Quotable* object that represents the "ratio" of two given * Tradable* objects
  - Example: *Discount Factor* type created by “dividing” a *Zero Bond* with its own *Currency*

- ** Quotable Group**
  - Manufactures a *Quotable* object that represents the "equivalence class" defined through a given *Quotable* object plus an "equivalence relationship"
  - Example: "*Yield Curve*" type created out of a specific *Discount Factor* object and an "equivalence relationship" that regards two *Discount Factor* objects equivalent if they have the same currency (*)
Part 2
User Interface
Deriscope is an Excel Add-In that enables the user to work with QuantLib in Excel.

Deriscope contains a wizard that allows the user to generate spreadsheet formulas by choosing predefined types and functions.

Deriscope places full context-relevant documentation at the user’s fingerprints. All relevant information is usually just a mouse click away.
Spreadsheet Formatting and Data Input

- All formulas generated by *Deriscope* have a similar look:

- This entry is **optional**. If present, it defines the **handle name** of the

- This cell contains the **Value** for the Payoff Key. The **Value** here is the object created in cell A15. The **green** colour indicates that the cell creates only a link.

- The formula is shown in formula bar. It returns a text in **red** colour that is the **handle name** of the created object.

- This **Key-Value** pair defines the **Type**. Its presence is **mandatory** only when the **Function** is **Static**.

- This **Key-Value** pair defines the **Function**. Its presence is **mandatory**.

- These two cells hold a **Key-Value** pair

- This is a **Key** and is coloured **black**. It has always an equal sign = at the end. It is part of the input to the ds formula, although rarely changed by the user.

- This is a **Value**. Its **blue** colour indicates it can be edited.
About “Objects” and “Handle Names”

- In Deriscope almost everything is an object.
- By “object” we mean any set of data (i.e. numbers and text) held in memory that can be accessed through a unique text identifier.
- This unique identifier is referred as “handle name”.
- Handle names act in spreadsheet as pointers to the respective objects.
Deriscope’s “building block” approach

♦ In Deriscope generally the user proceeds in two steps:

  – 1) Build each required object separately.

  – 2) Assemble the objects together and call the desired function.

♦ The next screen shows how a Stock Option is priced.
Stock Option Pricing Example
About Types

♦ Every Object has a unique Type associated with it.

So Types follow a hierarchy similar to the folders in a file structure.
About Functions

- Every Type has a set of predefined Functions associated with it.
- **Local Function**: If it is invoked by an already created object.
- **Static Function**: If it is invoked by a type.
- **Key-Value Pairs**: The typical form of a function’s input data.
Deriscope supports C++-style “inheritance” at spreadsheet level.

**Example:** An object of type “Fixed Coupon Bond” can be used in a context where an object of type “Bond” is required. In this case the object’s **dynamic (actual)** type is “Fixed Coupon Bond”, but its **static (context)** type is “Bond”.

**Language used:** Type “Fixed Coupon Bond” **derives** from type “Bond” or is a **subtype** of type “Bond”.

---

[Spreadsheet image with formulas and objects showing dynamic and static types.]
Functions can also be virtual!

A virtual function can be used with an object of unknown dynamic (actual) type.
Some types never have their own objects. They only have subtypes. These are called abstract types.

The type “Tradable” is abstract. No object of actual type “Tradable” exists!

The type “Bond”, which derives from “Tradable”, is also abstract. No object of actual type “Bond” exists!

In the folder analogy an abstract type is equivalent to a folder that is restricted to contain only subfolders.
The fundamental Types in Deriscope

Object

- ** Tradable**
  - Describes tradable financial products.
  - Examples of derived types: Stock, Bond, Currency, Swap

- ** Quotable**
  - Describes financial variables that can be evaluated at any time.
  - Examples of derived types: Stock Price, Interest Rate, FX Rate

- ** Market**
  - Collection of Quotables with their valuations for given date.

- ** Model**
  - Contains all the assumptions needed to carry out a particular calculation, that is not already part of the above types.
The “Valuation” Type

- The type "Valuation" derives from "Market".
- A typical Market object is just a collection of Valuation objects.
- Pricing takes in Valuation objects and returns a Valuation object.
Using the “Valuation” Type

This object is just a collection of the “elementary” Valuation objects defined below. It is used as input in the alternative valuation of the same swap rates in the boxes to the right.

This is a Valuation object containing the value of SwapRateObject_A.1 as calculated by SomeModel.1.

It may or may not agree with the original value of 4%.
Using the Wizard to View an Object

- Click on 🕵️ to view the contents of the object in that cell.
- Click on any cell to display relevant information within the wizard.
Using the Wizard to Create Formulas

Input Area
- Type Selector
- Object Selector
- Function Selector

Browse Area
- Press this Button to paste the selected function in the spreadsheet

Info Area
- Description of Stock Option
- QuantLib issues identified

QuantLib User Meeting, Deriscope, Nov 30, 2017
The Function Selector

- When clicked upon, the “**Function Selector**” displays all available Functions.

- **Origin** - Type where function is defined
  - **S** <-> **static** Functions
  - **L** <-> **local** Functions
  - **NV** <-> Non-Volatile

- All **concrete** types have a **Create** function
- All types deriving from ** Tradable** have a **Price** function

<table>
<thead>
<tr>
<th>Name</th>
<th>Origin</th>
<th>C</th>
<th>Vol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Stock Option</td>
<td>S</td>
<td>NV</td>
</tr>
<tr>
<td>Get Settlement Date</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
<tr>
<td>Price</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
<tr>
<td>Implied Vol</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
<tr>
<td>Get Market Spot Price</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
<tr>
<td>Clone</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
<tr>
<td>Get Equivalent Tradable</td>
<td>Tradable</td>
<td>L</td>
<td>NV</td>
</tr>
</tbody>
</table>

QuantLib User Meeting, Deriscope, Nov 30, 2017
The Browse Area

- The symbol 🟢 indicates that the input value is an object.

- Click on 🟢 to display the object’s contents and edit them.

- The symbol 🟢 indicates that the input value can be any value out of an enumerated list of choices.

- Click on 🟢 to display the choices and select one of them.

- Click on any element to display relevant information in the Info Area below.
No comments or dropdowns are needed within the spreadsheet!

- The user may ask for help by selecting any key.
- ... or get a dropdown with the possible values by selecting any value.
XML Export of Object

- Any object can be permanently stored (exported) as an xml file.
XML Import of Object

- Inversely an *object* may be created (**imported**) by reading a previously exported xml file.
Any _Deriscope formula_ can be permanently stored (exported) as an xml file and subsequently sent to a support person for analysis.

- If a comment is present, it will be also part of the xml file.
- This reduces the need for sending whole spreadsheets if issues arise.