

QuantLib User Meeting  
30 November 2017

# Deriscope

by

Ioannis Rigopoulos, owner of [deriscope.com](http://deriscope.com)

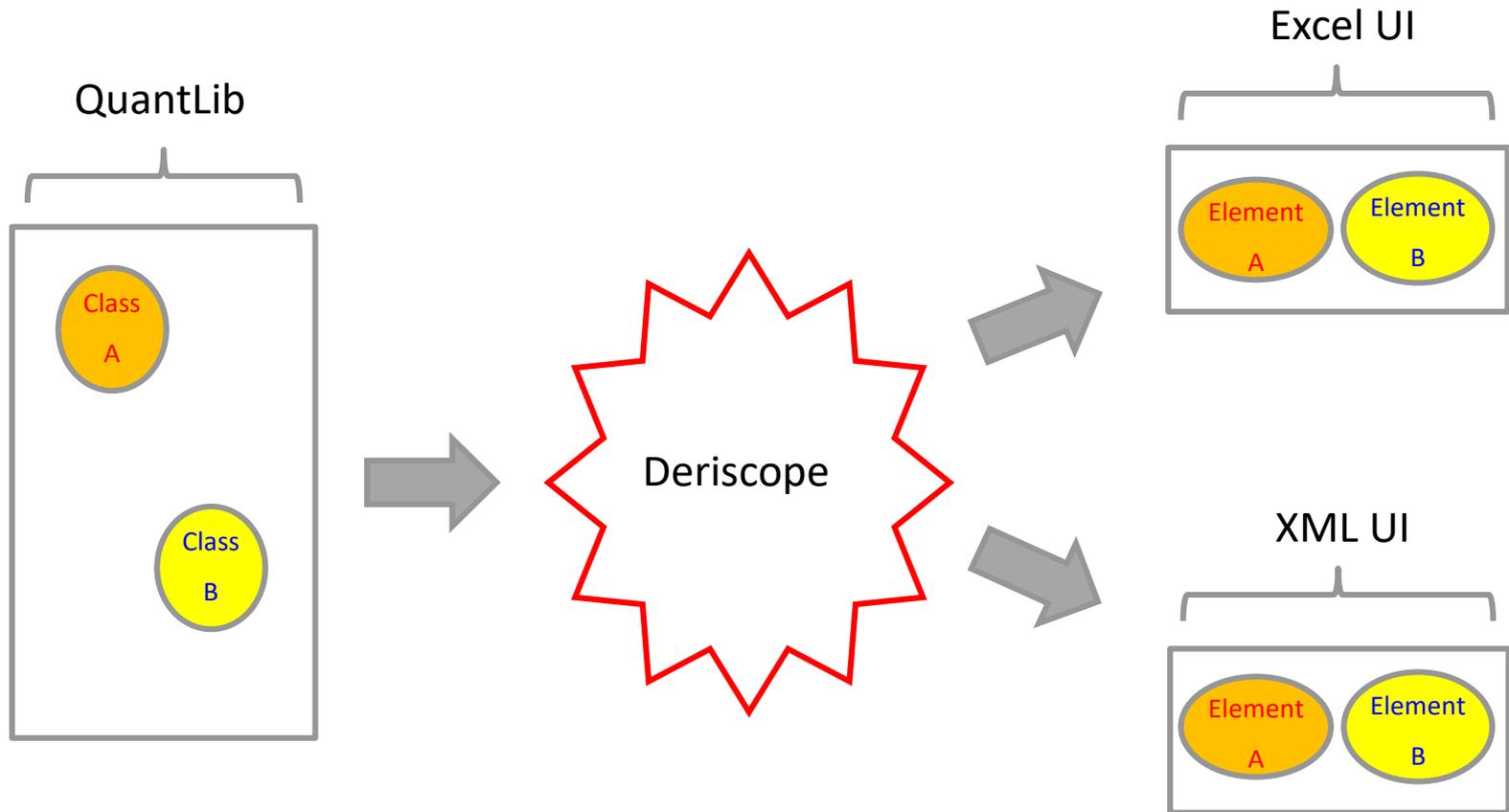
# Part 1

# Architecture

# The grand idea



- ◆ Map the compile-time C++ classes and respective run-time objects to User Interface compatible elements

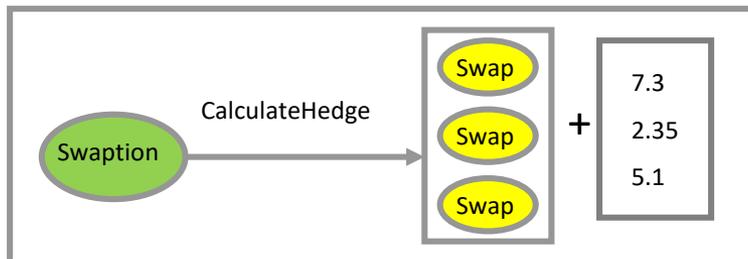


# The grand idea (example)

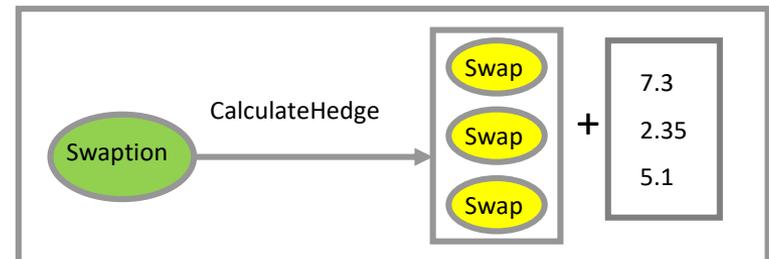


- ◆ A **QuantLib developer** has just implemented a new class called "**Swaption**" that has a method called "**CalculateHedge**" which returns a vector of **Swap** instruments plus a vector of notionals
- ◆ An **Excel user** who accesses QuantLib through Deriscope will see in the wizard a new type called "**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



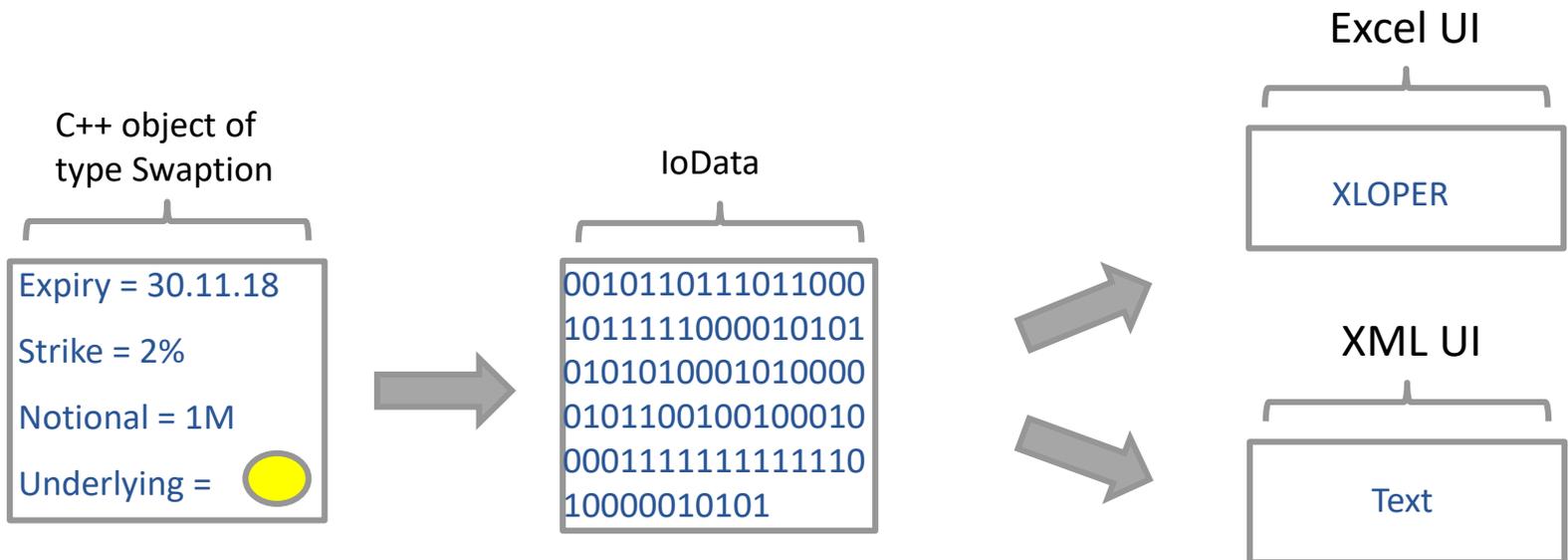
## Excel World (generated by Deriscope from the C++ world)



# IoData: Bridge between C++ and UI



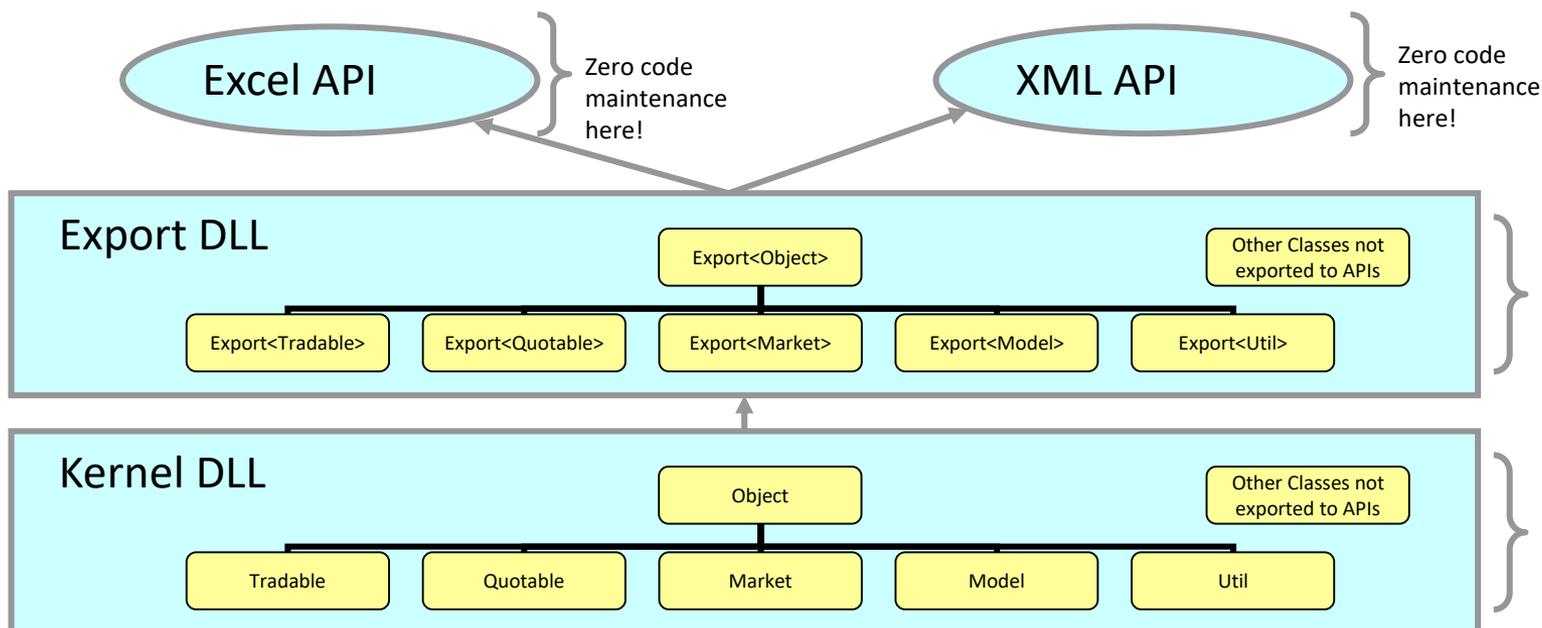
- ◆ C++ classes and data can be implemented in many different ways
- ◆ User Interfaces offer limited Data Representation
- ◆ Deriscope's trick:
  - 1. Map from **C++** to special class called **IoData**
  - 2. Map from **IoData** to each supported **User Interface** (\*)



# DLL & Class Structure



- ◆ 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 (\*)



# 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 (\*)

	A	B	C	D	E	F	G	H	I
1									
2									
3			1.17					1.17	
4									
5			30/360				DayCount=	30/360	
6			01-Apr-00				Start=	01-Apr-00	
7			01-Jun-01				End=	01-Jun-01	
8									
9									

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)



- ◆ 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 (\*)



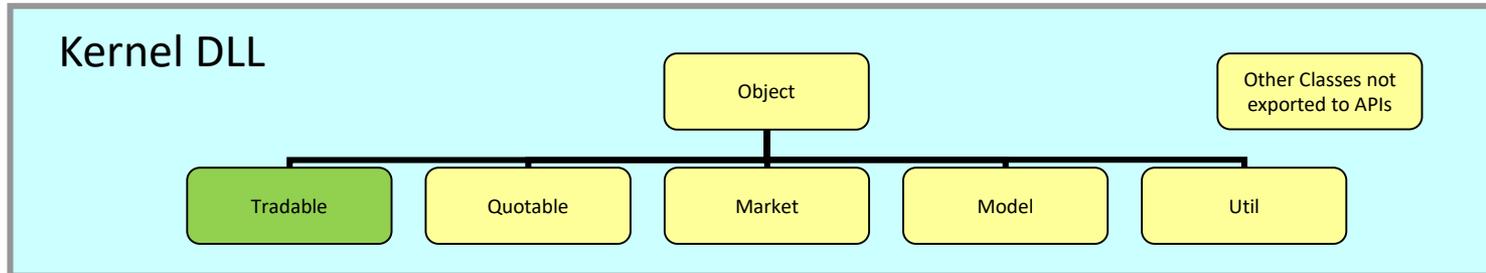
- ◆ List of available exported classes
- ◆ List of available exported functions within a given class
- ◆ Required input data for a given function
- ◆ Detailed description of any class
- ◆ Detailed description of any function
- ◆ Detailed description of any key-value pair input (\*)



- ◆ 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 (\*)

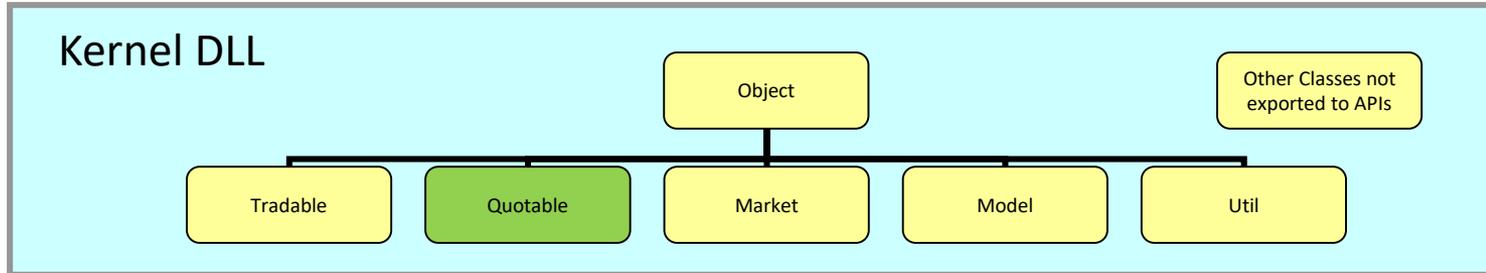


- ◆ C++ with great emphasis on **Object Oriented** principles.
- ◆ Proprietary **smart pointer**
- ◆ **Reflexion** capabilities through template-based **metaprogramming**
- ◆ Definition of Concepts in terms of a few **orthogonal** concepts
- ◆ Deriscope has a relatively large number of pure header files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

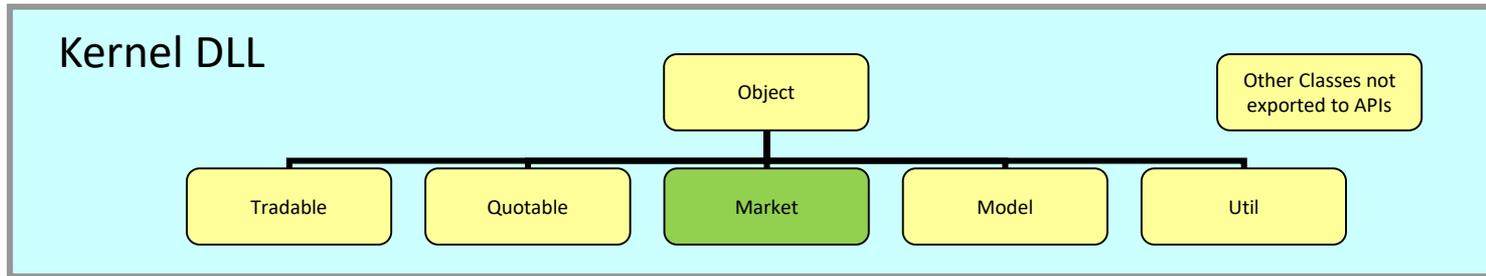


- ◆ Parent type of any object that describes a tradable instrument
- ◆ Examples of types deriving from **Tradable** are Bond, Stock, Swap etc
- ◆ The Market Price of a **Tradable A** with respect to another **Tradable B** at some time  $t$  is defined as the number of units of the **Tradable B** that 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 (\*)

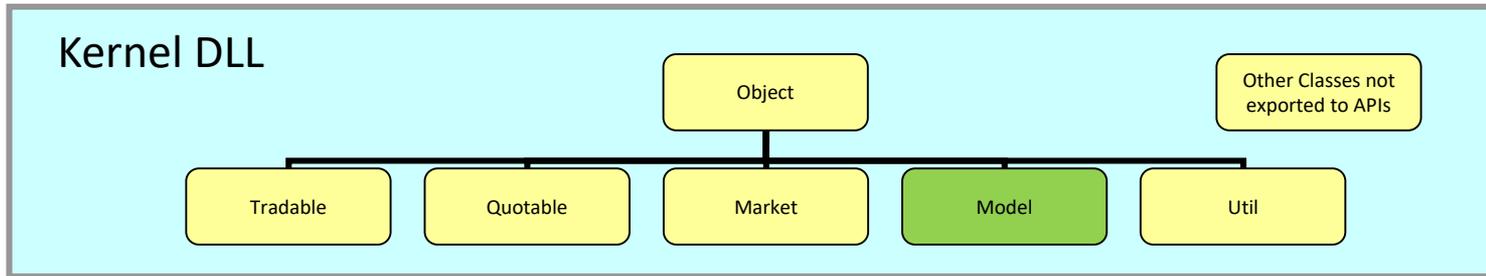
# Top Level Class Hierarchy - Quotable



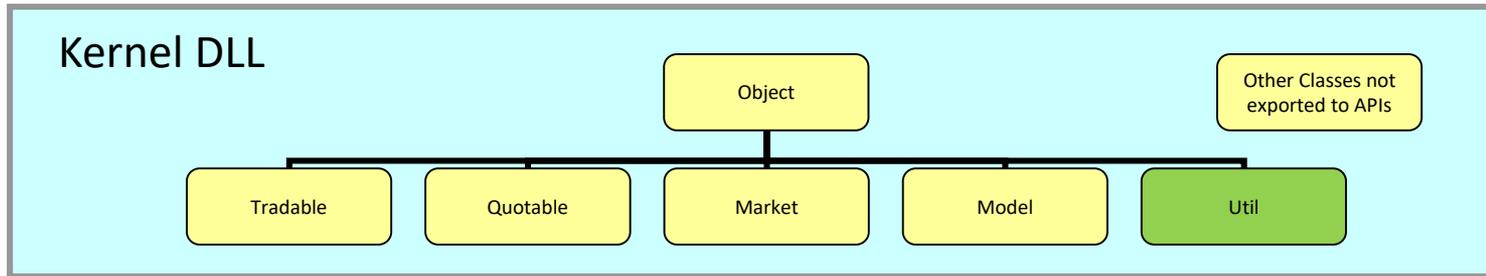
- ◆ Parent type of any object that describes a financial variable
- ◆ Examples of types deriving from **Quotable** are Stock Price, FX Rate
- ◆ The Value of a **Quotable** A at time t is a Measure
- ◆ For  $t = 0$ , the Measure typically collapses to a single number
- ◆ The Theoretical Value of a **Quotable** 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! (\*)



- ◆ 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 (\*)



- ◆ Concept Orthogonalization makes the maintenance 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 --> Model<T> (\*)**



- ◆ 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:

The screenshot shows a spreadsheet with two cells containing ds formulas. The first cell (C3) contains a formula for a 'Rip-off Option' and the second cell (C15) contains a formula for a 'Payoff'. Annotations explain the formatting and meaning of various parts of these formulas.

Cell	Formula
C3	<code>&amp;Rip-off Option.1 Type= Stock Option Function= Create Handle= Rip-off Option Stock= %SIEB.DE DE Payoff= &amp;Payoff_C15:2.1 Exercise Type= European Expiry= 17.11.2018 Barrier= No Barrier</code>
C15	<code>&amp;Payoff_C15:2.1 Type= Payoff Function= CREATE Payoff Type= Vanilla Direction= Call Strike= 100</code>

**Annotations:**

- Green box:** This entry is **optional**. If present, it defines the **handle name** of the
- Orange box:** 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.
- Yellow box:** The formula is shown in formula bar. It returns a text in **red** colour that is the **handle name** of the created **object**.
- Blue box:** This **Key-Value** pair defines the **Type**. Its presence is **mandatory** only when the **Function** is **Static**.
- Yellow box:** This **Key-Value** pair defines the **Function**. Its presence is **mandatory**.
- Green box:** These two cells hold a **Key-Value** pair
- Yellow box:** This is a **Value**. It's **blue** colour indicates it can be edited.
- Yellow box:** 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.



# Deriscope's “building block” approach



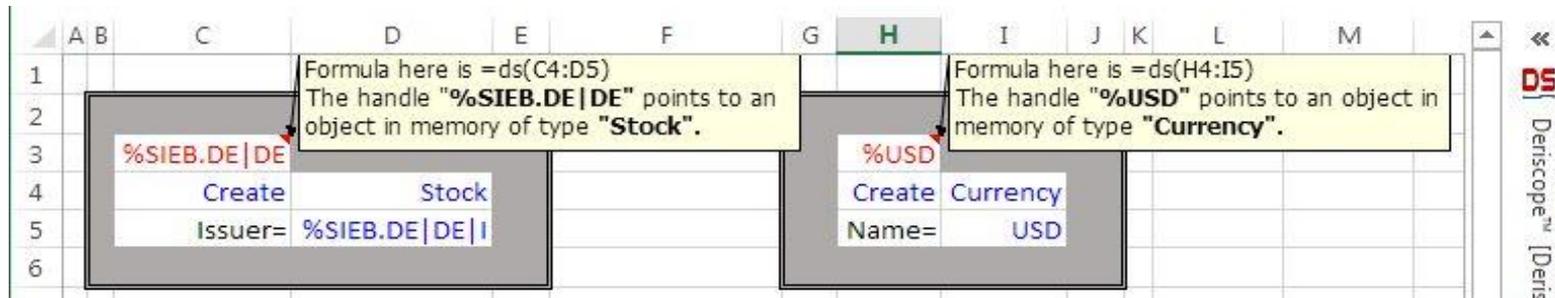
- ◆ In *Deriscope* generally the user proceeds in two steps:
  - **1)** Build each required **object** separately.
  - **2)** Assemble the **objects** together and call the desired function.
  
- ◆ The next screen shows how a Stock Option is priced.

# Stock Option Pricing Example



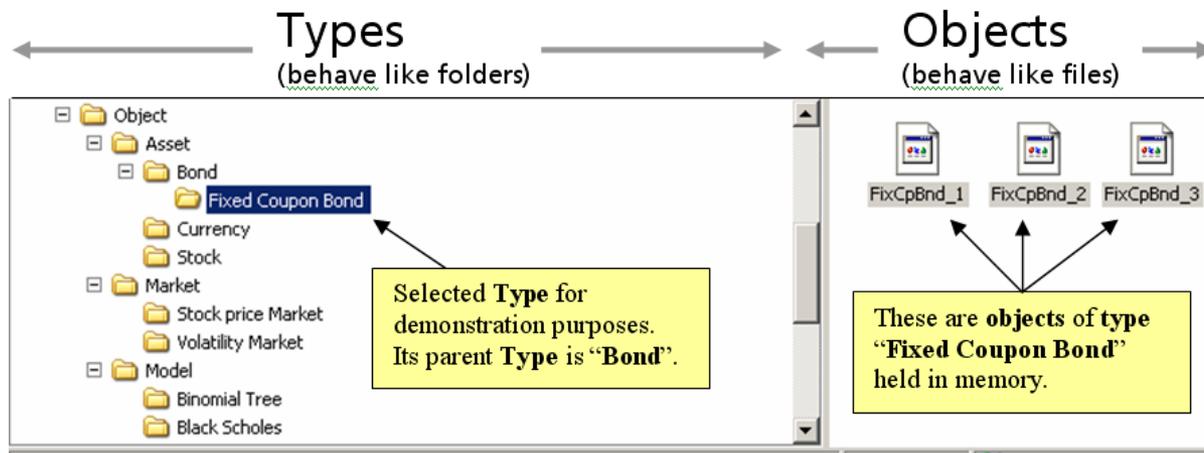
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															
30															
31															
32															
33															
34															

- ◆ Every *Object* has a unique **Type** associated with it.



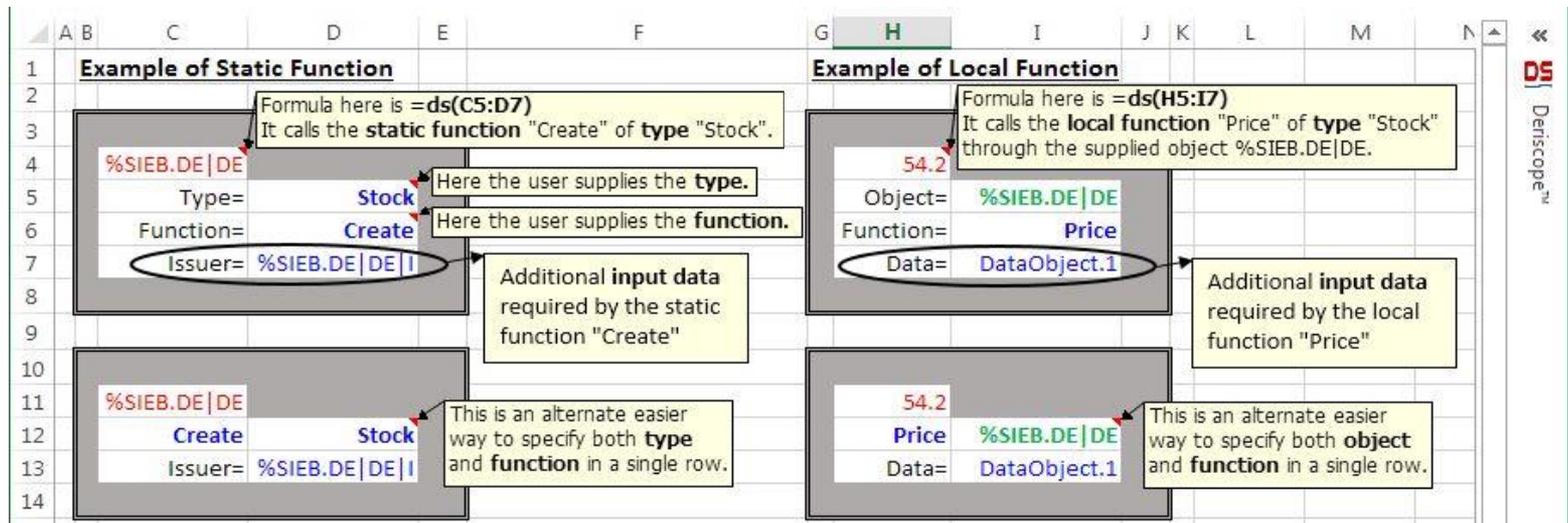
The screenshot shows a spreadsheet with two columns of object creation formulas. The first column (C4:D5) shows the formula `=ds(C4:D5)` with a tooltip explaining that the handle `"%SIEB.DE|DE"` points to an object of type `"Stock"`. The second column (H4:I5) shows the formula `=ds(H4:I5)` with a tooltip explaining that the handle `"%USD"` points to an object of type `"Currency"`. The spreadsheet also shows the underlying formulas: `Issuer= %SIEB.DE|DE|I` and `Name= USD`.

- ◆ 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.



The screenshot displays two columns of spreadsheet data, each with annotations explaining function types and input data.

**Example of Static Function (Columns C-E):**

- Row 2: Formula `=ds(C5:D7)` calls the static function "Create" of type "Stock".
- Row 4: Object `%SIEB.DE|DE`.
- Row 5: Type = `Stock`. Annotation: "Here the user supplies the type."
- Row 6: Function = `Create`. Annotation: "Here the user supplies the function."
- Row 7: Issuer = `%SIEB.DE|DE|I`. Annotation: "Additional input data required by the static function 'Create'".
- Row 11: Object `%SIEB.DE|DE`.
- Row 12: Function = `Create`, Type = `Stock`. Annotation: "This is an alternate easier way to specify both type and function in a single row."
- Row 13: Issuer = `%SIEB.DE|DE|I`.

**Example of Local Function (Columns H-M):**

- Row 2: Formula `=ds(H5:I7)` calls the local function "Price" of type "Stock" through the supplied object `%SIEB.DE|DE`.
- Row 4: Value `54.2`.
- Row 5: Object = `%SIEB.DE|DE`.
- Row 6: Function = `Price`.
- Row 7: Data = `DataObject.1`. Annotation: "Additional input data required by the local function 'Price'".
- Row 11: Value `54.2`.
- Row 12: Function = `Price`, Object = `%SIEB.DE|DE`. Annotation: "This is an alternate easier way to specify both object and function in a single row."
- Row 13: Data = `DataObject.1`.



- ◆ 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”.

Formula here is =ds(C10:D13)  
The created object's **type** - both **static** and **dynamic** - is Fixed Coupon Bond, which derives from **type** Bond.

Here an object of **dynamic type** Bond is expected, according to the specification of the function "Create". We pass the object %FxdCpnBnd\_C9:6.1 whose **static type** is Fixed Coupon Bond. Everything works since this **static type** happens to be a subtype of the **type** Bond expected here.

%FxdCpnBnd_C9:6.1	
Create	Fixed Coupon Bond
Maturity=	30. Nov 27
Coupon=	4%
Issuer=	6m

%FxdCpnBnd_C4:6.1	
Create	Bond Option
Maturity=	%FxdCpnBnd_C9:6.1
Expiry=	30. Nov 18
Strike=	95

# Spreadsheet C++ (Virtual Functions)



- ◆ Functions can also be *virtual*!
- ◆ A *virtual* function can be used with an *object* of unknown *dynamic (actual)* type.

Formula here is `=ds(E9:F10)`  
It will work as long as the calling object **SomeCallingObject** derives from **Bond**.

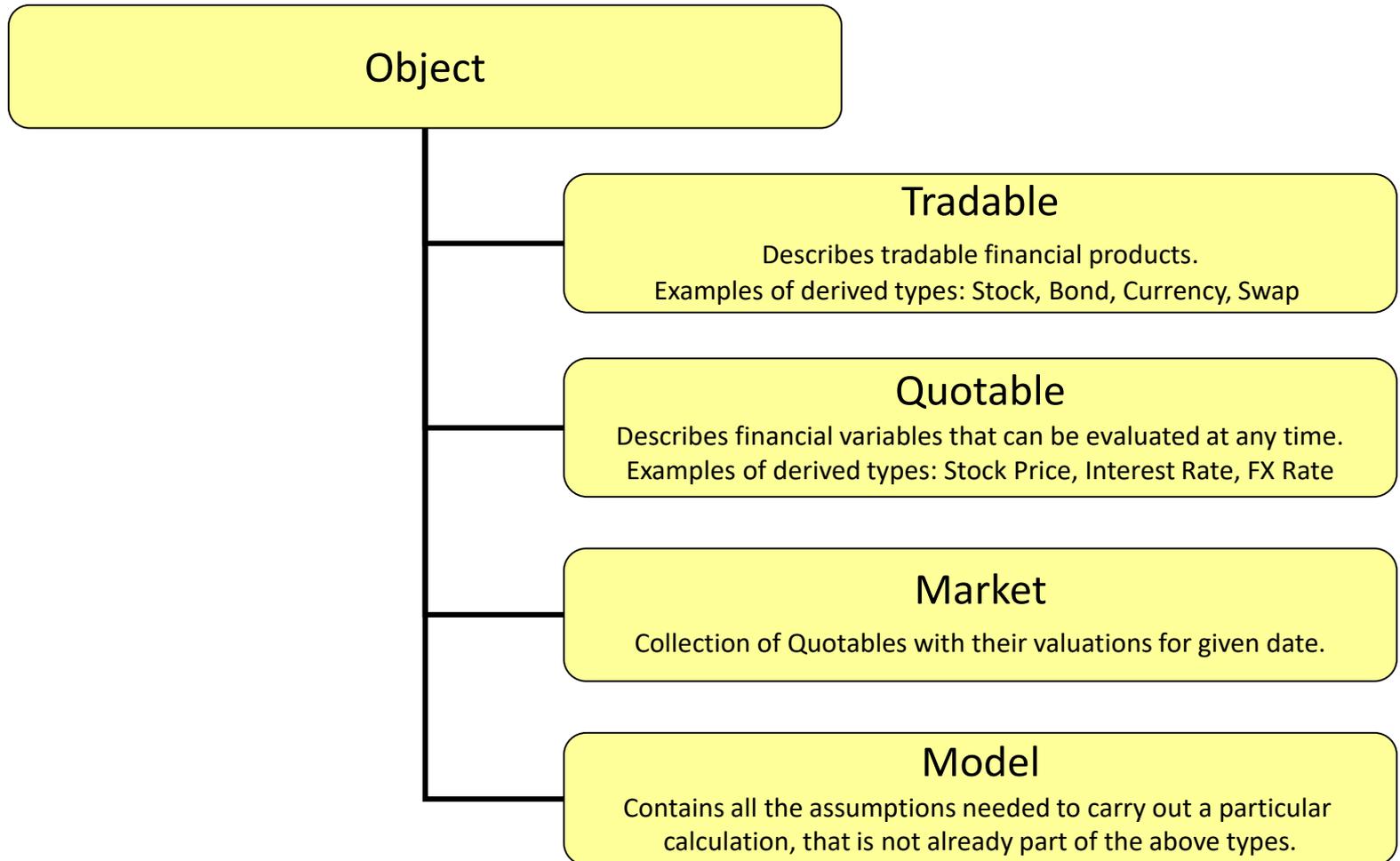
According to the "Calc Yield" specification, an object of **static (context) type** **Bond** is expected here. This means the **dynamic (actual) type** can be either **Bond** or some of its **subtypes**.

"Calc Yield" specification, is a **virtual function** declared within **type Bond**. It is implemented differently in each **Bond** subtype. Here "Calc Yield" is called by the object **SomeCallingObject** whose **dynamic (actual) type** is not necessarily known

DS | DeriscopeInterface.xlsx

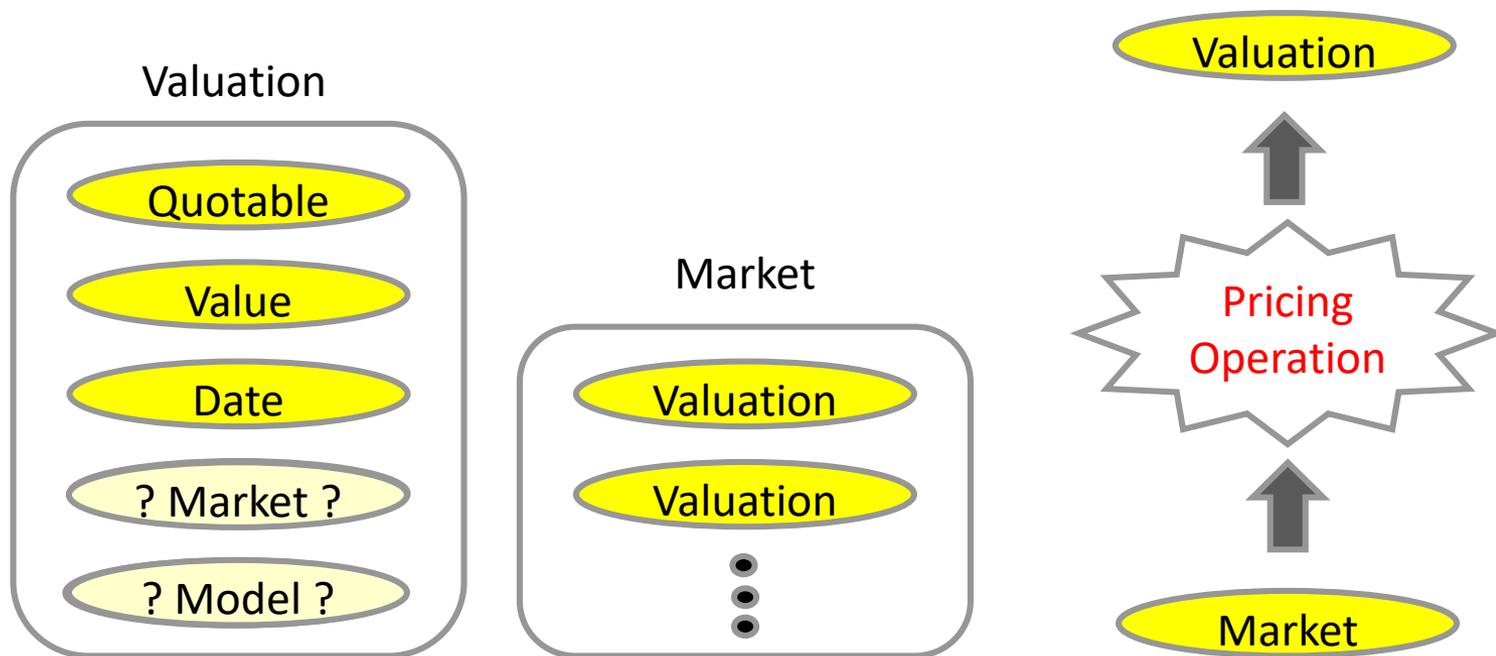


- ◆ Some *types* never have their own *objects*. They only have *subtypes*. These are called **abstract types**.
- ◆ The *type* “**Tradable**” is **abstract**. No *object* of *actual type* “**Tradable**” exists!
- ◆ The *type* “**Bond**”, which derives from “**Tradable**”, is also **abstract**. No *object* 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 “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*.

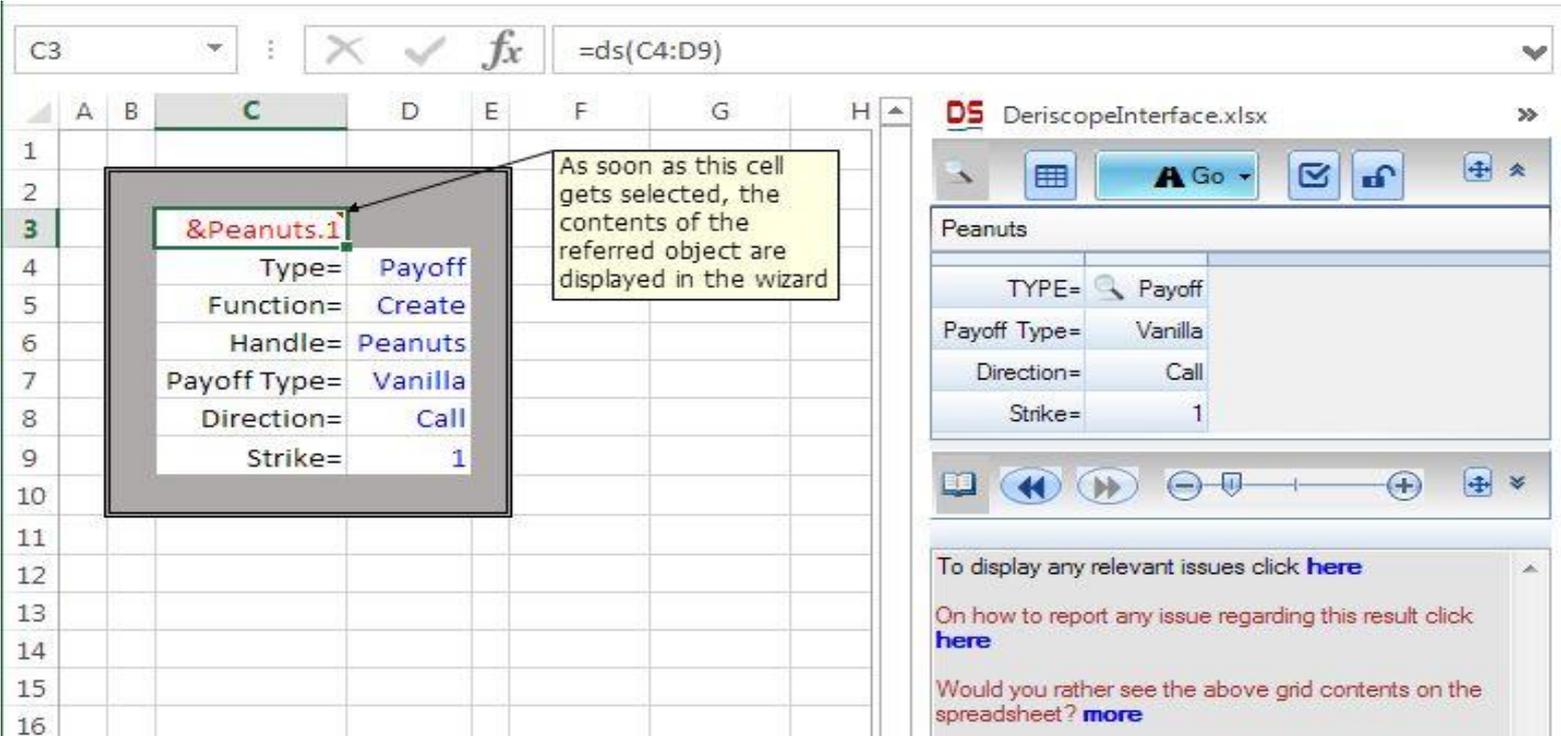




# Using the Wizard to View an Object



- ◆ Click on  to view the contents of the object in that cell.
- ◆ Click on any cell to display relevant information within the wizard.



The screenshot shows a spreadsheet with a wizard interface. The spreadsheet cell C3 contains the formula `=ds(C4:D9)` and the text `&Peanuts.1`. The wizard displays the object's properties:

Type=	Payoff
Function=	Create
Handle=	Peanuts
Payoff Type=	Vanilla
Direction=	Call
Strike=	1

As soon as this cell gets selected, the contents of the referred object are displayed in the wizard

DS DeriscopeInterface.xlsx

Peanuts

TYPE=	Payoff
Payoff Type=	Vanilla
Direction=	Call
Strike=	1

To display any relevant issues click [here](#)

On how to report any issue regarding this result click [here](#)

Would you rather see the above grid contents on the spreadsheet? [more](#)

# Using the Wizard to Create Formulas



The screenshot shows the Deriscope Wizard interface for creating a formula. The window title is "DeriscopeInterface.xlsx". The date is set to "18.11.2017". The breadcrumb navigation shows "Option > Vanilla Option > Stock Option". The "Create" button is highlighted. Below the navigation, there are several icons: a calendar, a "Go" button, a checkmark, and a lock. The main area displays the "Stock Option::Create" form with the following fields:

Stock=	%SIEB.DEIDE
Payoff=	\$Payoff#1
Exercise Type=	European
Expiry=	18.11.2018
Barrier=	No Barrier

At the bottom, there is a navigation bar with arrows and a zoom slider. Below the navigation bar, the "Type Stock Option" section provides a description: "Stock Option is a Tradable that represents a Vanilla Option where the underlying is a Stock. The pricing methodology is specified in Model[Vanilla Option]. The following QuantLib issues have been identified: here".

## Input Area

Type Selector

Object Selector

Function Selector

## Browse Area

Press this Button to paste the selected function in the spreadsheet

## Info Area

# The Function Selector

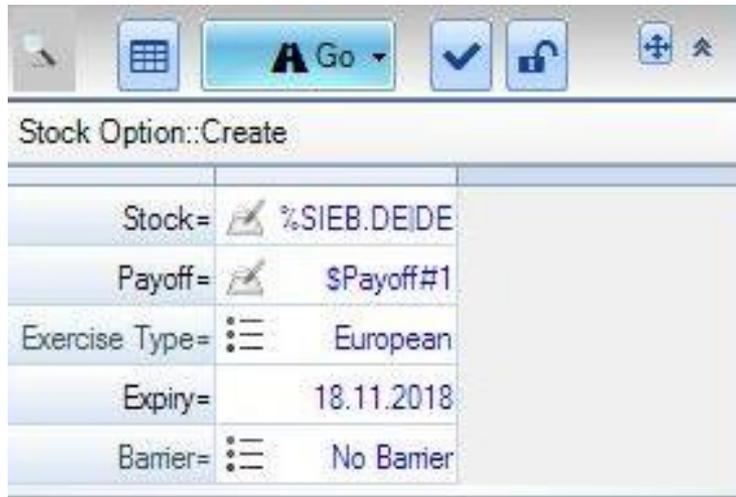


Name	Origin	C	Vol
Create	Stock Option	S	NV
Get Settlement Date	Tradable	L	NV
Price	Tradable	L	NV
Implied Vol	Tradable	L	NV
Get Market Spot Price	Tradable	L	NV
Clone	Type	L	NV
Get Equivalent Tradable	Tradable	L	NV

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

- ◆ When clicked upon, the “**Fuction Selector**” displays all available **Functions**.
- ◆ **Origin** -> Type where function is defined
- ◆ **S** <-> **static** Functions
- ◆ **L** <-> **local** Functions
- ◆ **NV** <-> **Non-Volatile**

# The Browse Area

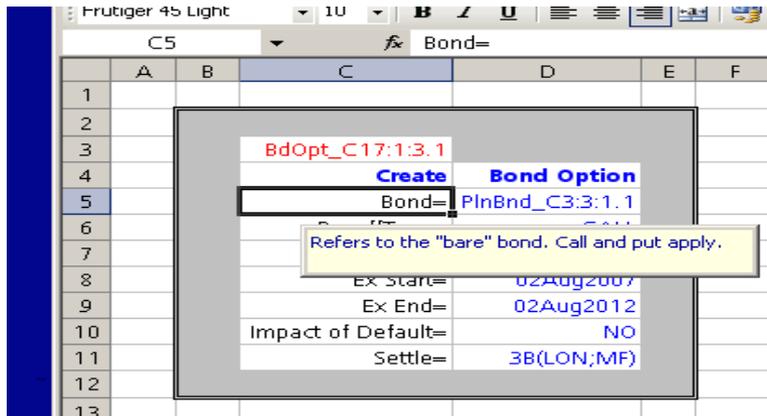


- ◆ The symbol indicates that the input value is an object.
- ◆ Click on to display the object's contents and edit them.
- ◆ The symbol indicates that the input value can be any value out of an enumerated list of choices.
- ◆ Click on to display the choices and select one of them.
- ◆ Click on any element to display relevant information in the Info Area below.

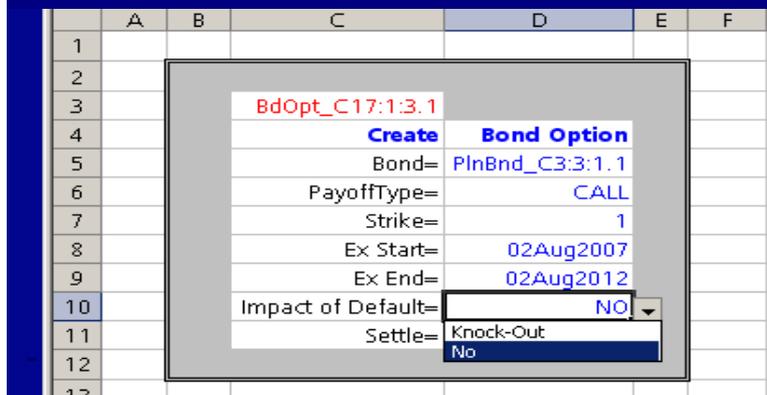
# Getting Information in Spreadsheet



- ◆ 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.

The screenshot shows an Excel spreadsheet with a context menu open over a cell containing the formula `=ds(C4:D11)`. The spreadsheet data is as follows:

	A	B	C	D	E	F
1						
2						
3			<code>&amp;Rip-off Option.1</code>			
4			Type=	Stock Option		
5			Function=	Create		
6			Handle=	Rip-off Option		
7			Stock=	%SIEB.DE DE		
8			Payoff=	&Payoff_C15:1.1		
9			Exercise Type=	European		
10			Expiry=	17.11.2018		
11			Barrier=	No Barrier		
12						
13						
14						
15						
16						
17						
18						

The context menu is open, showing the following options:

- Paste Function
- Paste Function w/o links (grouped input)
- Paste Function w/o links (split input)
- Run Function in Task Pane
- Create Demo Workbook
- Paste Object Contents
- Export Object to XML**
- Export Excel Formula to XML
- Load Object from XML
- Load Excel Formula from XML

The background shows the DeriscopeInterface.xlsx application window with a navigation pane on the right containing the following items:

- 18.11.2017
- eq << Option >> Vanilla Option >> Stock Option
- U Rip-off Option
- Go
- Stock Option
- %SIEB.DE|DE
- %SIEB.DE|DE
- &Payoff\_C15:1.1
- European
- Exercise Periods Start= 17.11.2018

# XML Import of Object



- ◆ Inversely an *object* may be created (**imported**) by reading a previously exported xml file.

The screenshot displays the Deriscope software interface. On the left, an Excel spreadsheet is visible with columns A, B, and C, and rows 1 through 18. Cell C2 is selected. On the right, the Deriscope task pane is open, showing a date of 18.11.2017 and various icons. A context menu is open over the task pane, listing several options: Paste Function, Paste Function w/o links (grouped input), Paste Function w/o links (split input), Run Function in Task Pane, Create Demo Workbook, Paste Object Contents, Export Object to XML, Export Excel Formula to XML, Load Object from XML (highlighted), and Load Excel Formula from XML.

# XML Export of Deriscope Formula



- ◆ 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.

The screenshot shows an Excel spreadsheet with the following data in cells C4:D7:

Object	Value
Object=	&Rip-off Option.1
Function=	Price
Models=	&VanOptMdl_C30:5.1
Markets=	&Mkt_M19:5.1

The formula in cell C3 is `=ds(C4:D7)`. The spreadsheet also shows a comment box in cell E3 with the text: "Happy Deriscope User: I think that the option price cannot be that big here. Could you please have a look?". The background shows the Deriscope interface with various settings and a "Go" button.