



# XLSTAT User Conference

## June 7 - 8 2007

### Paris, France

Programming with XLSTAT and VBA

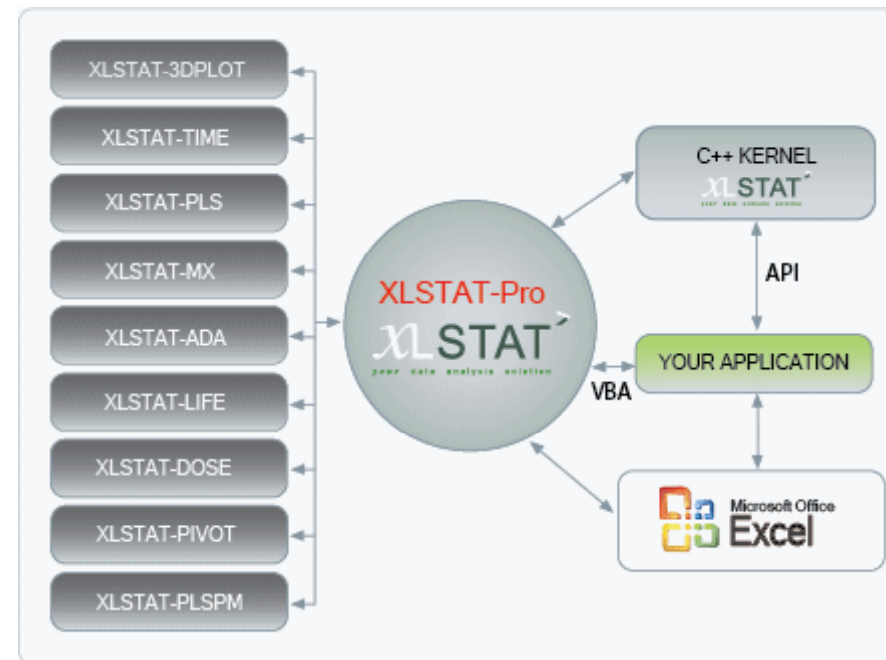


# Outline

- XL XLSTAT Architecture
- XL VBA – Introduction
- XL VBA – Syntax
- XL VBA – Debugging
- XL VBA – Using external code

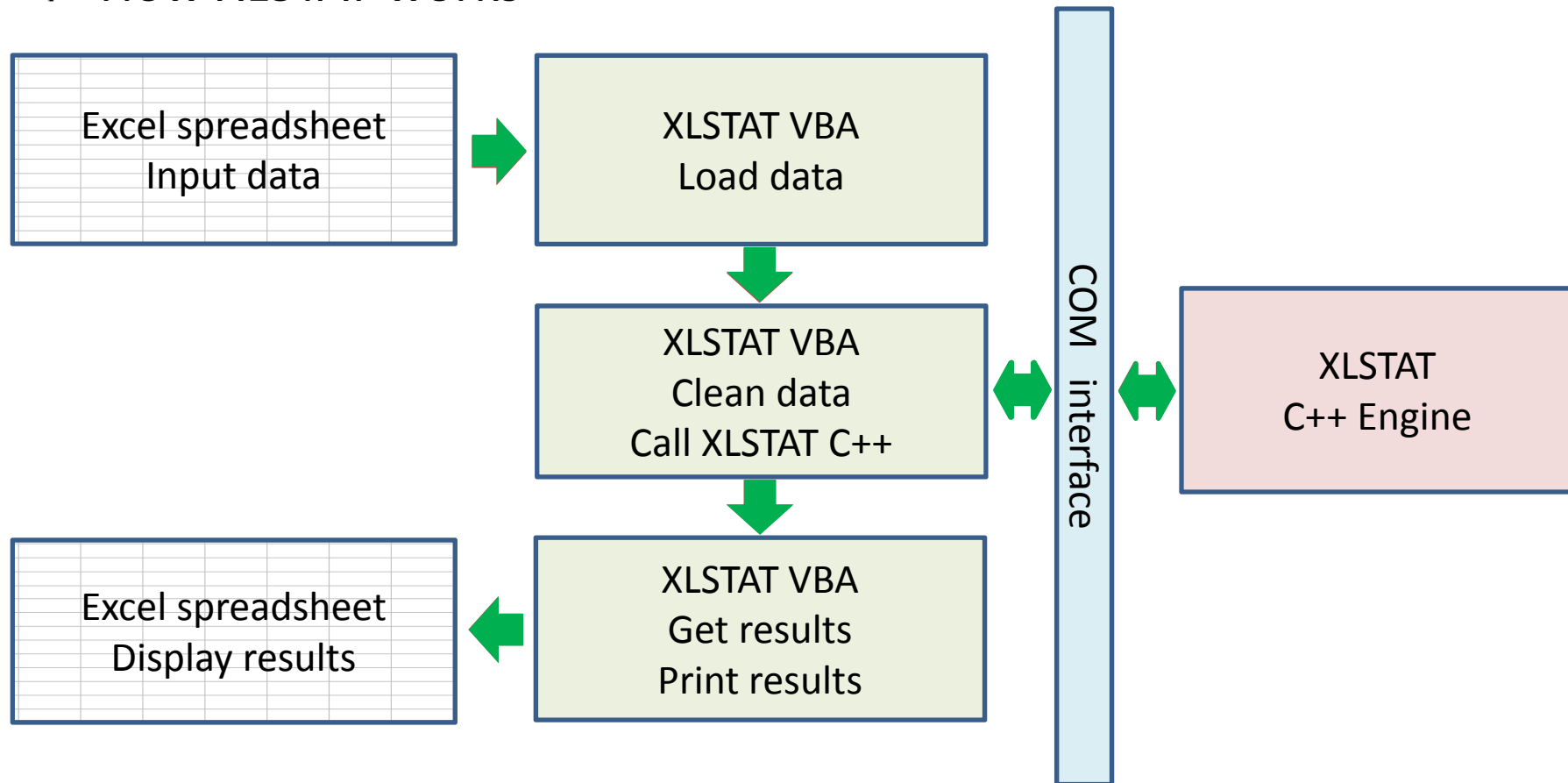
# XLSTAT Architecture

- Since version 2006
  - XLSTAT uses the COM interface to let the VBA and C++ layers communicate.
  - COM stands for Component Object Model. It is a standard that allows components developed with different languages to communicate
- **Consequence:** You can develop your own applications using either VBA, or other programming languages to call XLSTAT VBA or C++ functions



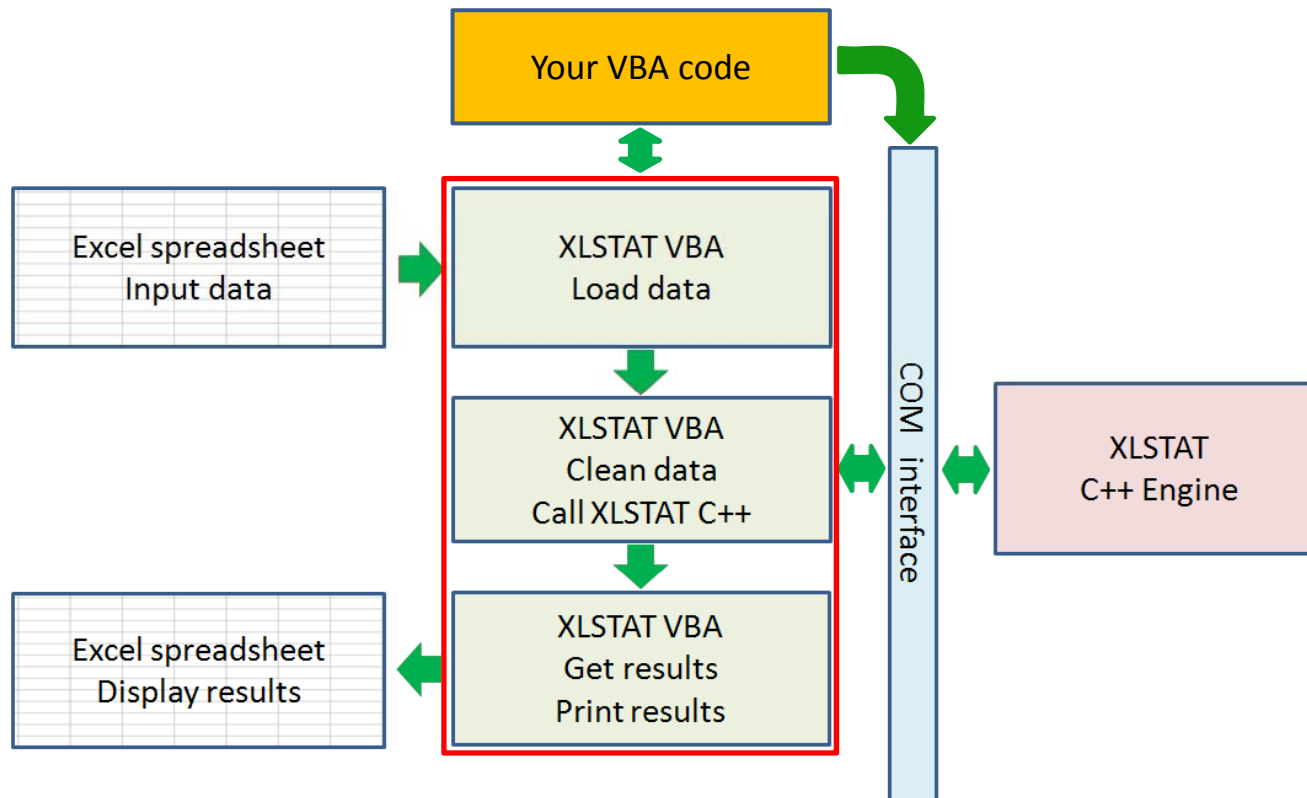
# How does it work?

## How XLSTAT works



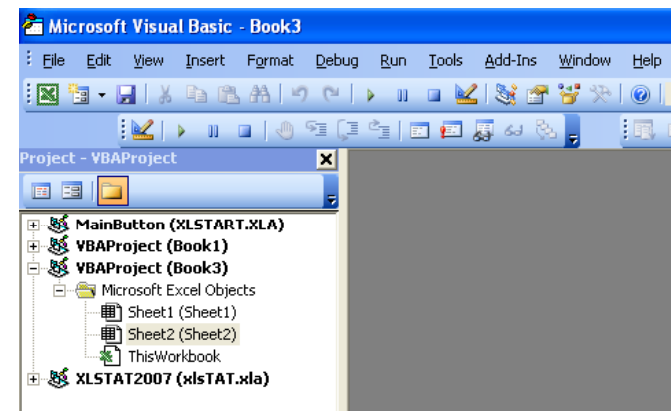
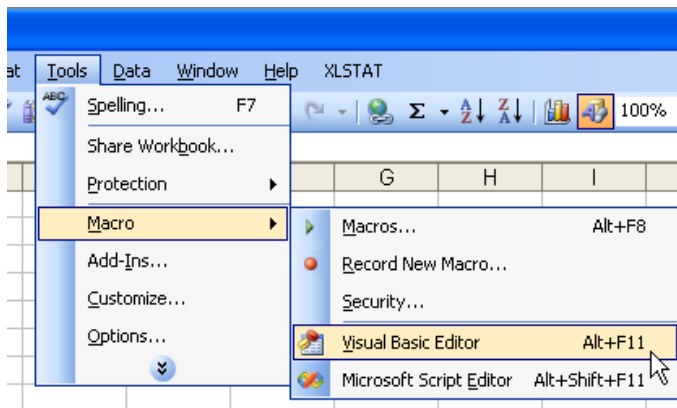
# Adding your own code

- You can use the XLSTAT VBA layer, or use XLSTAT as a bridge to the XLSTAT C++ layer



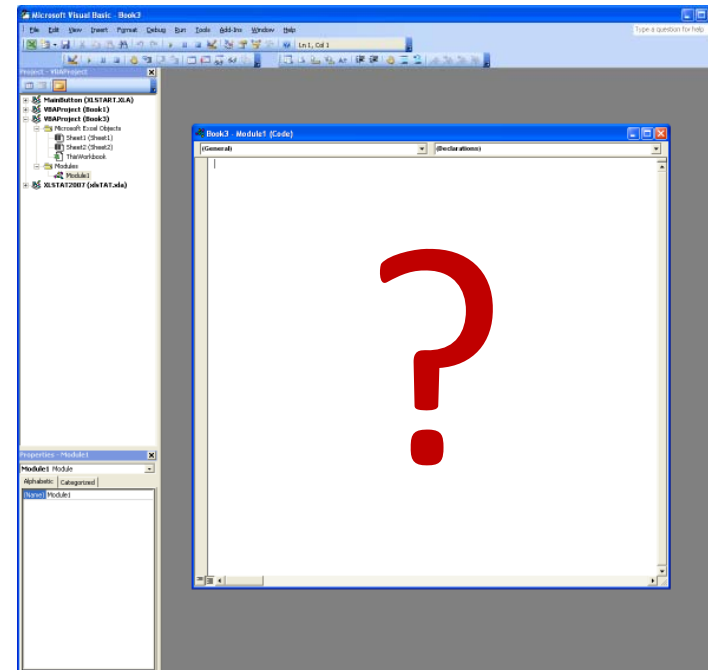
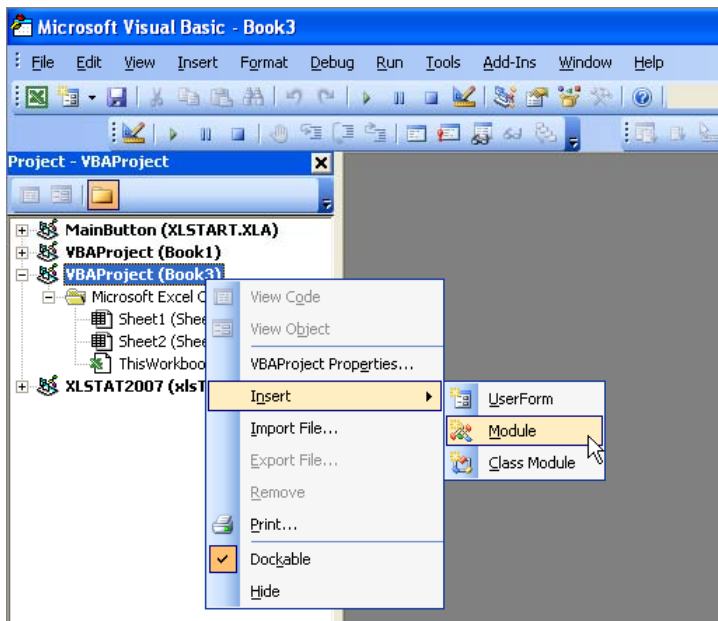
# VBA – What is it?

- VBA is a TLA (3 letters acronym) for Visual Basic for Applications. First version was released with Excel 97.
- VBA is a special version of Visual Basic aimed at automating simple tasks, and developing basic up to complex applications within Office programs.
- To start the VBA Editor:



# VBA – How does it look like?

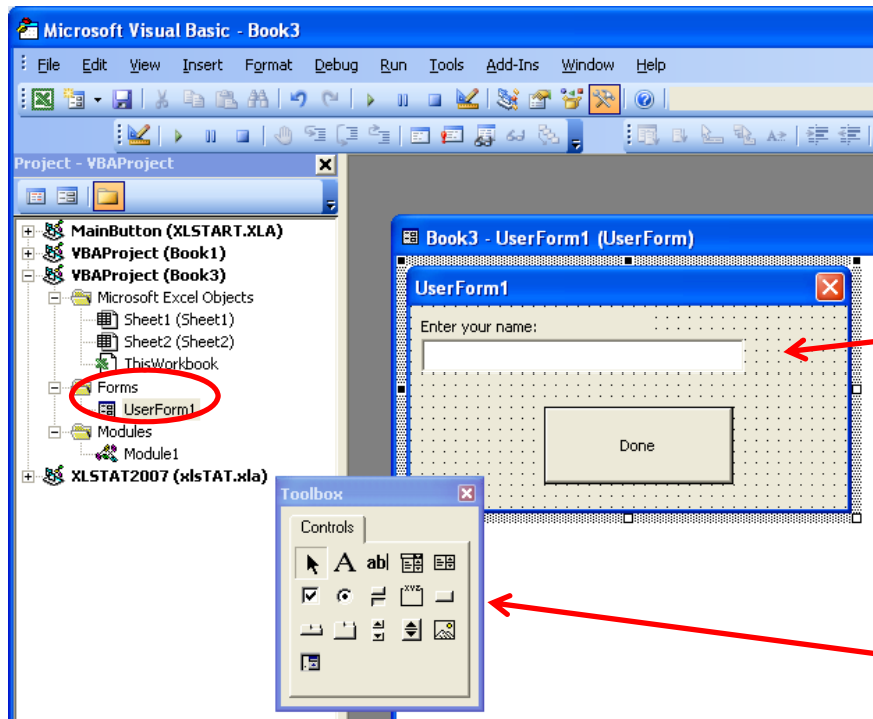
- 1 The Microsoft Visual Basic Editor allows managing VBA projects



- 1 Natural language is not (yet?) possible: you need to enter VBA code

# VBA – What you can do with it (1)

- ① Design forms:
  - Forms are dialog boxes used to interact with the user

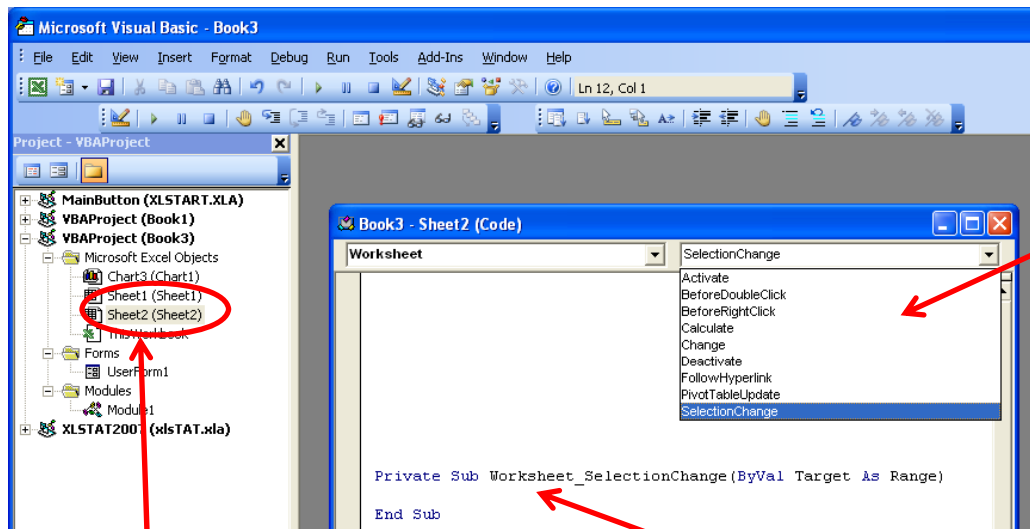


No programming is required to design the form. But it allows controlling it and manage the actions

The toolbox allows you to add controls to the form

# VBA – What you can do with it (2)

- Add code to control and react to events that happen in a workbook, a worksheet, or a chart
  - Example: A procedure that handles what happens when the selected cell changes



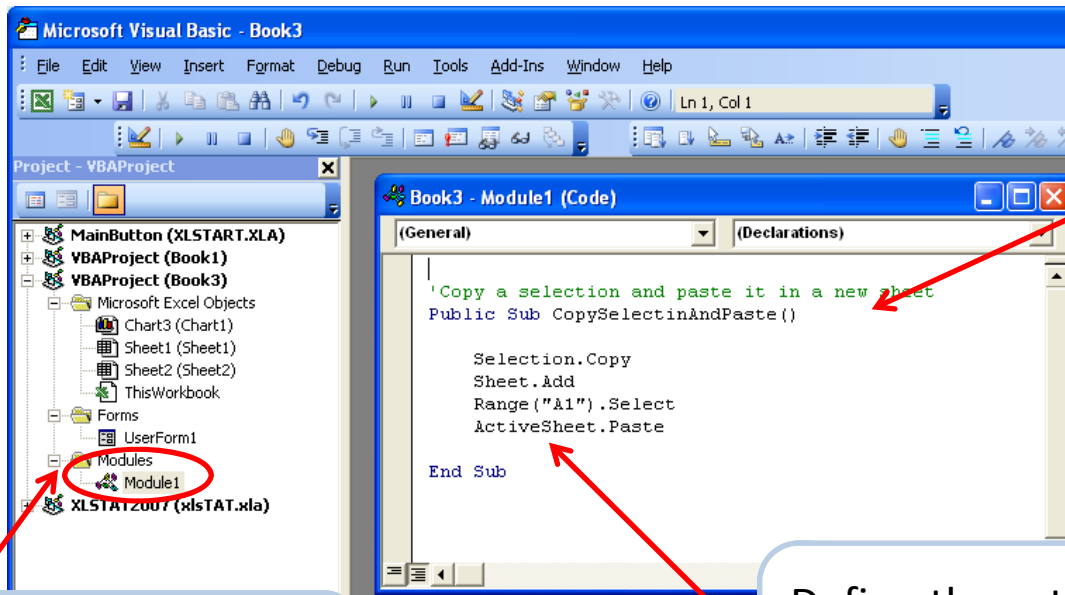
The events that correspond to Sheet2 are listed. Select one to automatically create the header of the subroutine

Double-click on the sheet name to display the corresponding code

You can type in what should happen when the selection on the sheet changes

# VBA – What you can do with it (3)

- Create you own routines and functions to perform simple or complex actions, or to automate tasks that you need to do on a regular basis
  - Example: A procedure that copies the selection and pastes it in a new sheet



Create your own subroutine

Insert a new module then type the code in it

Define the actions to be done in the subroutine

# VBA – What you can do with it (4)

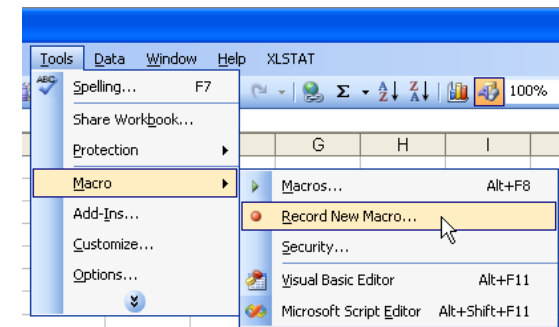
- Given the accessibility of Excel, its openness on data storage formats, you can virtually develop any type of solution in VBA
- The only drawback of VBA is that it is an interpreted language, not a compile language
- The consequence is that the speed of the computations is much lower than other languages
- This is the reason why XLSTAT is based on both VBA and C++
  - The VBA layer is used to develop interfaces
  - The C++ layer is used to perform all the mathematical and CPU intensive operations

# VBA – What you can't do with it



# Discovering VBA

- A well known way to discover VBA is to use the “**Record New Macro**” feature of Excel
- This is efficient for basic operations that do not require any condition, and they depend very much on the context
- This is also useful to discover objects types, methods, and properties, without having to go through the help
- Macros are recorded in the Modules section



# VBA Syntax – Key concepts

- VBA is an object oriented language
  - The language allows to manipulate **objects** (ex: Worksheet, Chart, Cell, ...) and **collections** (ex: Sheets, Workbooks, Points, ...)
  - Objects have **properties** (ex: size, font, ...), and **methods** that can be applied to them (add, delete, move, ...)
  - Many objects are part of the VBA environment. Sometimes it is necessary to build your own **type** of objects (ex: matrix of matrices), or special objects named **classes** that have specific properties and method
  - Some **events** can be detected by the environment and specific actions can be programmed to react to these events (ex: click, change, ...)

# VBA Syntax – Structure

- VBA is structured
  - Code is split into **subroutines** (actions), **functions** (can do actions, and return a value), **events**, and special definitions for classes.
  - A routine can be **Public** or **Private** (shared with the other modules of the project, or not)
- VBA syntax is simple
  - To apply a method to an object, you need to write `Object.Method`
  - To get/set the value of a property of an object, you need to write `Object.Property`
  - If you are applying several methods or getting/setting several properties of an object, you can factor the Object using the `[With Object ... End With]`

# Example 1

- Let's create a first procedure to better understand the concepts. This code can be pasted into the code project of a worksheet.

The screenshot shows the VBA editor window titled "Book3 - Sheet2 (Code)". The code editor contains the following VBA code:

```
Private Sub Worksheet_Activate()  
    Range("A1").Select  
    Range("A1").Font.Bold = True  
End Sub
```

Annotations with red arrows point to various parts of the code:

- Object of type "Range"**: Points to the "Range" object in the first line of code.
- Method**: Points to the ".Select" property access.
- Object of type "Font"**: Points to the "Font" object in the second line of code.
- Property**: Points to the ".Bold" property access.
- Value of the property**: Points to the "= True" value assigned to the property.
- The procedure sets what to do when you activate Sheet2 (event Activate)**: Points to the entire "Private Sub Worksheet\_Activate()" line.

This subroutine allows to select the A1 cell and set the style to bold when you activate the corresponding sheet

# VBA Syntax – Keywords

- The usual [If, Then, Else], [Do, Loop], [For, Next] structures are available, and the And, Or, Xor operators can be used
- To call a subroutine from another subroutine or from a function, you need to add Call before the routine name, followed by the arguments
- If you need to create temporary objects, you should declare their type. This allows to save memory and to better organize the code. Use Const or Dim to declare an object. The types of the arguments of a routine should be declared with the routine declaration.
- It is not possible to modify the value of an object that was declared with Const (ex: Const Precision=0.0001)

## Example 2

- The subroutine ColorBySign allows to color the selected cells in **blue** if  $<0$ , **green** if  $=0$  and **red** if  $>0$ . It is public, so you can call it with Tools / Macros / Run
- The function is private so it is only available in the module

```
Public Sub ColorBySign()

    Dim i As Long, j As Long

    With Selection
        For i = 1 To .Rows.Count
            For j = 1 To .Columns.Count
                With .Offset(i - 1, j - 1)
                    .Interior.ColorIndex = GetColorIdx(.Value)
                End With
            Next j
        Next i
    End With

End Sub

Private Function GetColorIdx(iValue As Double) As Long

    If iValue < 0 Then
        GetColorIdx = 41
    ElseIf iValue = 0 Then
        GetColorIdx = 4
    ElseIf iValue > 0 Then
        GetColorIdx = 3
    End If

End Function
```

Calling the function

# VBA Syntax – Basic Types

- The basic types are
  - Long (integers in the range [-2 147 483 648, -2 147 483 647])
  - Double (continuous values in the range [-1.79769313486231<sup>E</sup>308, 1.79769313486231<sup>E</sup>308])
  - String (strings of characters)
  - Boolean (True or False)
  - Variant (any type)
- When you declare an object using Dim, you can use special characters for some types:
  - Dim i As Long      <=>      Dim i&
  - Dim i As Double    <=>      Dim i#
  - Dim i As String    <=>      Dim i\$

## Example 3

- For a given X value this function returns the closest value among  $\text{Int}(X)$ ,  $\text{Int}(X)+0.5$  and  $\text{Int}(X+0.5)$
- $\text{Int}()$  is a built-in function that rounds to the closest lower integer (if X integer,  $\text{Int}(X)=X$ )
- The function being public you can use it within a worksheet

```
Public Function Int05(iMyValue As Double) As Double
```

```
    Dim MySign As Long
    Dim MyVal As Double, Temp As Double
```

```
    MySign = IIf(iMyValue < 0, -1, 1)
    MyVal = Int(iMyValue)
    Temp = MyVal
```

```
    If (iMyValue - MyVal) > Abs(iMyValue - (MyVal + 0.5)) Then
        Temp = MyVal + 0.5
    If Abs(iMyValue - MyVal - 0.5) > (MyVal + 1 - iMyValue) Then
        Temp = MyVal + 1
    End If
End If
```

```
    Int05 = Temp * MySign
```

```
End Function
```

# VBA Syntax – User Defined Types

- You can create your own types of objects by declaring them at the top of a module between [Public Type X, End Type]

```
Public Type StatisticalTest  
    CriticalVal As Double  
    DF As Long  
    StatValue As Double  
    pValue As Double  
End Type
```

# VBA Syntax – Range Object (1)

- Excel VBA allows to access the content of the worksheets through the Range Object

**Properties:** AddIndent, **Address**, AddressLocal, AllowEdit, Application, Areas, Borders, **Cells**, Characters, **Column**, **Columns**, ColumnWidth, Comment, Count, Creator, CurrentArray, CurrentRegion, Dependents, DirectDependents, DirectPrecedents, End, EntireColumn, EntireRow, Errors, Font, FormatConditions, **Formula**, FormulaArray, FormulaHidden, FormulaLabel, FormulaLocal, FormulaR1C1, FormulaR1C1Local, HasArray, HasFormula, Height, Hidden, HorizontalAlignment, Hyperlinks, ID, IndentLevel, Interior, Item, Left, ListHeaderRows, ListObject, LocationInTable, Locked, MergeArea, MergeCells, Name, Next, **NumberFormat**, NumberFormatLocal, **Offset**, Orientation, OutlineLevel, PageBreak, **Parent**, Phonetic, Phonetics, PivotCell, PivotField, PivotItem, PivotTable, Precedents, PrefixCharacter, Previous, QueryTable, Range, ReadingOrder, Resize, **Row**, RowHeight, **Rows**, ShowDetail, ShrinkToFit, SmartTags, SoundNote, Style, Summary, Text, Top, UseStandardHeight, UseStandardWidth, Validation, **Value**, Value2, VerticalAlignment, Width, Worksheet, WrapText, Xpath

**Methods:** Activate, AddComment, AdvancedFilter, ApplyNames, ApplyOutlineStyles, AutoComplete, AutoFill, AutoFilter, AutoFit, AutoFormat, AutoOutline, BorderAround, Calculate, CheckSpelling, **Clear**, ClearComments, ClearContents, ClearFormats, ClearNotes, ClearOutline, ColumnDifferences, Consolidate, Copy, CopyFromRecordset, CopyPicture, CreateNames, CreatePublisher, Cut, DataSeries, Delete, DialogBox, Dirty, EditionOptions, FillDown, FillLeft, FillRight, FillUp, **Find**, FindNext, FindPrevious, FunctionWizard, GoalSeek, Group, Insert, InsertIndent, Justify, ListNames, Merge, NavigateArrow, NoteText, Parse, PasteSpecial, PrintOut, PrintPreview, RemoveSubtotal, Replace, RowDifferences, Run, **Select**, SetPhonetic, Show, ShowDependents, ShowErrors, ShowPrecedents, **Sort**, SortSpecial, Speak, SpecialCells, SubscribeTo, Subtotal, Table, TextToColumns, Ungroup, UnMerge

**Parent Objects:** AllowEditRange, Application, Areas Collection, AutoFilter, ChartObject, HPageBreak, Hyperlink, ListColumn, ListObject, ListRow, Name, OLEObject, Pane, Parameter, PivotCell, PivotField, PivotItem, PivotTable, QueryTable, Range Collection, Scenario, Shape, SmartTag, VPageBreak, Window, **Worksheet**

**Child Objects:** Areas, Borders, Characters, Comment, Errors, Font, FormatConditions, Hyperlinks, Interior, ListObject, Phonetic, Phonetics, PivotCell, PivotField, PivotItem, PivotTable, QueryTable, Range, SmartTags, SoundNote, Validation, Worksheet, Xpath

# VBA Syntax – Range Object (2)

- Example 4.1: Set the style of cell D2 to bold

```
Range("A1").Offset(1,3).Font.Bold = True
```

- Example 4.2: Copy and paste cells

```
Range("A1:C3").Copy
```

```
Range("D2").Select
```

```
ActiveSheet.Paste
```

- Example 4.3: Get the number of columns in a range

```
NbCols = Range(MyRange).Columns.Count
```

- Example 4.4: Get the parent workbook of a Range

```
MyWbk = Range(MyRange.Address).Parent.Parent.Name
```

Or

```
MyWbk = Sheets(Range(MyRange.Address).Worksheet.Name).Parent.name
```

# VBA Syntax – Range Object (3)

- Example 4.5: Load the content of an Excel table into a table of Doubles
  - First copy the Range into an object to avoid reading too often in the worksheet which costs a lot of CPU time
  - Once it is in memory, copying it into another table is a lot faster
  - Doubles can easily be exchanged with other languages

```
Public Sub LoadTableDouble()

    Dim TempRange As Range
    Dim TempTable As Variant
    Dim MyTable() As Double
    Dim i As Long, j As Long, rws As Long, cols As Long

    'copy the "A1:B5" range into a range
    'set allows to create the object reference
    Set TempRange = Range("A1:B5")
    'sets TempTable the "A1:B5" range into an object of type variant
    Set TempTable = Range("A1:B5")
    'When set is not used copies the default property of Range into
    'TempTable, a table of variant
    TempTable = Range("A1:B5")

    'copy TempTable into a Table of Double
    rws = TempRange.Rows.Count
    cols = TempRange.Columns.Count
    Redim MyTable(1 To rws, 1 To cols)
    For i = 1 To TempRange.Rows.Count
        For j = 1 To TempRange.Columns.Count
            MyTable(i, j) = TempTable(i, j)
        Next j
    Next i

End Sub
```

# VBA Syntax – Tables

- You can create tables of a given type
  - Add parentheses in the Dim statement
    - ex: `Dim MyTable() As Double`
  - Then use Redim to set the dimensions
    - ex: `Redim MyTable(1 to 4, 1 to 2)` creates a 4 by 2 table (or matrix)
  - If the dimensions are set in the **Dim** statement, the size cannot be changed later
  - Redimensioning can be done later, either preserving the data if only the last dimension is modified, or erasing the current values
    - ex: `Redim Preserve MyTable(1 to 4, 1 to 3)` creates a 4 by 3 table (or matrix) without erasing the values stored in the first 2 columns
    - ex: `Redim MyTable(1 to 4, 1 to 3)` creates a 4 by 3 table (or matrix) and erases the values stored in the first 2 columns

## Example 5

- The subroutine Generate100Centered generates a normal random sample, centered on 0.
- To center the sample, it is necessary to memorize all the generated values
- Therefore the RandNormal() vector is created
- Values are output on the active worksheet

```
Public Sub Generate100Centered()

    Dim i As Long
    Dim RandNormal() As Double
    Dim Mean As Double

    ReDim RandNormal(1 To 100)

    Randomize

    Mean = 0
    For i = 1 To 100
        RandNormal(i) = Application.NormSInv(Rnd())
        Mean = Mean + RandNormal(i)
    Next i

    Mean = Mean / 100

    'Center the sample and display it
    With ActiveSheet.Range("A1")
        For i = 1 To 100
            RandNormal(i) = RandNormal(i) - Mean
            .Offset(i, 0) = RandNormal(i)
        Next i
    End With

End Sub
```

Calling an  
Excel function

Comments  
start with '

# VBA Syntax – Exchanging information

- Exchanging objects between VBA subroutines and functions:
  - When passing objects of basic types (a Long, a String, a Double, a Boolean), you can pass it **ByVal**, or **ByRef** (default). **ByVal** creates a copy in the **Child routine** and the value of the passed object is not modified in the **Parent routine** even if it is in the Child. Passing **ByVal** requires more memory as a copy is made.
  - More complex objects such as tables of basic types, object types (Shapes, Controls, ...) and custom types, can only be passed by reference (**ByRef**). This means that their address in memory is passed to the **Child routine**. Any change in the **Child routine** will be reflected in the **Parent routine**.
  - Note: to avoid confusion, do not use **ByVal**, and create a copy when you do not want to change the value in the parent routine.

# Example 6

- The subroutine ByRefExample does not change var1, while ByValExample does.

```
Public Sub Sub DemoPassingArguments()
```

```
    Dim var1 As Long
```

```
    var1 = 2
```

```
    Call ByRefExample(var1)
```

```
    MsgBox var1
```

```
    Call ByValExample(var1)
```

```
    MsgBox var1
```

```
End Sub
```

```
Private Sub ByRefExample(ByRef var1 As Long)
```

```
    var1 = 2 * var1
```

```
End Sub
```

```
Private Sub ByValExample(ByVal var1 As Long)
```

```
    var1 = 2 * var1
```

```
End Sub
```

MsgBox generates a dialog box that displays the value

var1 is only changed inside this sub

# VBA Syntax – Sharing information

- You can share a variable or a constant across all subroutines and functions
- To do so, put the Dim instruction at the top of the module
- Note: If a variable with the same name is declared in a function or a subroutine, it has priority, and the values it takes has no effect on the global variable

```

Public Test As Long

Public Sub GlobalVariableTest ()

    Dim Test As Long

    Test = 1

End Sub
    
```

Expression	Value	Type	Context
Test	0	Long	Module1
Test	1	Long	Module1.GlobalVariableTest

Global variable

Local variable

# VBA Syntax – In/Outputs, Optional

- There is no difference made between inputs and outputs in the arguments of a procedure.
- To better identify inputs, outputs and in/outputs, you can use the following simple rule:
  - Use an **i** or **i\_** prefix for inputs
  - Use an **o** or **o\_** prefix for outputs
  - Use an **io**
- Arguments can be **Optional**. Only the last arguments can be optional. Optional objects cannot be tables of Doubles, Longs, .... They can however be multidimensional variants.

# Example 7

- The function WeightedMean computes a weighted mean. If there are no weights, the argument “iWeights” does not need to be entered
- As it is using Ranges, it can be used in an Excel cell:

X	W
1	1
2	2
3	3
4	4
5	5
3	3,666667

Unweighted

Weighted

```
Public Function WeightedMean(iRange As Range, Optional iWeights As Range) As Double
```

```
Dim i As Long, j As Long
Dim WM As Double
```

```
If iWeights Is Nothing Then
```

```
WM = Application.Average(iRange)
```

```
Else
```

```
For i = 1 To iRange.Rows.Count
```

```
For j = 1 To iRange.Columns.Count
```

```
WM = WM + iWeights.Cells(i, j) * iRange.Cells(i, j)
```

```
Next j
```

```
Next i
```

```
WM = WM / Application.Sum(iWeights)
```

```
End If
```

```
WeightedMean = WM
```

```
End Function
```

If no weights are entered

# VBA Syntax – Optional Arguments

- When calling a function or a subroutine with optional arguments, if you omit one optional argument, you need to specify precisely the next optional arguments.
- Default value for optional parameters can be set

```
Public Function WeightedMeanRound(iRange As Range, Optional iWeights As Range, _
                                Optional iRound2DG As Boolean = True) As Double

    WeightedMeanRound = WeightedMean(iRange, iWeights)

    If (iRoundIt2DG) Then
        WeightedMeanRound = Int(WeightedMeanRound * 100) / 100
    End If

End Function

Public Sub TestWeightedMeanRound()
    MsgBox WeightedMeanRound(Range("E6:E10"), iRound2DG:=False)
End Sub
```

iWeights is omitted, so iRound2DG must be specified

# Debugging the code

- ❶ One of the great features of VBA is that it provides several tools that facilitate the debugging:
  - Clicking F5 or F8 (step by step mode) allows to start a subroutine that has no input/output parameters
  - You can add “toggle breakpoints” by pressing F9 or clicking in the margin
  - **Stop** allows stopping the code at a given line of the code
  - **Debug.Print** can be used to output some values in the “Immediate” window. To display this window, click Ctrl G. Note: The “Immediate” window is restricted to 256 lines. When you add a 257<sup>th</sup>, the 1<sup>st</sup> line is removed.
  - You can jump to a line by dragging the debugging cursor to the line you want

```
'sets TempObj the "A1:B5" range into an object of type range
Set TempObj = Range("A1:B5")
'sets TempTable the "A1:B5" range into an object of type variant
Set TempTable = Range("A1:B5")
'copies the default property of Range into TempTable
TempTable = Range("A1:B5")
```

# Locals Windows

- The “Locals” window allows to control the value of all the variables of a routine (arguments and internal objects)

The screenshot shows the 'Locals' window for 'VBAProject.Module1.LoadTableDouble'. The tree view is expanded to show 'MyTable', which contains five sub-folders: 'MyTable(1)', 'MyTable(2)', 'MyTable(3)', 'MyTable(4)', and 'MyTable(5)'. Each folder contains two variables, 'MyTable(i,1)' and 'MyTable(i,2)', all with a value of 0. Other variables like 'i', 'j', 'rws', and 'cols' are also visible at the bottom of the list.

Expression	Value	Type
Module1		Module1/Module1
TempObj		Object/Range
TempRange		Range/Range
TempTable		Variant/Variant(1 to 5, 1 to 2)
MyTable		Double(1 to 5, 1 to 2)
MyTable(1)		Double(1 to 2)
MyTable(1,1)	0	Double
MyTable(1,2)	0	Double
MyTable(2)		Double(1 to 2)
MyTable(2,1)	0	Double
MyTable(2,2)	0	Double
MyTable(3)		Double(1 to 2)
MyTable(3,1)	0	Double
MyTable(3,2)	0	Double
MyTable(4)		Double(1 to 2)
MyTable(4,1)	0	Double
MyTable(4,2)	0	Double
MyTable(5)		Double(1 to 2)
MyTable(5,1)	0	Double
MyTable(5,2)	0	Double
i	3	Long
j	3	Long
rws	5	Long
cols	2	Long

You can use it to explore tables

The screenshot shows the 'Locals' window for 'VBAProject.Module1.LoadTableDouble'. The tree view is expanded to show 'TempRange', which contains a large number of object properties such as 'AddInIdent', 'AllowEdit', 'Application', 'Areas', 'Borders', 'Cells', 'Column', 'ColumnWidth', 'Comment', 'Count', 'Creator', 'CurrentArray', 'CurrentRegion', 'Dependents', 'DirectDependents', 'DirectPrecedents', 'Errors', 'Font', 'FormatConditions', 'Formula', 'FormulaArray', and 'FormulaHidden'. Each property has a specific value and type.

Expression	Value	Type
Module1		Module1/Module1
TempObj		Object/Range
TempRange		Range/Range
AddInIdent	False	Variant/Boolean
AllowEdit	True	Boolean
Application		Application/Application
Areas		Areas/Areas
Borders		Borders/Borders
Cells		Range/Range
Column	1	Long
ColumnWidth	8,43	Variant/Double
Comment	Nothing	Comment
Count	10	Long
Creator	xICreatorCode	XICreator
CurrentArray	<No cells were found.>	Range
CurrentRegion		Range/Range
Dependents	<No cells were found.>	Range
DirectDependents	<No cells were found.>	Range
DirectPrecedents	<No cells were found.>	Range
Errors	<Application-defined or object-defin Errors	Errors
Font		Font/Font
FormatConditions		FormatConditions/FormatConditions
Formula		Variant/Variant(1 to 5, 1 to 2)
FormulaArray	""	Variant/String
FormulaHidden	False	Variant/Boolean

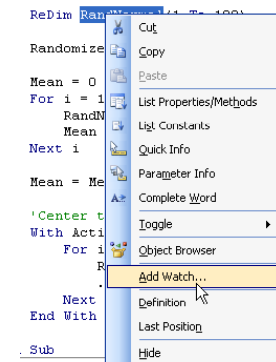
You can use it to explore objects

# Watches

- The “Watches” window allows to watch one or more objects that are of particular interest

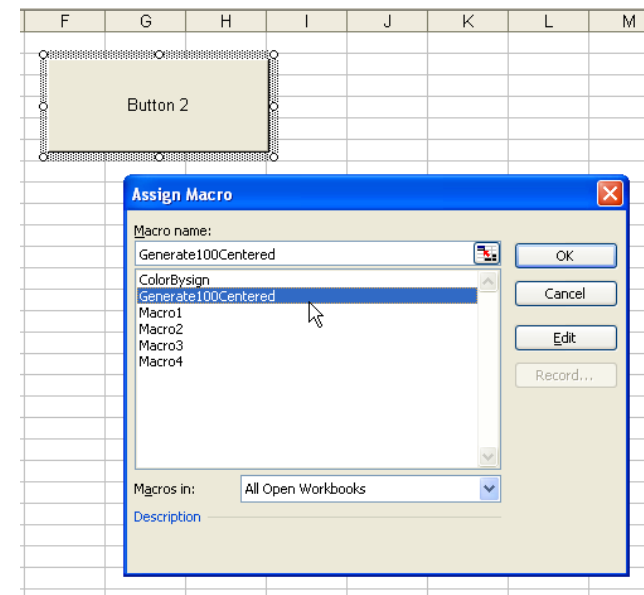
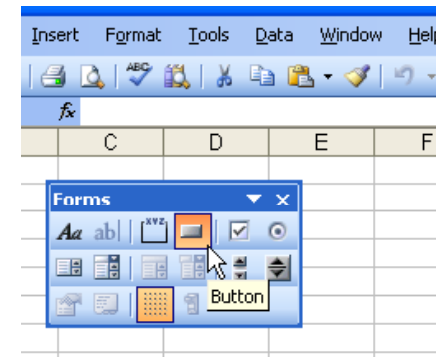
Expression	Value	Type	Context
ActiveSheet.Range("A1")		Variant/Object/Range	Module1.Generate100Center
RandNormal		Double(1 to 100)	Module1.Generate100Center
RandNormal(1)	-0,865874678825	Double	Module1.Generate100Center
RandNormal(2)	1,937298409121	Double	Module1.Generate100Center
RandNormal(3)	0	Double	Module1.Generate100Center
RandNormal(4)	0	Double	Module1.Generate100Center
RandNormal(5)	0	Double	Module1.Generate100Center
RandNormal(6)	0	Double	Module1.Generate100Center
RandNormal(7)	0	Double	Module1.Generate100Center
RandNormal(8)	0	Double	Module1.Generate100Center

- To add an object to the “Watches” window select the object in the code, and drag it into the window, or right click the object and choose Add to watch.



# Tip to save time

- If you want to avoid having to go to the Tools/Macros/Run command in the Excel menus, create a button that you can click to run the macro
- To do that, add a button using the Forms toolbar, and associate the macro of interest

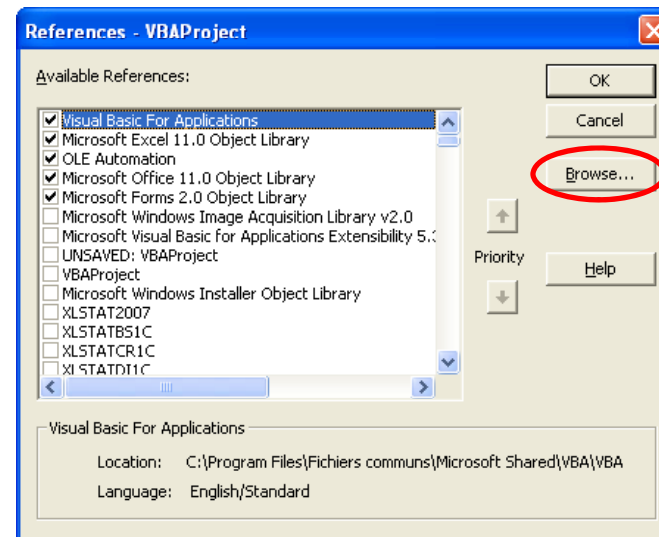
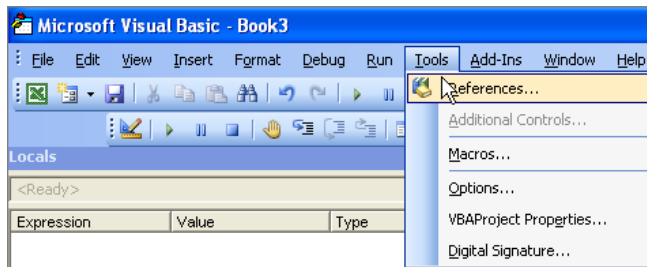


# Using external libraries

- What you don't want
  - Have to copy and paste old code all the time
  - Have to redo what has already been done many times by others
- What you want
  - Be able to develop little by little libraries of functions and routines that you can easily reuse
  - Be able to reuse components that were developed by others
- While this is very common in the open source community, this is much less common in the VBA community
- Since version 2006, a licensed user of XLSTAT can use the XLSTAT components to develop his own solution

# Making reference to a library

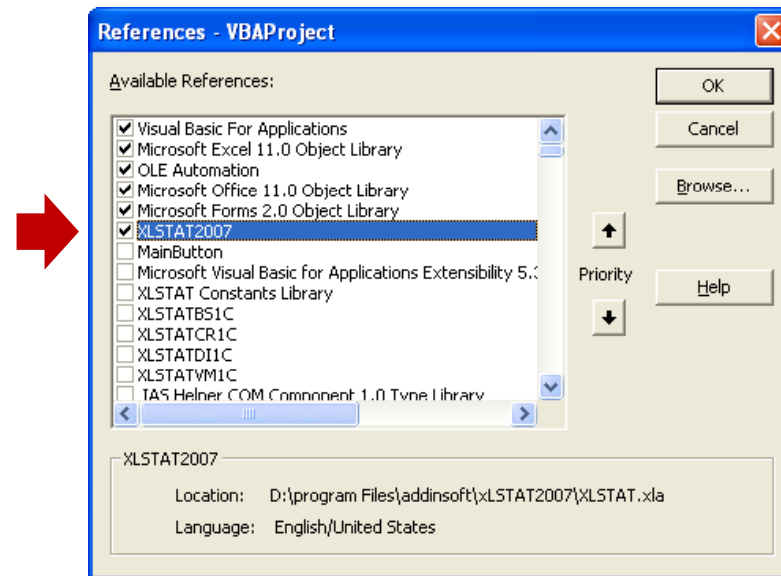
- To allow a project to use external code, you must set the reference of the project to the library



- If the file is not listed in the References window, you can use the Browse button to select it. By default Excel expects \*.dll files, but it can also be \*.xls or \*.xla files.

# Making reference to XLSTAT

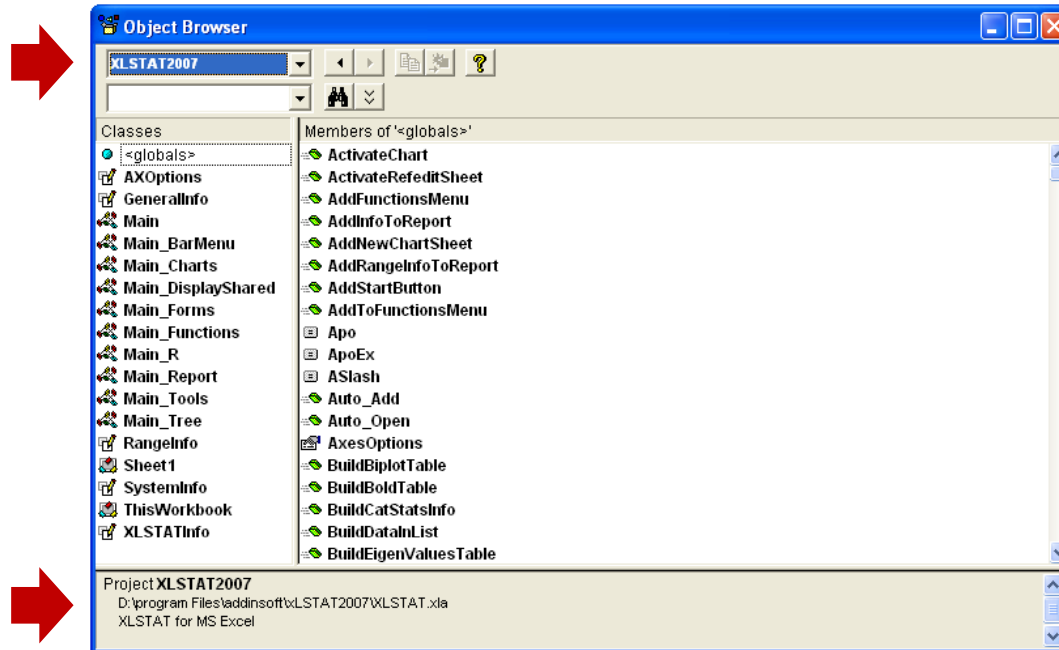
- To make reference to XLSTAT.xla, select the file in the XLSTAT installation folder. Then, you should see XLSTAT listed and checked.









- XLSTAT.xla is the main VBA based XLSTAT file

# Using XLSTAT elements

- You can then use the object browser to see what you can access in XLSTAT.xla (only public elements can be seen):



-  = Constant
-  = Sub or Function
-  = Object of special type
-  = Type
-  = Module
-  = Form / Wbook / sheet

# The RangeInfo type

- In XLSTAT, the RangeInfo type is used a lot, both for inputs and outputs. We recommend that you use it:

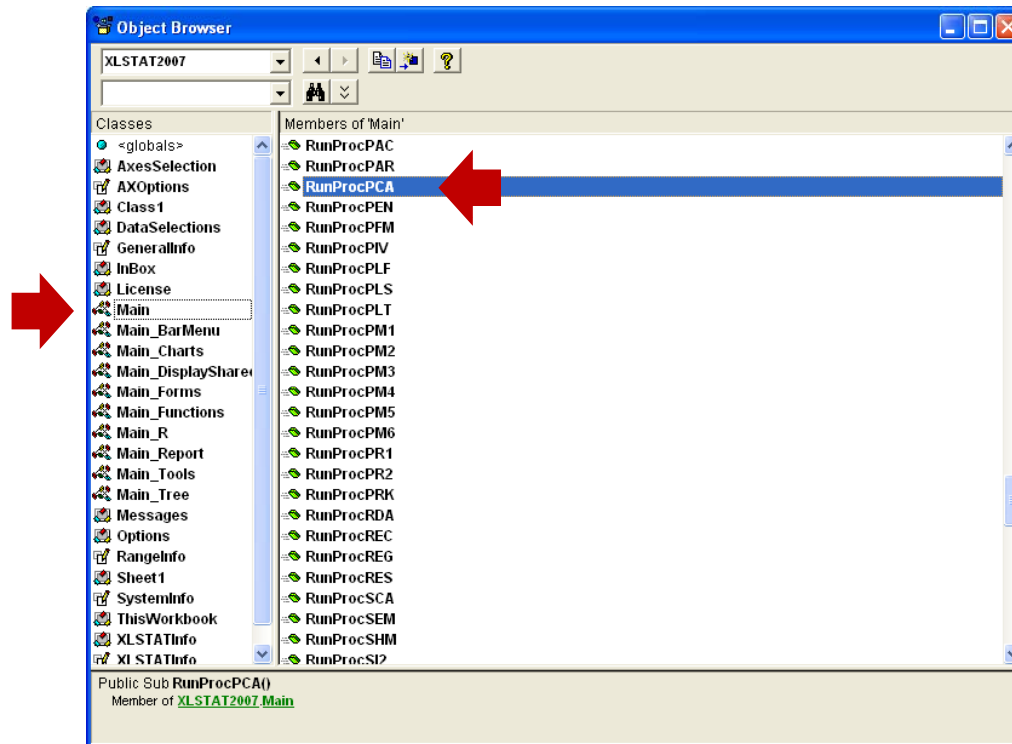
## Public Type RangeInfo

ByColumns As Boolean	-> Was the select done by columns
ByRows As Boolean	-> Was the select done by rows
RangeAddress As String	-> Address of the Range
Cell11 As String	-> Address of the upper left cell
Cell1n As String	-> Address of the upper right cell
CellInn As String	-> Address of the bottom right cell
CellStart As String	-> Address of the initial cell
MaxNbRow As Long	-> Maximum number of rows in an area
MaxNbCol As Long	-> Maximum number of columns in an area
NbAreas As Long	-> Nbr of areas in a range
NbCol() As Long	-> Nbr of columns in each area
NbRow() As Long	-> Nbr of rows in each area
OldCell11 As String	-> Address of the cell that was previously Cell11
SheetName As String	-> Name of the current sheet
SheetNameFirst As String	-> Name of the first sheet used by the current RangeInfo object
SheetPrefix As String	-> Name of the sheet prefix
TotalCol As Long	-> Total number of columns
TotalRow As Long	-> Total number of rows
WbookName As String	-> Name of the workbook where the RangeInfo is used

## End Type

# Example: calling an XLSTAT subroutine

- This example allows you to call a subroutine contained in the Main module of XLSTAT.xla:



```
Public Sub DemoShowPCADialogBox()  
  
    Call RunProcPCA  
  
End Sub
```

Running this subroutine displays the PCA dialog box

# Calling XLSTAT tools without showing dialog boxes (1)

- The XLSTAT projects where the various tools are available are stored in the XLSTAT-XYZ.dll files (XYZ: MCA, LOG, ...).
- These libraries include functions that allow to call silently the tools, without showing the dialog boxes.
- These functions start with LoadRun, followed by the name of the method (ex: LoadRunPCA for Principal Components Analysis, LoadRunREG for linear regression).

## Calling XLSTAT tools without showing dialog boxes (2)

- ① How to use the LoadRun functions:
  - Make reference to the XLSTAT-XYZ.dll library
  - Create a routine that prepares the information required by LoadRun
  - Call LoadRun with the necessary arguments
  - Postprocess the results if necessary

## Example: Calling LoadRunPCA (1)

### ● Definition of the LoadRunPCA function:

```
Public Sub LoadRunPCA(DataRange As Range, Optional MatrixType As Long = 1, _
    Optional WithVarLabels As Boolean = True, Optional ObsRange As Range, _
    Optional OutputType As Long = 2, Optional OutRange As Range, _
    Optional PCAType As Long = 1, _
    Optional NoScreenUpdating As Boolean)
```

Note 1: All options of the PCA dialog box are not available in the LoadRunPCA function. If you need to change the options, do it first in the dialog box. If you really need to access them programmatically, contact Addinsoft.

Note 2: When set to True, **NoScreenUpdating** allows to ignore the dialog box that prompts you for the choice of axes.

## Example: Calling LoadRunPCA (2)

### Call example:

```
Public Sub Sub VBA_PCA_Demo()
```

```
With Workbooks("demoVBA-PCA.xls")
```

```
.Activate
```

```
Call LoadRunPCA(Sheets("Data").Range("$B:$G"), 1, True, Sheets("Data").Range("$A:$A"), PCAType:=1)
```

```
End With
```

```
End Sub
```

# Other LoadRun procedures

The following LoadRun procedures are available:

Method(s)	Subroutine name	Library
Dissimilarity matrices	LoadRunDSS	XLSTAT-CLU.dll
K-means clustering	LoadRunKMN	XLSTAT-CLU.dll
Agglomerative Hierarchical Clustering	LoadRunAHC	XLSTAT-CLU.dll
Normality tests	LoadRunNormTests	XLSTAT-DIS.dll
Distribution sampling	LoadRunDistSampling	XLSTAT-DIS.dll
Distribution fitting	LoadRunDistribFit	XLSTAT-DIS.dll
Classification trees	LoadRunTree	XLSTAT-DTM.dll
Kaplan Meier Analysis	LoadRunLife	XLSTAT-Life.dll
Logistic regression	LoadRunLOG	XLSTAT-LOG.dll
Linear regression	LoadRunREG	XLSTAT-LOG.dll
Discriminant analysis	LoadRunDisc	XLSTAT-LOG.dll
Principal Components Analysis	LoadRunPCA	XLSTAT-MCA.dll
Create a contingency table	LoadRunDCA	XLSTAT-MCA.dll
Correspondence analysis	LoadRunCA	XLSTAT-MCA.dll
Multiple Correspondence analysis	LoadRunMCA	XLSTAT-MCA.dll
MDS	LoadRunMDS	XLSTAT-MCA.dll
Non linear regression	LoadRunNLI	XLSTAT-NLN.dll
Non parametric regression	LoadRunNPRreg	XLSTAT-NLN.dll
PLS regression	LoadRunPLS	XLSTAT-PLS.dll
Transformations	LoadRunTransform	XLSTAT-PRP.dll
Time series smoothing	LoadRunSmooth	XLSTAT-TSA.dll
ARIMA	LoadRunARIMA	XLSTAT-TSA.dll
Non parametric tests	LoadRunNonParam	XLSTAT-TST.dll
Correlation tests	LoadRunCorrelTests	XLSTAT-TST.dll

Demos are available at <http://www.xlstat.com/demoVBAzip>

Should you need to access any additional tool, just tell us

# Example: K-means and AHC (1)

- This example shows how to do first a k-means and then an AHC: this is necessary if you have a lot of observations and if you want to use AHC. AHC is a very intensive method and it should not be used with more than 2000 observations.

The code is available in the DemoVBA.zip file in demoVBA-KMN-AHC.xls

```
Public Sub ChainKmeansAHC()
    Dim ClassesAddress As String, WRangeAddress As String
    Dim ColFormats() As String
    Dim NbrClustersKMN As Long, NbrClustersAHC As Long, NbrCols As Long, NbrRows As Long
    Dim Classes As Variant, Classes2 As Variant
    Dim RangeOut As RangeInfo

    NbrClustersKMN = 10
    NbrCols = 4
    NbrRows = 150

    Call LoadRunKMN(Sheets("Sheet1").Range("B:E"), False, NbrClustersKMN, 1, NbrRepetitions:=10, _
        Center:=True, Reduce:=True, ResultsInOriginalSpace:=False, NoScreenUpdating:=False)

    With ActiveSheet
        'store the results by objects
        ReDim Classes(1 To NbrRows)
        ClassesAddress = .Cells.Find("Results by object").Address
        With Range(ClassesAddress)
            Classes = ActiveSheet.Range(.Offset(3, 1).Address & ":" & .Offset(NbrRows + 2, 1).Address)
        End With

        'locate the weights to know where the clusters are described
        WRangeAddress = .Cells.Find("Sum of weights").Address

        NbrClustersAHC = 3

        'Dissimilarity Criterion 10 is Euclidean distance
        'clustering method 6 is Ward

        With Range(WRangeAddress)
            Call LoadRunAHC(ActiveSheet.Range(.Offset(1, -NbrCols).Address & ":" & .Offset(NbrClustersKMN, _
                -1).Address), _
                False, True, 10, 6, False, _
                RowWeights:=ActiveSheet.Range(.Offset(1).Address & ":" & _
                    .Offset(NbrClustersKMN).Address), _
                WithTruncation:=True, TruncNbrClasses:=NbrClustersAHC, _
                NoScreenUpdating:=False)
        End With

        End With

        'locate the results by objects
        With ActiveSheet
            ReDim Classes2(1 To NbrClustersKMN)
            ClassesAddress = .Cells.Find("Results by object").Address
            With Range(ClassesAddress)
                Classes2 = ActiveSheet.Range(.Offset(3, 1).Address & ":" & .Offset(NbrClustersKMN + 2, 1).Address)
            End With
        End With

        'find the final cluster for each obs
        For i = 1 To NbrRows
            Classes(i, 1) = Classes2(Classes(i, 1), 1)
        Next i

        'display the results below the previous results
        RangeOut.Cells(1) = Range(ClassesAddress).Offset(NbrClustersKMN + 6).Address
        RangeOut.SheetName = ActiveSheet.Name
        RangeOut.WbBookName = ThisWorkbook.Name

        Call DisplayTitle(RangeOut, "Final classification for initial observations")

        Call SetFormat(ColFormats, 1, -2)
        Call DisplayTable(RangeOut, Classes, NbrRows, 1, iColumnsFormats:=ColFormats)
    End Sub
```

## Example: K-means and AHC (2)

- Step 1: Call the K-means routine

```
Public Sub Sub ChainKmeansAHC()

    Dim ClassesAddress As String, WRangeAddress As String
    Dim ColFormats() As String
    Dim NbrClustersKMN As Long, NbrClustersAHC As Long, _
        NbrCols As Long, NbrRows As Long
    Dim Classes As Variant, Classes2 As Variant
    Dim RangeOut As RangeInfo

    NbrClustersKMN = 10
    NbrCols = 4
    NbrRows = 150

    Call LoadRunKMN(Sheets("Sheet1").Range("B:E"), False, _
        NbrClustersKMN, 1, NbRepetitions:=10, _
        Center:=True, Reduce:=True, _
        ResultsInOriginalSpace:=False, _
        NoScreenUpdating:=False)

    ○ ○ ○
```

- Step 2: Extract the classes information, and find where the cluster description is located

```
With ActiveSheet

    'store the results by objects
    ReDim Classes(1 To NbrRows)
    ClassesAddress = .Cells.Find("Results by object").Address
    With Range(ClassesAddress)
        Classes = ActiveSheet.Range(.Offset(3, 1).Address & ":" &
            .Offset(NbrRows + 2, 1).Address)
    End With
    'locate the weights to know where the clusters are described
    WRangeAddress = .Cells.Find("Sum of Weights").Address

    ○ ○ ○
```

## Example: K-means and AHC (3)

- ④ Step 3: Call the AHC routine, and specify the row weights option
  
- ④ Step 4: Extract the classes information, and compute the final classification for the original observations. Then display the final classification

```
NbrClustersAHC = 3

'Dissimilarity Criterion 10 is Euclidean distance
'clustering method 6 is Ward

With Range(WRangeAddress)
  Call LoadRunAHC(ActiveSheet.Range(.Offset(1, _
    -NbrCols).Address & ":" & .Offset(NbrClustersKMN, _
    -1).Address), _
    False, True, 10, 6, False, _
    RowWeights:=ActiveSheet.Range(.Offset(1).Address & ":" _
    & .Offset(NbrClustersKMN).Address), _
    WithTruncation:=True, TruncNbrClasses:=NbrClustersAHC, _
    NoScreenUpdating:=False)

End With
```

End With

○ ○ ○

```
'locate the results by objects
With ActiveSheet
  ReDim Classes2(1 To NbrClustersKMN)
  ClassesAddress = .Cells.Find("Results by object").Address
  With Range(ClassesAddress)
    Classes2 = ActiveSheet.Range(.Offset(3, 1).Address & ":" & .Offset(NbrClustersKMN +
    2, 1).Address)
  End With
End With

'find the final cluster for each obs
For i = 1 To NbrRows
  Classes(i, 1) = Classes2(Classes(i, 1), 1)
Next i

'display the results below the previous results
RangeOut.Cells11 = Range(ClassesAddress).Offset(NbrClustersKMN + 6).Address
RangeOut.SheetName = ActiveSheet.Name
RangeOut.WbookName = ThisWorkbook.Name

Call DisplayTitle(RangeOut, "Final classification for initial observations")

Call SetFormat(ColFormats, 1, -2)
Call DisplayTable(RangeOut, Classes, NbrRows, 1, iColumnsFormats:=ColFormats)

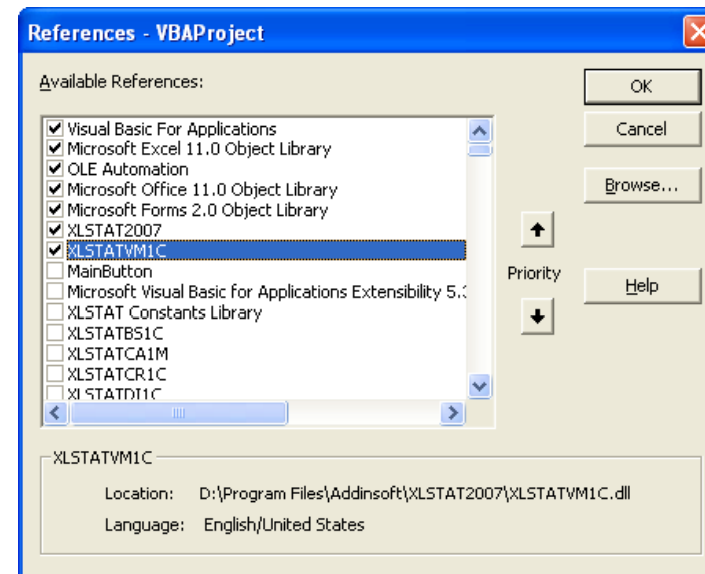
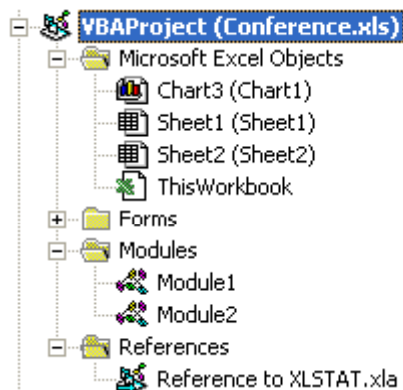
End Sub
```

# Using XLSTAT C++ routines

- In the object browser, besides the XLSTAT main library, XLSTAT.xla, you can see a series of other projects that are C++ libraries.
- You can also add reference to the C++ libraries in order to access the C++ functions of XLSTAT
- This requires more caution as all the exchanges between VBA and C++ are by address, and any mistake in the declarations might lead to crashing Excel
- To give the address of an object, you need to specify the address of the first element, and the size

# References to C++

- When you make reference to a C++ library, you do not see it in the References list of the project explorer, but only in the references box



# XLSTAT C++ / Example 1

- Many matrix operations are available in the XLSTATVM1C.dll library
- This example shows how to compute a matrix product

```
Public Sub TestMatrixProduct()
```

```
    Dim i As Long, j As Long
```

```
    Dim Matrix1(1 To 2, 1 To 3) As Double
```

```
    Dim Matrix2(1 To 3, 1 To 4) As Double
```

```
    Dim Matrix3(1 To 2, 1 To 4) As Double
```

```
'note: a matrix is considered in C++ as a vector of size n x p
```

```
'so you can use vector routines if you do not need to use the row/column information
```

```
Call Cpp_V_SetToValue(Matrix1(1, 1), 2 * 3, 1)
```

```
Call Cpp_V_Set1toNdbl(Matrix2(1, 1), 3 * 4)
```

```
'run the matrix product: Matrix1 x Matrix2 = Matrix3
```

```
Call Cpp_M_ProductAB(Matrix1(1, 1), 2, 3, Matrix2(1, 1), 4, Matrix3(1, 1))
```

```
End Sub
```

Matrix3		Double(1 to 2, 1 to 4)
Matrix3(1)		Double(1 to 4)
Matrix3(1,1)	6	Double
Matrix3(1,2)	15	Double
Matrix3(1,3)	24	Double
Matrix3(1,4)	33	Double
Matrix3(2)		Double(1 to 4)
Matrix3(2,1)	6	Double
Matrix3(2,2)	15	Double
Matrix3(2,3)	24	Double
Matrix3(2,4)	33	Double

# XLSTAT C++ / More about matrices

- Here are a few of the matrix functions available in XLSTATVM1C.dll

Routine name	Description
Cpp_M_Center	Center a matrix
Cpp_M_Center2	Center a matrix and return the result in the original matrix
Cpp_M_CholDecomp	Cholesky decomposition of a square matrix $A = LL'$
Cpp_M_Dedupe	Deduplicate a matrix of doubles
Cpp_M_Determinant	Determinant of a matrix
Cpp_M_DuplicateDbl	Duplicate a matrix of doubles
Cpp_M_DuplicateStr	Duplicate a matrix of strings
Cpp_M_Eigen	Compute the eigen decomposition of a square matrix
Cpp_M_Inverse1	Inverse of a square matrix
Cpp_M_InverseTriang	Inverse of a triangular matrix
Cpp_M_InvGeneralized	Generalized inverse of a square matrix
Cpp_M_ProductAB	Matrix product $C=AB$
Cpp_M_ProductABC	Matrix product $D=ABC$
Cpp_M_ProductBV	Multiply a matrix by a vector ( $V2 = B.V1$ )
Cpp_M_ProductTrAA	Multiply a matrix by its transpose
Cpp_M_ProductTrABA	Matrix product $C=A'BA$
Cpp_M_ScalarsA	Multiply a matrix by a constant
Cpp_M_Standardize2	Standardize a matrix and return the result in the original matrix
Cpp_M_Sums1As2B	Sum of 2 matrices multiplied by constants: $C = s1.A + s2.B$
Cpp_M_SVDdecomp	Singular value decomposition of a matrix
Cpp_M_Trace	Trace of a matrix
Cpp_M_Transpose	Transpose a matrix
Cpp_M_Transpose2	Transpose a matrix and return the result in the original matrix

# Creating an Excel function that requires both VBA and C++

- We want to create a spreadsheet function that allows to compute the **Chi-square independence test** on a contingency table, and returns either the Chi-square value, the DF, the critical value or the p-value
  - We need to be able to accept a range as input
  - We need to convert it to a table of doubles before passing it to the C++ routine
- 1. Routine declaration: Needs to be a function to return a result.
  - iResult is 1 for Chi-square, 2 for p-value, 3 for critical value, 4 for DF

```
Public Function XLSTAT_IndepTest(iRange As Range, iResult As Long, _
                                Optional alpha As Double = 0.05) As Double
```

- alpha is used only to compute the critical value

# XLSTAT\_IndepTest (2)

## 2. Transforming the Range into a Table

- A function of XLSTAT allows to quickly transform a Range into a vector of size (1 to n, 1 to 1)

Dim isMissing As Boolean, Missing() As Boolean, n As Long, p As Long

Call ProcessRanges(iRange, isMissing, Missing(), ContTable(), n, p)

- We don't want any missing value, so if there is one we should exit

If isMissing Then

XLSTAT\_IndepTest = CVErr(2015)

Exit Function

End If

- ContTable is now (1 to n, 1 to 1). So the true size of the contingency table is (1 to n/p, 1 to p)

# XLSTAT\_IndepTest (3)

## 3. Calling Cpp\_IndepChiSquare

- Cpp\_IndepChiSquare allows to obtain the 4 possible outputs

Dim ChiSQ As Double, pVal As Double, ChiCrit As Double, DF As Double, RetVal As Long

Call Cpp\_IndepChiSquare(ContTable(1, 1), n / p, p, alpha, ChiSQ, pVal, ChiCrit, DF)

- We then prepare the Output. Built-in function Choose, allows to pick the  $n^{\text{th}}$  element in a list, where n is the first argument

XLSTAT\_IndepTest = Choose(iResult, ChiSQ, pVal, ChiCrit, DF)

# XLSTAT\_IndepTest (4)

4. Application: Effect of a drug on 4 groups of mice

Effect	G1	G2	G3	G4
positive	3	2	0	6
none	3	1	3	1
negative	2	2	3	0

Function Arguments

XLSTAT\_IndepTest

IRange: C24:F26 = {3|2|0|6;3|1|3|1;2|2}

IResult: 2 = 2

Alpha: = 0,096371032

No help available.

IRange

Formula result = 0,096371032

[Help on this function](#)

OK Cancel

Microsoft Excel - Conference.xls

File Edit View Insert Format Tools Data Window Help XLSTAT

G29 =XLSTAT\_IndepTest(C24:F26;2)

	A	B	C	D	E	F	G	H
22								
23		Effect	G1	G2	G3	G4		
24		positive	3	2	0	6		
25		none	3	1	3	1		
26		negative	2	2	3	0		
27								
28								
29							0,096371	
30								



# Programming with XLSTAT ... next steps

- The C++ library of matrix functions allows to quickly develop most data analysis techniques
- XLSTAT includes several functions that facilitate the management of **dialog boxes** and the extraction of data
- XLSTAT is designed to welcome your own home made interfaces and code through the scheme of “**Plug-ins**”. As a proof of concept, we have developed one for a major food company: **a specific toolbar is automatically added to XLSTAT**, and it allows the end users to access specific functions where several XLSTAT functions and specific actions are successively processed

# Conclusion

- VBA is a powerful and easy to use language. It allows
  - Automating tasks in Excel
  - Creating interfaces to interact with the end-user
  - Using XLSTAT built-in VBA functions
  - Using XLSTAT built-in C++ functions
  - Chaining several XLSTAT tasks
  - Creating your own solution based on XLSTAT
  
- In order to further develop the access of the XLSTAT users to programming we will
  - Open a forum
  - Publish additional examples