### QUICKSTART GUIDE TO ACTIVE-X WITH EXCEL

Active-X is a powerful link to Excel. It gives the user through VBA® complete control over Excel from within ED. The use of Active-X is very simple, but the user should be aware of a few difficulties and exceptions. This small guide names a number of these exceptions and gives some examples to make it even easier to start with Active-X.

This guide will through the build up of an Excel® sheet and during this process shows the different problems that can occur and how to solve them. A basic knowledge of VBA is being assumed.

#### Step 1: Creating an application:
To be able to send information to an Active-X application, a handle or pointer must exist. In ED a variable to hold this handle can be created with the `DIM` command.

```
Dim( [ExcelApp], vbOle )
```

Now we have a variable that can hold this handle. The next step is to create the Excel application and store the handle in this variable.

```
ExcelApp := CreateOle( [Excel.Application] )
```

We now have created an Excel application, but the application is not yet visible and it contains no workbooks. But because of the handle we can start to send VBA commands to Excel and make it visible and create these workbooks. The 4Dscript command to do this is `OLE`. The first parameter of `OLE` is the handle. The second is method or property that you want to execute and the third parameter can be the value of a property or a return value.

The first thing that we want to do know is to make the application visible.

```
Ole(ExcelApp, [Visible], 1)
```

In VBA you would write this as "Application.Visible = True". In ED the handle ExcelApp functions as the "Application." part of VBA and the 3th parameter is the value of the property.

---

**TIP 1:**
If you want to know which events, methods and properties are available for use with the Active-X object, go the VBA environment by pressing Alt and F11 simultaneous. Within the environment press F2 to get the object browser and in the box that says "All libraries" select "Excel". Then all objects that are available with their events, methods and properties can be looked at.

---

#### Step 2: Having a workbook
Now we can either create a new workbook or open an existing workbook. In this guide we will do both.

First create a new workbook:

Ole(ExcelApp, [Workbooks.Add])

And set the name of the activesheet (which is automatically the first sheet when opening a new workbook):

```
Ole(ExcelApp, [ActiveSheet.Name], [A name])
```

Here we again see the use of the 3th parameter as the value of a property.

Now we want to open an existing workbook.

---

**TIP 2:**

In VBA we use the "" to determine a string or text. In ED we the [ ] for the same purpose. This means that in stead of using the "" characters, we use the [ ] characters.

Example: ActiveSheet.Cells("A1:B4") becomes ActiveSheet.Cells([A1:B4])

---

**TIP 3:**

In ED the [ ] characters are used to determine that a piece of code contains text or is a string. If the [ ] characters have to be passed as part of this piece of code, the 4Dscript expressions Sbo (Square bracket open) and Sbc (Square bracket close) can be used.

Example: Concat( [ActiveSheet.Cells(], Sbo, [A1:B], String(Count), Sbc, [)] )

---

Suppose we have a workbook called "Attempt.xls" at the location "C:\". The 4Dscript code to open this workbook is the following:

```
Ole(ExcelApp, [WorkBooks.Open([C:\\Attempt.xls])])
```

---

**TIP 4:**

The character \ has a different meaning in VBA. To pass this character correctly you have to pass with an extra \.
Using pdir directly in a concat is not possible either because of this effect. The following function excepts a path and returns it with double backslashes:

```
RegisterFunction(
  [Xpdir],
  [Files],
  1,1,
  [Do(
    Model.XString := [],
    Repeat(
     SubstrCount(p(1), [\]),
     Model.XString := Concat(
      Model.XString, StrSeperate(p(1), [\], Count), [\\]
     )
    ),
    Model.XString
   )
  ]
 )
```

---

If the "Attempt.xls" file is not stored on C:\ but in the work directory of ED, the following syntax would open that file.

```
Ole(
 ExcelApp,
 Concat(
  [WorkBooks.Open(],
  Sbo, Concat(XpDir(Pdir), [Work\\Attempt.xls]), Sbc,
  [)]
 )
)
```

### Step 3: Changing the sheet.
All the formatting that can be performed on a sheet, can also be done through VBA. An example that contains a few different elements is setting a different border around a cell.
In this example a range of cells is selected and the bottom edge is made a thick line.

```
ole([excelapp],
[ActiveSheet.Range([A1:D1]).Borders(4).Weight], [3])
```

---

**TIP 5:**
Within Excel constants have a name. For instance the bottom line of cell in VBA is called "xlEdgeBottom". These constants have to pass as the value that they contain. To find out which constants are available have a look at the help file or at the object browser (see TIP 1)

---

**TIP 6:**
To know which commands have to be used to perform a certain action, record a macro of that action and than look at the code of that macro.

---

For more information about the commands that are available in VBA or how to use them take a look at the help files supplied with VBA.