1 introduction

One of the new features in ED 8.1 is the new ExcelActiveX atom. In contrast with the old Excel atom, which used the DDE mechanism to link ED and Excel, the ExcelActiveX atom uses the more modern ActiveX mechanism. This yields better performance, better interoperability with non-English versions of Excel (decimal comma vs. decimal point) and the ability to access more than one workbook or worksheet at a time.

The ExcelActiveX atom allows you to:

- connect to an Excel workbook
- have more than one ExcelActiveX atom in the model if you have multiple workbooks
- create atom references for easy access from 4DScript
- use 4DScript functions to read/write individual cells and sheets in the workbook
- create tables which are automatically copied to/from Excel on reset or at a specified time

Existing models using the old Excel atom will of course continue to work, but for new models we recommend to use the ExcelActiveX atom instead of the Excel atom. Both can be found in the Data section of the library tree.

This tutorial teaches how to use the ExcelActiveX atom. It is created for people that have some experience in ED in building simple models and have some knowledge of 4DScript.

2 Connecting to Excel

Before you can start working with Excel you first have to establish a connection to it.

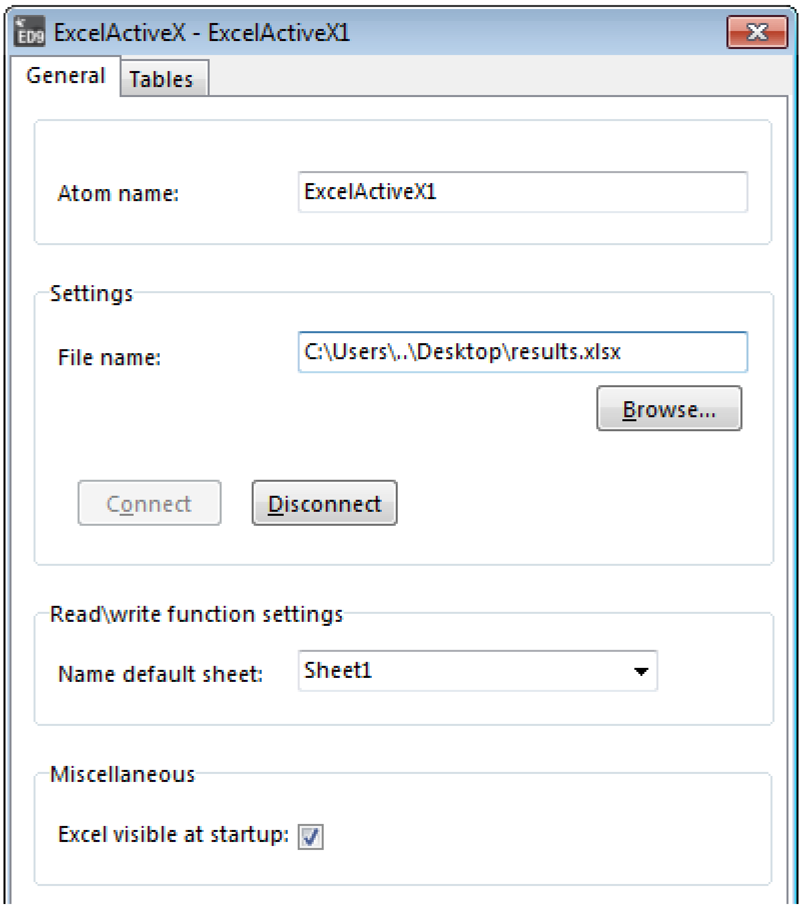
Step 1: Drag an ExcelActiveX atom in your model

Easiest is to connect to a ;workbook that already exists. In this example we assume you have created a new Excel workbook named Results.xls. Drag an ExcelActiveX atom from the Data section of the Library into your model.

Step 2: Connect to the workbook

Double–click on the ExcelActiveX atom to open the user interface of the atom, see Picture 1. Browse to the Results.xls workbook. Click Connect, the Excel file will open. Make sure the option *Excel visible at start-up* is switched on, otherwise the connection will be established but the Excel workbook will not appear on the screen.

The link to the Excel workbook has now been established. If you save the model and re-open it later, the model will establishes the link automatically, step 1 and 2 do not have to be repeated manually.



Atom name: ExcelActiveX1

Settings

File name: C:\Users\..\Desktop\results.xlsx

Read\write function settings

Name default sheet: Sheet1

Miscellaneous

Excel visible at startup: ✓

Picture 1: ExcelActiveX General page

3 Using ExcelActiveX_Write

In the previous section we have established a connection to the Excel workbook. We can now use `ExcelActiveX_Write` to write data to the workbook. We can use this 4DScript function to write to any of the sheets of the workbook. We will be writing mostly to *Sheet1*, so we make this sheet the default sheet.

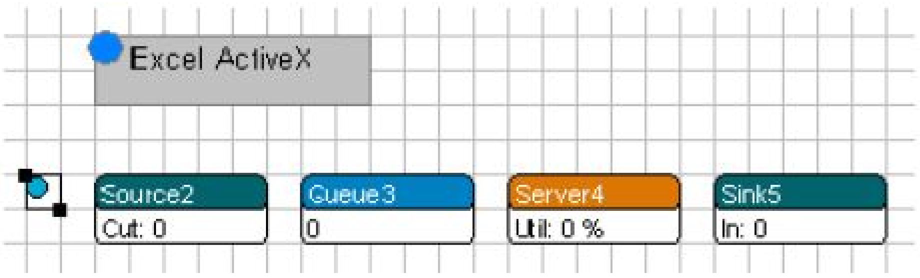
Step 1: Setting the default sheet

In the User-interface of the ExcelActiveX atom open the list behind *Name default sheet*. If the connection is established the list will contain the names of all sheets in the workbook. Select the sheet you want to write to, in this case Sheet1.

If you later change the name of the sheet in Excel, make sure to come back to the ExcelActiveX user interface to change the name here as well.

Step 2: Build a model

Now build a simple source-queue-server-sink model in ED, see Picture 2. We will use this model to generate some data to write to Excel.



Picture 2: Enter caption here

Step 3: Writing code in the model to write data to Excel

In the example we will use Excel to keep track of the lead times. Write the following command on the *Trigger on Entry* of the Sink:

Trigger on Entry: Write data to Excel

```
ExcelActiveX_Write(Input(c), 1, Age(i))
```

The command `ExcelActiveX_Write(e1, e2, e3)` writes the result of expression e3 to the cell at row e1, column e2 of the default sheet. The `Age` command returns the number of seconds that have passed in the model since a product was created by its Source. When measured at the Sink this is the lead time of that product. In this *Trigger on Entry* the `i` in `Age(i)` represents the product that just entered the Sink. Similarly, `Input(c)` returns the number of products that have entered c, where c represents the Sink itself. This means that the age of the first product entering the sink goes to row 1, the second to row 2, etc.

Step 4: Run the model

Set the speed of the simulation model to a low rate, reset the simulation and start a run, then switch from ED to Excel. If everything went correctly you can now watch the lead times being written to the first column of Sheet1 while the model runs. Note that if we reset the model, the old data is not cleared in the Excel sheet. If we start a new run the data will be written over the old data, which means that if we do a shorter run only the fist part of the Excel data will be from this new run. The data is written over the old data, but if you do a shorter run, the data will be a mix of both.

Step 5: Adjust the column to write to

To write the waiting times in the queue to the second column, write the following command on the *Trigger on exit* of the Queue:

Trigger on Entry: Write data to Excel

```
ExcelActiveX_Write(Output(c), 2, Age(i))
```

Note that we use not `Input(c)` but `Output(c)`. Here `c` is the queue and `Output(c)` is the number of products that have left the queue so far. We use `Output(c)` instead of `Input(c)` because a queue can hold more than one product. If three products have entered the queue before the first leaves, `Input(c)` would be 3 but we want to write the waiting time of the first product to row 1. It is very important to always consider carefully whether to use `Input(c)` or `Output(c)`.

The basic rule is: on *Trigger on Entry* use `Input(c)`, on *Trigger on Exit* use `Output(c)`.

If you are reading this tutorial purely to record lead times and similar measurements, we also suggest you look at the simpler Data Recorder atom. This atom helps the user to collect data and write it to the correct row in Excel without using too much 4DScript. The ExcelActiveX atom in combination with 4DScript is of course a much more flexible approach.

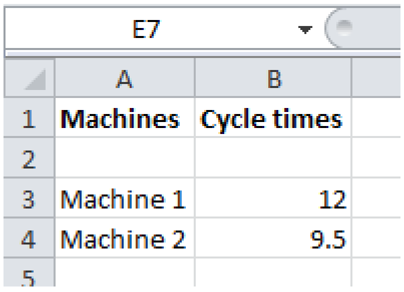
4 Using ExcelActiveX_Read

Until now we have described how to export data to Excel, such that the data gathered in during the simulation run can be further analyzed. But reading data from Excel can also be useful. There might be historical data available of the service times of the machines or processes or of the inter-arrival time of customers and you would like to use these to test how different strategies would have worked out. Another reason is that you can group all settings of interest in the model in one Excel sheet to have a clear overview of all important parameters of the model.

In this example we will use ExcelActive_Read to read the average cycle times of the servers from an Excel sheet instead of setting them directly in the model.

Step 1:

Go to Sheet2 of your Excel sheet and enter some settings, see Picture 3.



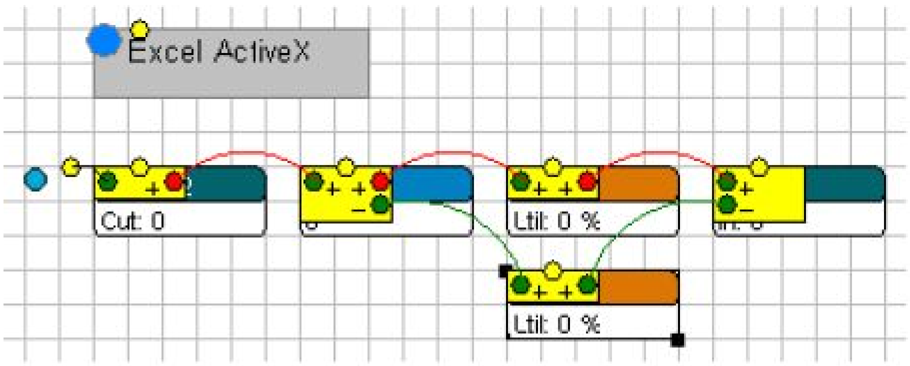
E7

	A	B
1	Machines	Cycle times
2		
3	Machine 1	12
4	Machine 2	9.5
5		

Picture 3: The Excel file Result.xls

Step 2:

Add a second server to the model and connect it to the queue and sink, see Picture 4.



Picture 4: Model with two parallel servers

Step 3:

Make sure to change the *Send To* option of the queue to send products to both servers. Furthermore, set the *Cycletime* of the first server to:

Cycletime: Read data from Excel

```
ExcelActiveX_Read(3, 2, [Sheet2])
```

and that of the second server to:

Cycletime: Read data from Excel

```
ExcelActiveX_Read(4, 2, [Sheet2])
```

The first read command reads cell B3 (= row 3, col 2), the second B4. Both commands explicitly state they want to read from Sheet2 by specifying it as their third argument. If no third argument is given, the default sheet is used. In the previous section we have set the default sheet to Sheet1.

Step 4:

Reset the model and let it run for a while. If no error messages pop up, increase the speed and let it run some more. Then stop it and check the Summary Report to verify the average stay times in the servers. The first should be precisely the value specified in Excel while the second, which has `NegExp()` applied to it, should be near the average specified in Excel.

It is certainly convenient to group all settings together on a single Excel sheet, but it is relatively slow to read data from Excel every time a service time has to be determined. Reading data into ED through a table results in an increase in speed and is therefore recommend in most situations.

5 Copying tables from and to Excel

In this example we will not separately read and write individual values, but use the more efficient block copy functionality of the ExcelActiveX atom. The tables tab of the ExcelActiveX atom allows you to create tables and give each of these tables settings to read or write a block of data to Excel on reset or at a specified time.

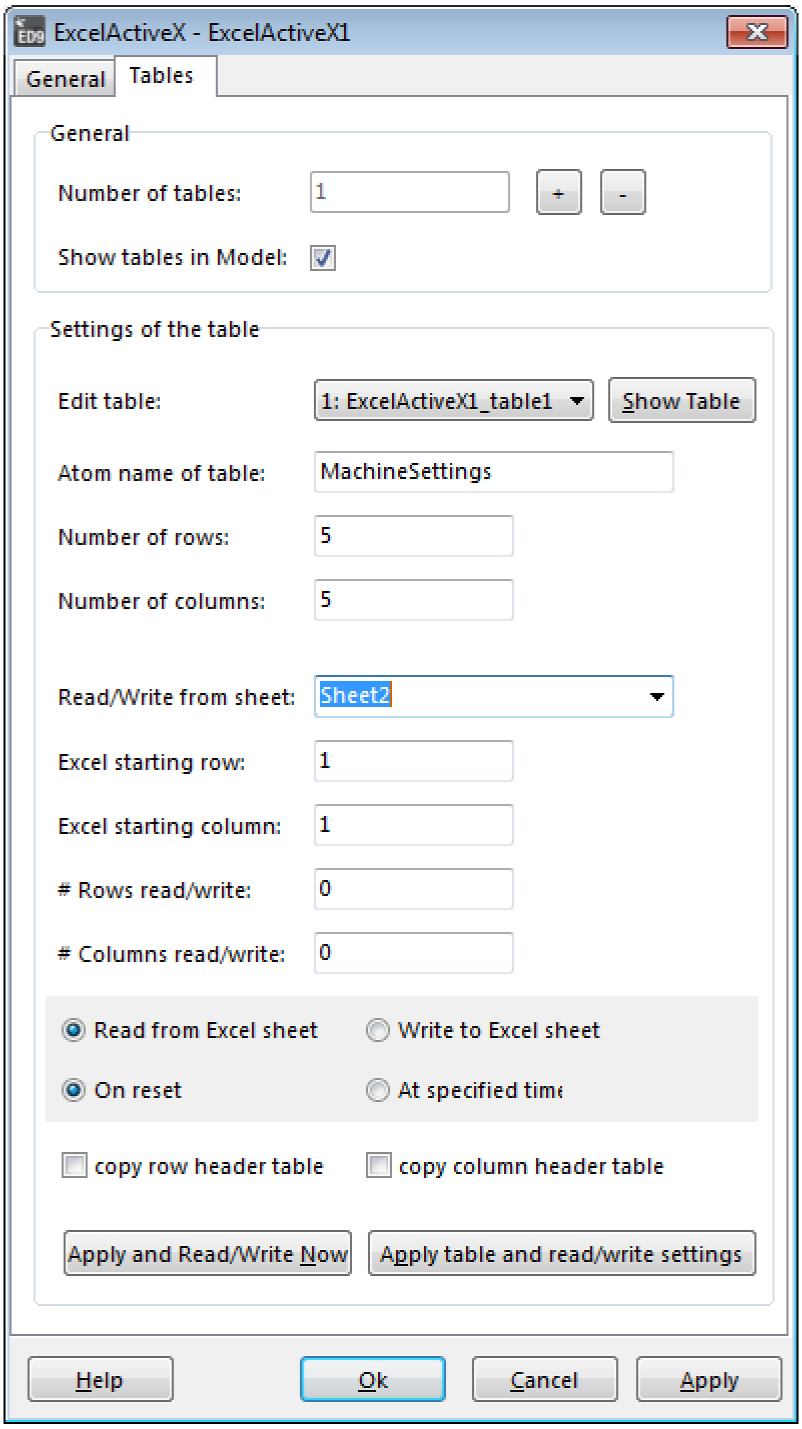
As a first example we will read the table with the machine times on Sheet2 of results.xls

Step 1:

Open the User Interface of the ExcelActiveX atom and go to the Tables tab. Click the '+' button to create a new table. The new table should immediately become visible in the model, attached to the ExcelActiveX atom.

Step 2:

Change the name of the table to MachineSettings. Leave the table at size 5x5. Set it to read/write from Sheet2 and set the number of rows and columns to copy also to 5x5. Finally, set it to Read from Excel on Reset. First click *Apply table and read/write settings* before you click OK.



General

Number of tables: 1

Show tables in Model: ☑

Settings of the table

Edit table: 1: ExcelActiveX1_table1

Atom name of table: MachineSettings

Number of rows: 5

Number of columns: 5

Read/Write from sheet: Sheet2

Excel starting row: 1

Excel starting column: 1

Rows read/write: 0

Columns read/write: 0

[✓ Read from Excel sheet] Write to Excel sheet

[✓ On reset] At specified time

☐ copy row header table ☐ copy column header table

Picture 5: User interface ExcelActiveX atom Tables tab

Step 3:

Reset the model. The data will now be imported from Excel. Right click on the table atom to check if the data has been copied to the table. To access the data that is now stored in the MachineSettings table of ED it is useful to create aliases for the table.

Step 4:

Double-click on the table atom and click in the check box to select *Create alias*. Click OK. If the option *Use atom name* is checked ED will now create three aliases based on the atom name which can be used to access the data in the table. To read data from the table the alias `MachineSettings(e1,e2)` is created. This function will return the value stored in cell(e1,e2). The second alias is `setMachineSettings(e1,e2,e3)` which allows you to set the value e3 in cell(e1,e2) of the MachineSettings table.

Step 5:

Change the *cycletime* of the first and second server such that the data is read from the MachineSettings table instead of from Excel. Use the aliases created in the previous step.

We can also store the data gathered during a simulation run in an ED table and write this table as a whole to Excel.

Step 1:

First add a table to the ExcelActiveX atom. Double-click on the Excel atom and go to the tables tab. Click on the '+' button to add another table. Open the list box 'Edit table' and select the new table. You can now change the settings for the new table.

Step 2:

Change the name of the table to Results. As in section 1.3 we would like to write the lead times and waiting times to this table. We thus set the size of the table to 1000 rows and 2 columns we can gather the results of 1000 products. Select Sheet 3 and select *Write to Excel* and set the number of rows to read to 1000 and columns to 2. Don't forget to press the *Apply table and read/write settings*.

We first need to run the model and write the data to the results table before we can write the data to Excel. As we did for the MachineSettings we can also create aliases for the Results table.

Step 3:

Create the aliases Results and setResults for the results table. Change the code in the *Trigger on Exit* of the Queue and the *Trigger on Entry* of the Sink, such that the data is written to the Results table in ED instead of to Excel. Run the model for 8 hours and right click on the table to check whether the data is written to the table.

There are several ways to export the data in the Results table to Excel. You can either do this by hand or automate it by specifying the time during the simulation at which the data should be written to Excel.

Step 4:

Double-click on the ExcelActiveX atom and go to the tables tab. Select the Results table in the *Edit table* field. Click the button *Apply and read/write now*. We have not changed any settings so the only thing that will happen is that the data gathered in the results table is now exported to Sheet3 of the Results Workbook. Check if the data is indeed in written to Excel.

Step 5:

We can also automate the export to Excel. Clear Excel Sheet3. Go to the tables tab in the ExcelActiveX atom and select the Results table in the *Edit table* field. Check the option *At specified time*. After you have checked the option a textbox will appear.

Here you can type in `hr(8)`. In the Run Control, enable *Run until stop time* and set the stop time at 8 hours. Then run the model for 8 hours. The data will be send to Excel just before the end of the run.