

Simple User Interfaces for PCRaster with Excel

Willem van Deursen, PCRaster Environmental Software

willem@pcraster.nl

October 29, 2009

@2009, Willem van Deursen

Introduction

The fact that PCRaster is a scripting environment for model building offers enormous flexibility to steer and control PCRaster runs from other software. This paper describes the use of Excel (and other MS Office applications such as PowerPoint) to create Simple User Interfaces to control PCRaster models. In this paper, 'controlling is model' means user control over the start of a model run, the definition of parameter values and some control over output visualization. Standard Excel functionality is used, and only limited use is made of macros and VBA: techniques a PCRaster modeler should be able to handle or learn.

This paper describes various simple techniques to build Simple User Interfaces. Once you understand these basic techniques, as an Excel or PowerPoint user you can use all your Excel and PowerPoint tricks to enhance the Simple User Interfaces. The example dataset that accompanies this paper can be obtained from www.carthago.nl under PCRaster.

This paper describes the following steps:

1. Visualization of maps and timeseries under control of Excel
2. Assigning model parameter values and starting PCRaster runs under control of Excel
3. Using PCRaster model results (timeseries) in the spreadsheet

This paper resulted from Simple User Interfaces we built for the "Perspectives in Integrated Water Resources Management" project in cooperation with Marjolijn Haasnoot and Onno Van Logchem (Deltares, The Netherlands). (<http://public.deltares.nl/display/CAW/Perspectives+in+Integrated+Water+Resources+Management>)

The paper uses one of my first PCRaster models, the CalGIS model. Annex 1 gives a short description of the CalGIS model. You should run this model once, to make sure all data is generated (PCRCalc -f CalGIS.pcrmod).

NOTE: Applying the techniques presented here allows for strict personal use unless you or your organization do own a PCRaster developers license. From the free license terms of PCRaster, only strict personal use of wrapper-applications is allowed. You are not allowed to deploy wrapper-applications of any kind built with PCRaster, nor are you allowed under the free license to redistribute (parts of) PCRaster.

1. Visualization of maps and timeseries under control of Excel

Start in Excel with a blank sheet. Save the Excel file somewhere on your system, so the Excel workbook has a filename and (more important): a file path and a default folder. I saved my Excel workbook in the same folder as where the maps of my simple model are located. This means the default folder for this Excel workbook is now the same as where my maps live.

For most of these tasks, you will need the 'Control Toolbox'. You can open the Control Toolbox by going to the View menu. Select Toolbars and then click the check of the Control Toolbox (figure 1). This will bring up the 'Control Toolbox'. Move this control box out of your way, but don't close it: you will need it later!

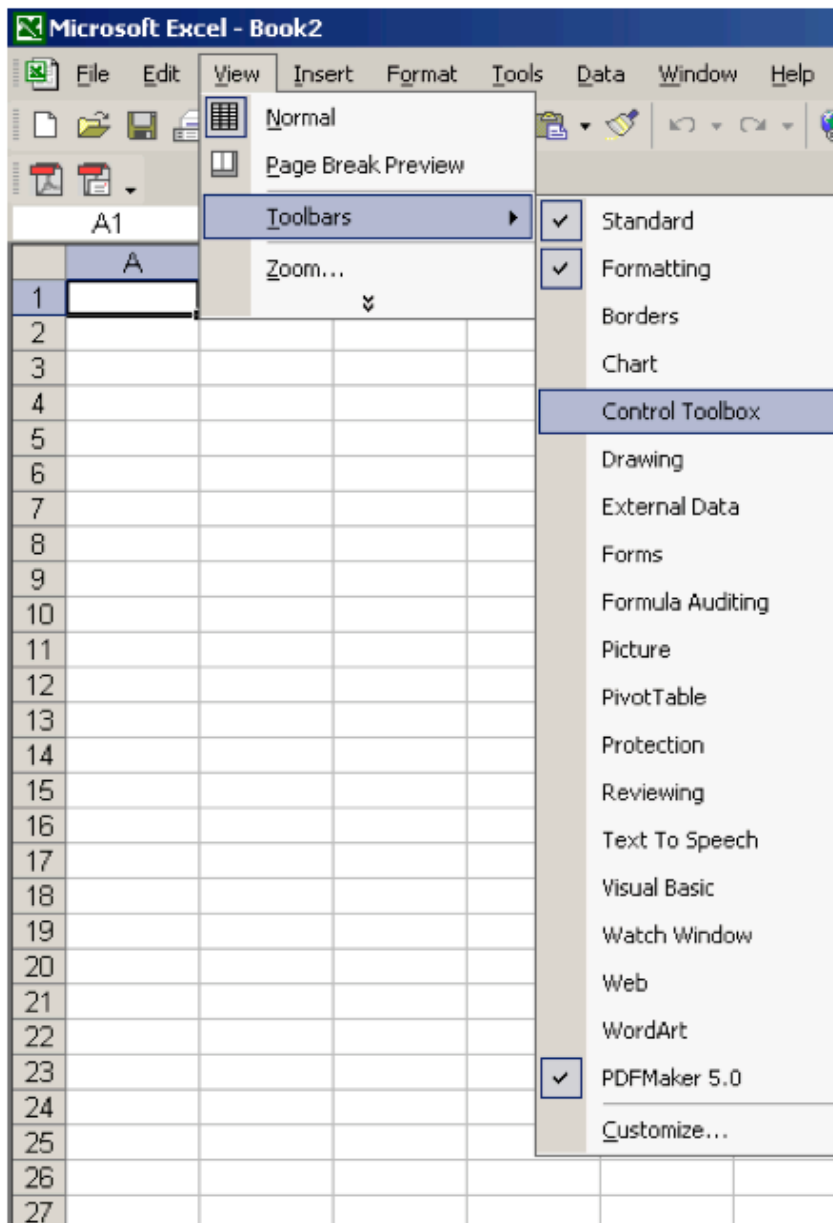


Figure 1

The core of it all is a simple technique to call an external program from Excel.

Cell B1 is going to be the name of the program you want to use. In this example, we want to use the Aguila program to show a map. In cell B1, type the entire path to the Aguila program. On my system the folder that contains my PCRaster applications is C:\Program Files\PCRaster\apps\, so in cell B1 I type

C:\Program Files\PCRaster\apps\aguila.exe.

Cell B2 is going to be the filename of the map I want to display. This filename has to be relative to the default folder of my Excel workbook. I saved my workbook in the same folder as my PCRaster maps, so I don't need to specify the relative path to the folder. In cell B2 I type SoilType.pcrmap, since that is the map I want to display.

In cell B3 I store the path to the folder of my PCRaster maps. Although it is not absolutely necessary to store this path, it makes it easier to refer to the path and allows Excel to refind the default folder.

	B3	fx C:\PCRasterTestModel	
	A	B	C
1		C:\Program Files\PCRaster\apps\aguila.exe	
2		soiltype.pcrmap	
3		C:\PCRasterTestModel	
4			
5			
6			
7			

Figure 2

Now I need to set up a CommandButton that will execute the program defined in cell B1 with the argument (parameter) specified in cell B2. In the 'Control Toolbox' click the 'Design Mode Toggle' to enter Design Mode.

Design Mode Toggle

Command Button Control



Figure 3

Once you are in 'Design Mode' (meaning: The Design Mode Toggle is Down) you can click the 'Command Button Control'. Your cursor will change to a + as you move off the control toolbox and you can then click in cell A3 to draw a CommandButton on your spreadsheet. You can use the resize squares on the CommandButton and/or drag the CommandButton to the right location. Now right-click on the CommandButton you've just drawn and select Properties from the menu of options that appears.

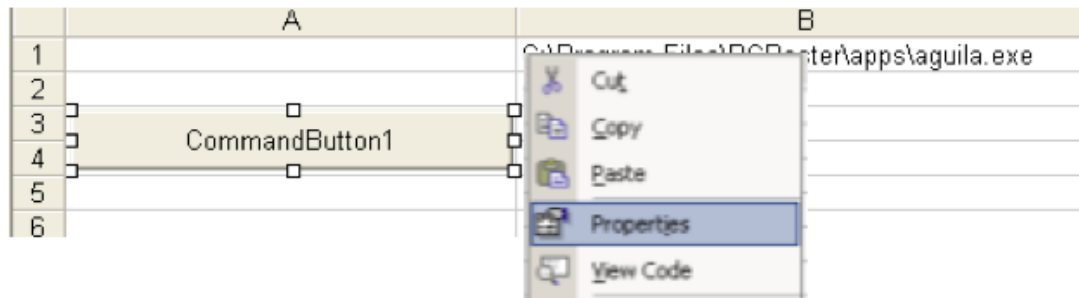


Figure 4

Selecting Properties will open the Properties box. The properties can be listed either alphabetically or grouped by category depending on which of the two tabs at the top you select.

The property you are most interested in is the Caption property: change this into some meaningful caption of your CommandButton.

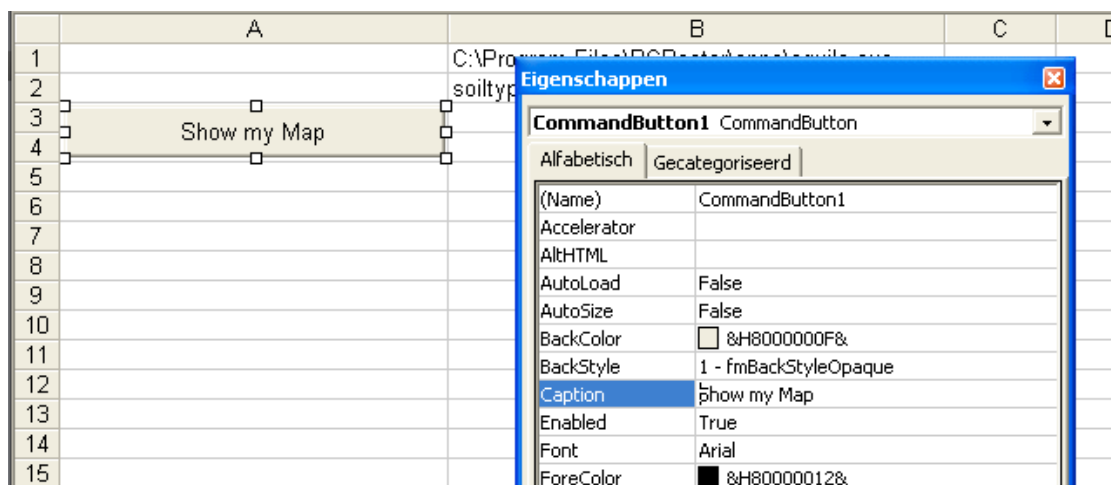


Figure 5

Close the Properties Box.

While still in Design Mode, double-click the CommandButton. This will open the Visual Basic Code Editor, where you can type in the code to be executed when clicking your CommandButton.

In this Visual Basic Code Editor, type the following lines:

```
Private Sub CommandButton1_Click()  
    cmdLine = Chr(34) + Range("b1") + Chr(34)
```

```

cmdLine = cmdLine + " " + Range("b2")
ChDir Range("b3")
Shell cmdLine, vbNormalFocus
End Sub

```

(Note that Excel already provided the first line and the last line of this code).

What does this code do?

The line

```
cmdLine = Chr(34) + Range("b1") + Chr(34)
```

creates a variable called cmdLine and puts the value of cell b1 in that variable. It puts double quotes (that's the meaning of Chr(34)) around the b1 value, so that the command processor correctly interprets the spaces that occur in the Aguilas path.

The line

```
cmdLine = cmdLine + " " + Range("b2")
```

extends the variable cmdline with a single space and the value of cell b2. (Note that we do not put double quotes around the value of b2! The reason why we do not put the double quotes here will be explained in the section on passing other parameters to the called program).

cmdLine now contains (including the double quotes!!)

```
"C:\Program Files\PCRaster\apps\aguila.exe" soiltype.pcrmap
```

The line

```
ChDir Range("b3")
```

will make sure the default folder for Excel is actually the folder where your maps are stored. (ChDir actually means Change Directory, the Visual Basic command to change the default folder or directory. Note that ChDir does not work over different drives, so keep your spreadsheet and maps on the same drive!!)

The line

```
Shell cmdLine, vbNormalFocus
```

will execute the cmdLine variable in the default folder.

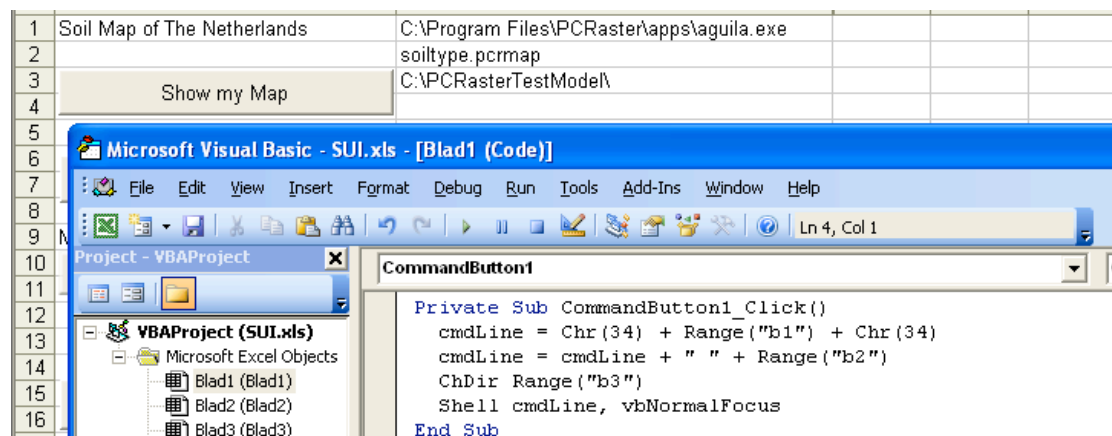


Figure 6

Now we are ready to see if everything works as designed. Close the Microsoft Visual Basic window, and exit Design Mode (meaning: click the Design Mode Toggle on the Control Toolbox again, so the Design Mode Toggle is Up). Your spreadsheet is now in Running Mode, the normal operation mode. You can click your CommandButton now, and that should bring up Aguila, showing the map you named in cell B2.

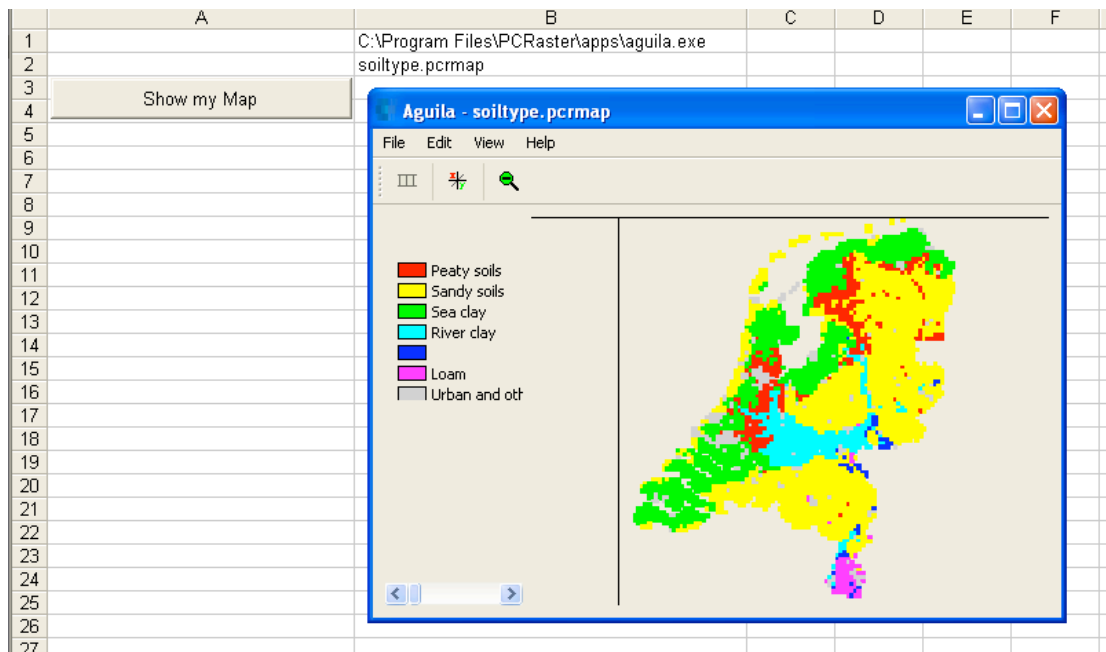


Figure 7

If this all works as designed, we are going to extend this Simple User Interface with a listbox where we can select the maps we want to see.

Use column C to set up a list of menu entries of the maps you want to display. Use column D to list the filename (and file path) for the maps in column C.

C	D
Map Title	Path to Map
Soil Map of The Netherlands	soiltype.pcrmap
Sandy Soils in the Netherlands	sandysoils.pcrmap

Figure 8

Because we saved our Excel file in the same folder as where the maps are located, and we ChDir to the folder with the maps, we don't have to specify a file path to the maps, only a filename will suffice.

In cell A1 we want to create a drop down list of the Map Titles. Excel offers a very convenient way to do this: The validation drop down list

These lists are contained within a cell on your worksheet and the drop down arrow (to the right of the cell) does not appear unless the cell is selected. To add a list to a worksheet place the cursor in the required cell (In our case: A1) and then select Data | Validation | Settings. In the dialog box select 'Allow: List' and ensure the 'In-cell dropdown' box is ticked. The source data refers to a range of cells containing the selection of options, in our case: C2:C3.

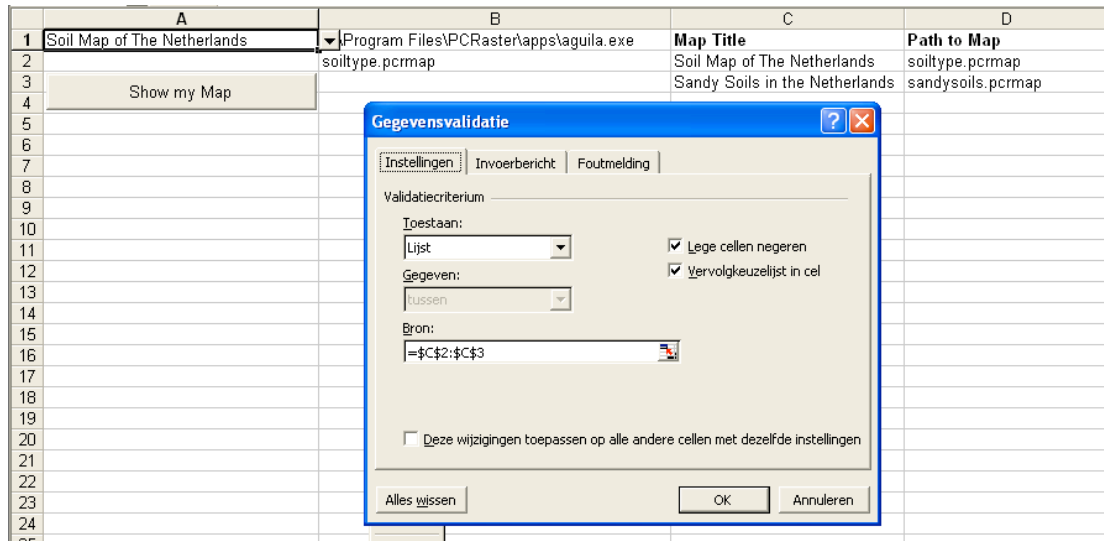


Figure 9

Now check that in cell A1 we have a drop down list box from which we can select one of the titles defined in range C2:C3.

The map to be displayed when we press the CommandButton is defined in cell B2. Currently, this definition is static: it doesn't change according to the selection in cell A1. Standard Excel functions allow us to make the link: erase the static file name in cell B2 and replace it with the function
`=VLOOKUP(A1,C2:D3,2,FALSE)`

Check that your file name definition field in B2 now actually reflects the choice you make in cell A1. Select a map with the drop down list in cell A1 and check to see that your CommandButton still works as expected.

You can also mix and match visualization of maps, timeseries and mapstacks, as shown in the next figure. Be aware however that you have to adjust the range of the Data Validation in cell A1, and you have to adjust the formula in cell B2 to reflect the entire range where your maps are defined.

C	D	E
Map Title	Path to Map	
Soil Map of The Netherlands	soiltype.pcrmap	
Sandy Soils in the Netherlands	sandysoils.pcrmap	
Heather Map Stack	heather --timesteps [1,100]	
Grass timeseries	grass.pcrts	

Figure 10

For a slightly more advanced Simple User Interface, hide the columns B, C and D, so that only the user controls of your User Interface are visible.

Using filenames with spaces and adding extra parameters to the commandline

We can use column D not only to specify the path to the map to be displayed, but also to specify extra commandline switches for the external program. In cell D4, we could also specify the map stack with the new Aguila syntax, thus using

heather --timesteps [1,100]

as the commandline.

Now it becomes clear why we didn't use double quotes in our VBA commands for this part of the cmdLine: enclosing the entire parameter list in double quotes would make it impossible for Aguila to interpret the parameters. However, this leaves one problem unsolved: how do we visualize maps when the path to the map contains spaces? This is simply solved by using the necessary double quotes in the actual string in column D (figure 11).

C	D	E	F	G	H	I	J	K	L
Map Title	Path to Map								
Soil Map of The Netherlands	soiltype.pcrmap								
Sandy Soils in the Netherlands	sandysoils.pcrmap								
Heather Beetle Map Stack	"Z:\carthago On My Mac\Documents\PCRaster\SimpleUserInterface\SUI_CalGIS\heather" --timesteps [1,100]								
Grass timeseries	grass.pcrts								

Figure 11

Use of absolute or relative path names

We have made our live relatively simple by using the folder were we stored our maps as the default folder for Excel. There are many situations in which this is not the preferred setup of your model, and there are many situations in which Excel loses track of what the default folder location was.

An argument can be made that it is better to not depend on Excel to keep track of default folder locations, but to always use absolute folder paths to describe the location of your maps. This means you have to completely specify the path to your folder, and so my relatively simple example would become something as shown in the next figure.

	C	D	E	F	G	H	I	J	K	L
Map Title		Path to Map								
Soil Map of The Netherlands		"Z:\carthago On My Mac\Documents\PCRaster\SimpleUserInterface\SUI_CalGIS\soiltype.pcrmap"								
Sandy Soils in the Netherlands		"Z:\carthago On My Mac\Documents\PCRaster\SimpleUserInterface\SUI_CalGIS\sandysoils.pcrmap"								
Heather Beetle Map Stack		"Z:\carthago On My Mac\Documents\PCRaster\SimpleUserInterface\SUI_CalGIS\heather" --timesteps [1,100]								
Grass timeseries		"Z:\carthago On My Mac\Documents\PCRaster\SimpleUserInterface\SUI_CalGIS\grass.pcrts"								

Figure 12

Note the double quotes around each filepath, and note that the end-user will not see this as you will be hiding these columns.

This will make your Simple User Interface more robust (we are not depending on relative path names anymore), but this approach will make it harder to move your Simple User Interface from one computer to another: you have to adjust all absolute pathnames to be correct on the new computer!

Both approaches have their advantages and disadvantages, but be aware of the consequences of using absolute versus relative pathnames!

2. Assigning parameter values and starting PCRaster runs under control of Excel

We can use the same technique described above to create a CommandButton that allows us to run a specific model. To do this, we specify the PCRCalc program in cell B7, and we specify the model to run in cell B8. To specify the model to run, we have to use the -f switch. If we try to type the string

-f calgis.pcrmod

in cell B8, Excel will start to complain about incorrect formulas and incorrect names: Excel is trying to interpret your string as a formula. So, tell Excel that it is not a formula you are typing, but a simple string. Type in

'-f calgis.pcrmod

(Note the single quote as first character in this string!)

in cell B8 and note that Excel now handles the string correctly.

Add a CommandButton in cell A7 (Use the 'Control Toolbox' to toggle to design mode, and add the CommandButton to your sheet). Use the properties box of your CommandButton to assign a meaningful caption and doubleclick on this new CommandButton to open the Visual Basic Editor for this CommandButton. Type in the following lines for CommandButton2_Click:

```
Private Sub CommandButton2_Click()
    cmdLine = Chr(34) + Range("b7") + Chr(34)
    cmdLine = cmdLine + " " + Range("b8")
```

```
MsgBox cmdLine
```

```
ChDir Range("b3")
Shell cmdLine, vbNormalFocus
```

```
End Sub
```

These lines are similar to those for CommandButton1, only now we are referring to cell B7 for the program to execute and to cell B8 for the commandline. I added

the statement “MsgBox cmdLine”, because that will actually show the constructed cmdLine before Excel tries to execute the commandline.

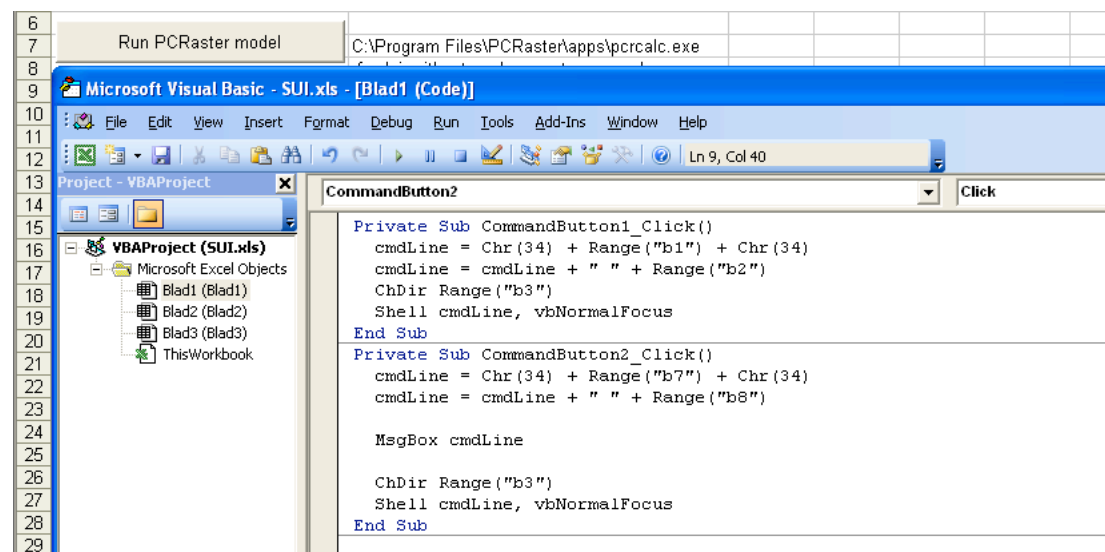


Figure 13

Closing the Visual Basic Editor, toggling off ‘Design Mode’ and clicking the CommandButton2 will now execute PCRCalc with the appropriate commandline. (For a thorough discussion on the Visual Basic Shell function see <http://msdn.microsoft.com/en-us/library/xe736fyk.aspx>)

(Note that we need a fully debugged PCRastermodel, because you won’t have time to read the error messages. Excel immediately closes the output window of pcrcalc. More advanced techniques allow you to start the PCRCalc program from within a CMD-box and allow you to keep this box open. It all boils down to replacing the code for CommandButton2_Click with the following code:

```

Private Sub CommandButton2_Click()
    cmdLine = "cmd /k "+Chr(34) + Range("b7") + Chr(34)
    cmdLine = cmdLine + " " + Range("b8")

    MsgBox cmdLine

    ChDir Range("b3")
    Shell cmdLine, vbNormalFocus
End Sub

```

Now the user has to explicitly close the CMD box where PCRCalc was running. For developing the model, that is good. For an end-user that’s less convenient.

Parameters from the user interface can be passed to the model. I modified my original model to obtain one of its parameters from the commandline. This modified version is the PCRaster model ‘CalGISwithExternalParameter.pcrmod’, to be found in the accompanying dataset. This model makes use of the argument

substitution with a \$-construct (see PCRaster manual or Annex B). Instead of defining the value for the parameter 'beetleinfluence' in the PCRCalc script, I modified my script to obtain this value from the commandline using

binding

```
beetleinfluenceparameter = $1; # heather beetle as external parameter
```

initial

```
beetleinfluence = beetleinfluenceparameter/100;
```

Switch to 'Design Mode' again, and drop a Scroll bar (Slider) control on your worksheet. Open the Properties Box for this Scrollbar control and set the following properties:

```
Min      0
Max      50
LinkedCell B9
```

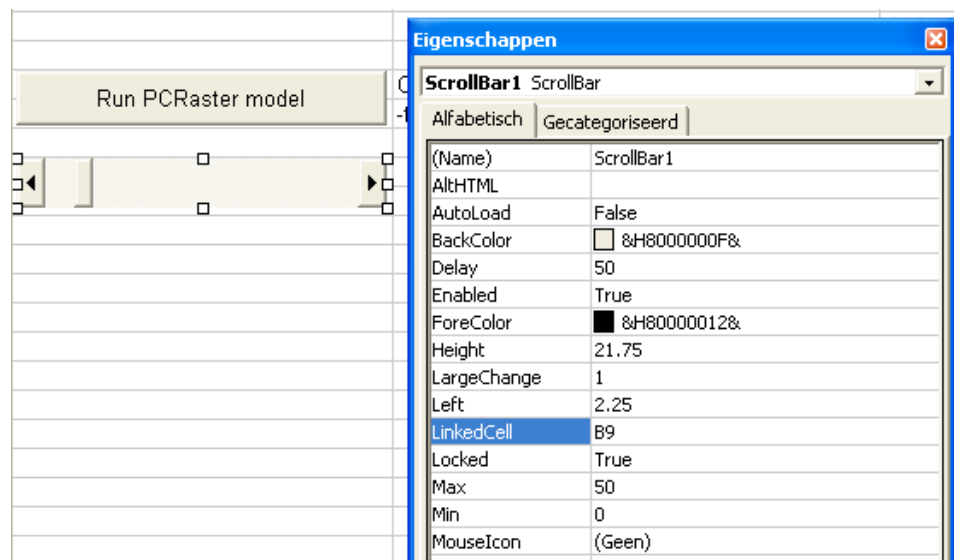


Figure 14

Note especially the LinkedCell property: this will link the position of the slider thumb to a cell in Excel. In this example we linked the scrollbar to cell B9.

Exit Design Mode and modify your pcrastermodel to be called in cell B8 to be 'CalGISwithExternalParameter.pcrmod', so cell B8 should contain

```
'-f CalGISwithExternalParameter.pcrmod
```

(Remember the single quote as first character in this string!)

Modify the Visual Basic Code of your CommandButton2 to include this extra parameter in your commandline, as shown in the code (Remember to switch to Design Mode!).

```
Private Sub CommandButton2_Click()
  cmdLine = "cmd /K " + Chr(34) + Range("b7") + Chr(34)
  cmdLine = cmdLine + " " + Range("b8") + " " + Range("b9").Text
```

MsgBox cmdLine

ChDir Range("b3")

Shell cmdLine, vbNormalFocus

End Sub

Now exit 'Design Mode', type a title for your scrollbar in a cell just above the scrollbar and see if you can run and control your model from the Simple User Interface as you just developed. Hide columns C and D again for a clean interface. Hide column B to get an even cleaner interface.

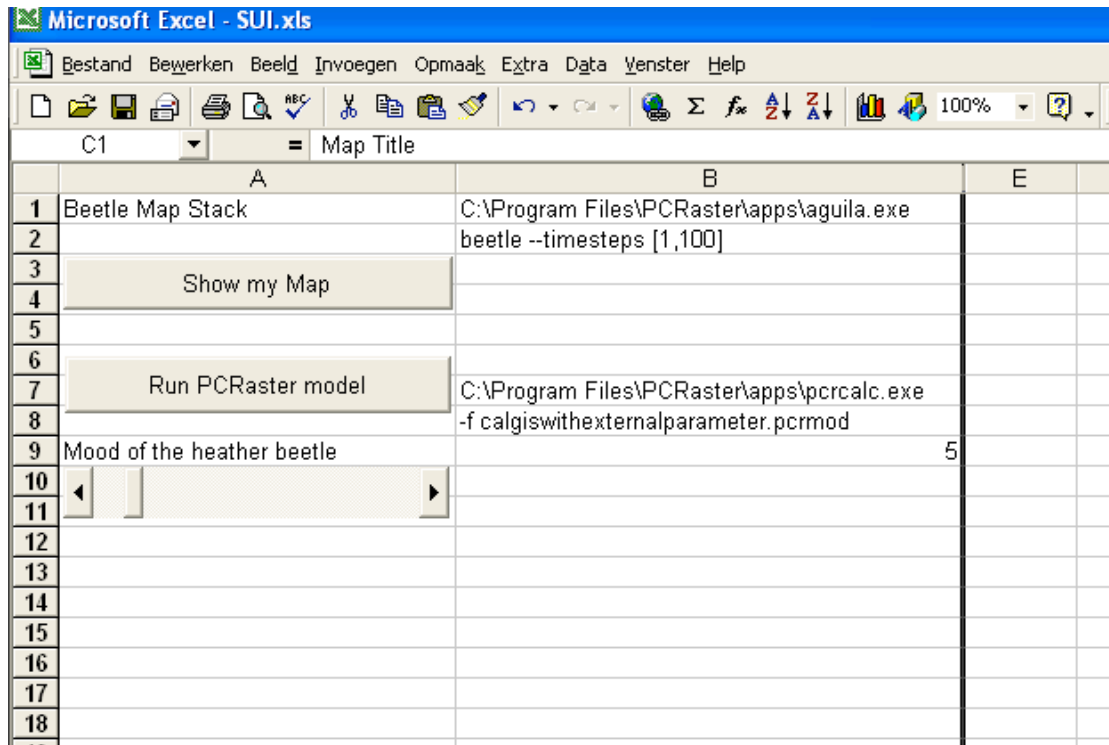


Figure 15

3. Using PCRaster model results (timeseries) in the spreadsheet

The resulting timeseries of the model can be imported and used in Excel by setting up an external data link. Move to cell E1 and select Data -> Import External Data -> Import Data... from the Excel menu. Excel opens a dialog to 'Select Data Source'. In this dialog box, choose 'All files (*.*)' as 'Files of type'. Move to the folder where your model results are stored and select the file 'grass.pcrtss'.

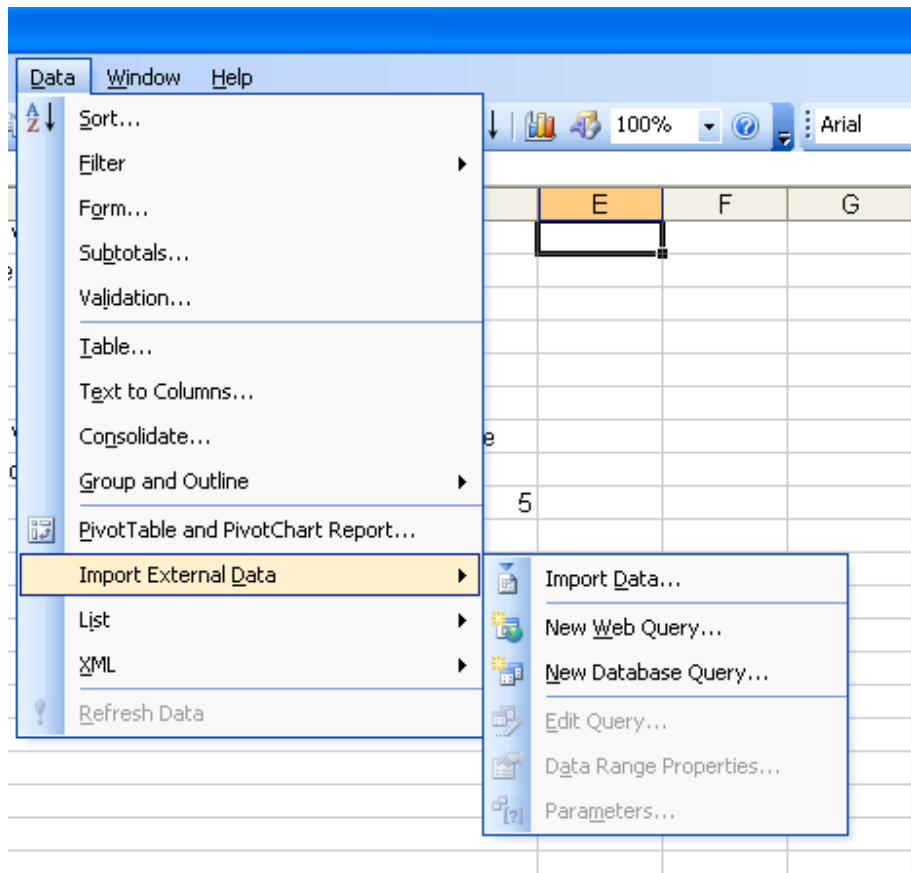


Figure 16

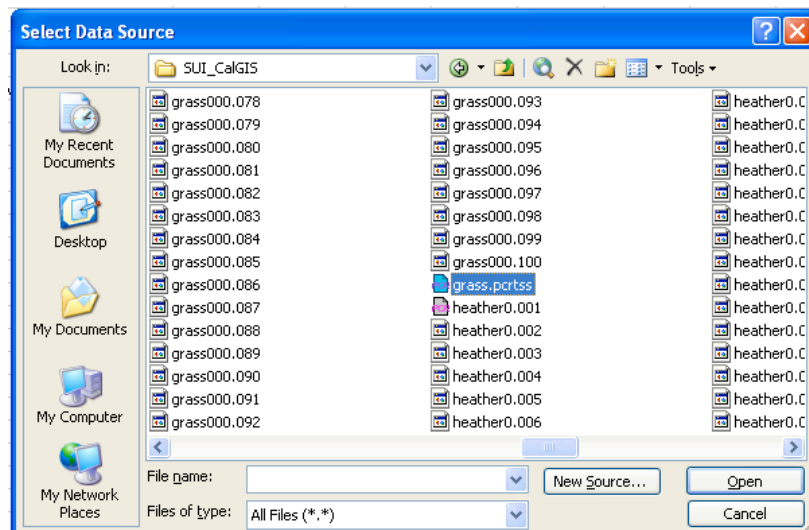


Figure 17

The next dialog is the Text Import Wizard, familiar to most Excel users. In step 1 of the Wizard, make sure that you select 'Delimited' as original data type and click 'Space' as delimiters in step 2. The timeseries file will now be imported. Once we imported this dataset, we can open the Toolbar for External Data (Excel menu View -> Toolbars -> External Data). In the External Data Toolbar select the 'Data Range Properties'

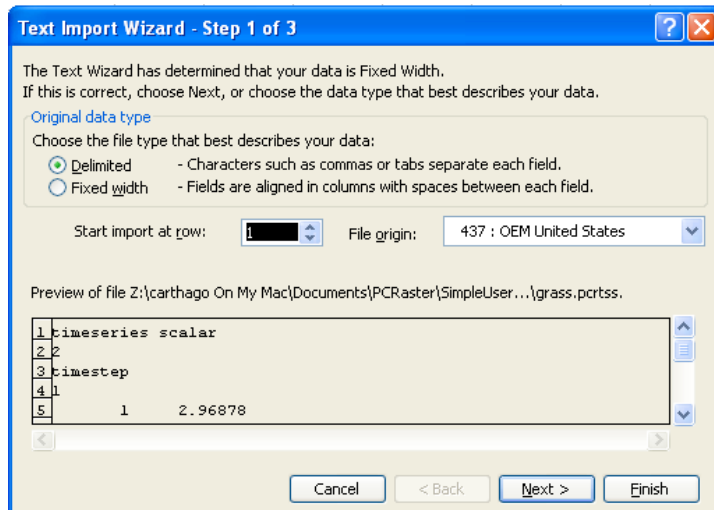


Figure 18

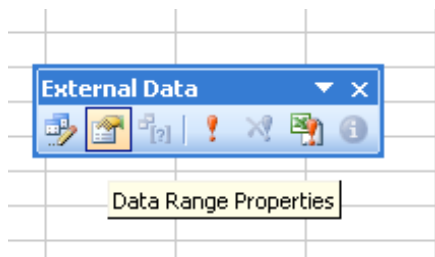


Figure 19

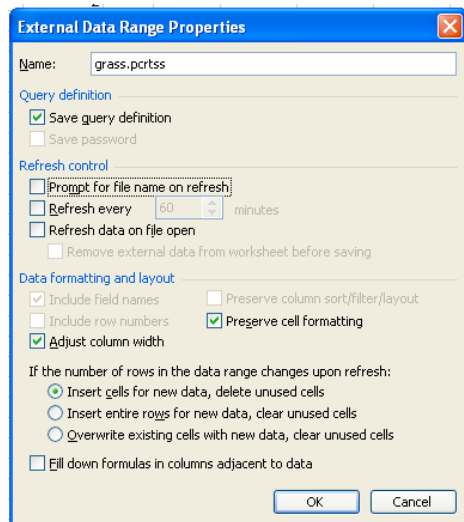


Figure 20

In the 'External Data Range Properties' dialog uncheck the option 'Prompt for file name on refresh' and note the name of the Data Range (grass.prtss, the same as the filename we imported). Now create a graph of this timeseries using your Excel skills. Each time you run the model (pressing the command button for running the model), you can now update your table and your graph by pressing the exclamation mark (Refresh Data or Refresh All buttons) on the External Data Toolbar.

You can automate this process by another CommandButton, with the following VBA code attached:

```
Private Sub CommandButton3_Click()  
    Range("grass.pcertss").QueryTable.Refresh False  
End Sub
```

Note that you would want to add this code to the code to run the PCRaster model (the code of CommandButton2). That way, pressing this button would run the model AND update the timeseriesgraph. However, that's an approach that is a little more involved. The challenge here is that Shell function of Excel doesn't wait for the external program to finish. It starts the PCRCalc program and immediately returns control to Excel. So Excel would immediately start refreshing the External Data. PCRCalc doesn't have the time to finish, so no reliable timeseriesdata would be loaded. On the internet one can find plenty of approaches for Excel to ExecuteAndWait, but the implementation of these functions is beyond the scope of this paper. See <http://www.cpearson.com/excel/ShellAndWait.aspx> for a possible approach.

Annex 1: Description of the CalGIS model



In the Netherlands, many heathlands formerly dominated by *Calluna* have become dominated by grasses (*Deschampsia*). The environmental policy of the Netherlands regards this replacement of heather by grasses as unfavourable. It has been suggested that eutrophication of these heathlands may be a significant factor in the replacement of heather by grasses and field. The heather – grassland system is further influenced by the heather beetle or heather bug. During heather beetle outbreaks it is observed that *Calluna* is more severely affected by heather beetles in the more heavily fertilized vegetation. This suggests that the competition between heather and grasses is not simply due to the higher potential growth rate of the grasses, but may also be caused by a higher susceptibility of the fertilised *Calluna* to the heather beetle.

It is hypothesized that the outbreaks of the beetle are stimulated by nitrogen enrichment, and become more severe and occur more frequently under eutrophic conditions. No outbreaks are observed in young *Calluna* stands (< 5 years), and we assume that the canopy cover of *Calluna* must be more than approximately 50% before outbreaks occur.

During an outbreak of this beetle plague, *Calluna* plants die off almost completely over large area. The opening up of the dwarf shrub canopy enhances the growth of grasses due to an elimination of the competition from *Calluna*.

The CalGIS model contains a simplified process model to experiment with these processes. The CalGIS model is based on the vegetation research of Dr. Gerrit Heil of the University of Utrecht. It has been one of the first models ever to be built in PCRaster. The pictures included are also courtesy of Dr Gerrit Heil. More information can be found in Van Deursen W.P.A. and G.W. Heil, 1994: Analysis of heathland dynamics using a spatial distributed GIS model. *Scripta Geobotanica* 21: 17-27.

CalGIS (*Calluna* GIS model) simulates the heather-grasses competition in the Netherlands using the soil map as the basic input map. CalGIS very optimistically assumes all sandy soils in the Netherlands to be potential heather habitat. This version does not

include the quite substantial disturbances on this habitat in such a densely populated region as the Netherlands.

The CalGIS model is described extensively in my PhD thesis and in a Quickstart tutorial I developed (both can be obtained from www.carthago.nl under PCRaster).

Annex 2: substitution of arguments with a \$-construct

In both command line expressions and model scripts, parts of the model can be substituted from shell-like arguments.

For example, `sum_2.mod` :

```
$1 = $2 + $3;
```

Calling

```
pcrcalc -f sum_2.mod sum_2.map add_1.map add_2.map
```

Is equal to

```
pcrcalc "sum_2.map = add_1.map + add_2.map;
```

On the command line, the model and the arguments need to be separated by `;;` .

For example:

```
pcrcalc "$1 = $2 + $3 ;;" sum_2.map add_1.map add_2.map
```

Basic substitution rules:

- Everything may substituted, even operators, functions, etc.
- If a substitution fails, blanks are returned: no error message is printed, the resulting string might be incorrect. (use `-t` to check substitution without running the model)
- A `$`-sign followed by a number refers to an argument on the command line, e.g. `$1 $2`
- If the `$`-sign is followed by a non-numerical string the argument is a shell or environment variable, for example `$RESULT` .
- Simple arguments may be enclosed in curly braces, e.g. `$1 = $2 + $3` equals `#{1} = $2 + $3`;
This enables prefix-substitution of a variable such as `#{1}.map = $2 + $3`;

Advanced substitution rules:

- A range of parameters can be given in the `#{from,to}` construct. For example:
`$1 = max(#{2,3}) ;; max.map in1.map in2.map` becomes
`max.map = max(in1.map,in2.map)`
- `n` in the first or second arguments denotes the number of arguments:
`$1 = max(#{2,n}) ;; max.map in1.map in2.map in3.map` becomes
`max.map = max(in1.map,in2.map,in3.map)`
- The `#{from,to}` construct prints an `,` between every argument. Another argument separator can be given explicitly:
`$1 = #{2,n,+} ;; sum.map in1.map in2.map in3.map` becomes
`sum.map = in1.map+in2.map+in3.map`
- A wrapper around each arguments can be given:
`$1 = #{2,n+,sqrt($)} ;; sumsqrt.map in1.map in2.map in3.map` becomes
`sumsqrt.map = sqrt(in1.map)+sqrt(in2.map)+sqrt(in3.map)`
In the 4th argument, the wrapper, a `$` is given for where the argument should be inserted. That `$`-sign is optional, allowing the following construct:

$\$1 = (\${2,n,+}) / (\${2,n,+,1})$;; av.map in1.map in2.map in3.map becomes
av.map = (in1.map+in2.map+in3.map) / (1+1+1)

- White space in argument 3 and 4 is kept in the substitution:
 $\$1 = \${2,n, \text{and}, \text{not} \$}$;; andnot.map in1.map in2.map in3.map becomes
andnot.map = not in1.mapandnot in2.mapandnot in3.map
while
 $\$1 = \${2,n, \text{and} , \text{not} \$}$;; andnot.map in1.map in2.map in3.map becomes
andnot.map = not in1.map and not in2.map and not in3.map

Additional notes:

- Currently, comments (#) are not allowed within a substitution
- Note that $\${n}$ denotes the last argument, while $\$n$ means an environment variable named n .
- Note that is an extra substitution level, so one needs to give $\$\1 in an UNIX shell, if the command line expression is enclosed in "-symbols. This is not necessary if it is enclosed in '-symbols.
- If you are trying \$-constructs and you get the message:
ERROR: parse error near line '?' near symbol '?'
then two errors are frequent:
 - you forgot the delimit the expression and the arguments by ;;
 - the substitution went wrong. First try -t, to see what the result of the substitution is. As said before, the substitution mechanism silently ignores most errors, and returns blanks. So be warned.