



FLUIDFLOW

FluidFlow

SCRIPTING HANDBOOK

©Flite Software 2019

| | | |
|---|--|----|
| 1 | Introduction | 2 |
| 2 | Acrylic Valve Turndown Script (Pascal & Basic) | 3 |
| 3 | Butterfly Valve Pressure Loss vs Opening Position Script (Pascal)..... | 14 |
| 4 | Orifice Pressure Drop Script with Excel Report (Pascal) | 19 |
| 5 | Mine Dewatering Script with Excel Report (Pascal)..... | 25 |

1 Introduction

FluidFlow is designed to allow the modelling of fluid behaviour within complex piping systems and accurately predict how the system will work for a given set of boundary conditions. The software uses a number of well-established models and correlations to solve the system.

The FluidFlow *Designer Handbook* gives users an overview on how to develop models using the software. This handbook takes these models a step further and considers how to write scripts to perform dynamic simulations.

Scripting can be used to perform a wide range of dynamic simulations such as;

1. The study of tank fill/drain times based on a set of design pump operating conditions.
2. Analyse system pressure as demands vary.
3. Investigate system control philosophies.
4. Evaluate valve performance for variable speed pumps.
5. Flare stack depressurisation.
6. Analyse scale build-up in systems and study the effect on flow rate.
7. Optimise pump and system performance.

The above is just a brief list of some of the studies which can be completed. Scripting is a powerful tool which helps engineers optimize system performance, producing lower operating costs and lowering carbon emissions.

This manual should be used in conjunction with the software Help file. You will find further information in the Help file at the following locations:

Contents | Scripting.

Contents | Script References.

If you have a specific design application and wish to use an intuitive user friendly program to speed up your design process, contact us at: support@fluidflowinfo.com.

Testimonial:

"FluidFlow is fast, easy to use, accurate and a reliable package. The software drastically cuts design time - these benefits apply not only to the designer but also to the peer review team. During operation of the built systems, the agreement between running plant pressure readings against design data was highly accurate. That bought my full trust in the package".

Mat Landowski, Lead Process Engineer



2 Acrylic Valve Turndown Script (Pascal & Basic)

In this example, we have a single variable speed centrifugal pump transporting acrylic acid to two demand points via two control valves. The control valve at the top of the model (User Number 13) is set to control the flow to 2.87 kg/s and the control valve at the bottom of the model (User Number 14) is set to control the flow to 5.75 kg/s.

We shall write a script which will allow us to review the performance of each control valve as the pump speed is gradually reduced.

In completing the study, we wish to determine if either of the two control valves will lose control of the flow with reduction in pump speed or if we simply reach the minimum pump speed with the valves still in control.

We can plot a curve of valve position v's pump speed so that we can better understand the 'sensitivity' of the butterfly control valves we are using.

The model of the system is shown in Figure 2.1.

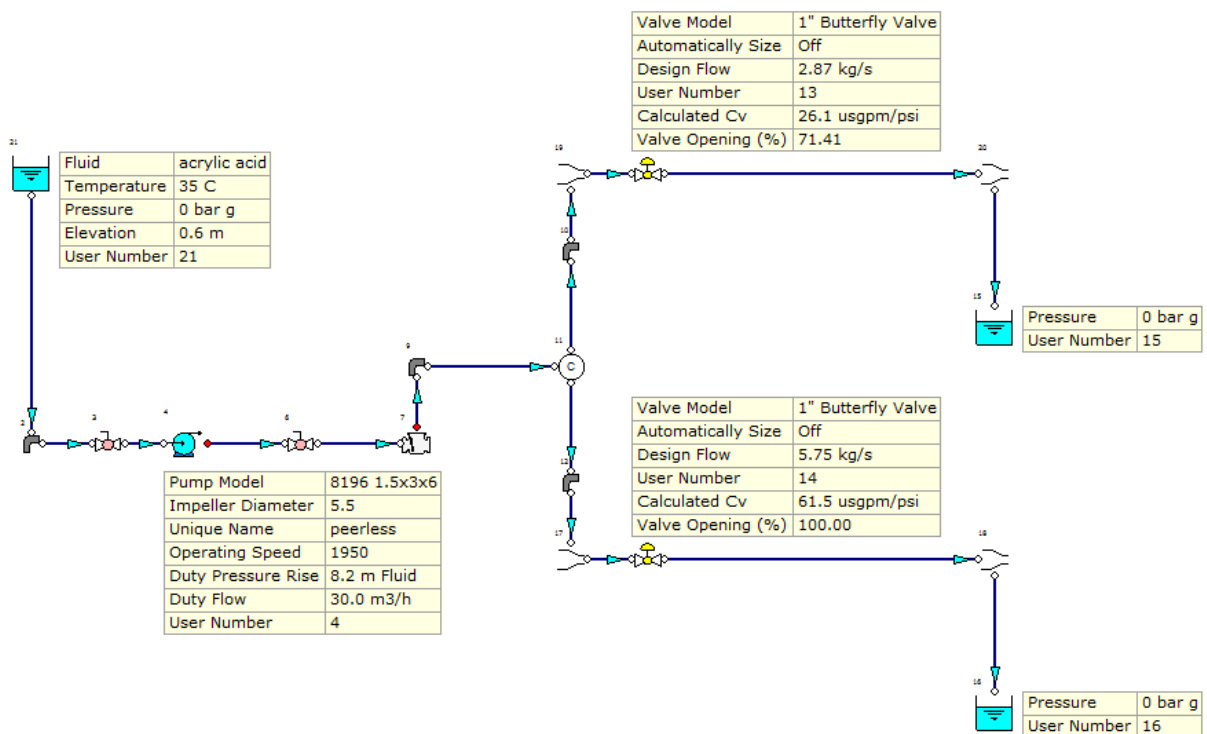


Figure 2.1: Acrylic Valve Turndown Model.

Before we write our script, let's take note of some model-specific data (Table 2.1).

Table 2.1: Node Data

| Node Description | Node User Number | Design Flow rate (kg/s) |
|-------------------------|------------------|-------------------------|
| Centrifugal Pump | 4 | N/A |
| Butterfly Control Valve | 13 | 2.87 |
| Butterfly Control Valve | 14 | 5.75 |

The centrifugal pump has an operating speed range of 2850 to 1450 RPM. It is intended to run the simulation starting at the maximum speed of 2850 RPM and reduce the pump speed in steps of 20 RPM.

Let's now consider writing the script in Pascal. Note, any text in blue below is supplementary and not included as part of the actual script.

Before we start, we need to set the programming language for the script. To do this, select: Script | Pascal (or Basic) from the Script window (Figure 2.2).

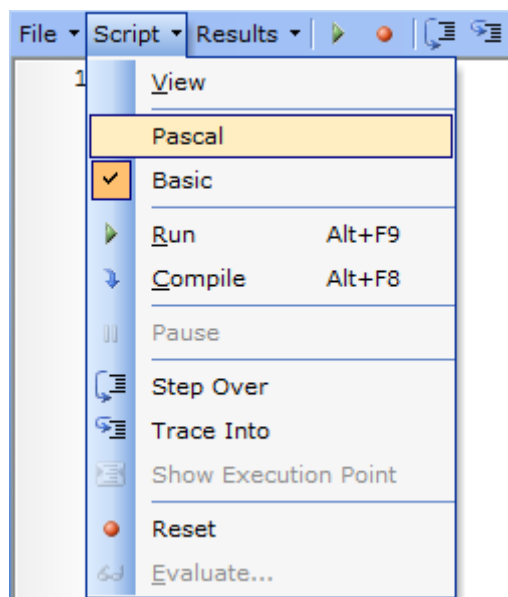


Figure 2.2: Pascal/Basic Script Setup.

2.1.1 Acrylic Valve Turndown – Pascal.

| Script | Description of Script Steps | | | |
|---|--|-----------------------|-----|-----------------|
| <pre>begin PumpSpeed := 2850; MinOpSpeed := 1450; SpeedDecrement := 20;</pre> | <p>Initialise the pump speed and other variables. Note, we need to give consideration to the descriptive terms we use here as they will form a key part of the script used later.</p> <p>(2850 RPM Max. Pump Speed). (1450 RPM Min. Pump Speed). (20 RPM Speed Decrement reducing from Max to Min Speed).</p> <p>Note, the pump in this model is a Peerless 8196 1.5x3x6 model and the capacity curve for this pump model is defined in the pumps database.</p> | | | |
| <pre>Output.Clear; Results.Clear;</pre> | <p>Clear the output window and chart results.</p> | | | |
| <pre>Network.Set(4, 'BoosterOperatingSpeed', PumpSpeed);</pre> | <p>Reset the pump speed and recalculate in case we have made previous calculation runs. The number 4 in this line refers to the User Number of the pump (unique node number – available from Results tab). Now, we need to set the pump operating speed (initial starting speed). We can retrieve the relevant property from the Helpfile. This data can be found at:</p> <p>Contents Script References Property Constants Boosters.</p> <p>We can see from the list that what we need is Property Id 173 "BoosterOperatingSpeed".</p> <table border="1" data-bbox="1167 1161 2103 1206"> <tr> <td data-bbox="1167 1161 1574 1206">BoosterOperatingSpeed</td> <td data-bbox="1574 1161 1742 1206">173</td> <td data-bbox="1742 1161 2103 1206">Operating Speed</td> </tr> </table> <p>The initial booster operating speed can be set to 2850 RPM by adding the term "PumpSpeed" to the script ("PumpSpeed" is 2850 RPM as described at the start of the script).</p> | BoosterOperatingSpeed | 173 | Operating Speed |
| BoosterOperatingSpeed | 173 | Operating Speed | | |

| | | | | |
|---|--|------------------------|-----|-------------------|
| Network.Calculate; | Calculate the network. | | | |
| <pre>while ((PumpSpeed > MinOpSpeed) and (Network.Get(13, 'CalculatedValveOpening') < 99) and (Network.Get(14, 'CalculatedValveOpening') < 99)) do</pre> | <p>Make the main loop using a while statement. Note the “and” condition in the “while” statement.</p> <p>The numbers 13 and 14 refer to the User Number of the control valves. The scenario we are creating here is, as long as the pump is operating at a speed greater than the minimum speed and the position of both control valves is below 100% fully open, run the simulation loop. Note, much like the pump, we have retrieved the relevant valve opening property from the Helpfile. This data can be found at:</p> <p>Contents Script References Property Constants Controllers.</p> <table border="1" data-bbox="1167 683 2089 730"> <tr> <td>CalculatedValveOpening</td> <td>167</td> <td>Valve Opening (%)</td> </tr> </table> <p>We can see from the list that what we need is Property Id 167 “CalculatedValveOpening”.</p> | CalculatedValveOpening | 167 | Valve Opening (%) |
| CalculatedValveOpening | 167 | Valve Opening (%) | | |
| <pre>begin Network.Calculate;</pre> | Calculate the network. | | | |
| <pre>Results.Update;</pre> | Update the chart results by including. | | | |
| <pre>PumpSpeed := PumpSpeed - SpeedDecrement;</pre> | Now we need to reduce the pump speed from the initial starting speed which has been set in the script (2850 RPM). We can therefore use the descriptive terms we have used at the beginning of this script. | | | |
| <pre>Network.Set(4, 'BoosterOperatingSpeed', PumpSpeed); end;</pre> | We need to update the reducing pump speed in the network during the simulation. | | | |

| | |
|--|--|
| <pre> if (PumpSpeed <= MinOpSpeed) then ShowMessage('The system is able to control at the minimum operating speed of the pump') </pre> | <p>Now we can consider the results for the simulation. Write the result summary to a message box.</p> <p>The condition we are considering here is; if the network runs through the simulation and both valves are in control for all pump speed conditions, a message box will appear advising:</p> <p><i>'The system is able to control at the minimum operating speed of the pump'.</i></p> <p>However, this may not actually be the case so we need to include the conditions where either of the valves reach 100% fully open before the pump reaches its minimum speed. Let's consider the next line of the script.</p> |
| <pre> else if (Network.Get(13, 'CalculatedValveOpening') > 99) then ShowMessage('Control valve (13) is fully open at a pump speed of ' + IntToStr(PumpSpeed+SpeedDecrement)) else ShowMessage('Control valve (14) is fully open at a pump speed of ' + IntToStr(PumpSpeed+SpeedDecrement)); </pre> | <p>Now we are considering the condition where either of the valves reach 100% fully open (more specifically, over 99% open) prior to the pump reaching its minimum speed, i.e. either of the valves lose control of the set flow rate.</p> |
| <pre> Network.Refresh; Results.Refresh; end; </pre> | <p>Refresh the network and results</p> |

2.1.2 Acrylic Valve Turndown – Basic.

| Script | Description of Script Steps | | | |
|--|---|-----------------------|-----|-----------------|
| <p>PumpSpeed = 2850 MinOpSpeed = 1450 SpeedDecrement = 20</p> | <p>Initialise the pump speed and other variables. Note, we need to give consideration to the descriptive terms we use here as they will form a key part of the script used later.</p> <p>(2850 RPM Max. Pump Speed). (1450 RPM Min. Pump Speed). (20 RPM Speed Decrement reducing from Max to Min Speed).</p> <p>Note, the pump in this model is a Peerless 8196 1.5x3x6 model and the capacity curve for this pump model is defined in the pumps database.</p> | | | |
| <p>Output.Clear Results.Clear</p> | <p>Clear the output window and chart results.</p> | | | |
| <p>Network.Set(4, "BoosterOperatingSpeed", PumpSpeed)</p> | <p>Reset the pump speed and recalculate in case we have made previous calculation runs.</p> <p>The number 4 in this line refers to the User Number of the pump (unique node number – available from Results tab). Now, we need to set the pump operating speed (initial starting speed). We can retrieve the relevant property from the Helpfile. This data can be found at:</p> <p>Contents Script References Property Constants Boosters.</p> <p>We can see from the list that what we need is Property Id 173 "BoosterOperatingSpeed".</p> <table border="1" data-bbox="1167 1201 2112 1246"> <tr> <td>BoosterOperatingSpeed</td> <td>173</td> <td>Operating Speed</td> </tr> </table> <p>The initial booster operating speed can be set to 2850 RPM by adding the term "PumpSpeed" to the script ("PumpSpeed" is 2850 RPM as described at the start of the script).</p> | BoosterOperatingSpeed | 173 | Operating Speed |
| BoosterOperatingSpeed | 173 | Operating Speed | | |

| | | | | |
|--|--|------------------------|-----|-------------------|
| Network.Calculate; | Calculate the network. | | | |
| WHILE ((PumpSpeed > MinOpSpeed) AND (Network.Get(13, "CalculatedValveOpening") < 99) AND (Network.Get(14, "CalculatedValveOpening") < 99)) | <p>Make the main loop using a while statement. Note the "and" condition in the "while" statement.</p> <p>The numbers 13 and 14 refer to the User Number of the control valves. The scenario we are creating here is, as long as the pump is operating at a speed greater than the minimum speed and the position of both control valves is below 100% fully open, run the simulation loop. Note, much like the pump, we have retrieved the relevant valve opening property from the Helpfile. This data can be found at:</p> <p>Contents Script References Property Constants Controllers.</p> <table border="1" data-bbox="1167 683 2101 730"> <tr> <td>CalculatedValveOpening</td> <td>167</td> <td>Valve Opening (%)</td> </tr> </table> <p>We can see from the list that what we need is Property Id 167 "CalculatedValveOpening".</p> | CalculatedValveOpening | 167 | Valve Opening (%) |
| CalculatedValveOpening | 167 | Valve Opening (%) | | |
| Network.Calculate | Calculate the network. | | | |
| Results.Update | Update the chart results by including. | | | |
| PumpSpeed = PumpSpeed - SpeedDecrement | Now we need to reduce the pump speed from the initial starting speed which has been set in the script (2850 RPM). We can therefore use the descriptive terms we have used at the beginning of this script. | | | |
| Network.Set(4, "BoosterOperatingSpeed", PumpSpeed) END WHILE | We need to update the reducing pump speed in the network during the simulation. | | | |

| | |
|---|--|
| <pre> IF (PumpSpeed <= MinOpSpeed) THEN ShowMessage("The system is able to control at the minimum operating speed of the pump") </pre> | <p>Now we can consider the results for the simulation. Write the result summary to a message box.</p> <p>The condition we are considering here is; if the network runs through the simulation and both valves are in control for all pump speed conditions, a message box will appear advising:</p> <p><i>'The system is able to control at the minimum operating speed of the pump'.</i></p> <p>However, this may not actually be the case so we need to include the conditions where either of the valves reach 100% fully open before the pump reaches its minimum speed. Let's consider the next line of the script.</p> |
| <pre> ELSE IF (Network.Get(13, "CalculatedValveOpening") > 99) THEN ShowMessage("Control valve (13) is fully open at a pump speed of " + CStr(PumpSpeed+SpeedDecrement)) ELSE ShowMessage("Control valve (14) is fully open at a pump speed of " + CStr(PumpSpeed+SpeedDecrement)) END IF END IF </pre> | <p>Now we are considering the condition where either of the valves reach 100% fully open (more specifically, over 99% open) prior to the pump reaching its minimum speed, i.e. either of the valves lose control of the set flow rate.</p> |
| <pre> Network.Refresh Results.Refresh </pre> | <p>Refresh the network and results</p> |

2.1.3 Acrylic Valve Turndown – Results.

Now we can start the script simulation by clicking the Run button as shown below.

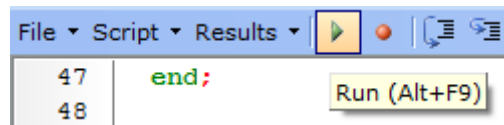


Figure 2.3: Script Run Icon.

When you select Run, the model will run through a series of iterations for each decremented pump speed. Control valve 14 reaches the fully open position (100% open) when the pump reaches a speed of 1970 RPM.

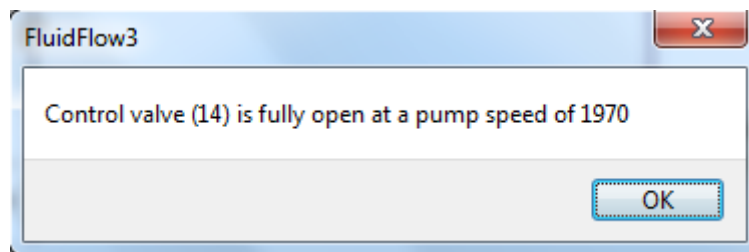


Figure 2.4: Script Message Box.

We can also view the pump and valve performance curves (based on a decrement of 20 RPM) by selecting; Results | View from the Script Window (see below).

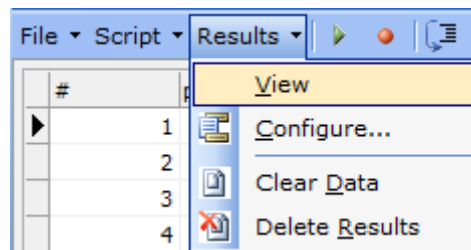


Figure 2.5: Results Data.

Firstly, you will see the graph plot of the pump performance which will be a linear plot since we are reducing the speed in constant steps of 20 RPM (see below).

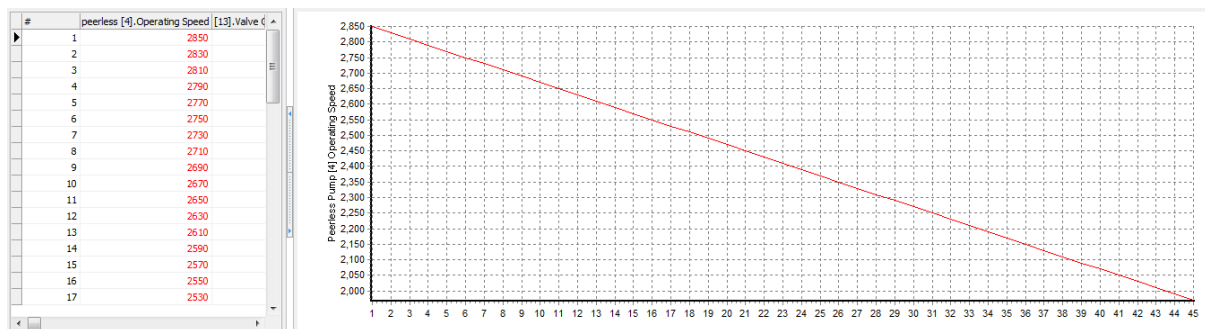


Figure 2.6: Pump Performance Curve.

To view the performance curve for the valves, select the drop-down menu as shown below and choose the valve of interest.

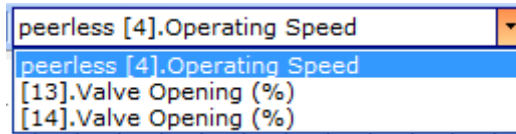


Figure 2.7: Drop-Down Menu for Graph Plots.

Below is the performance curve for valve node 13 in this case which as we can see, reached approximately . 75% open when the simulation stopped (stopped as valve 14 was fully open at this point).

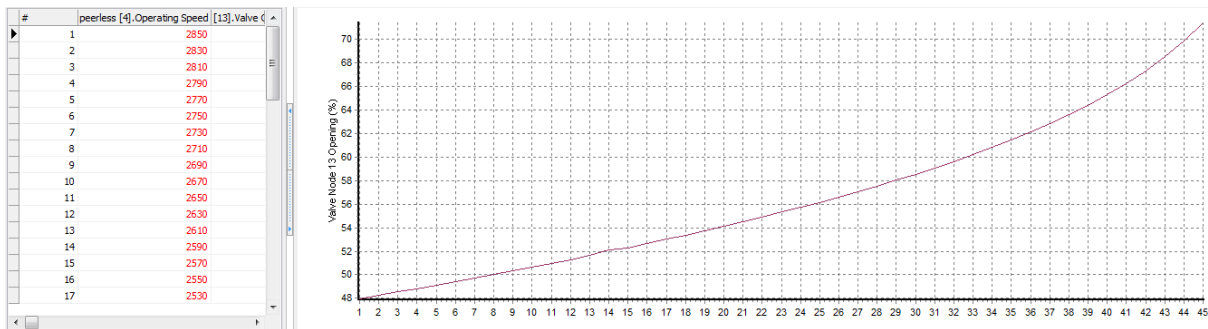


Figure 2.8: Valve 13 Performance Curve.

Finally we have the performance curve for valve node 14 which reached the fully open position at 1970 RPM (see below).

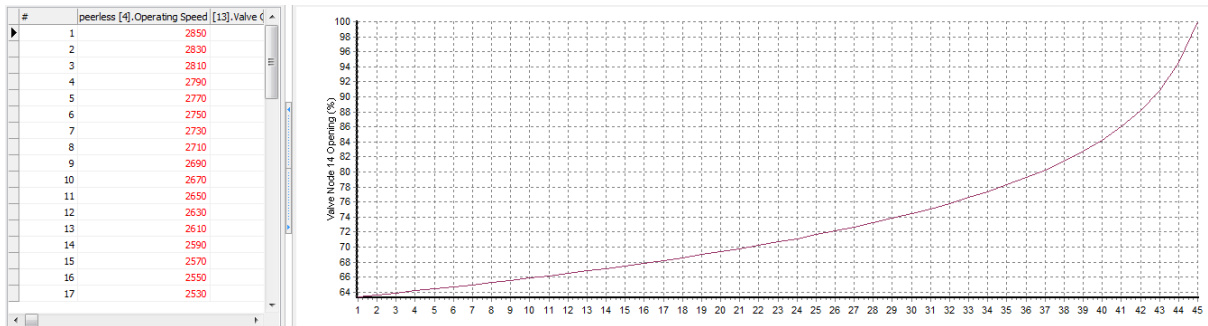


Figure 2.9: Valve 14 Performance Curve.

You can edit the setup of your charts by selecting: Results | Configure (Figure 2.10).

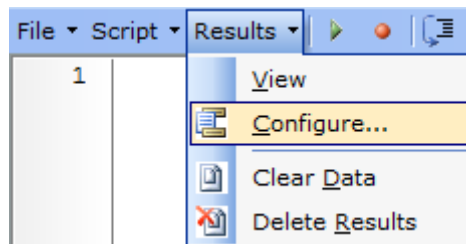


Figure 2.10: Configure Charts.

There are three charts for this particular example as shown in Figure 2.11.

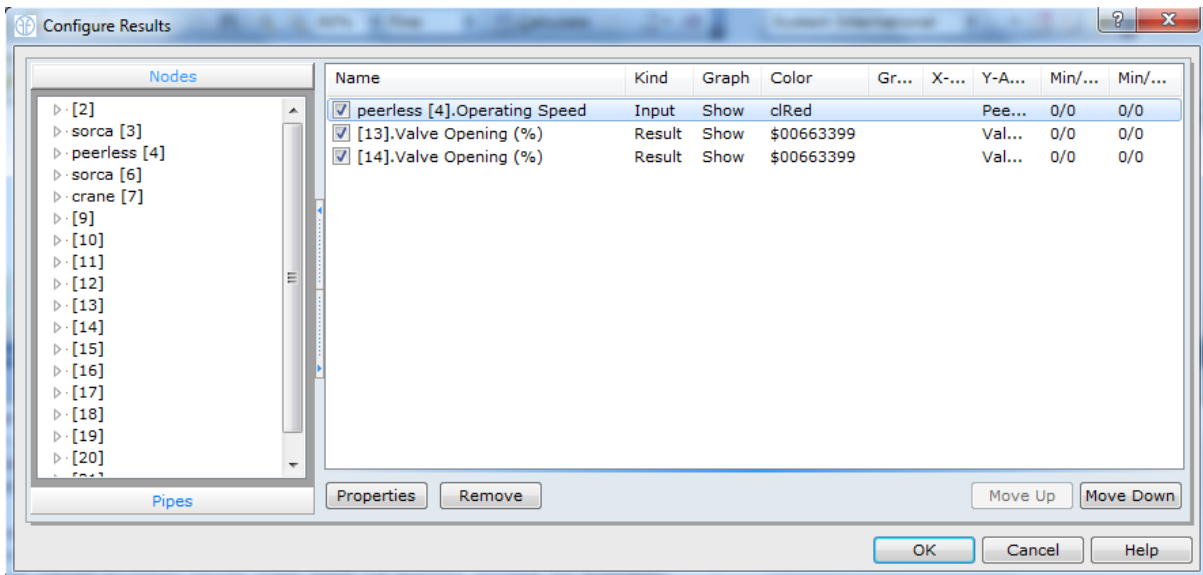


Figure 2.11: Configure Chart Display.

If you select any of the three charts, you can click the properties button which opens a new dialog as per Figure 2.12.

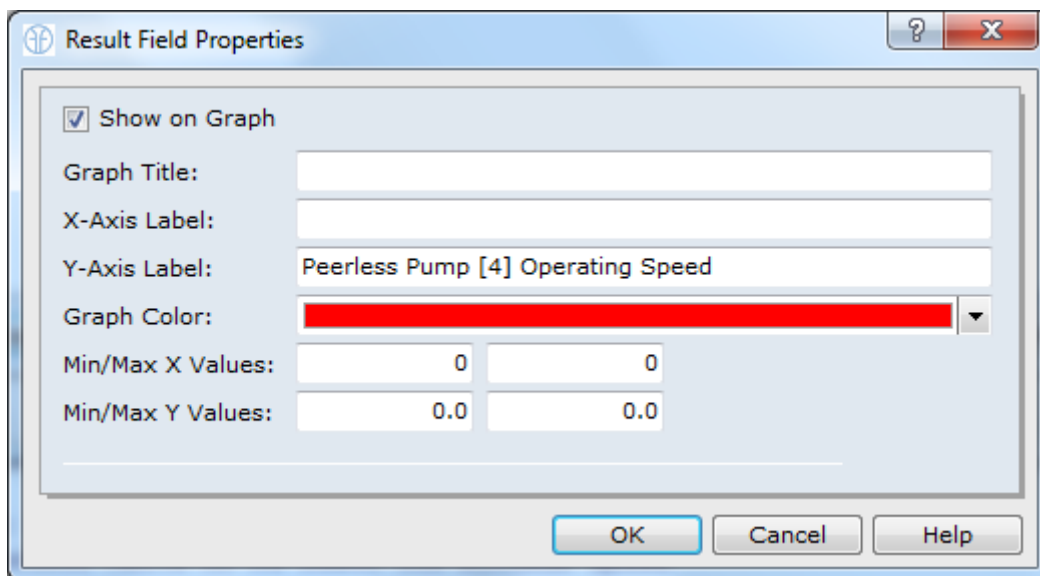


Figure 2.12: Result Field Properties Dialog.

You can edit the chart display and add labels to the chart from this dialog.

Note, this example file has been installed with your trial copy of FluidFlow. You can access the script for this example at;

C:\Program Files (x86)\Flite\FluidFlow3\QA Scripting\Pascal\Acrylic Supply Control Valve Turndown

C:\Program Files (x86)\Flite\FluidFlow3\QA Scripting\Basic\Acrylic Supply Control Valve Turndown (BASIC)

3 Butterfly Valve Pressure Loss vs Opening Position Script (Pascal)

This example shows how to plot a chart of pressure loss across a manual butterfly valve as a percentage of opening. We also wish to display the pressure loss results in kPa units. All results after a calculation are stored as SI values, this means that all pressure loss values are stored as Pa. If we just wish to display pressure loss in Pa we can use the Result Properties Dialog to automatically set up this column. Since we need to process the pressure loss results we must write a few lines of script to do this for us.

In this example there is no need to explicitly declare any variables but we will be using a reference to color variables and therefore we need to include Graphics in a uses clause

The model of the system is shown in Figure 3.1.

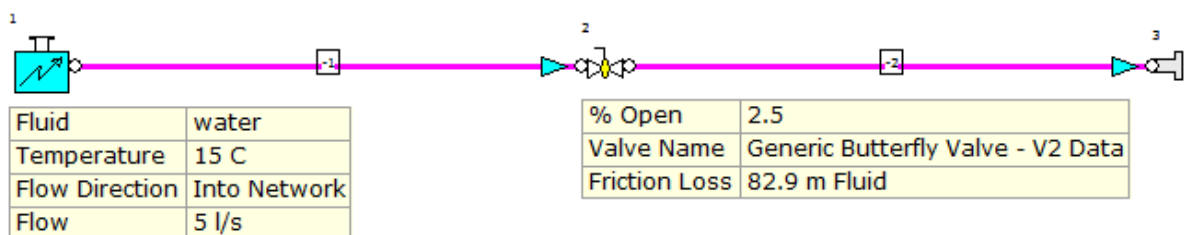


Figure 3.1: Butterfly Valve Model.

Before we start, we need to set the programming language for the script. To do this, select: Script | Pascal (or Basic) from the Script window (Figure 3.2). In this case, we are selecting Pascal.

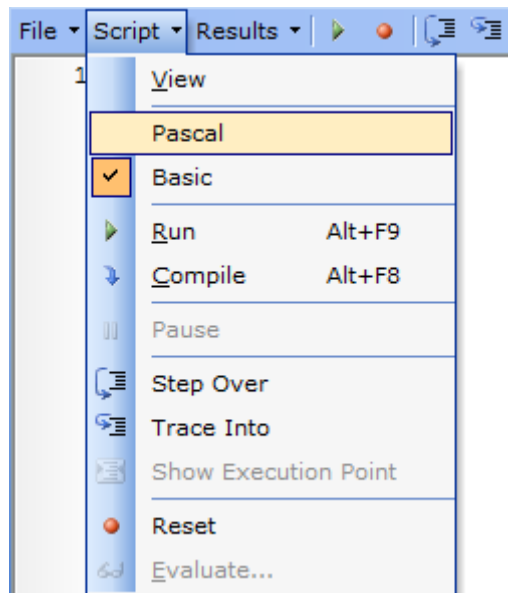


Figure 3.2: Pascal/Basic Script Setup.

3.1.1 Butterfly Valve Pressure Loss vs Opening Position – Pascal.

| Script | Description of Script Steps |
|--|---|
| <pre>uses Graphics;</pre> | <p>As we will be using a reference to color variables we need to include Graphics in a uses clause.</p> |
| <pre>begin ValveOpening := 100; MinimumValveOpening := 10; IncrementalChange := 2.5; PressureLossInkPa := 0;</pre> | <p>Initialise variables we will use.</p> |
| <pre>Output.Clear Results.Clear</pre> | <p>Clear and previous results.</p> |
| <pre>Network.Set(2, 'ManualValveOpening', ValveOpening); Network.Calculate;</pre> | <p>Set the initial manual valve opening to fully open.</p> |
| <pre>PressureLossColumn := Results.Add('Pressure Loss (kPa)', True, clWebRoyalBlue, 'Valve Total Pressure Loss', 'Valve Opening 100 -> 5 in 2.5 increments', 'Pressure Loss in kPa');</pre> | <p>Add a new field (PressureLoss) to be shown on the chart in blue. See the Results.Add method in the help file for information on the parameters together with additional sample code.</p> |
| <pre>while (ValveOpening >= 5) do begin Network.Calculate; NextRowIndex := Results.Update;</pre> | <p>Loop to change the valve opening.</p> |
| <pre>PressureLossInkPa := Units.Convert('Pa', 'kPa', Network.Get(2, 'PressureLoss')); Results.Set(PressureLossColumn, NextRowIndex, PressureLossInkPa);</pre> | <p>Set the pressure loss column values in kPa after conversion.</p> |
| <pre>ValveOpening := ValveOpening - IncrementalChange; Network.Set(2, 'ManualValveOpening', ValveOpening); end; end;</pre> | <p>Change the valve opening.</p> |

3.1.2 Butterfly Valve Pressure Loss vs Opening Position – BASIC.

| Script | Description of Script Steps |
|--|--|
| uses Graphics | As we will be using a reference to color variables we need to include Graphics in a uses clause. |
| ValveOpening = 100 MinimumValveOpening = 10 IncrementalChange = 2.5 PressureLossInkPa = 0 | Initialise variables we will use. |
| Output.Clear Results.Clear | Clear and previous results. |
| Network.Set(2, "ManualValveOpening", ValveOpening) Network.Calculate | Set the initial manual valve opening to fully open. |
| PressureLossColumn = Results.Add("Pressure Loss (kPa)", True, clWebRoyalBlue, "Valve Total Pressure Loss", "Valve Opening 100 -> 5 in 2.5 increments", "Pressure Loss in kPa") | Add a new field (PressureLoss) to be shown on the chart in blue. See the Results.Add method in the help file for information on the parameters together with additional sample code. |
| WHILE (ValveOpening >= 5) Network.Calculate NextRowIndex = Results.Update | Loop to change the valve opening. |
| PressureLossInkPa = Units.Convert("Pa", "kPa", Network.Get(2, "PressureLoss")) Results.Set(PressureLossColumn, NextRowIndex, PressureLossInkPa) | Set the pressure loss column values in kPa after conversion. |
| ValveOpening = ValveOpening - IncrementalChange Network.Set(2, "ManualValveOpening", ValveOpening) END WHILE | Change the valve opening. |

3.1.3 Orifice Pressure Drop – Results.

Now we can start the script simulation by clicking the Run button as shown below.

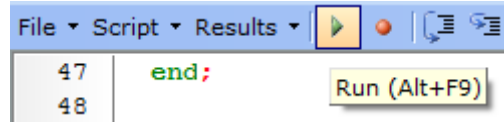


Figure 3.3: Script Run Icon.

When you select Run, the software will automatically run through a series of iterations. When complete, you can view the graph results by selecting: Results | View from the Script window.

You will firstly be presented with the graph result for the % opening position of the valve which is changing in steps of 2.5% as set in the script code. Since we have a constant % change, we have a linear type graph plot as shown in Figure 3.4.

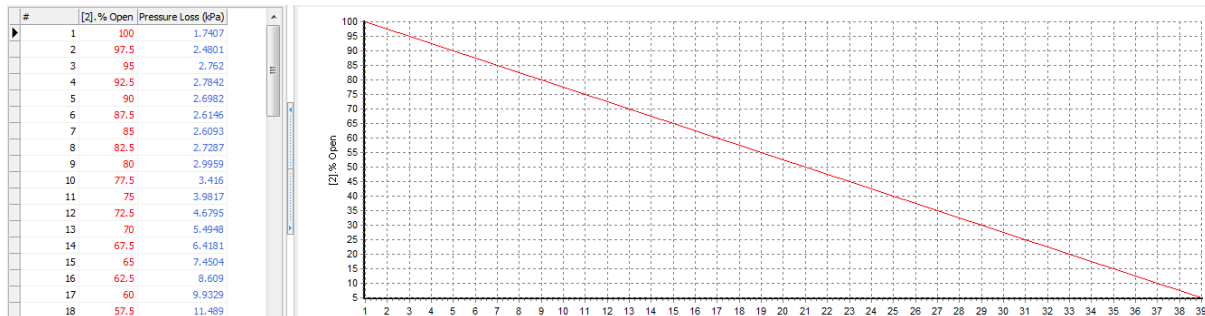


Figure 3.4: Valve Position Chart Result.

If we select the drop-down menu (Figure 3.5), we can select the graph for valve pressure loss in kPa.

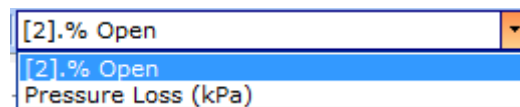


Figure 3.5: Chart Results.

Selecting Pressure Loss (kPa) reveals the chart result as shown in Figure 3.6.

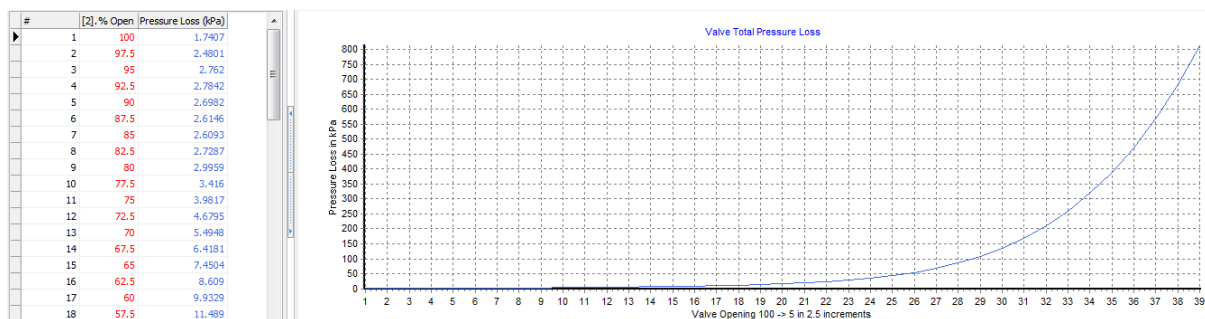


Figure 3.6: Valve Pressure Loss Chart Result (kPa).

Note, this example file has been installed with your trial copy of FluidFlow. You can access the script for this example at;

C:\Program Files (x86)\Flite\FluidFlow3\QA Scripting\Pascal\Pressure Loss v's Opening for a Manual Butterfly Valve

C:\Program Files (x86)\Flite\FluidFlow3\QA Scripting\Basic\Pressure Loss v's Opening for a Manual Butterfly Valve

4 Orifice Pressure Drop Script with Excel Report (Pascal)

In this example, we have a digestion and flash tank plant with two-phase liquor entering the system with a set pressure of 2975 kPa g. We wish to write a script to predict the pressure loss across the orifice plate based on upstream pressure, flow and quality. Note, the quality shall change from 0.01 to 0.012.

The model of the system is shown in Figure 4.1.

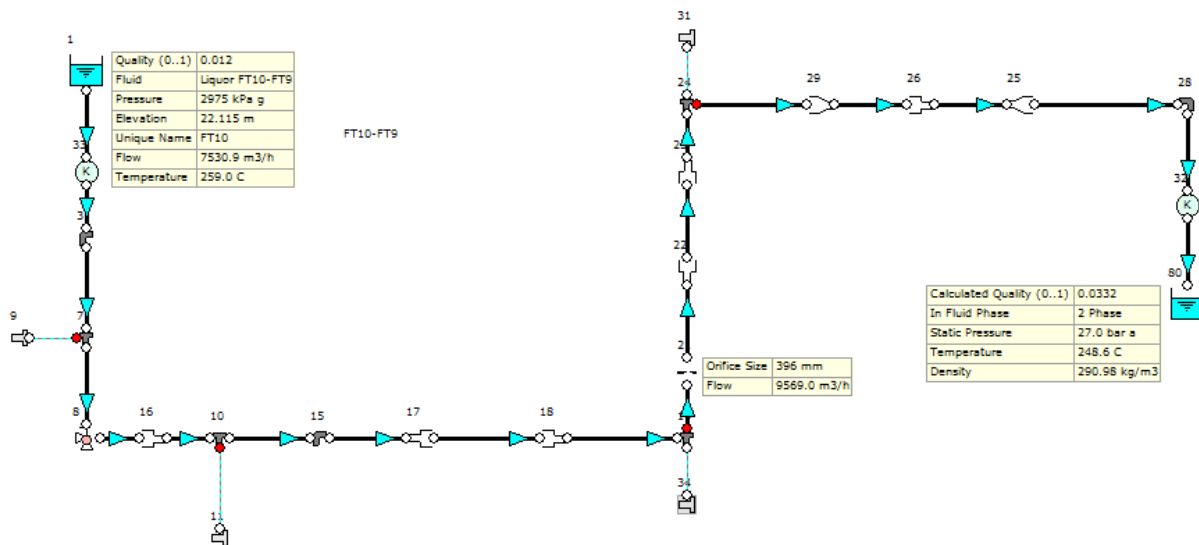


Figure 4.1: Digestion & Flash Tank System.

An Excel spreadsheet (titled 'Orifice Pressure Drop') is saved to the hard drive where we have stored the Input information for the model. The image below provides an overview of this data.

| | A | B | C | D | E |
|---|-------|---------------|--------|--------|--------|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | Case 1 | Case 2 | Case 3 |
| 4 | Input | FT10 Pressure | 2975 | 2975 | 2975 |
| 5 | | Flow | 4000 | 4000 | 4000 |
| 6 | | Quality | 0.01 | 0.011 | 0.012 |

Figure 4.2: Excel Data to be used in simulation.

4.1.1 Orifice Pressure Drop – Pascal.

| Script | Description of Script Steps |
|--|---|
| <pre> const ExcelFileFolder = 'C:\A1\' ExcelFileName = 'Orifice Pressure Drop'; </pre> | <p>Define the constants to be used in the script.</p> <p>ExcelFileFolder denotes the filepath of the Excel file which contains data you wish to import to the simulation. Note, you will need to change this filepath to reflect the position where you are storing your .xlsx file.</p> <p>ExcelFileName denotes the name of the Excel file which you wish to use as part of the simulation.</p> |
| <pre> InputDataColumnStart = 3; InputDataRowStart = 3; PressureRow = 4; QualityRow = 6; </pre> | <p>Denotes the Excel start column (column C) of the input data.</p> <p>Denotes the Excel start row (row 3) of the input data.</p> <p>Denotes the row that holds the pressure data.</p> <p>Denotes the row that holds the quality data.</p> |
| <pre> SupplyNodeID = 1; OrificeID = 21; </pre> | <p>SupplynodeID: is the number FluidFlow assigns to the node where we wish to change values.</p> <p>OrificeID is the node number of the node whose results we are interested in. The orifice plate User Number is 21.</p> |
| <pre> OrificeTotalPressureLossRow = 10; OrificeFlowRow = 11; OrificeInQuality = 12; OrificeOutQuality = 13; </pre> | <p>Declare the rows where we want the results to start in Excel.</p> <p>Add more property rows here and then add a row for each addition in the WriteResultDescriptions and ProcessCaseStudies procedures.</p> |
| <pre> var objXL; </pre> | <p>Declare variables that will be used in the calculation.</p> |

| | |
|--|--|
| <pre> procedure StartExcel; var EmptyParam: OleVariant; begin objXL := CreateOLEObject('Excel.Application'); objXL.Visible := True; XLWorkBooks := objXL.WorkBooks; XLWorkBooks.Open(ExcelFileFolder+ExcelFileName, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam); end; </pre> | <p>Start Excel Workbook.</p> |
| <pre> procedure StopExcel; begin objXL := UnAssigned; end; </pre> | <p>Stop Excel Workbook.</p> |
| <pre> function GetColumnCount: Integer; var col: Integer; </pre> | <p>This returns the number of columns in the result set.</p> |
| <pre> begin Result := 0; col := InputDataColumnStart; while (Length(objJXL.Cells[InputDataRowStart, col].Value) > 0) do begin Inc(col); Inc(Result); end; end; </pre> | |

| | |
|--|--|
| <pre> procedure WriteResultDescriptions; const ResultDescriptionColumn = 2; begin objXL.Cells[OrificeTotalPressureLossRow, ResultDescriptionColumn].Value := 'Orifice Loss (Pa)'; objXL.Cells[OrificeFlowRow, ResultDescriptionColumn].Value := 'Orifice Flow (kg/s)'; objXL.Cells[OrificeInQuality, ResultDescriptionColumn].Value := 'Orifice In Quality'; objXL.Cells[OrificeOutQuality, ResultDescriptionColumn].Value := 'Orifice Out Quality'; end; </pre> | <p>The column title is assigned here. For instance, the Excel row for orifice total pressure loss will be described as 'Orifice Loss (Pa)'.</p> |
| <pre> procedure ProcessCaseStudies; const InputTwoPhaseQuality = 69; var currentColumn: Integer; nDataColumns: Integer; thisPressure, thisQuality: Double; </pre> | <p>nDataColumns is the number of columns of data we have. This is the number of case studies.</p> |
| <pre> begin WriteResultDescriptions; nDataColumns := GetColumnCount; for currentColumn := InputDataColumnStart to InputDataColumnStart + nDataColumns - 1 do begin thisPressure := objXL.Cells[PressureRow, currentColumn].Value; thisQuality := objXL.Cells[QualityRow, currentColumn].Value; Network.Set(SupplyNodeID, 'BoundaryPressure', thisPressure); </pre> | <p>GetColumnCount refers to getting the number of case studies we need to process.</p> <p>thisPressure refers to reading the pressure variable data from Excel.</p> <p>thisQuality refers to reading the quality variable data from Excel.</p> |

| | |
|---|--|
| <pre> Network.Set(SupplyNodeID, InputTwoPhaseQuality, thisQuality); Network.Refresh; Network.Calculate(True); Results.Update; objXL.Cells[OrificeTotalPressureLossRow, currentColumn].Value := Network.Get(OrificeID, 'PressureLoss'); objXL.Cells[OrificeFlowRow, currentColumn].Value := Network.Get(OrificeID, 'InternalFlow'); objXL.Cells[OrificeInQuality, currentColumn].Value := Network.Get(OrificeID, 'TwoPhaseQuality'); objXL.Cells[OrificeOutQuality, currentColumn].Value := Network.Get(OrificeID, 'OutTwoPhaseQuality'); end; end; </pre> | <p>Change the Input Data.</p> <p>Recalculate.</p> <p>Write Out the newly calculated results to Excel - All Results are returned in SI Units. (If we wish to display in any other units, refer to the example in the help file at: Scripting Scripting Objects Units Object).</p> |
| <pre> begin StartExcel; try ProcessCaseStudies; finally StopExcel; end; end. </pre> | <p>Script code execution starts here.</p> |

4.1.2 Orifice Pressure Drop – Results.

Now we can start the script simulation by clicking the Run button as shown below.

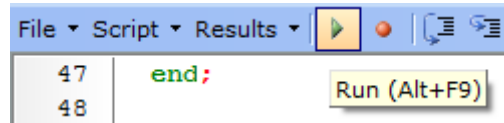


Figure 4.3: Script Run Icon.

When you select Run, the software will automatically start the relevant Excel file and use the data contained in this file to perform the simulation. The results will also be exported to the same Excel file based on the settings assigned in the script code.

The Excel report appears as shown below.

| | A | B | C | D | E |
|----|--------|---------------------|----------|----------|----------|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | Case 1 | Case 2 | Case 3 |
| 4 | Input | FT10 Pressure | 2975 | 2975 | 2975 |
| 5 | | Flow | 4000 | 4000 | 4000 |
| 6 | | Quality | 0.01 | 0.011 | 0.012 |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | Result | Orifice Loss (Pa) | 84549.87 | 85411.56 | 85533.42 |
| 11 | | Orifice Flow (kg/s) | 1269.522 | 1234.061 | 1213.526 |
| 12 | | Orifice In Quality | 0.015284 | 0.017199 | 0.018251 |
| 13 | | Orifice Out Quality | 0.020393 | 0.022365 | 0.023427 |

Figure 4.4: Excel Input and Results Data.

5 Mine Dewatering Script with Excel Report (Pascal)

In this example, we wish to consider the optimum pump speed and most cost effective means of mine water removal using any number of pumps up to a maximum of 7 pumps (in parallel).

This system involves the transportation of mine water from an elevation of 4 M to an elevation of 30 M over a distance of approximately 364 M. As part of this script, we will export a unique set of results to an Excel report.

The model of the system is shown in Figure 5.1.

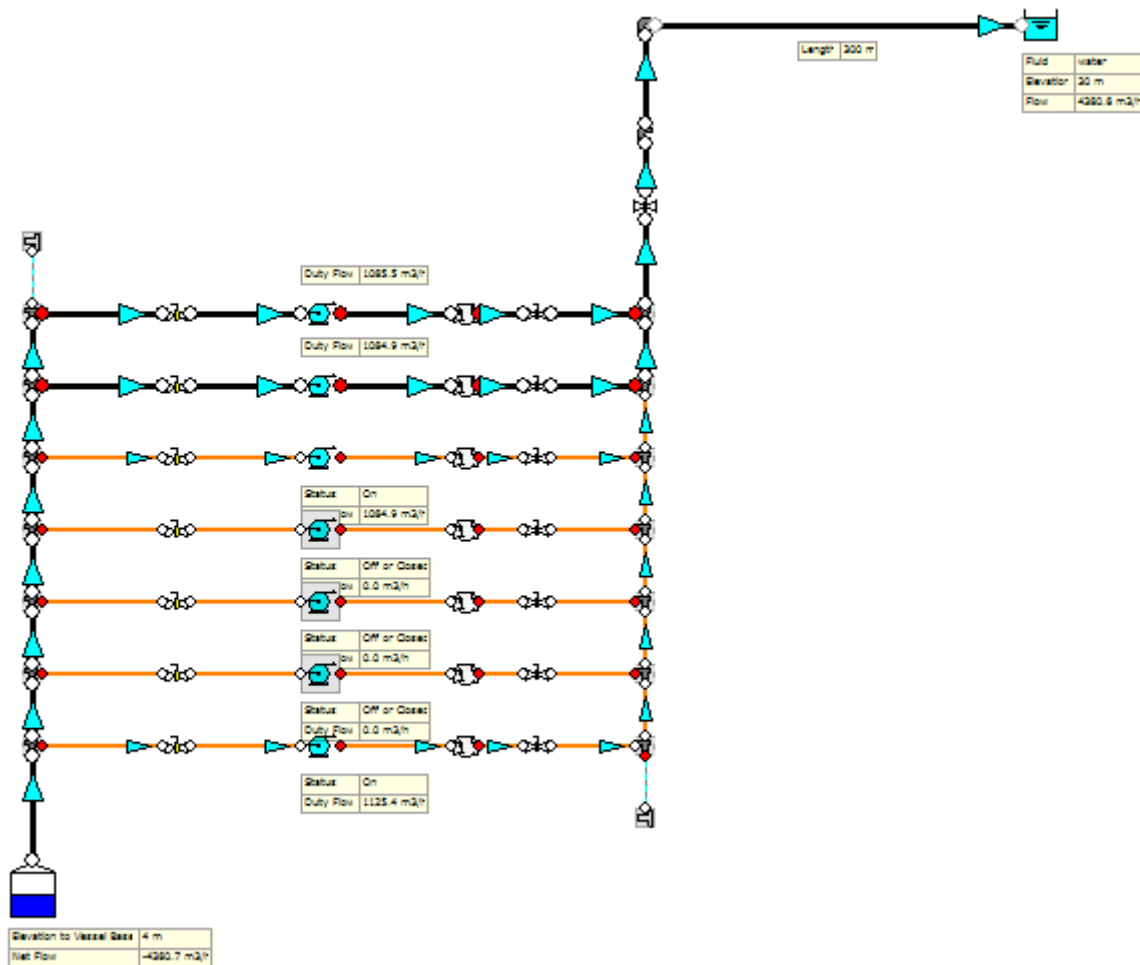
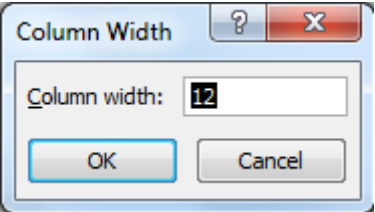


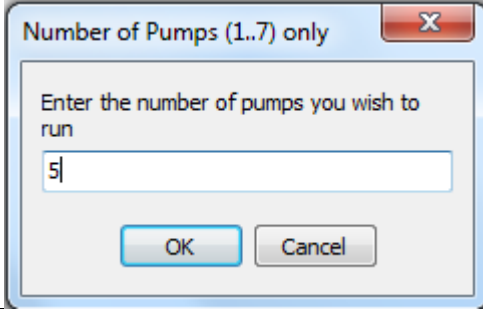
Figure 5.1: Mine Water Removal Plant.

5.1.1 Mine Dewatering – Pascal.

| Script | Description of Script Steps |
|--|--|
| <pre> const PumpMinSpeed = 900; PumpMaxSpeed = 1480; PumpSpeedIncrement = 20; On = 1; Off = 0; MaxPumps = 6; </pre> | <p>Define the constants to be used in the script. Note, we need to give consideration to the descriptive terms we use here as they will form a key part of the script used later.</p> <p>(1480 RPM Max. Pump Speed which is fixed by pump manufacturer). (900 RPM Min. Pump Speed. Below this figure the pump has insufficient head to overcome the discharge static height). (20 RPM Speed Increment increasing from Min to Max Speed).</p> <p>Note, all pumps in this network are Flygt CP3231 000*(450, 170kW) model and the capacity curve for this pump is defined in the database.</p> |
| <pre> xLColumnWidth = 12; xLColumnStart = 1; xLColPumpSpeed = xLColumnStart+1; xLColPumpFlow = xLColumnStart+2; xLColPumpEfficiency = xLColumnStart+3; xLColPumpPower = xLColumnStart+4; xLColPumpSpecificPower = xLColumnStart+5; xLColumnEnd = xLColPumpSpecificPower; xLRowStart = 2; xLLine = 4; xLColumn = 2; </pre> | <p>This is where we declare the setup of our Excel file which we will generate with the script. Note, the Excel report will be created automatically by the software.</p> <p>xLColumnWidth = 12; - This sets the width of the Excel column to 12 (see below).</p>  <p>xLColumnStart = 1; - This sets the results to start at column 1.</p> |

| | |
|---|---|
| <pre> var Flow: Double; Efficiency: Double; Power: Double; SpecificPower: Double; nPumps: Integer; PumpCurrentSpeed: Integer; </pre> | <p>Declares the variables for the simulation.</p> <p>nPumps denotes the number of pumps under consideration. PumpCurrentSpeed denotes the current speed of the pumps.</p> |
| <pre> PumpId; PumpBESpeed; </pre> | <p>PumpId denotes a variant array of element numbers. PumpBESpeed denotes an array to hold the speed of the lowest specific energy (i.e. speed of BEP).</p> |
| <pre> procedure SetPumpSpeed(PumpNo, Speed: Integer); begin Pump := Network.GetElement(PumpNo); Pump.Set('BoosterOperatingSpeed', Speed); end; </pre> | <p>Helper Methods.</p> |
| <pre> procedure TurnPumpOn(PumpNo: Integer); begin Pump := Network.GetElement(PumpNo); Pump.Set('Status', On); end; </pre> | <p>The script will start by turning the pumps on and then to end the script simulation, turn the pumps off.</p> |
| <pre> procedure TurnPumpOff(PumpNo: Integer); begin Pump := Network.GetElement(PumpNo); Pump.Set('Status', Off); end; </pre> | <p>The script will start by turning the pumps on and then to end the script simulation, turn the pumps off.</p> |

| | |
|---|--|
| <pre> procedure GetPumpValues(PumpNo: Integer; var Q, Eff, P, SpecificP: Double); var rho, MassFlow: Double; begin Pump := Network.GetElement(PumpNo); rho := Pump.Get('CalculatedDensity'); MassFlow := Pump.Get('InternalFlow'); if (rho <= 0) then Q := 0 else Q := MassFlow*3600/rho; Eff := Pump.Get('BoosterDutyEfficiency'); P := Pump.Get('BoosterDutyPower'); if (MassFlow > 0) then SpecificP := P/MassFlow else SpecificP := 0; end; </pre> | <p>Setup the calculation procedure.</p> |
| <pre> begin xLRow := xLRowStart; xLCol := xLColumnStart; </pre> | <p>This represents the start of the main execution point where we initialise local variables. The row and column start point are obtained here based on the setup declared earlier in the script.</p> |
| <pre> PumpId := [1,2,3,4,5,6,7]; PumpBESpeed := [0,0,0,0,0,0,0]; PumpCurrentSpeed := PumpMinSpeed; for i := 0 to MaxPumps - 1 do TurnPumpOff(PumpId[i]); </pre> | <p>Initialise all var arrays and pump status.</p> <p>PumpId denotes the unique pump user numbers (1 to 7). PumpBESpeed denotes an array to hold the speed of the lowest specific energy (i.e. speed of BEP).</p> |

| | |
|---|---|
| <pre> strPumps := ''; InputQuery('Number of Pumps (1..7) only', 'Enter the number of pumps you wish to run', strPumps); try nPumps := StrToInt(strPumps); except raise('Not a valid number'); end; </pre> | <p>Determine how many pumps we wish to run. Note, a dialog box (Input Query) as shown below will appear asking how many pumps we wish to include in the simulation.</p>  |
| <pre> for i := 0 to nPumps - 1 do begin TurnPumpOn(PumpId[i]); SetPumpSpeed(PumpId[i], PumpMinSpeed); end; </pre> | <p>Update the array structures.</p> |
| <pre> objXL := CreateOleObject('Excel.Application'); objXL.Visible := True; XLWorkbooks:=objXL.WorkBooks; XLWorkbook := XLWorkbooks.Add; XLSheets := XLWorkbook.sheets; XLSheet := XLSheets.item(1); </pre> | <p>Set up the Excel spreadsheet for results.</p> |
| <pre> objXL.Cells[xLRow, xLCol].Value := IntToStr(nPumps) + ' Pumps'; objXL.Cells[xLRow, xLColPumpSpeed].Value := 'Speed'; objXL.Cells[xLRow, xLColPumpFlow].Value := 'Flow (m3/h)'; objXL.Cells[xLRow, xLColPumpEfficiency].Value := 'Efficiency'; objXL.Cells[xLRow, xLColPumpPower].Value := 'Power (W)'; objXL.Cells[xLRow, xLColPumpSpecificPower] := 'Specific Power (W/kg)'; for i := xLColPumpFlow to xLColumnEnd do objXL.Columns(i).ColumnWidth := XLColumnWidth; </pre> | <p>Write out the headers on the Excel sheet.</p> |

```
try
```

```
while (PumpCurrentSpeed < PumpMaxSpeed) do  
begin  
  Network.Calculate;  
  Results.Update;  
  
  GetPumpValues(PumpId[0], Flow, Efficiency, Power,  
SpecificPower);  
  xLRow := xLRow + 1;  
  
  objXL.Cells[xLRow, xLColPumpSpeed] :=  
PumpCurrentSpeed;  
  objXL.Cells[xLRow, xLColPumpFlow] := Flow;  
  objXL.Cells[xLRow, xLColPumpEfficiency] := Efficiency;  
  objXL.Cells[xLRow, xLColPumpPower] := Power;  
  objXL.Cells[xLRow, xLColPumpSpecificPower] :=  
SpecificPower;  
  
  PumpCurrentSpeed := PumpCurrentSpeed +  
PumpSpeedIncrement;  
  
  for i := 0 to nPumps-1 do  
  begin  
    SetPumpSpeed(PumpId[i], PumpCurrentSpeed);  
  end;  
end;  
end;
```

Update results for current pump speed in the Excel report.

| | |
|---|---|
| <pre> XLCharts := XLSheet.ChartObjects; XLChartObject := XLCharts.add(440, 20, 400, 400); XLChart:=XLChartObject.Chart; XLChart.ChartType := xlLine; cell := XLSheet.cells(XLRowStart+1, xLColPumpSpecificPower); range := cell.Resize(xlRow-1, 1); XLChart.setSourceData(range, xlColumn); cell := XLSheet.cells(XLRowStart+1, xLColPumpSpeed); range := cell.Resize(xlRow-1, 1); XLChart.SeriesCollection(1).XValues := range; </pre> | <p>Set border of the chart.</p> <p>Produce the Excel chart.</p> |
| <pre> finally objXL := UnAssigned; PumpId := UnAssigned; PumpStatus := UnAssigned; PumpBESpeed := UnAssigned; PumpCurrentSpeed := UnAssigned; end; end; </pre> | <p>Tidy Up the Excel report.</p> |

5.1.2 Mine Dewatering – Results.

Select Run from the Script window as shown below.

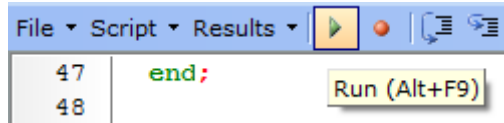


Figure 5.2: Script Run Icon.

When you select Run, you will be presented with a dialog asking how many pump you wish to consider in the simulation (1 to 7 pumps). For this purposes of this example, we have entered 5 pumps and selected OK.

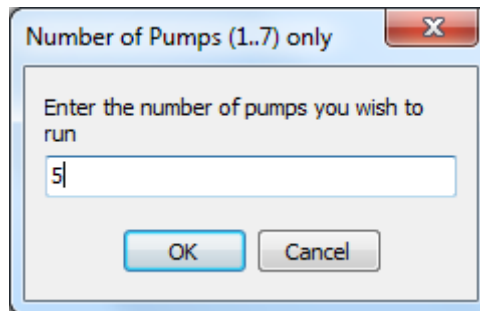


Figure 5.3: Number of Pumps to be considered.

The software will run through the simulation exporting results for Flow, Efficiency, Power and Specific Power to Excel. A graph curve relationship of specific power vs speed will also be generated from where you can clearly identify the most optimum specific power/speed. In this case with 5 pumps in operation, the optimum speed would be 1040 RPM as this produces the lowest specific power requirement (425 W/kg).

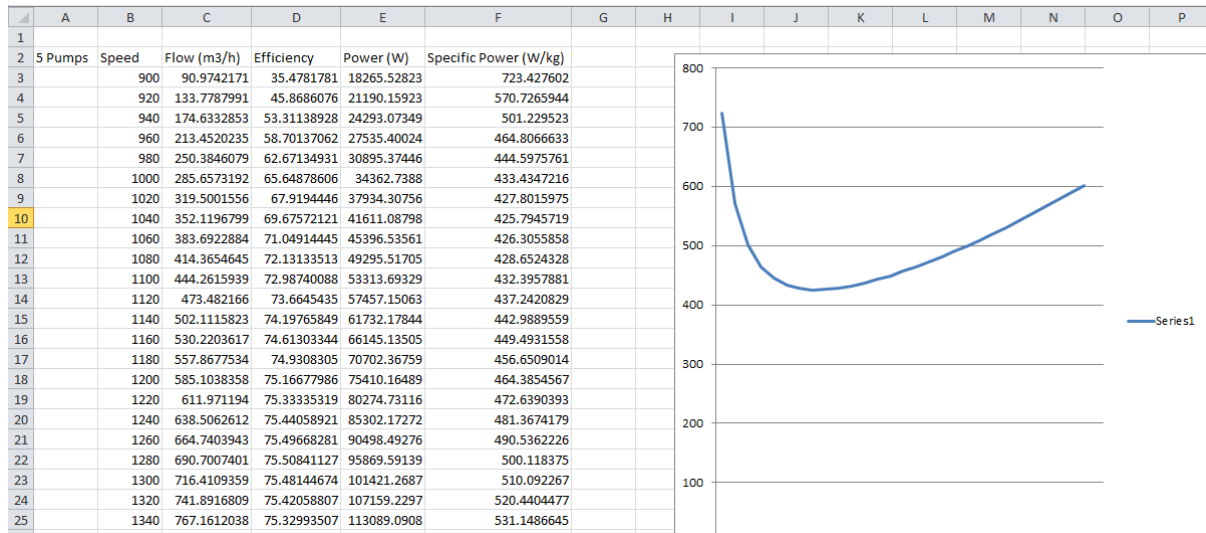


Figure 5.4: Excel Results - Specific Power vs Speed (5 Pump).

Contact us at:

support@fluidflowinfo.com

Flite Software NI Ltd
Block E
Balliniska Business Park
Springtown Road
Derry
Northern Ireland
BT48 0LY

T: +44 2871 279 227

F: +44 2871 279 806

www.fluidflowinfo.com