Exercise 1 – Lubrication System

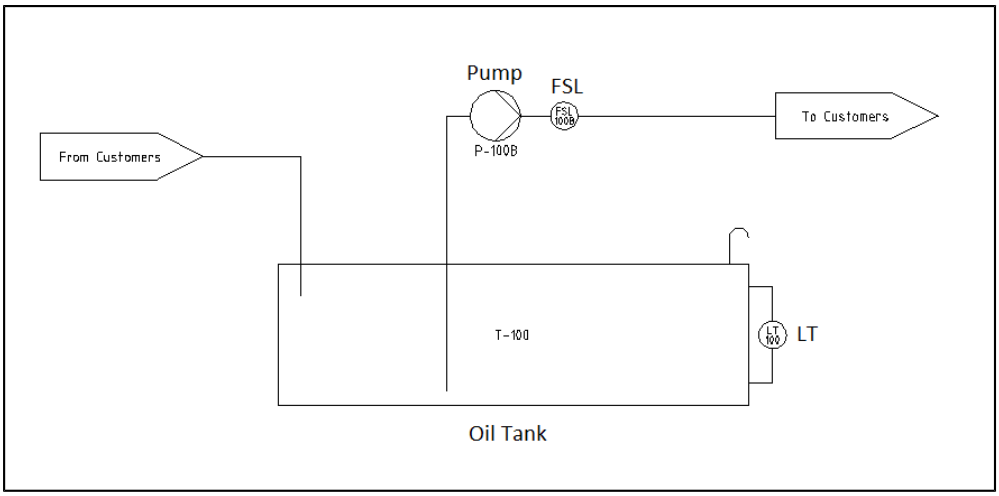
Preface

With this simple system, you'll be able to take your first steps in WonderLogix Studio. It will train you in modular design, class creation and the use of basic logic. Let's get started!

Task

The lubrication system circulates oil to lubricate the bearings of a certain critical piece of equipment (ex. a turbine). Oil is pumped from an oil tank to the consumers and returns to the tank by gravity. You need to define the control logic for this system.

1. **Components:**
    - Oil tank with a level transmitter
    - Oil pump
    - Flow-switch-low on the pump’s discharge
2. **Functionality:**
    - When the oil system is enabled, it will turn on the pump.
    - The pump will not be turned on if the oil tank’s level is below 20%.
    - If the pump is requested to operate for at least 3 seconds **and** the flow switch indicates low flow, the oil system will report a Failure.



Guidelines for Solution

1. Think of the **structure** (you'll be able to change it later):
    a. Which modules can you create here? – A module can be thought of as a black box with a certain functionality. Modules can be duplicated or reused in other places.
    b. Which of the components act together to create a common functionality? – They can be good candidates to be parts of the same module.
    c. Try to represent the reality as well as you can: The level transmitter is not just "a level transmitter." It's a level transmitter inside a _____.
2. Now, create the structure. Open a new project in WonderLogix Studio. Add subcomponents to create your structure. Note: Physical components like the pump or flow-switch-low should be marked as physical when creating them. Their background is purple and a wrench icon appears on their top-right corner. Other components are white. They represent logical levels, which contain physical and other logical components.
3. **Interface**: Design what every component can be asked to do (input) or report (output).
    a. To do that, take a look at the task's description. Pay attention to the verbs: "When the Oil System is **enabled**, it will **turn on** the pump." So, there should be an **Enable** input to the oil system (the tree's root) and another input to turn on the pump. We recommend using **Operate**.
    b. When signals pass from a certain component to a higher one in the hierarchy, they should usually undergo some abstraction. For example, the level transmitter's reading (an analog signal) is not interesting for the pump. What is interesting is the answer to a very simple question: Is the oil level low or not? Get it? True/False instead of an analog signal.
4. **Logic**: Connect the attributes (inputs, outputs, properties and parameters) using logical sentences. Construct them by selecting the adequate option among the suggested sections. Refer to the Logic Editor in the User's Manual.
5. Finish when all the attributes have logic and you are satisfied with the design.

Change Order #1

1. In order to increase the system's reliability, another pump and flow-switch-low are added.

2. When the flow-switch-low of the first pump is active for 3 consecutive seconds after being turned on, the second pump is started. The oil system will only report failure if the same thing happens for this pump as well.

Guidelines for Solution

1. Now that we need to repeat the pump + flow switch once again, it's clear that they should be defined as a module. However, we don't just want to create one and copy it. We want to keep them linked so that every change we make to one of them will reflect to the other. Look for "Classes" in the User's Manual.
2. If you haven't defined a module, don't worry – you don't need to start all over again. Simply create a component now (Pumping Unit can be a good name) and use the Lower command to place the pump and flow-switch-low beneath it in the hierarchy. You can also drag-and-drop them!
3. If you need to rename a component, do it through the component's Details.

Change Order #2

1. In order to make the design more flexible and understandable, the "magic number" of 3 seconds should be replaced by the parameter Delay To Failure.
2. The same goes for the oil level – the parameter Minimal Level should replace the number "20."

Guidelines for Solution

1. Enter the component in which the logic containing the 3-second delay is defined. Create the parameter Delay To Failure (note: be sure to capitalize the first letter of every word). Edit the logic to replace the constant number with the new parameter.
2. Now you can enjoy the fact that you are using two instances of the same class – the change you made to one Pumping Unit reflects to the other one automatically.
3. Repeat this process with the oil level parameter.

Change Order #3

1. Currently, when the oil level is low, the pump is not operated. However, there is no indication of a failure to the outer world (a bigger system which this oil system is a part of). A logic change is thus required so that the oil system will also report failure in the case of low level in the tank.

Guidelines for Solution

1. All you have to do is edit the relevant logic.

Change Order #4

1. The system after the last change worked fine for a while. When the oil tank level was low one day, someone went to fill it up. But when the oil level was no longer low, the pump suddenly started working, causing a hazard to the technician. For safety reasons, they decided to eliminate the operation command to the pumps when there is a failure, latch the failure when it happens and require an external reset to continue the operation.

Guidelines for Solution

1. Two changes are required here. First, we want the pump to depend on the Failure output (i.e. not to operate the pump when there is failure). But it's not possible to refer to an output of a component in the logic of one of its internal attributes. So, we need to define a property. Properties are like internal variables. Read about them in the User's Manual.
    a. Now, define a property that will be set when there are conditions to failure (backup pump failed or low oil level).
    b. Change the main pump's logic to refer to this property instead of directly to the oil level.
    c. Change the output Failure to refer just to this property.
2. The second change is in the way we create the logic of this property. You remember that we need to latch the failure until an external reset command is received.
    a. Create another input to the system – Reset.
    b. Define the logic of this property to set when either the backup pump fails or the oil tank is low. However, this time, use the "from the moment" connection instead of the usual "only while."
    c. After confirming, a "clear" sentence will appear automatically asking you when to clear ("unlatch") the failure. Use the Reset input.
    d. Another small sentence will pop up: "In case of conflict, Oil System sets/clears Failure." This is a conflict resolver: What should be done when both conditions are true? For instance, there is low oil level, but someone clicked Reset. Should the failure be cleared or not? This is for you to decide in any particular case. Here, the correct answer should usually be to keep the Failure set and allow reset only when there are no conditions to failure.

Congratulations!

You have finished the first exercise.

You can now keep playing with the model. Add descriptions if you like and explore other ways to make this example more complicated and interesting.

You can also start your own project and create a system that reflects your own world.

Good luck! We are here to help at support@wonderlogix.com.

See you on the next exercise!