Creating Advanced Triggers and Validation in OFS

When starting out with OFS Flow and Alerts, most users quickly become familiar with the ways in which triggers can be used to create alerts and forms in response to a specific event. As time progresses, OFS users find the power of Flow is limited only by the scenarios it is applied to - with the most complex workflows and checks yielding huge benefits in process repeatability and reliability.

As complexity increases, it is common that more unusual validation and triggering requirements emerge. Within this document, we review some of the more complex formulas you might encounter in OFS and try to untangle them with you.

As a precursor to this resource, we recommend you are familiar with our relevant training resources:
Fusion Flow - Advanced Triggers (2m 48s)
Fusion Flow - Validating Form Inputs (4m 13s)

Matching Specific Text Formats

When building a check which is relevant to a specific product group, you may find a need to build a trigger condition based on matching text. For example, in the below product list, a Plastic Bottle check will only be required for some products:

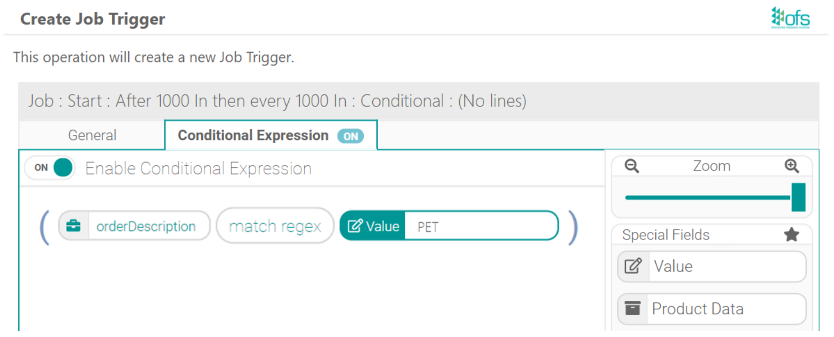
Order Description	**Bottle Type**	**Trigger Check?**
CHERRY-GLSBTL300	Glass	No
CHERRY-PETBTL300	PET	**Yes**
LIME-GLSBTL300	Glass	No
LIME-PETBTL300	PET	**Yes**

Building a Trigger Condition

When building a trigger, any Job/Shift/State/Relay trigger can have a condition added. Via the conditional triggers tab, it is possible to examine the metadata for a job, shift or product, and allow the trigger only to fire when a stated condition is met.

The example, right, shows how “match regex” can be used to match specific text within a parameter. Here, the Order Description is searched, to see if it contains the letters “PET”. When it does, the expression evaluates to true, and the trigger fires.

The rule shown right can be used to create the desired behavior from the table above.


Create Job Trigger

This operation will create a new Job Trigger.

Job : Start : After 1000 In then every 1000 In : Conditional : (No lines)

General
Conditional Expression ON
ON Enable Conditional Expression
(orderDescription match regex Value PET)
Zoom
Special Fields
Value
Product Data

In this example, “match regex” is used like the Find tool in a word processor, but it is a much more powerful tool. Continue overleaf to learn more about Regular Expressions (regex).

Matching an Expression based on Pattern

Within a factory environment, there are a number of examples of data which follows a specific pattern.

2019-04-18 10:01AM	pH = 5.39	BAT-394-3JD	25M
Time and Date Formats	Chemical Test Results	Batch Codes	Date Codes

Within OFS Flow, we provide basic validation patterns for Times, Dates, Integers and Decimal numbers, but Regular Expressions provide a way for you to define **your own custom format** for validation.

Regular Expressions

Regular Expressions **(or regex)** are used in programming to **match** the **pattern** of an input. They are a common tool, used in many programming languages. Regular Expressions represent a way of describing the components of text input. In the previous example, the pattern used was simply a part of the input:

Input	**Regular Expression**	**Match?**
CHERRY-GLSBTL300	PET	No
CHERRY-PETBTL300	PET	**Yes**

Regular Expressions can also evaluate patterns in inputs, where you might not know the exact value:

Input	**Regular Expression**	**Match?**
CHERRY-1GLSBTL330	\d{3}	**Yes**
CHERRY-2GLSBTL650	\d{3}	**Yes**

Why does the above pattern create a match?

\d represents any digit and {3} means that we are looking for 3 consecutive digits. For a comprehensive reference to the codes available to represent patterns in regex, refer to this guide: **Regex Cheat Sheet**

Putting this into Practise - Validating a Batch Code

Consider an example where you wish to ensure that operators have correctly entered the batch code printed on your product. An example code is given below:

BAT-25M-3JD

Batch Code Composition
BAT - 3 character product reference, changes with each SKU. Can be any characters, no numbers.
25M - 2 numbers to represent a week number, single letter to represent production year.
3JD - Alphanumeric reference to the workcentre and shift.

Within this batch code, we have a number of opportunities to practise our regex skills. The pattern (rather than the specific letters and numbers) from each section can be represented as a separate component, which we will group together to create an overall pattern match:

Section	BAT	25M	3JD
Regex	[A-Z]{3}	\d{2}[A-Z]	[A-Z0-9]{3}
Meaning	Matches 3 consecutive uppercase letters.	Matches 2 consecutive digits followed by a single uppercase letter.	Matches 3 consecutive alphanumeric characters.

Combining this into a single rule, inclusive of the dashes in the batch code creates:

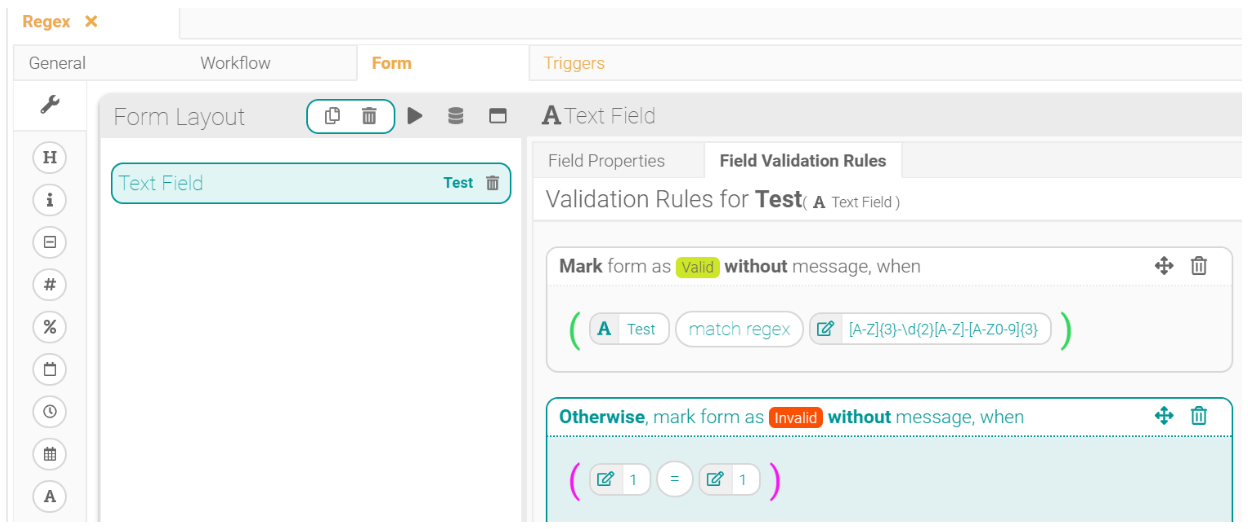
^[A-Z]{3}-\d{2}[A-Z]-[A-Z0-9]{3}$

This might look like a spillage at the alphabetti spaghetti factory, but this is valid regex, which will accurately validate the pattern we are looking for.

When combining the patterns, notice that we've added a character at the beginning (^) and the end ($). These characters indicate that your patterns should be at the beginning and end of the text to be evaluated, ensuring that no additional text is entered before or after the matched pattern.

Adding Regular Expressions to OFS

When you add regular expressions in OFS, there is no way of negating them, and querying whether the pattern **does not match**. In this circumstance, you can add a fall-back rule:



Validation Rules for Test(A Text Field)

Mark form as Valid without message, when

(A Test match regex [A-Z]{3}-\d{2}[A-Z]-[A-Z0-9]{3})

Otherwise, mark form as Invalid without message, when

(1 = 1)

Why 1=1?
The second validation rule for this field is only reached when the first is **not true**, meaning that the field will always be invalid, and red, when the regular expression is not satisfied.