Ajouter un test LTF pour une nouvelle règle de production

Dans le document suivant je prends exemple sur la règle de production « Marker Validation » avec test LTF « StepMarkerValidation »

Structure des tests existants

0. Initialisation

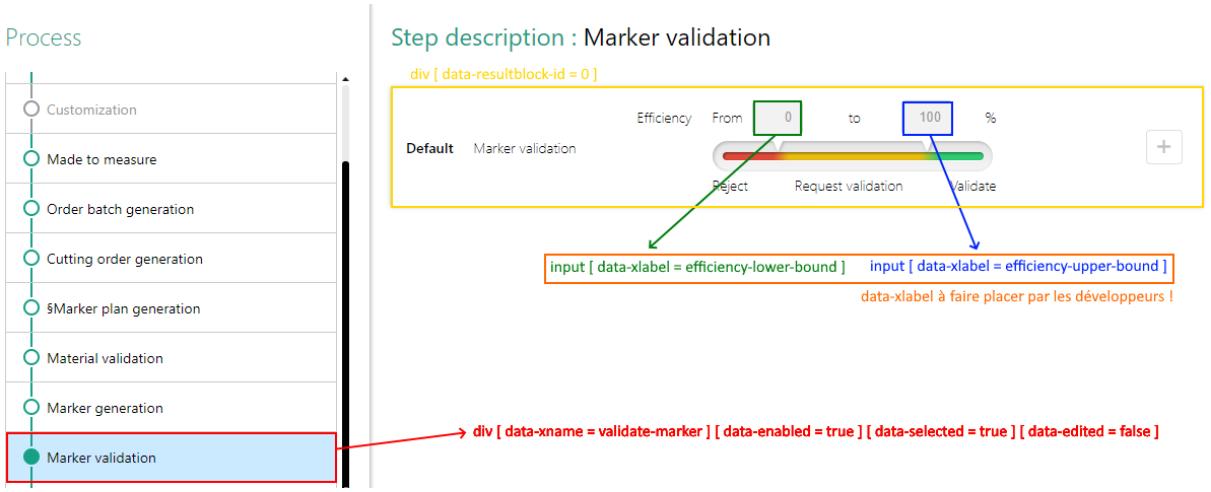
Reset les règles de production, ouvrir cutting-room-production-process et se login avec le user dédié : ltf-production-rules@framboise.com (!Lectra33 !)

(TODO : avoir un user dédié On Demand en plus de ltf-production-rules qui est Mass Prod)

```
chapter -> ***** Step marker validation TEST ***** -> Web production process ->

callScript -> scripts.ResetProductionRules -> [ ltf-production-rules@framboise.com , !Lectra33! ] ->

channel -> start -> prodprocess -> ltfs://cutting-room-production-process.dev.mylectra.com/

callScript -> subscript.Login -> [ ltf-production-rules@framboise.com , !Lectra33! ] ->

callScript -> subscript.activity_list.WaitForActivityListLoad -> [  ] ->
```

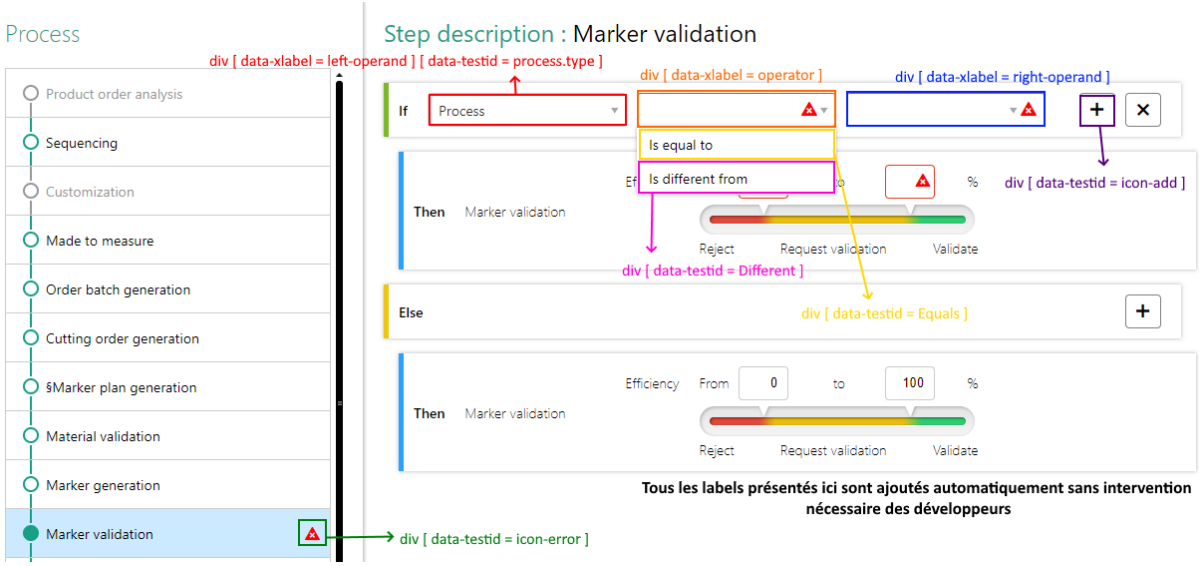
1. Trouver la règle activée dans la liste, vérifier les valeurs par défaut

```
callScript -> subscript.activity_list.CheckActivityEnabledState -> [ validate-marker , true ] ->

comment1 -> Initial state : two checkboxes unclicked

callScript -> subscript.activity_list.SelectActivity -> [ validate-marker ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 0 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 100 , efficiency-upper-bound , 0 ] ->
```



2. Vérifier la liste des opérandes possibles (et opérateurs associés) de l'arbre conditionnel

```
callScript -> subscript.ribbon.ClickEdit -> [  ] ->

callScript -> subscript.conditions.AddFirstCondition -> [  ] ->

callScript -> subscript.activity_list.CheckActivityInError -> [ validate-marker ] ->

callScript -> subscript.conditions.operand_type.UnfoldOperators -> [ process.type ] ->

callScript -> subscript.conditions.operand_type.CheckOperator -> [ Equals ] ->

callScript -> subscript.conditions.operand_type.CheckOperator -> [ Different ] ->
```



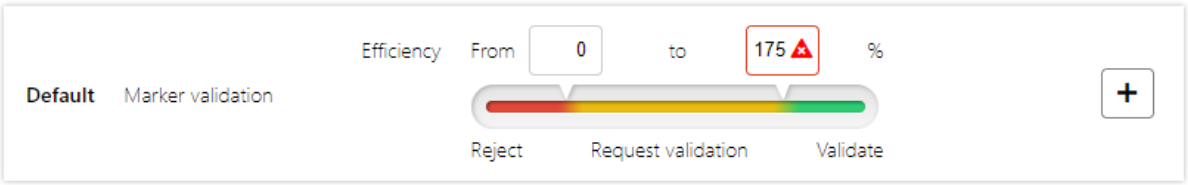
Comme le nombre de conditions et d'opérateurs associés est important, pour garder le test propre je crée un subscript pour cette étape (CheckLeftOperands).

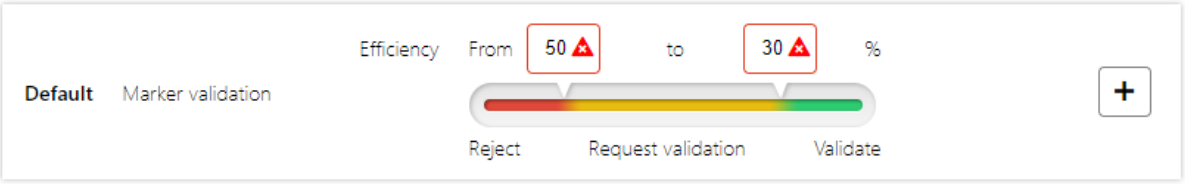
3. Vérifier le statut en cours d'édition et le bouton annuler

```
paragraph -> Test edit mode and cancel

callScript -> subscript.ribbon.ClickEdit -> [  ] ->

callScript -> subscript.result_block.FillInput -> [ 10 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.FillInput -> [ 80 , efficiency-upper-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 10 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 80 , efficiency-upper-bound , 0 ] ->

callScript -> subscript.activity_list.CheckActivityEditedState -> [ validate-marker , true ] ->

callScript -> subscript.ribbon.ClickCancel -> [  ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 0 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 100 , efficiency-upper-bound , 0 ] ->

callScript -> subscript.activity_list.CheckActivityEditedState -> [ validate-marker , false ] ->
```

4. **Tester le bloc de résultat complètement (cas d'erreur, option possible, affichage conditionnels…)**

Dans le cas de marker validation, trois tests sont réalisés : l'efficacité ne peut dépasser 100%, la borne inférieure ne peut pas dépasser la borne supérieure d'efficacité, et les inputs sont numériques seulement.





5. **Créer une structure de règle conditionnelle avec des opérandes. Privilégier les opérandes avec picker ou input numérique.**

```
paragraph -> Add conditional block with two conditions

callScript -> subscript.ribbon.ClickEdit -> [  ] ->

callScript -> subscript.result_block.FillInput -> [ 23 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.FillInput -> [ 88 , efficiency-upper-bound , 0 ] ->

callScript -> subscript.conditions.AddFirstCondition -> [  ] ->

callScript -> subscript.activity_list.CheckActivityInError -> [ validate-marker ] ->

callScript -> subscript.conditions.FillConditionList -> [ 0-0 , process.type , Equals , MTO ] ->

callScript -> subscript.conditions.AddCondition -> [ 0-0 ] ->

callScript -> subscript.conditions.FillCondition -> [ 0-1 , material.nature , Contains , Linen ] ->

comment1 -> Remove the product category and check the activity is in error
```

6. Sauver la règle et vérifier sa structure (conditions, blocs de résultats)

On vérifie que toutes les infos sont conservées après sauvegarde, blocs conditionnels comme blocs de résultats.

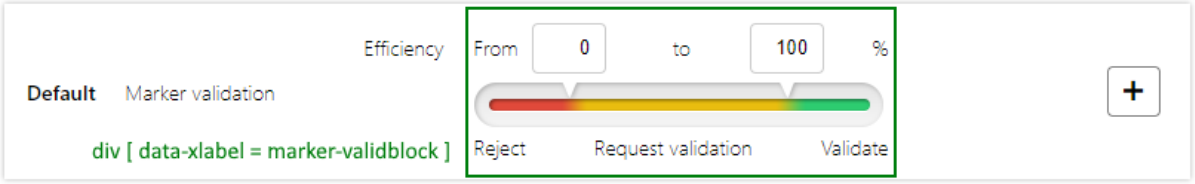
```
paragraph -> Check state of activity after saving

callScript -> subscript.activity_list.CheckActivityEditedState -> [ validate-marker , false ] ->

callScript -> subscript.conditions.CheckConditionList -> [ 0-0 , process.type , Equals , MTC/MTM ] ->

callScript -> subscript.conditions.CheckCondition -> [ 0-1 , material.nature , Contains , Linen ] ->

callScript -> subscript.conditions.CheckCondition -> [ 1-0 , marker.length , Above , 85 ] ->

callScript -> subscript.conditions.CheckConditionList -> [ 1-1 , material.motif , Different , Plaid ] ->

callScript -> subscript.conditions.CheckCondition -> [ 1-2 , batch.supplier , Equals , Textile-producer-INC ] ->

callScript -> subscript.conditions.CheckConditionList -> [ 1-3 , cuttingorder.isrecut , Equals , True ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 22 , efficiency-lower-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 44 , efficiency-lower-bound , 1 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 23 , efficiency-lower-bound , 2 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 75 , efficiency-upper-bound , 0 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 95 , efficiency-upper-bound , 1 ] ->

callScript -> subscript.result_block.CheckInputValue -> [ 88 , efficiency-upper-bound , 2 ] ->
```

7. Tester l'aide en ligne

Un subscript existe déjà : CheckOnlineHelp, il suffit d'y affecter l'id de la règle testée et le data-xlabel d'une div du bloc de résultat sur laquelle l'aide en ligne est montrée.

Dans le cas de la règle Marker Validation, les deux items du bloc de résultat sont des inputs et pas des div, il a fallu ajouter un data-xlabel exprès pour cette étape sur un élément non utilisé par ailleurs. A demander aux développeurs pour cette étape !

```
paragraph -> Test online help

callScript -> subscript.ribbon.CheckOnlineHelp -> [ validate-marker , marker-validblock ] ->
```



8. Ajouter un paragraphe relatif à la règle dans le test SupportMode.

Le mode support est relatif à l'utilisateur, il existe donc un test indépendant pour le mode support. Dans ce test, on passe sur toutes les règles, on édite le bloc de résultats, on vérifie que le bouton Save reste désactivé, et on Cancel.

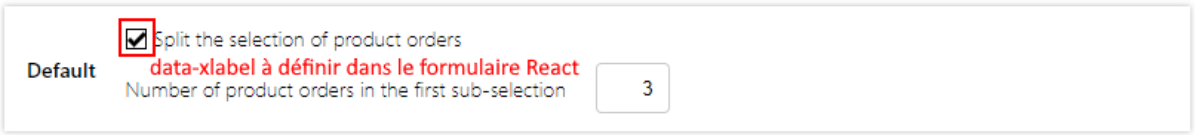
```
paragraph -> Step Marker Validation

callScript -> subscript.activity_list.SelectActivity -> [ validate-marker ] ->

callScript -> subscript.ribbon.ClickEdit -> [  ] ->

callScript -> subscript.result_block.FillInput -> [ 95 , efficiency-upper-bound , 1 ] ->

callScript -> subscript.ribbon.CheckSaveDisabled -> [  ] ->

callScript -> subscript.ribbon.ClickCancel -> [  ] ->
```

Tester un bloc de résultats.

Pour permettre le test automatique du bloc de résultat, les développeurs doivent placer quelques balises sur les objets du formulaire qui doivent permettre de les manipuler et de consulter leur contenu. Plusieurs de ces objets correspondent à des composants Kiwui existants dont je fais la liste ici, et pour lesquels j'ai déjà écrit des subscripts génériques (folder result_block des subscripts).

CheckBox

Il suffit d'ajouter un data-xlabel dans le formulaire React, le composant Kiwui permet déjà de savoir si la checkbox est cochée ou non. La checkbox est une div, on peut pointer sur son xlabel pour afficher l'aide en ligne.




```
return (
  <Form onSubmit={e => e.preventDefault()}>
    <FormLine helpUrl={urls[0]}>
      <CheckBox
        disabled={disabled}
        label={formatMessage({ id: 'rule.sequencing.split.selection' })}
        checked={statementResult.splitList!}
        onChange={value => updateSequencing('splitList', value)}
        xlabel="splitList"
        tickSize={13}
      />
    </FormLine>
```

Sous scripts liés : ClickCheckbox.ltf IsCheckboxFilled.ltf IsCheckboxEmpty.ltf

Input

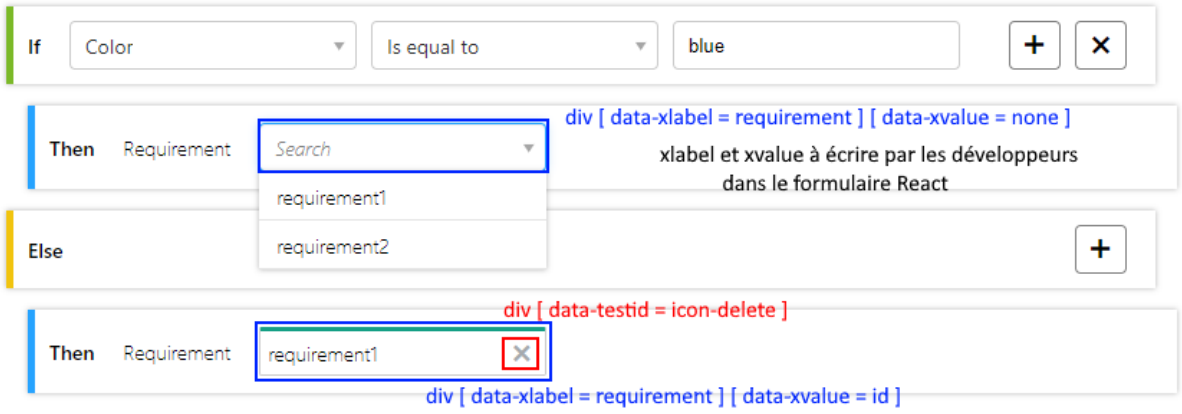
Il suffit d'ajouter un data-xlabel dans le formulaire React.


```
<FormLine helpUrl={urls[1]}>
  <label htmlFor="orders-number">{formatMessage({ id: 'rule.sequencing.number.orders.
  <Input
    disabled={disabled}
    id="orders-number"
    data-xlabel="orders-number"
    type="number"
    numberMaxDigits={0}
    value={statementResult.firstSubListSize}
    error={!isFirstSubListSizeValid(statementResult.firstSubListSize)}
    icon={<ErrorIcon errorKey="error.not.positive.field" />}
    width={50}
    min={0}
    onBlur={evt => updateSequencing('firstSubListSize', evt.target.value)}
  />
</FormLine>
```

Sous scripts liés: FillInput.ltf CheckInputValue.ltf

Picker

Il faut ajouter dans le formulaire React un data-xlabel et un data-xvalue qui vaut « non » quand aucune valeur n'est sélectionnée pour le Picker. Le picker est une div et peut être utiliser pour afficher l'aide en ligne.

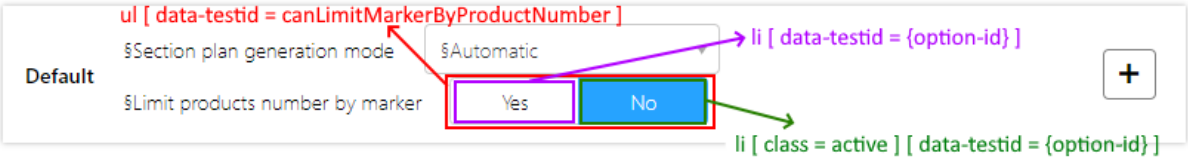



```
<Form>
  <FormLine helpUrl={urls[0]}>
    <label htmlFor={`requirement-${statementIndex}`}>{formatMessage({ id: 'cutting.requirement' })}</label>
    <DropDownSearch
      data-xlabel="requirement"
      data-xvalue={statementResult.requirementId ? statementResult.requirementId : 'none'}
      listItems={requirements}
      value={statementResult.requirementId}
      onChange={handleRequieremementChange}
      customRenderSelection={(item: any) => <DropDownSearchRenderer item={item} disabled={disabled} onDelete={() => hand
      disabled={disabled}
      placeholder="Search"
      width={200}
    />
  </FormLine>
</Form>
```

Sous scripts liés: CheckPickerEmpty.ltf CheckPickerValue.ltf SetPickerEmpty.ltf SetPickerValue.ltf

Item Switcher

Aucune action de la part des développeurs, les labels sont générés avec les infos obligatoires à la création du composant. L'ItemSwitcher n'est pas un div et ne peut pas servir pour l'aide en ligne.



Sous scripts liés: ChangeItemSwitcher.ltf CheckActiveItemSwitcher.ltf