Préambule

Les expressions ici utilisent la syntaxe reconnue par python qui suit le standard POSIX (à quelques exceptions près [1)]).

Capture

Les parenthèses **( )** de groupement permettent de délimiter un ensemble d'alternatives et d'opérer des captures. Le plus souvent si nous réalisons une capture nous voudrons pouvoir y faire référence via un nom que nous aurons choisi.

En python la syntaxe qui permet cela est la suivante

```
(?P<nom_du_groupe_de_capture>...)
```

La sous-expression rationnelle(ici symbolisée par ...) qui aura été trouvée sera accessible par le nom ***nom_du_groupe_de_capture***

Dès lors nous pouvons y faire référence dans la suite de l'expression rationnelle elle-même :

```
(?P=nom_du_groupe_de_capture)
```

Mais dans le cas de EzGED nous y ferons le plus souvent référence lors d'un post formatage grâce à la syntaxe @<groupname>.

Tout de suite un petit exemple.

Numéro de facture

Soit le texte extrait suivant

```
RueDuCommerce SA
Facture 44-50, avenue du Capitaine Glarner
93585 Saint-Ouen Cedex
EzDEV
                                                                    2
Rue Adolphe Pégoud
90130 PETIT-CROIX
        Numéro de facture : 56497        Référence client : 100132
        Date de la facture : 04/11/2015
```

Cet exemple illustre bien l'un des cas que vous retrouvez le plus souvent. Il s'agit de récupérer un numéro (ici de facture).

Nous pourrions être tentés de faire une recherche avec le filtre de champ

```
Numero de facture : .*
```

puis un post formatage

```
@keepdigits(@self)
```

Mais dans notre exemple ci-dessus si la longueur du champ ou les paramétrage de positionnement sont trop laxistes nous avons le risque de récupérer aussi la référence client.

Nous pouvons procéder autrement en capturant explicitement le premier numéro de la ligne:

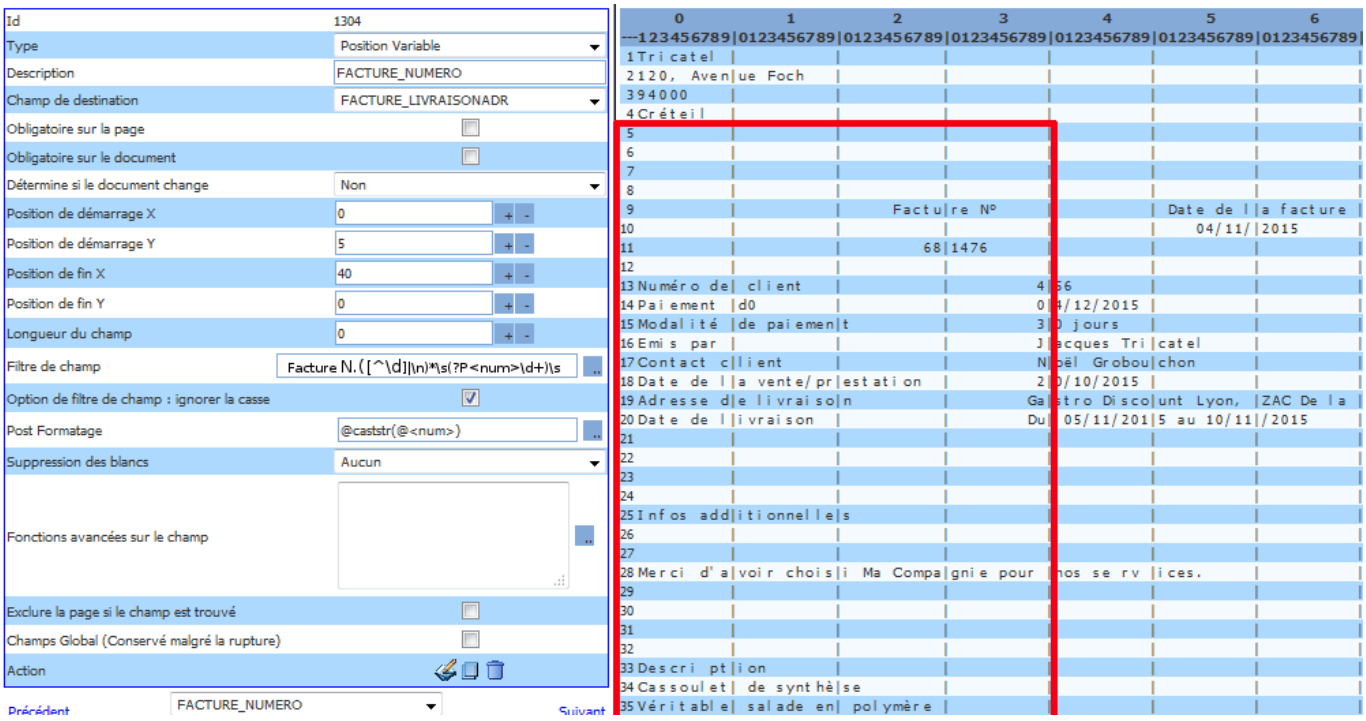
```
Numero de facture : (?P<numero>\d+)\D*
```

Le premier numéro est la première suite de chiffres non interrompue. Elle prend fin dès lors que l'on rencontre autre chose qu'un chiffre (comme un espace ou une lettre).

\D est équivalent à la séquence [^0-9]. Elle identifie tout caractère sauf un chiffre.

Recherche multi-ligne

Prenons cet exemple de texte extrait



Si l'on souhaite récupérer le numéro de facture on pourrait le faire assez simplement en utilisant la macro @relativeto. Ce serait une bonne solution mais il existe une alternative qui ne nous obligera pas à créer deux champs d'analyse.

Comme le montre la capture nous utilisons un champ de type variable et nous ciblons à l'aide des paramètres de position la zone de recherche. La zone de recherche est encadrée en rouge (l'image

est tronquée, elle s'étend jusqu'au bas du document).

Et l'expression rationnelle que nous utilisons est la suivante :

```
Facture N.([^\d]|\n)*(?P<num>\d+)
```

Décomposons l'expression.

Expression	Signification
Facture N.	Ce que l'on recherche commence par cette chaîne. Le point remplace le signe ° qui n'est pas bien géré.
([^\d]|\n)*	Ce qui suit doit être un caractère (excluant les chiffres) ou un retour à la ligne. Ce motif peut se répéter 0 ou plusieurs fois
(?P<num>\d+)	On doit trouver au moins un chiffre. Cette suite de chiffre est capturée dans la variable num

En résumé : Nous cherchons une chaîne qui **commence par *“Facture N”***, qui est suivit **éventuellement** (mais pas nécessairement) par **tout type de caractère sauf les chiffres** mais en **incluant les sauts de ligne** et cet enchaînement est suivi par **au moins un chiffre**.

L'exemple dans la capture ci-dessus nous renvoi comme valeur

```
681476
```

1)

Article wikipedia sur les expression rationnelles
https://fr.wikipedia.org/wiki/Expression_rationnelle#Python

From:
http://wiki.ezdev.fr/ - **EzGED Wiki**

Permanent link:
http://wiki.ezdev.fr/doku.php?id=cold:tutoriaux:regex&rev=1462262255

Last update: **2023/03/17 09:56**