Cette page se propose d'expliquer en détails quelques aspects du paramétrage d'un COLD et d'un modèle d'analyse COLD.

Centre d'apprentissage

Le centre d'apprentissage est l'interface qui vous permet de paramétrer un modèle d'analyse COLD.

Filtre de champ

Le filtre de champ permet de spécifier un motif à rechercher dans la page au moyen d'une expression régulière.

Exemple : Votre Cold porte sur l'indexation de factures clients. Chaque facture comporte une ligne “Numéro client : xxxxxx”. Plutôt qu'indiquer une position absolue on préfèrera rechercher le motif “Numéro client : <suite de chiffres>” En expression régulière cela se traduit par “Numéro client : \d*” (/!\ sans les doubles quotes, ou guillements en français)

Afin de tirer parti des expressions régulières, qui sont un outil fondamental pour l'analyse de documents, nous vous recommandons vivement de vous familiariser avec leurs notions et syntaxes. Vous pouvez consulter la page wikipedia sur les expressions régulières ou encore celle-ci Module python pour les expressions régulières sachant que c'est ce module qui traitera vos expressions régulières.

Les macros

Ce que nous appelons macro sont simplement des fonctions spécifiques à EzGED

Macros de formatage

Ces macros sont utilisables dans le champ **post formatage**

@self fait référence à la valeur du champ.
@<x> fait référence à une variable (nommée arbitrairement x) capturée dans l'expression régulière du filtre de champ.

Macro	Description	Exemple
@caststr(@<x>)	Convertit en chaîne de caractères la variable	
@concat('@sep:<separateur>',@<groupe1>,@<group2>,...)	Concatène, avec un séparateur si précisé, les valeurs de chaque groupe capturé	@concat(“@sep:-”,@<jour>,@<mois>,@<annee>)

Macro	Description	Exemple
@concatdate('@sep:<separateur>',@<groupe1>,@<group2>,...)	Idem que la macro concat mais applique à sa sortie la macro Smartdate	@concatdate(“@sep:-”,@<jour>,@<mois>,@<annee>)
@getinlist('table','champ_recherche','champ_retour')	cherche dans *table* l'enregistrement pour lequel le champ *champ_recherche* a pour valeur celle trouvée par l'analyse et retourne la valeur du champ *champ_retour*	
@getinlist('table','champ_recherche','champ_retour')		
@getfromdatasource(2,'table','champ_recherche','champ_retour')		
@int2month(<nombre>)	Convertit un mois sous forme numérique en sa représentation littérale	@int2month(3) ou @int2month(“03”) = “mars”
@int2day(<jour>)	Convertit un jour sous forme numérique en sa représentation littérale	@int2day(3) ou @int2day(“03”) = “mercredi”
@keepdigits(@self)	Ne conserve que les caractères numériques.	
@monthint(@self)	Convertit un mois sous forme littérale en sa représentation numérique	février deviendra 2
@Smartdate(@self)	Convertit la chaîne de caractères trouvée en date valide (si possible).	
@Smartfloat(@self)	Convertit la chaîne de caractères trouvée en flottant.	

@concat (depuis la révision 4244)

Pour préciser un séparateur (i.e une chaine de caractères à insérer entre chaque valeur) il faut que le premier argument de la macro soit **'@sep:<*separateur*>'** où <séparateur> est à remplacer par le caractère (ou la chaine de caractères) de votre choix.

En dehors des variables (exemple : **@<x>**) tous les paramètres de la macro doivent être entourés de guillemets simples. Les guillements simples sont donc à proscrire en dehors de cet usage.

Exemple de concaténation d'une capture (stockée dans la variable numero) sans séparateur :

```
@concat('N° ',@<numero>)
```

Exemple de concaténation avec séparateur. Nous choisirons ici le tiret :

```
@concat('@sep:-',@<nombre1>,@<nombre2>,@<nombre3>) => exemple : 63-54-108
```

Recherche par SIREN

La macro **@society** cherche dans le document à analyser un numéro de SIREN et retourne le nom de la société correspondante. Pour bien comprendre les résultats retournés par @society, voici comment procède la macro :

1. Elle interroge tout d'abord votre base données en lançant une requête sur la table societycache. Si un résultat est trouvé elle le retourne.
2. Si elle interroge notre base de données et retourne le résultat si trouvé.
3. En dernier recours elle effectue une interrogation sur différents moteurs de recherches et analyse le flux de résultats.
4. Enfin, si aucun résultat n'est trouvé à la suite des étapes 1,2 et 3 elle retournera la valeur “Inconnu”.

/!\ MAJ revision 4111 si @society ne parvenant pas à joindre le serveur il ne renvoie pas *Inconnu* et donc l'analyse bloquera si le champ est obligatoire. Cette maj fait suite à ce bug : http://mantis.ezdev.fr/view.php?id=439

Recherche par Numéro de TVA

La macro **@societyeu** cherche dans le document à analyser un numéro de TVA Intracommunautaire et retourne le nom de la société correspondante.

ATTENTION !cette macro n'est pas disponible en version ONE.

Pour bien comprendre les résultats retournés par @societyeu, voici comment procède la macro :

La macro cherche par ordre de préférence les numéro de TVA dans cet ordre par défaut,

“FR,BE,IT,GB,LU,DK,EL,IE,NL,AT,PT,FI,SE,HU,PL,CY,CZ,EE,LV,SI,RO,DE,ES,SK,BG,HR,MT,LT”

puis interroge un web service de la base européenne : Pour l'instant les bases des pays suivants ne sont pas interrogeables : DE,ES,SK,BG,HR,MT,LT c'est pour cela qu'ils sont en dernier dans l'ordre de recherche par défaut, toutefois vous pouvez utiliser la base de cache pour les faire fonctionner, pour cela modifier le champs societycache_code en Varchar de 20.

Vous pouvez changer l'ordre de recherche ou le limiter à certains pays uniquement en ajoutant la variable suivante dans instance.conf exemple ici limitation aux pays limitrophes à la France avec francophones de préférence + Angleterre et Irlande:

[ezged]

```
mytvacountries = FR,BE,LU,IT,GB,IE,DE,ES
```

Vous pouvez également renseigner la variable suivante pour ignorer votre ou vos numéros de TVA

```
mytvaeu = FR72515180115, BE1234567890
```

Fonctions avancées

Vous disposez d'une possibilité très intéressante, celle d'utiliser des fonctions pythons dans la zone de texte “Fonctions avancées sur le champ”. Toute fonction python n'est pas utilisable et vous serez principalement intéressés par celles s'appliquant aux chaînes de caractères.

Manipulations de chaînes de caractères

Remplacement

str**.replace***(ancien,nouveau)* : remplace dans la chaîne identifiée par *str* toutes les occurrences de *ancien* par *nouveau*.

Exemple :

```
"Ceci est un fruit".replace("fruit","légume") affichera "Ceci est un légume"
```

Exemple dans EzGED : Nous avons un champ de recherche appelé *RECHERCHE_nom* dont le numéro est 7 et contenant un nom. Comme vous le savez déjà nous pouvons récupérer cette valeur par *@FLD42*. Imaginons que notre champ *RECHERCHE_nom* contienne la valeur **Dupont** après recherche du nom dans le document. Pas de chance, sur l'ensemble de nos documents nous avons confondu Dupont et Dupond. Nous voudrions néanmoins que le document soit indexé avec le nom correct. Voici comment faire :

```
@FLD3.replace("t","d")
```

Remplacement + expression régulière

On peut utilisation la fonction **re.sub**(expr,nouveau,chaine)

Découpage

str**.split***(séparateur)* : retourne une liste des mots contenus dans *str*. Le découpage des mots s'effectue en fonction du paramètre *séparateur*. S'il n'est pas spécifié le séparateur sera l'espace. Exemple :

```
"tomate,poivron,courgette".split(",") retourne la liste
["tomate","poivron","courgette"].
```

Exemple dans EzGED : Le champ de recherche numéro 3 se voit affecter lors de l'analyse le nom du fichier qui est “747_reservation_sydney.pdf” (sans les “) Pour retrouver le numéro de réservation avec la fonction split c'est très simple :

```
@FLD3.split("_")[0]
```

Ce qui nous retourne **747**. Vous remarquerez ici que nous avons fait suivre l'appel de la fonction par *[0]*. Pour comprendre de quoi il s'agit voir la section sur les indices.

Découpage / Slicing

En python les indices sont notamment utilisés pour les listes et les chaînes de caractères, ces dernières pouvant être vues comme des listes de caractères. Un indice permet de se positionner dans une liste. Il est important de noter qu'en python (comme dans la plupart des langages informatiques) la numérotation des indices débute à 0. Le premier élément d'une liste aura donc pour indice 0.

Exemple :

```
chiffres = ["un","deux","trois"]
print chiffres[1]
```

Ceci affichera **deux** puisqu'il s'agit de la valeur de l'élément de la liste en position 1.

Nous comprenons maintenant l'exemple vu un peu plus haut avec notre champ contenant la valeur **747_reservation_sydney.pdf**

```
@FLD3.split("_")[0]
```

Tout d'abord la fonction **.split** va retourner la liste

```
["747","reservation","sydney.pdf"]
```

puis avec *[0]* nous indiquons à Python que nous souhaitons récupérer le premier élément (position 0) de cette liste.

Indices négatifs

Voilà une fonctionnalité extrêmement pratique que nous offre python.

Reprenons l'exemple précédent. Nous avons notre liste résultat du split sur le nom de fichier.

```
>>> liste = "747_reservation_sydney.pdf".split("_")
>>> ["747","reservation","sydney.pdf"]
```

Supposons que vous souhaitiez récupérer le dernier élément de cette liste. Vous pourriez faire ceci :

```
>>> liste[2]
>>> 'sydney.pdf'
```

Mais imaginez que la taille de cette liste ne soit pas connue. Vous savez que le nom de la ville est en dernière position dans le nom du fichier (juste avant l'extension) mais il se peut que d'autres chaînes de caractères s'intercalent entre le mot *reservation* et le nom de la ville. En python néanmoins le problème se résout facilement comme ceci :

```
>>> liste[-1]
>>> 'sydney.pdf'
```

On indique que l'on souhaite récupérer l'élément en première position à partir de la fin de la liste, c'est-à-dire le dernier élément.

Extractions

Ce que nous avons vu sur les indices jusqu'ici, et ce que nous savons des listes et des chaînes de caractères, va nous permettre de réaliser des extractions au sein, par exemple, d'une chaîne de caractère.

Prenons la chaîne de caractère suivante :

```
>>> s = "un_nom_de_fichier_quelconque.pdf"
```

Nous souhaitons extraire le nom du fichier sans son extension. Évidemment la longueur du nom n'est pas connue à l'avance. Nous allons donc extraire les caractères depuis le début de la chaîne (position 0) jusqu'à 4 caractères avant la fin.

```
>>> s[0:-4]
>>> 'un_nom_de_fichier_quelconque'
```

Exemple complet

Combinons l'ensemble des notions abordées afin d'extraire le seul nom de la ville dans un nom de fichier, en prenant soin de capitaliser sa première lettre. Dans notre exemple le nom de fichier est **747_reservation_sydney.pdf** et il est contenu dans le champ numéro 3.

```
@FLD3.split("_")[-1][0:-4].capitalize()
```

Nous retournera bien **Sydney**

Dates et échéance

On peut facilement récupérer la date courante :

```
_common.tstamp()
```

Plus intéressant encore, la fonction *tstamp()* accepte un paramètre, à savoir un nombre de seconde depuis une certaine époque (cf. Heure Unix

Pour récupérer le nombre de seconde depuis cette date jusqu'à maintenant on utilise en python :

```
time.time()
```

On comprend alors facilement comment obtenir une date d'échéance de X jours après la date courante. Il suffit d'additionner le nombre de secondes représentant la date courante avec le nombre de secondes dans X jours.

Exemple si l'on veut une date d'échéance à 30 jours après la date courante :

```
>>>_common.tstamp(time.time()+30*24*3600)
'2014-11-14 11:12:52'
```

Mise en garde

Utilisation de @filename et @filedate

L'analyse passe dans le centre d'apprentissage mais échoue en mode travail. Il faut noter que la macro **@filename** récupère le nom du fichier apprenti lorsque l'on est dans le centre d'apprentissage. Or, ce fichier apprenti est préfixé par le numéro de job. Il faut prendre en compte que ce numero ne sera pas contenu dans le nom du fichier lors de l'exécution “normale”.

Nb: la même précaution s'applique à la macro @filedate qui dans le centre d'apprentissage vous retournera la date de dernière modification du fichier apprenti (le fichier apprenti étant une copie du fichier résultat de l'ocr).