w.c.s. - Development #32403

Enregistrement de pièces jointes à l'historique via RegisterComment

16 avril 2019 18:16 - Michaël Bideau

Statut:	Fermé	**Début:**	16 avril 2019
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Planning:**	Non

Description

Aujourd'hui dans WCS il est possible de recevoir une réponse WebService de type:

1. JSON: uniquement des données (plutôt textuelles)
2. Fichier: qui peut éventuellement être associé à une données de traitement du formulaire

Mais il est impossible d'envoyer le contenu d'un (ou plusieurs) fichier en même temps que d'autres informations (en JSON).

Le patch ci-joint à pour but de permettre cela via une action de Workflow d'enregistrement d'un commentaire avec pièces jointes (qui suivrait une réponse JSON d'un WebService).

Il nécessite deux éléments:

- un retour de WS en structure JSON dont l'une des clés contiendra le contenu du fichier (encodé en base64 ou non), et éventuellement d'autres métadonnées telles que 'filename', 'content_type', etc.
- une action de Workflow configurée avec en pièce jointe une expression Python faisant appel à utils.attachment(...) et exploitant les données de la structure JSON du WS stockées dans le Workflow

Exemple d'usage de la fonction utils.attachment():

```
utils.attachment(varname='afile'
, filename=ws1_response_afile_filename, content_type=ws1_response_afile_content_type, b64_content=
ws1_response_afile_b64_content)
```

Ci-joint également, une capture d'écran du rendu du paramétrage de l'action de Workflow d'enregistrement d'un message avec pièces jointes à l'historique. Ce cas teste 3 sources de pièces jointes différentes:

1. la fonction utils.attachment() exploitant les données de la structure JSON du retour d'un appel WebService stockées dans le Workflow
2. une donnée de traitement du Workflow de type fichier (automatiquement proposée)
3. une expression python utilisant une référence à un fichier fourni par l'usager via un champs de type Fichier dans le formulaire

D'un point de vue métier, cela permettra de simplifier grandement les worklfows devant faire des appels à des WebService renvoyant données ET fichiers et permettra aussi de simplifier le développement des connecteurs Passerelle (notamment openADS) devant gérer ce type de cas.

J'attends impatiemment vos retours.

Merci d'avance.
Bien cordialement,

Demandes liées:

Lié à w.c.s. - Development #32535: Dans wcs.formdata.flatten_dict ne plus sup...	**Rejeté**	**23 avril 2019**
Lié à w.c.s. - Development #54081: Filtrer les fichiers attachés dans l'actio...	**Fermé**	**18 mai 2021**

Révisions associées

Révision 8bf94c5e - 09 juin 2019 17:42 - Michaël Bideau

workflows: add possibility to attach files to notes in history (#32403)

Historique

#1 - 17 avril 2019 09:28 - Frédéric Péters

- Assigné à Frédéric Péters supprimé

- Priorité changé de Haut à Normal

#2 - 17 avril 2019 09:56 - Frédéric Péters

(je vais commencer à faire quelques commentaires de style)

```
    assert d == {'varname'     : 'var1'
                ,'filename'    : ''
                ,'content_type': 'application/octet-stream'
                ,'b64_content' : ''}
```

De manière générale on est assez proche de http://docs.python-requests.org/en/master/dev/contributing/#kenneth-reitz-s-code-style (pas cette construction avec les virgules en début de ligne).

```
b64_content if b64_content else base64.encodestring(content) if content else ...
```

On essaie de limiter la forme "x if y else z" aux cas simples, certainement jamais en enchainer plusieurs sur une même ligne.

```
- class RegisterCommenterWorkflowStatusItem(WorkflowStatusItem):
+ class RegisterCommenterWorkflowStatusItem(WorkflowStatusItemWithAttachments):
```

Que les quelques méthodes utiles à partager soient plutôt tapées sur WorkflowStatusItem directement.

```
# copy-pasted from class SendmailWorkflowStatusItem
```

Le code n'y étant plus, pas utile de mentionner son origine.

```
    def get_attachments(self, as_uploads=False):
        if as_uploads:
            return self.get_attachments_as_uploads()
        return self.attachments

    def get_attachments_as_uploads(self):
        return self.uploads
```

get_attachments() ne me semble pas utilisé et dans l'ensemble ça sonnait bizarre d'avoir un get_attachments(as_uploads=True) qui ne fasse pas la même chose que get_attachments_as_upload().

get_attachments_as_upload() étant juste l'accès à un attribut, on peut s'en passer, et à l'endroit où la méthode était utilisée directement faire self.uploads.

Mais cet attribut qui conserve un état entre deux appels m'ennuie assez, plutôt retourner le résultat, côté appelant, que ça donne :

```
            attachments = self.convert_attachments_to_uploads()
            self.attach_uploads_to_form(formdata, attachments)
```

```
context='[%s/attachments]' % self.__class__)
```

Mettre une vraie chaine, genre workflow/attachments, qu'un grep puisse trouver directement l'affaire.

```
    def get_attachment_varname(self, attachment, upload=None):
```

Toutes ces situations ne m'ont au final pas l'air d'être utilisées, j'ai l'impression qu'on a soit xx['varname'] (dictionnaire), soit xx.varname (objet).

```
    def form_has_attachment(self, formdata, varname=None, attachment=None):
```

Avoir les paramètres optionnels donne l'impression que ça pourrait être utile sans eux, que genre ça retournerait True s'il y avait un fichier attaché.

Aussi cette méthode m'a ensuite l'air utilisée pour ne pas écraser un contenu existant; on a plutôt tendance à faire l'inverse, que du nouveau contenu écrase l'ancien.

De manière générale aussi, utiliser formdata plutôt que form, réserver form aux formulaires HTML.

~~

J'ai déjà poussé ce patch dans une branche pour notre jenkins, ça devrait bientôt apparaitre sur https://jenkins2.entrouvert.org/job/wcs-wip/

#3 - 17 avril 2019 10:51 - Benjamin Dauvergne

Quelque chose m'échappe, je ne comprends pas ce qu'on ne pouvait pas déjà faire avec une donnée de traitement de type fichier (on gère déjà des WS qui renvoient des fichiers encapsulé dans du JSON), étant donné un ws avec un varname "ws1" et un sous champ "fichier" ayant la structure:

```
{
  "fichier1": { filename:.., b64_content: ...},
  "fichier2": { filename:.., b64_content: ...}
}
```

il suffit d'une action "affecter à une variable de traitement" avec l'expression "ws1['fichier1']" pour affecter contenu à une variable de traitement de type fichier.

#4 - 17 avril 2019 10:52 - Frédéric Péters

On veut proposer à l'usager de télécharger le document.

#6 - 18 avril 2019 12:00 - Michaël Bideau

- Fichier 0001-workflow-added-attachments-for-RegisterComment.patch ajouté

Bonjour Frédéric et Benjamin,

Merci pour vos retours rapides.

Ci-joint le patch intégrant les remarques.

Ci-dessous quelques réponses/commentaires.

Merci encore pour votre co-construction.

Frédéric Péters a écrit :

> (je vais commencer à faire quelques commentaires de style)
> [...]
> De manière générale on est assez proche de http://docs.python-requests.org/en/master/dev/contributing/#kenneth-reitz-s-code-style (pas cette construction avec les virgules en début de ligne).

C'est corrigé.
Pour info, rien n'est indiqué pour la gestion des virgules, seulement l'indentation, dans le style Kenneth ou PEP8. Mais les exemples contenant des virgules vont dans votre sens.
Personnellement, je trouve mon style plus efficace (détection d'erreur et plus lisible), mais j'accepte aisément cette petite concession. Envisagez-vous d'utiliser un linter prochainement (black)?

> On essaie de limiter la forme "x if y else z" aux cas simples, certainement jamais en enchainer plusieurs sur une même ligne.

C'est corrigé (j'avais sur-compliqué l'instruction).

> Que les quelques méthodes utiles à partager soient plutôt tapées sur WorkflowStatusItem directement.

Là je suis plutôt en désaccord car je trouve que la gestion des pièces jointes n'est pas adaptée à toute les actions de Workflow. Néanmoins j'ai remonté les méthodes dans la classe *WorkflowStatusItem* comme demandé.

> `# copy-pasted from class SendmailWorkflowStatusItem`
>
> Le code n'y étant plus, pas utile de mentionner son origine.

C'est retiré (c'était pour faciliter la revue de code).

> get_attachments() ne me semble pas utilisé et dans l'ensemble ça sonnait bizarre d'avoir un get_attachments(as_uploads=True) qui ne fasse pas la même chose que get_attachments_as_upload().
>
> get_attachments_as_upload() étant juste l'accès à un attribut, on peut s'en passer, et à l'endroit où la méthode était utilisée directement faire self.uploads.
>
> Mais cet attribut qui conserve un état entre deux appels m'ennuie assez, plutôt retourner le résultat, côté appelant, que ça donne :
>
> [...]

C'est corrigé. J'ai retiré les deux méthodes get_attachments() et get_attachments_as_uploads() et passé en paramètre de la fonction attach_uploads_to_form() les *uploads*

> Mettre une vraie chaine, genre workflow/attachments, qu'un grep puisse trouver directement l'affaire.

C'est corrigé.

> `def get_attachment_varname(self, attachment, upload=None):`
>
> Toutes ces situations ne m'ont au final pas l'air d'être utilisées, j'ai l'impression qu'on a soit xx['varname'] (dictionnaire), soit xx.varname (objet).

Si si, je confirme que toutes ces situations peuvent être rencontrées. J'ai donc laissé la méthode telle quelle.

> `def form_has_attachment(self, formdata, varname=None, attachment=None):`
>
> Avoir les paramètres optionnels donne l'impression que ça pourrait être utile sans eux, que genre ça retournerait True s'il y avait un fichier attaché.

J'ai rendu les paramètres obligatoires.

> Aussi cette méthode m'a ensuite l'air utilisée pour ne pas écraser un contenu existant; on a plutôt tendance à faire l'inverse, que du nouveau contenu écrase l'ancien.

Non ce n'est pas pour éviter un écrasement, mais pour éviter d'avoir deux fois la même pièce jointe (justement car il n'y a pas d'écrasement). J'ai donc laissé cette méthode et ce comportement tels quels.

> De manière générale aussi, utiliser formdata plutôt que form, réserver form aux formulaires HTML.

J'ai renommé les méthodes:

- *form_has_attachment* en *formdata_has_attachment*
- *attach_uploads_to_form* en *attach_uploads_to_form*

J'attends une nouvelle fois vos retours.

Bien cordialement,

#7 - 18 avril 2019 12:15 - Frédéric Péters

> Envisagez-vous d'utiliser un linter prochainement (black)?

C'est justement en cours de discussion côté Django et s'ils s'accordent dessus, et une fois basculés en Python 3, je pense qu'on pourra l'envisager. (il te reste quantité de virgules en début de ligne dans les tests)

> > Toutes ces situations ne m'ont au final pas l'air d'être utilisées, j'ai l'impression qu'on a soit xx['varname'] (dictionnaire), soit xx.varname (objet).
>
> Si si, je confirme que toutes ces situations peuvent être rencontrées. J'ai donc laissé la méthode telle quelle.

Je ne suis pas à l'aise avec le bout avec la regex, c'est pour gérer quelle situation ?

> attachments = [x for _,x in self.convert_attachments_to_uploads().items()]

self.convert_attachments_to_uploads().values() fait le job.

~~

J'ai déjà poussé vers jenkins, je regarderai plus attentivement un peu plus tard.

#8 - 18 avril 2019 12:21 - Benjamin Dauvergne

Pourquoi empêche-t-on les pièces jointes multiples pour un même varname ? On a pas cette limitation sur AddAttachmentWorkflowStatusItem.

attach_uploads_to_formdata et formdata_has_attachment sont partagés mais il me semble qu'ils ne sont utilisés que par RegisterCommenterWorkflowStatusItem, ils devraient rester là.

#9 - 18 avril 2019 13:26 - Michaël Bideau

- Fichier 0001-workflow-added-attachments-for-RegisterComment.patch ajouté

Super réactifs, merci.

Ci-joint un nouveau patch tenant compte de vos remarques.

> il te reste quantité de virgules en début de ligne dans les tests

Oui pardon, c'est corrigé (j'avais oublié de faire une passe dans le code des tests).

> > Toutes ces situations ne m'ont au final pas l'air d'être utilisées, j'ai l'impression qu'on a soit xx['varname'] (dictionnaire), soit xx.varname (objet).
>
> Si si, je confirme que toutes ces situations peuvent être rencontrées. J'ai donc laissé la méthode telle quelle.
>
> Je ne suis pas à l'aise avec le bout avec la regex, c'est pour gérer quelle situation ?

C'est pour tous les cas sauf les *dict* en fait (où l'on a explicitement la *varname* passée).
Lorsque l'on a un *PicklableUpload* dans ce traitement il n'est pas associé à un *varname* (enfin pas que je sache), donc je m'en remet à la valeur de la variable *attachment* qui contient généralement quelque chose comme form_var_XXX_raw.
Idem lorsque le *PicklableUpload* retourné est *None*.
Si vous connaissez une méthode qui permettrait de retrouver l'identifiant du champs de type fichiers qui fourni le *PicklableUpload* à partir de la valeur de la variable *attachment* (ou autre), je suis preneur.

> attachments = [x for _,x in self.convert_attachments_to_uploads().items()]
>
> self.convert_attachments_to_uploads().values() fait le job.

Oups, héhé. Bien vu merci.

> J'ai déjà poussé vers jenkins, je regarderai plus attentivement un peu plus tard.
>
> Pourquoi empêche-t-on les pièces jointes multiples pour un même varname ? On a pas cette limitation sur AddAttachmentWorkflowStatusItem.

J'ai supprimé la méthode formdata_has_attachment() et désormais la méthode attach_uploads_to_formdata() effectue toujours l'ajout de pièce jointe.

> attach_uploads_to_formdata et formdata_has_attachment sont partagés mais il me semble qu'ils ne sont utilisés que par RegisterCommenterWorkflowStatusItem, ils devraient rester là.

J'ai déplacé la méthode attach_uploads_to_formdata dans la classe RegisterCommenterWorkflowStatusItem.

Merci encore.

#10 - 19 avril 2019 18:59 - Benjamin Dauvergne

- Il ne sert à rien de répéter les valeurs par défaut qui sont déjà dans la signature de la fonction :

```
        'filename'    : filename if filename else '',
        'content_type': content_type if content_type else 'application/octet-stream',
```

  mettre directement filename et content_type comme avant.
- Même fonction : encodestring est déprécié, on utilise plutôt b64encode

La façon de passer varname me semble quand même très moche, je me demande si on ne devrait pas avoir un widget exprès, sous forme de tableaux à deux colonnes:

- première colonne varname obligatoire
- deuxième colonne le widget attachment actuel (et à la fin il ne devrait pas rester énormément de code commun avec l'envoi de mail)

Ainsi ça ne nécessite plus de modification à utils.attachment qui ne sert en fait qu'à ajouter varname

Aussi veut-on absolument rendre varname obligatoire ?

#11 - 20 avril 2019 08:15 - Frédéric Péters

Je suis en congés et ne vais plus suivre ce ticket pendant quelques jours; je reste aussi plutôt ennuyé par la partie varname, pour avancer j'aurais comme proposition de ne simplement pas s'en soucier, cette action sert à exposer à l'usager un fichier dans l'UI et le code qui fait ça n'a pas besoin de varname.

#12 - 23 avril 2019 10:51 - Michaël Bideau

Bonjour messieurs,

Merci pour vos retours.

Je vois que ça vous embête de toucher à la fonction evalutils.attachment(), alors bien que je ne vois le problème, je vous propose de ne pas l'utiliser du tout.
Ainsi l'expression Python (dans le widget des pièces jointes) au lieu d'être actuellement (avec le dernier patch):

```
utils.attachment(varname='afile',filename=ws1_response_afile_filename, content_type=
ws1_response_afile_content_type, b64_content=ws1_response_afile_b64_content)
```

deviendrait

```
{'varname':'afile', 'filename':ws1_response_afile_filename, 'content_type': ws1_response_afile_content_type,
'b64_content': ws1_response_afile_b64_content}
```

C'est plus court, et sans magie noire (totalement inutile ici).

Est-ce que cela vous conviendrait?

Sinon pour répondre en détail à Benjamin:

> Il ne sert à rien de répéter les valeurs par défaut qui sont déjà dans la signature de la fonction

Effectivement, c'est un oubli de ma part après avoir remis les valeurs par défaut (chaîne vide au lien de None). Je suis plus focalisé sur le fond actuellement et il m'arrive de zaper ces petites modifs. Bravo/merci à toi de le voir et me le remonter.

> Même fonction : encodestring est déprécié, on utilise plutôt b64encode

La fonction magique qui convertit le dictionnaire Python en *PicklableUpload* utilise un decodestring() donc j'ai utilisé l'équivalent en amont.

Merci encore.

Bien cordialement,

#13 - 23 avril 2019 11:25 - Benjamin Dauvergne

Michaël Bideau a écrit :

> Est-ce que cela vous conviendrait?

Je ne crois pas, le sens de l'intervention de Fred c'est de dire que tu n'as pas besoin de varname du tout, donc tu n'a pas besoin de construire un dico contenant un champ varname, donc tu dois pouvoir directement extraire du contenu JSON le dico de fichier attaché sans le reconstruire ni y toucher. Ça simplifie grandement le code de ne plus s'occuper de varname, et on pourra voir plus tard quand on en aura l'usage comment amener proprement la fonctionnalité dans l'interface. Ici l'expression deviendrait simplement

```
ws1_response.afile
```

simplifiant grandement les choses.

De mon coté je reste sur ma position que le contenu du fichier et le nom de la variable affectée à l'AttachmentPart sont deux choses bien différentes à ne pas mélanger (surtout dans la mesure ou ça complexifie l'usage grandement par rapport à l'envoi de mail par exemple).

> > Même fonction : encodestring est déprécié, on utilise plutôt b64encode
>
> La fonction magique qui convertit le dictionnaire Python en *PicklableUpload* utilise un decodestring() donc j'ai utilisé l'équivalent en amont.

La doc python a priorité sur nos usages, donc on corrigera les autres usages un jour de pluie, il reste qu'il vaut mieux ne pas en introduire de nouveaux.

#14 - 23 avril 2019 12:27 - Michaël Bideau

Bonjour Benjamin, merci pour ton retour rapide.

> Je ne crois pas, le sens de l'intervention de Fred c'est de dire que tu n'as pas besoin de varname du tout

Effectivement, pour simplement convertir un *PicklableUpload* en *AttachmentEvolutionPart* la *varname* n'est pas obligatoire. Cela répondrait donc au besoin de faire apparaître à l'usager la pièce jointe provenant du fichier décodé depuis le JSON du WS.

Cependant, si l'on souhaite utiliser ce fichier dans une action de Workflow de type Alerte, ou autre, il me semble qu'il est nécessaire de lui attribuer une *varname* pour pouvoir y faire référence? De plus il faudrait que cette *varname* ne soit pas dépendante du contenu du fichier ou du nom du fichier (donc fournie indépendamment du fichier).
Il existe la possibilité de parcourir les attachments du formulaire mais sans *varname* impossible (ou presque) de savoir quel fichier correspond à quelle notion/étape métier (par exemple si c'est un fichier fourni par l'utilisateur ou bien un retour de WS).

C'est pourquoi j'ai (depuis le départ, peut être à tort) essayé de répondre à ce besoin d'identifier la pièce jointe afin qu'elle soit exploitable par/dans d'autres actions de Workflow.
En effet, il est très probable que nous souhaitions afficher un message à l'usager pour lui indiquer ce qu'est le fichier qui apparaît dans ses pièces jointes. Et, bien que nous pourrions le faire de manière descriptive sans le référencer directement, le réafficher (lien/icone) dans le message me semble plus "user-friendly".

Si répondre à ce besoin est trop gênant, je consens à reporter celui-ci à plus tard.
Je trouverais cela dommage dans la mesure où le code est fonctionnel et même testé, et n'empêche en rien de le refactorer ultérieurement.

> tu dois pouvoir directement extraire du contenu JSON le dico de fichier attaché sans le reconstruire ni y toucher

Il ne me semble pas, mais je peux me tromper. En effet les données JSON de la réponse du WS sont "applaties" à la réception et la structure JSON n'est plus dispo, d'où le besoin de la reconstruire.

> Ici l'expression deviendrait simplement
>
> ```
> ws1_response.afile
> ```

Aucune variable, dans le contexte de l'évaluation de l'expression Python, ne permet d'accéder à la structure JSON renvoyée par le WS. En l'occurrence, ni ws1, ni ws1_response et ni ws1_response.afile, n'existent/ne sont définies.

Comme expliqué ci-dessus, je dois donc reconstruire un dictionnaire Python à partir des données JSON "applaties" sotckées dans les données du workflow.
Il est possible que je n'ai pas connaissance d'un autre moyen, si tel est le cas je suis preneur.

> > Même fonction : encodestring est déprécié, on utilise plutôt b64encode
>
> La fonction magique qui convertit le dictionnaire Python en PicklableUpload utilise un decodestring() donc j'ai utilisé l'équivalent en amont.

> La doc python a priorité sur nos usages, donc on corrigera les autres usages un jour de pluie, il reste qu'il vaut mieux ne pas en introduire de nouveaux

Certes, mais il faut bien que l'encodage corresponde au décodage et réciproquement. J'ai d'abord utilisé b64encode() mais ça pose des soucis.

Encore merci.

#15 - 23 avril 2019 15:16 - Benjamin Dauvergne

Michaël Bideau a écrit :

> Certes, mais il faut bien que l'encodage corresponde au décodage et réciproquement. J'ai d'abord utilisé b64encode() mais ça pose des soucis.
>
> Encore merci.

Il n'y a pas de différence entre les deux, juste des sauts lignes qui sont ignorés :

```
In [3]: base64.decodestring(base64.b64encode(s)) == s
Out[3]: True

In [4]: base64.b64decode(base64.encodestring(s)) == s
Out[4]: True
```

#16 - 23 avril 2019 15:23 - Benjamin Dauvergne

Michaël Bideau a écrit :

> tu dois pouvoir directement extraire du contenu JSON le dico de fichier attaché sans le reconstruire ni y toucher

> Il ne me semble pas, mais je peux me tromper. En effet les données JSON de la réponse du WS sont "applaties" à la réception et la structure JSON n'est plus dispo, d'où le besoin de la reconstruire.

Effectivement, mais donc je m'attacherai plutôt à résoudre ce problème qui me semble plus embêtant (impossibilité d'utiliser des sous-parties des réponses JSON); peut-être qu'on devrait supprimer le del d[k] dans flatten_dict.

La fonction flatten_dict servait surtout à l'ancien système de template mais avec les templates Django c'est désormais inutile.

#17 - 23 avril 2019 15:31 - Benjamin Dauvergne

- Lié à Development #32535: Dans wcs.formdata.flatten_dict ne plus supprimer la version non aplatie des données ajouté

#18 - 24 avril 2019 11:35 - Michaël Bideau

Bonjour Benjamin,

Merci à nouveau pour ton retour.

> peut-être qu'on devrait supprimer le del d[k] dans flatten_dict

Ce serait la meilleure conception, évitant les palliatifs ensuite.
Par curiosité: quelle est l'intérêt de la structure JSON "applatie" dans ce cas (puisqu'on pourrait accéder aux données de la structure JSON "telle quelle" de la même manière)?

Juste pour souligner un point: après cette modification il va y avoir l'information (le contenu du fichier) stocké simultanément de 4 manières différentes:

- dans les données de Workflow avec la structure JSON "applatie"
- dans les données de Workflow avec la structure JSON "telle quelle"
- dans le fichier décodé transformé en *PicklableUpload*
- dans la pièce jointe de type *AttachmentEvolutionPart*

J'avais essayé dans le premier patch proposé (touchant à wscall.py) de n'avoir l'information stockée qu'une seule fois (dans la pièce jointe de type *AttachmentEvolutionPart*, après avoir "vidé" la structure JSON "applatie").

Personnellement ça ne me gêne pas du tout, mais je me demande si cela ne vas poser des soucis de "purge" ultérieurement? Notamment comment savoir quoi purger, comment et quand?
C'était juste pour être sûr que vous ayez ça en tête au niveau des aspects de conception.

Merci encore.

Bien cordialement,

#19 - 26 avril 2019 18:49 - Benjamin Dauvergne

Michaël Bideau a écrit :

> Bonjour Benjamin,
>
> Merci à nouveau pour ton retour.
>
> > peut-être qu'on devrait supprimer le del d[k] dans flatten_dict
>
> Ce serait la meilleure conception, évitant les palliatifs ensuite.
> Par curiosité: quelle est l'intérêt de la structure JSON "applatie" dans ce cas (puisqu'on pourrait accéder aux données de la structure JSON "telle quelle" de la même manière)?

L'ancien format des templates (EZT au lieu de Django) ne supportait que l'accès à des attributs (a.b.c) et pas à des dictionnaires (a['b']['c']), on devait donc aplatir les dictionnaires pour les rendre utilisables. Django n'a pas cette limitation il ne fait pas de différence entre a.b, a['b'] et a[0], c'est la même syntaxe avec un point.

> Juste pour souligner un point: après cette modification il va y avoir l'information (le contenu du fichier) stocké simultanément de 4 manières différentes:
>
> - dans les données de Workflow avec la structure JSON "applatie"
> - dans les données de Workflow avec la structure JSON "telle quelle"

Non il n'est stocké qu'une fois, flatten_dict n'est utilisé qu'à la consommation, pas avant le stockage (workflow_data contient la donnée JSON pure telle qu'elle a été reçue) et même à la consommation les chaînes étant immutable en python elles ne sont pas dupliqués donc le base64 qui constitue le gros du contenu n'est présent qu'une fois en RAM; au moins durant le traitement d'un unique formdata.

- dans le fichier décodé transformé en *PicklableUpload*

En fait dans les deux cas qui nous intéressent où le fichier n'est que temporaire, on pourrait se contenter d'un Upload classique de quixote et pas d'un PicklableUpload vu que l'upload ne sera pas stocké (il sera transformé en AttachmentEvolutionPart qui a son propre stockage dans un cas ou attaché au mail généré dans l'autre cas).

- dans la pièce jointe de type *AttachmentEvolutionPart*

Oui j'ai conscience que c'est bien embêtant d'autant plus d'un point de vue utilisation correcte de postgres, workflow_data est un champ de type bytea, qui peut atteindre 1 Go mais je ne sais pas si c'est excellent pour les performances de la base (même si en fait postgres ne stocke pas ça inline au milieu des autres colonnes, ça prend de la place pour rien et ça pourrit le cache disque).

On pourrait expérimenter un jour avec un stockage dans un evolution part spécifique qui gèrerait un stockage sur disque quand c'est trop gros.

> J'avais essayé dans le premier patch proposé (touchant à wscall.py) de n'avoir l'information stockée qu'une seule fois (dans la pièce jointe de type *AttachmentEvolutionPart*, après avoir "vidé" la structure JSON "applatie").

Je te conseille de ne pas te lancer dans des travaux trop compliqués à ce stade, nous sommes prudents sur ce genre d'évolutions.

> Personnellement ça ne me gêne pas du tout, mais je me demande si cela ne vas poser des soucis de "purge" ultérieurement? Notamment comment savoir quoi purger, comment et quand?

Pour ce point c'est déjà bon, workflow_data est effacé lors de l'action d'anonymisation, nous avons seulement un problème du coté du stockage des fichiers.

#20 - 06 mai 2019 20:50 - Frédéric Péters

Il me semble que dans les derniers échanges il y introduction d'une dépendance à #32535, je viens de le regarder, je suis frileux.

En alternative moche, mais moins que :

```
{'varname': 'afile',
 'filename': ws1_response_afile_filename,
 'content_type': ws1_response_afile_content_type,
 'b64_content': ws1_response_afile_b64_content
}
```

il y aurait la création d'un utils.dict_from_prefix(' ws1_response_afile'), qui aurait l'avantage de ne prendre aucun risque sur le reste.

~~

À part ça, du dernier commentaire de Benjamin, je note juste qu'il reste celui-ci :

> en fait dans les deux cas qui nous intéressent où le fichier n'est que temporaire, on pourrait se contenter d'un Upload classique de quixote et pas d'un PicklableUpload

#21 - 15 mai 2019 14:40 - Michaël Bideau

- Fichier 0001-workflow-added-attachment-support-for-RegisterCommen.patch ajouté

Bonjour Frédéric, Benjamin et EO,

Je reviens vers vous avec une nouvelle version du patch W.C.S., cette fois-ci sans gestion du *varname* et avec le code utilitaire dans une fonction utils.dict_from_prefix().

Qu'en dites-vous?

Si vous le souhaitez je peux également essayer de le refactorer pour ne pas créer de PicklableUpload mais seulement un upload *quixote*, comme suggéré par Benjamin?

Merci d'avance pour votre retour.

Bien cordialement,

#22 - 15 mai 2019 14:46 - Michaël Bideau

- Fichier 0001-workflow-added-attachment-support-for-RegisterCommen.patch ajouté

Oops, petite erreur de fichier de patch, celui fourni contient quelques typos dans la descriptions de la fonction utils.dict_from_prefix(), corrigées dans le patch ci-joint.

#23 - 06 juin 2019 15:44 - Benjamin Dauvergne

Michaël Bideau a écrit :

> Si vous le souhaitez je peux également essayer de le refactorer pour ne pas créer de PicklableUpload mais seulement un upload *quixote*, comme suggéré par Benjamin?

Finalement ce serait trop compliqué, les fonctions autour des emails dépendent de PicklableUpload et ne peuvent pas prendre un Upload.

#24 - 06 juin 2019 19:23 - Benjamin Dauvergne

- Dans dict_from_prefix ne pas utiliser re et juste faire k[len(prefix):]
- pour

```
                # useless but required to restore upload.fp from serialized state, needed by 'AttachmentEv
olutionPart.from_upload()
                fp = upload.get_file_pointer()
```

> à confirmer par fred mais je pense qu'on peut remplacer upload.fp par upload.get_file_pointer() dans l'implémentation de from_upload() ou au moins faire un upload.fp or upload.get_file_pointer(); ainsi on évite ce commentaire

C'est Fred qui aura le dernier mot.

#25 - 09 juin 2019 17:47 - Frédéric Péters

- Statut changé de Nouveau à Résolu (à déployer)

J'ai rebasé, corrigé les conflits et appliqué une série de modifications de style, également revu le message de commit pour suivre nos normes, et poussé.

Au final je reste pas bien convaincu par tout ça, avec toujours l'impression qu'il aurait fallu remonter plus haut dans la discussion du besoin.

```
commit 8bf94c5e7a6904bf25c37c538445b8c0f90b64b5
Author: Michael Bideau <mbideau@atreal.fr>
Date:   Wed May 15 14:44:18 2019 +0200

    workflows: add possibility to attach files to notes in history (#32403)
```

#26 - 10 juin 2019 10:15 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

#27 - 03 juin 2021 16:31 - Nicolas Roche

- Lié à Development #54081: Filtrer les fichiers attachés dans l'action "Message dans l'historique" en fonction des rôles auquel le message s'applique ajouté

Fichiers

Screenshot-2019-4-16 Backoffice de Démarches - Workflow - Wk1.png	98,7 ko	16 avril 2019	Michaël Bideau
0001-workflow-added-attachments-for-RegisterComment.patch	25,3 ko	16 avril 2019	Michaël Bideau
0001-workflow-added-attachments-for-RegisterComment.patch	24,1 ko	18 avril 2019	Michaël Bideau
0001-workflow-added-attachments-for-RegisterComment.patch	22,9 ko	18 avril 2019	Michaël Bideau
0001-workflow-added-attachment-support-for-RegisterCommen.patch	19,9 ko	15 mai 2019	Michaël Bideau
0001-workflow-added-attachment-support-for-RegisterCommen.patch	19,8 ko	15 mai 2019	Michaël Bideau