

==== Documentation de l'Opener pour développeur ==== Cet article s'adresse à toute personne souhaitant étendre les possibilités de l'Opener. Pour votre bonne compréhension, il est nécessaire d'avoir des connaissances basiques en python. ==== Introduction ==== L'opener est un outil ouvert à tous ce qui signifie que vous pouvez l'étendre ou l'adapter selon vos besoins. Le code est structuré de manière à ce que vous puissiez vous repérer à l'aide de cet article. ----

==== Librairie nécessaire ==== L'addon utilise la librairie Naming qui est chargée de réaliser la nomenclature des différents fichiers/dossiers. Dans notre cas, l'addon l'implémente sous sa forme la plus simple possible. Cette dernière est disponible en téléchargement au lien suivant: [\[\[https://github.com/LesFeesSpeciales/lib.git\]\]](https://github.com/LesFeesSpeciales/lib.git) Dans le cas où vous souhaiteriez développer vos propres outils avec, un guide existe sur le wiki à l'adresse suivante: [\[\[kabaret.naming|Guide naming\]\]](#). ----

==== Structure des fichiers ==== Les fonctions et différentes parties de l'addon sont regroupées dans différents fichiers par catégories. Les différents modules sont les suivants: **__init__**: Gère l'initialisation de l'addon, c'est lui qui crée les variables de scène (stockant les chemins, séquences...). **__files__**: Gère l'interaction entre l'addon et les fichiers (Listage des dossiers, fichiers avec vérifications...). **__gui__**: Ce module est chargé de dessiner la GUI de l'addon dans les outils de Blender, si vous souhaitez modifier l'interface de l'addon, cela se passe dans la fonction **naming_panel** de ce module. **__interface__**: interface.py fait le pont entre Blender, le système de fichier et la librairie naming. C'est ses fonctions qui, avec Naming, génèrent les chemins. **__operators__**: Operator contient tout les opérateurs nécessaires à l'addon, parmi eux, certains ont pour fonction d'ouvrir simplement un explorateur de fichier (OBJECT_OT_custompath), etc... (Voir section opérateurs) **__persistence__**: Comme son nom l'indique, persistence permet de rendre certains champs de l'interface persistents, c'est-à-dire qu'il va stocker certaines valeurs dans des fichiers localement. **__ressources__**: ressources permet la mise en place de variable globale à tout les modules. ----

==== Fonctions clefs ==== Dans cette partie, nous allons éclaircir le fonctionnement de certaines fonctions clefs de l'addon. === a. **initSceneProperties** === C'est ici que se déroule l'enregistrement des variables de scènes, nécessaires à toutes sortes de choses. On retrouve par exemple tous les champs, que ce soit la séquence, l'asset ou autre sous forme de variable de scène ici. Cette fonction se divise en deux parties, l'instanciation de variables destinées à être **dynamiques** ou **statiques**. Ici nous suivons le code séquentiellement, et donc commencerons par la partie dynamique. Dans Blender, l'actualisation des propriétés de scènes EnumProperty pose toujours problème: il est impossible de les mettre à jour sans les supprimer et les recréer. Ainsi, pour mettre en place les Enum dynamique, j'ai simplement créé une fonction (UpdateEnum) dont l'unique utilité est de supprimer et de recréer les Enums passés en paramètre. Dans le cas de l'Opener, les propriétés dynamique servent à mémoriser les assets, séquences, shots, stores, ... ajoutés par l'utilisateur. L'exemple qui suit est juste une partie succincte du code de la fonction initSceneProperties concernant la création de la propriété stockant le(s) répertoires choisis par l'utilisateur.

```
<code python> #DYNAMIC ATTRIBUTES-----
-----> #Liste contenant toutes les propriétés dynamiques properties = [] #Valeurs par défaut pour la propriété Store (le répertoire racine) drives = (('Store','Store directory',''),('/u/Project','/u/Project/'),('test2_1','test2_2'),('','')) #Création de la sceneProperty vide bpy.types.Scene.drives = EnumProperty(name="none",description="none",items=('',),update=interface.update_naming) s = len(properties) #Ajout de la propriété pour l'enregistrement ressources.Items.append(('none','none','')) #Pour chaque propriétés for i in range(s): #Setup store dir if properties[i][0][0] == 'Store': #Dans le cas du store #Si il n'y a pas de fichier persistant déjà existant, on remplit avec les valeurs par défaut if not persistence.load_config(): for line in range(1,len(properties[i])): ressources.Items.append((str(properties[i][line][0]),str(properties[i][line][1]),str(properties[i][line][2]))) #On met à jour la variable grâce à la fonction UpdateEnum interface.UpdateEnum(bpy.types.Scene,ressources.Items,properties[i][0][0],properties[i][0][1],ressources.Items[0][0]) </code>
```

Ci-dessous, un exemple de déclaration de propriété de scène statique, pour

la propriété root. <code python> #STATICS ATTIBS-----> #ROOT-----
 -----> bpy.types.Scene.roots = EnumProperty(name="Root", description="root",
 items=(('LIB', "LIB", ""), ('MOVIE', "MOVIE", ""), ('', "", "")), default='LIB', update =
 interface.update_naming) </code> La partie intéressante est la ligne suivante: <code python>
 update = interface.update_naming </code> Cette dernière consiste à appeler la fonction
****update_naming(self,context)**** à chaque modification de la propriété_. Ce qui nous amène à
 cette fonction. **=== b. update_naming ===** <code python> #Copie du dictionnaire pour des
 vérificationq temp = ressources.path.copy() #Nettoyage du dictionnaire path ressources.path.clear()
 #Séparation des éléments en fonction de l'OS if sys.platform == 'win32': dest =
 bpy.context.scene.drives.split("\\") else: dest = bpy.context.scene.drives.split('/') #Mise à jour du
 dictionnaire if bpy.context.scene.roots == 'LIB': ressources.path['Lib'] = 'LIB'
 ressources.path['Family']=bpy.context.scene.famille #Si jamais l'asset sélectionné dans la liste est
 Other lors de la MAJ, on l'ajoute aux assets if bpy.context.scene.asset == 'other': #Construction du
 tuple temporaire pour vérifier son unicité dans la liste des asset tempAsset =
 (str(bpy.context.scene.newA),str(bpy.context.scene.newA),") #Si il n'existe pas alors on l'ajoute if
 tempAsset not in ressources.Items_asset: #On récupère le nouvel asset à ajouter dans le dictionnaire
 ressources.path['Asset']=bpy.context.scene.newA #On ajoute cet asset à la liste
 ressources.Items_asset.append((str(bpy.context.scene.newA),str(bpy.context.scene.newA),"))
 UpdateEnum("",ressources.Items_asset,'asset',"") bpy.context.scene.asset = bpy.context.scene.newA
 else: bpy.context.scene.newA = "" else: ressources.path['Asset']=bpy.context.scene.asset
 #.....Quelques lignes plus tard..... #Si jamais lors de l'exécution de la fonction, on se
 trouve dans la fenêtre MOVIE alors: elif (bpy.context.scene.roots == 'MOVIE') and
 (bpy.context.scene.drives != 'none') and (temp['Dept'] == ressources.path['Dept']) and ('Shot' in
 temp) and ('Sequence' in temp): temps = persistence.load_seq() for i in range(len(temps)): y =
 (str(temps[i]),str(temps[i]),") if y not in ressources.Items_seq:
 ressources.Items_seq.append((str(temps[i]),str(temps[i]),")) change = True #Si il y a de nouvelles
 séquences, on les ajoute if change: UpdateEnum("",ressources.Items_seq,'seq',"none")
 bpy.context.scene.shot = ressources.Items_shot[0][0] elif temp['Shot'] == bpy.context.scene.shot:
 temps = persistence.load_shots() ressources.Items_shot.clear()
 ressources.Items_shot.append(('none','none',"")) for i in range(len(temps)): y =
 (str(temps[i]),str(temps[i]),") if y not in ressources.Items_shot:
 ressources.Items_shot.append((str(temps[i]),str(temps[i]),"))
 UpdateEnum("",ressources.Items_shot,'shot',"none") #.....Quelques lignes plus tard.....
 try: bpy.context.scene.newF = create_naming(self,context,"",ressources.path,ressources.command)
 files.Update_ListFile(bpy.context.scene.newF) except: print('naming no setup, clearing list')
 bpy.context.scene.custom.clear() #Clearing scene ressources.command.append("! missing field !")
 </code> ---- <WRAP center round important 60%> En construction </WRAP>

From:

<https://les-fees-speciales.coop/wiki/> - **Les Fées Spéciales**

Permanent link:

https://les-fees-speciales.coop/wiki/doc_opener?rev=1439997527Last update: **2015/08/19 16:18**