Workflow vers Process

Introduction

Dans le cadre de la suite Efalia, il est possible de lancer des workflows Efalia Process depuis MG9 à la place de docflow purement MG9.

Le concept est d'initier un docflow MG9 suivant la méthode habituelle, mais celui-ci débranche immédiatement vers Process suivant la méthode d'un lancement de connecteur standard. Il faut donc paramétrer l'appel vers Efalia Process comme un connecteur.

Le reste imite l'utilisation des connecteurs MG9 actuels. Le moteur de MG9 appelle l'API d'Efalia Process régulièrement pour vérifier l'état d'avancement du workflow en l'affichant dans MG9.

Lorsque le workflow atteint l'un de ses termes possibles, le connecteur MG9 termine l'étape et libère le document une fois le docflow MG9 terminé.

Conseil

L'objectif est de limiter le docflow MG9 à la seule étape d'envoi à Efalia Process.

Prérequis

Côté Efalia Process

- Les versions d'**Efalia Process à partir de la 6.24.0** vérifient l'intégrité du JSON (facultatif) grâce à un composant dédié
- Un gestionnaire de workflow qui puisse servir de compte de service pour MG9

Côté MG9

- Une installation sur un **build 9.6.2 SVN 17302 (SP8)**

Paramétrage du connecteur Efalia Process dans MG9

Dans le répertoire filesystem.installation.multigest/bin/mgworkey, ajouter le fichier de configuration (config.ini) suivant :

- URL de l'instance d'Efalia Process
- Identifiants d'un gestionnaire de workflow Efalia Process (utilisé par

défaut)

```
[SERVICE]
BASEURI=https://installation.workey.extension/workey
URLAPI=api
LOGIN=<login du gestionnaire de workflow côté Efalia Process>
PASSWORD=<mot de passe du gestionnaire de workflow côté Efalia Process>
AUTHTYPE=0
SSLVERIFYPEER=0
TIMEOUT=600
SESSIONTIMEOUT=600
TEMPLATE=MG9_JSON

[TRACE]
LEVEL=7
LOGS=D:\Deve\www\multigest\bin\mgworkey\logs
```

Paramétrage du docflow MG9

Lancer un workflow sur Efalia Process se fait comme un lancement de connecteur standard.

On définit :

- Un nom de processus côté *MG9*
- Le nom du processus côté *Process*
- Le nom du document (instance de workflow) à générer côté Process
- Le rôle pour la création du workflow côté *Process*
- Le nom interne du champ input côté *Process*
- Un flag si on lance le workflow en tant que l'acteur connecté ou pas
    - Si on lance le workflow pour l'acteur connecté, l'url du POST est du type https://urlProcess/workey/api/document/<idDuProcess>/<idDuDoc>/<rôle>?output=js&actor=<loginActeur>
    - Si on lance le workflow en tant que Gestionnaire de workflow, le document doit être créé côté Workey à un nouvel état à paramétrer dans le lancement de workflow. Le paramètre est l'état du workflow à flagger à true dans le template de workflow. L'url du POST est alors du type https://urlProcess/workey/api/document/<idDuProcess>/<idDuDoc>/<rôle>?output=js

Modélisation côté Efalia Process

Dans le designer

MG9 envoie à Efalia Process un payload sous la forme d’un json stringifié placé dans un champ décidé en amont (celui paramétré côté MG9).

Le json est constitué sous la forme d’un objet contenant une liste de documents et une liste de fichiers potentiellement liés aux documents :


```
{
    "documents": [
        {
                "id": "123456",
                "name": "nomDoc.doc",
                "mimetype": "application/msword",
                "createdOn": "2023-06-22T12:00:01+02:03",
                "updatedOn": "2023-06-22T12:00:01+02:03",
        "template": {
                        "folder": "Annexe",
                        "subfolder": "Personnelle",
                        "document": "Personnelle",
                }
                "filingCabinet": "Ressource Humaine",
                "folderId": "1337",
                "sheet": "nom_fiche_métadonnée",
                "metadata": [
                {
                    "name": "nom",
                    "type": "string",
                    "length": 50,
                    "value": "Dupond"
                },
                {
                    "name": "prenom",
                    "type": "string",
                    "length": 40,
                    "value": "Martin"
                },
                {
                    "name": "date_naissance",
                    "type": "date",
                    "length": 10,
                    "value": "1991-10-01T01:02:03+04:05"
                }
            ],
            "versions": [
                {
                    "id": 1,
                    "mimetype": "application/msword",
                    "createdOn": "2023-06-22T12:00:01+02:03"
```

```
                }
            ]
        }
    ],
    "folders": [
        {
            "folderId": "1337",
            "name": "Martin Dupond - Fondeur",

            "filingCabinet": "Ressource Humaine",

            "metadata": [
                {
                    "name": "NomEtPrenom",
                    "type": "string",
                    "length": 100,
                    "value": "Martin Dupond"
                },
                {
                    "name": "matricule",
                    "type": "number",
                    "value": 1337
                },
                {
                    "name": "nb_dents",
                    "type": "number",
                    "value": 31
                }
            ],
        }
    ]
}
```

L’intérêt de ce payload est de fournir toutes les informations nécessaires pour effectuer deux actions :
– Peupler le composant de visionnage des fichiers GED
– Utiliser les métadonnées pour enrichir les données présentes dans le workflow

Information

Pour l’heure, il n’existe pas de fonction WKYJS pour faciliter la tâche et la gestion du payload se fait en javascript.

Exemple de traitement de payload pour peupler des composants Process :


```
WKYJS.onLoad(function() {
    let inputMG9 = WKYJS.getFieldValue("Input_MG9");
    if (inputMG9 !== null && inputMG9 !== ""){
          //Préparation du découpage du payload
          inputMG9 = JSON.parse(inputMG9);
          //Répartition des données du payload dans le formulaire Process
      if (inputMG9.folders != undefined)
      {
        //**********  PARTIE A MODIFIER **********//
        //**********       DEBUT       **********//
        //Récupération des métadonnées TECHNIQUES du document pour peupler
des champs Process
        //On peuple le champ Process "idDoc" avec la valeur de la métadonnée
technique du fichier "id" du payload MG9
                setInFieldMetadataFileTech ("idDoc","id");
        //On peuple le champ Process "Montant" avec la valeur de la
métadonnée technique du fichier "template" du payload MG9
                //A noter qu'il s'agit d'un objet json constitué de folder,
subfolder et file.
                setInFieldMetadataFileTech ("gabaritDoc","template");
        //Récupération des métadonnées METIER du document pour peupler des
champs Process
        //On peuple le champ Process "ID_piece" avec la valeur de la
métadonnée du fichier "ID_PIECE" du payload MG9
                setInFieldMetadataFileFonc ("ID_piece","ID_PIECE");
        //On peuple le champ Process "Montant" avec la valeur de la
métadonnée du fichier "MONTANT" du payload MG9
```

```
                setInFieldMetadataFileFonc ("Montant","MONTANT");
        //Récupération des métadonnées TECHNIQUES du dossier lié au document
pour peupler des champs Process
        //On peuple le champ Process "idDossier" avec la valeur de la
métadonnée technique (du dossier lié au document) "id" du payload MG9
                setInFieldMetadataFolderTech ("idDossier","id");
        //On peuple le champ Process "filingCabinet" avec la valeur de la
métadonnée technique (du dossier lié au document) "filingCabinet" du payload
MG9
                setInFieldMetadataFolderTech
("filingCabinet","filingCabinet");
        //Récupération des métadonnées METIER du dossier lié au document pour
peupler des champs Process
        //On peuple le champ Process "Nom_demandeur" avec la valeur de la
métadonnée du dossier "NOM" du payload MG9
                setInFieldMetadataFolderFonc ("Nom_demandeur","NOM");
        //On peuple le champ Process "date_naissance" avec la valeur de la
métadonnée du dossier "DATE_NAISSANCE" du payload
                setInFieldMetadataFolderFonc
("date_naissance","DATE_NAISSANCE");
        //Peuplement d'un composant pièces-jointes GED (MGDoc pas MGFolder)
pour visualiser les documents depuis un formulaire Process
        //Le composant "document MG" aura ici pour nom interne "Document_MG"
                chargePreview("Document_MG")
        //**********  PARTIE A MODIFIER **********//
        //**********        FIN        **********//
        }else{WKYJS.errorBox("Erreur de chargement, je ne retrouve pas le
dossier. Désolé !");}
    }else{WKYJS.errorBox("Je ne détecte aucun fichier en entrée du processus.
Est-ce bien normal ?");}
        WKYJS.refresh();
        //Fonctions utilisées, ne pas modifier
        //Fonction pour récupérer les métadonnées techniques du fichier (id,
nom, suffix, date de maj, reference...) et les mettre dans un champ Process
        function setInFieldMetadataFileTech
(ProcessFieldName,inputMetadataName){
        if(inputMG9.documents[0][inputMetadataName] != undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.documents[0][inputMetadataName]
);
                }else{
                        console.log("Métadonnée technique du fichier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.documents[0]);
                }
        }
        //Fonction pour récupérer les métadonnées métier du fichier (définies
par les utilisateurs dans le plan de classement) et les mettre dans un champ
Process
        function setInFieldMetadataFileFonc
(ProcessFieldName,inputMetadataName){
        if(inputMG9.documents[0].metadata.length > 0 &&
```

```
inputMG9.documents[0].metadata.find(item => item.name === inputMetadataName)
!= undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.documents[0].metadata.find(item
=> item.name === inputMetadataName).value);
                }else{
                        console.log("Métadonnée fonctionnelle du fichier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.documents[0]);
                }
        }
        //Fonction pour récupérer les métadonnées template du fichier
(définies par les utilisateurs dans le plan de classement) et les mettre dans
un champ Process
        function setInFieldMetadataFileTemplate
(ProcessFieldName,inputMetadataName){
        if(inputMG9.documents[0].template != undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.documents[0].template[inputMeta
dataName]);
                }else{
                        console.log("Métadonnée template du fichier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.documents[0]);
                }
        }
        //Fonction pour récupérer les métadonnées techniques du dossier
rattaché au fichier (id, nom, filingCabinet, comments) et les mettre dans un
champ Process
        function setInFieldMetadataFolderTech
(ProcessFieldName,inputMetadataName){
        if(inputMG9.folders[0][inputMetadataName] != undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.folders[0][inputMetadataName]);
                }else{
                        console.log("Métadonnée technique du dossier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.folders[0]);
                }
        }
        //Fonction pour récupérer les métadonnées métier du dossier rattaché
au fichier (définies par les utilisateurs dans le plan de classement) et les
mettre dans un champ Process
        function setInFieldMetadataFolderFonc
(ProcessFieldName,inputMetadataName){
        if(inputMG9.folders[0].metadata.length > 0 &&
inputMG9.folders[0].metadata.find(item => item.name === inputMetadataName) !=
undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.folders[0].metadata.find(item
=> item.name === inputMetadataName).value);
                }else{
                        console.log("Métadonnée fonctionnelle du dossier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.folders[0]);
```

```
                }
    }
        //Fonction pour récupérer les métadonnées template du dossier
rattaché au fichier (id, nom, filingCabinet, comments) et les mettre dans un
champ Process
        function setInFieldMetadataFolderTemplate
(ProcessFieldName,inputMetadataName){
        if(inputMG9.folders[0].template != undefined){
WKYJS.setFieldValue(ProcessFieldName,inputMG9.folders[0].template[inputMetada
taName]);
                }else{
                        console.log("Métadonnée template du dossier non
trouvée : " + inputMetadataName);
                        console.log(inputMG9.folders[0]);
                }
        }
        //Nécessite le composant MGDoc de visualisation de fichier MG9
(différent de MGFolder)
        function chargePreview(previewField){
       WKYJS.setFieldValue(
        previewField,
        inputMG9.documents.map((i) => ({
            id: i.id,
            name: i.name,
            mimetype: i.mimetype,
            updatedOn: i.updatedOn,
            folder: i.template.folder,
            subfolder: i.template.subfolder,
            filingCabinet: i.filingCabinet,
       })));
    }
});
```

Dans l’administration

Quel que soit le mode de lancement du workflow, il faut utiliser un gestionnaire de workflow paramétré côté MG9 pour lancer les workflows, soit en son compte, soit au compte de l’utilisateur connecté – le pivot est alors l’identifiant (login) de l’utilisateur connecté qui doit être identique dans les deux solutions – soit au compte d’un utilisateur paramétré en dur.
Dans le cas où le workflow est lancé directement pour le compte du gestionnaire de workflow, celui-ci doit être dans le premier rôle du workflow puisqu’il va ouvrir une instance de ce dernier. Le document passe alors automatiquement la première opération qui ne doit être qu’une opération d’initialisation du workflow afin d’être pris en charge par un premier acteur physique.

Attention

Dans le cas d’un lancement de workflow par compte de service et non par

impersonnation, la gestion du payload ne se fera pas lors de la première opération de création.
Elle se fait côté navigateur donc le moteur ne peut pas la gérer. Il ne faut donc pas prévoir de mécanique utilisant le payload lors de cette étape de création.
En revanche, la gestion du payload pourra s’opérer lors de la première action faite par un utilisateur physique.