Processus et automatisations

ELO Automation Services

Table des matières

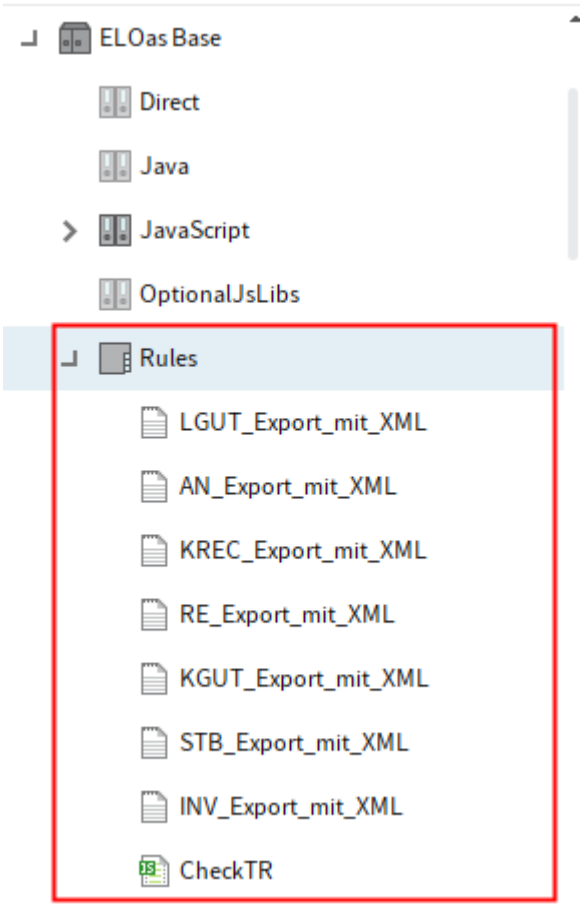
Autres thèmes 121

Bases

Fonctionnement de ELOas

ELOas est un servlet pouvant retravailler des documents ELO dans un processus opérant à l'arrière-plan. Par exemple: l'entrée ultérieure des données d'indexation à partir d'autres sources de données, le déplacement de documents ou la mise en place de structures de dépôt. En raison de la grande flexibilité, l'interface JavaScript intégrée est en mesure de créer de nombreuses autres fonctions.

Un ruleset forme la base du traitement. Il se compose d'une configuration XML, qui a été créée avec une interface utilisateur graphique depuis la console d'administration. Il est possible de définir plusieurs rulesets, ceux-ci sont alors exécutés de façon consécutive avec une commande par intervalles ("toutes les dix minutes", "une fois par jour à 13 heures"). Le ruleset contient par ailleurs une demande et une suite de règles permettant de traiter les données.



Illustr. : classeur Rules dans ELO

ELOas active chaque ruleset de sa liste placé sous "\ELOas\Rules". Tout d'abord, une vérification est effectuée pour chaque ruleset, à savoir si la condition d'intervalle est remplie (par exemple: 10 minutes sont-elles passées depuis le dernier parcours?). Si la condition n'est pas encore remplie, c'est le prochain ruleset qui est traité. Mais si au contraire il s'agit d'exécuter la tâche au moment même, la recherche spécifique est effectuée. A partir de la liste des occurrences, la liste des règles est traitée pour chaque entrée. Ici, il est possible de modifier la cible dans ELO,

de compléter les métadonnées ou d'effectuer d'autres actions. Ensuite, le document est enregistré et l'entrée suivante est traitée jusqu'à ce que la fin de la liste des occurrences soit atteinte. Finalement, le nouveau point d'exécution est calculé et le traitement avec le prochain ruleset est poursuivi.

Il est très simple d'ajouter d'autres rulesets par le biais de l'interface utilisateur graphique. Toutefois, ils ne seront actifs qu'après le chargement de la configuration, tout comme les rulesets modifiés.

En alternative, la configuration XML des règles et le code JavaScript peuvent également être enregistrés dans un fichier document, à la place d'un enregistrement dans le texte complémentaire. L'on voit alors dans la structure des fichiers texte à la place de classeurs pour les sous-entrées.

Pour ELOwf, un autre typ de ruleset a été introduit: l'appel direct. Ces rulesets sont créés dans un propre classeur *Direct*, tournent dans leur propre thread et livrent directement un résultat. Pour cette raison, ils ne peuvent pas être définis avec un intervalle, mais doivent être créés en tant que déclencheurs (pour 0 minutes: utiliser 0M en tant qu'intervalle). Par ailleurs, il faut veiller à ce que seules des actions courtes soient exécutées ici, étant donné que le processus doit attendre le résultat et qu'il l'interrompt après un certain temps.

Méthodes de recherche (recherche d'indexation, treewalk, liste de tâches, boîte de réception, chrono-timbre)

En premier lieu, ELOas a été conçu pour traiter une liste de résultats, résultant d'une recherche d'indexation. Au fil du temps, d'autres options ont été ajoutées; celles-ci peuvent être sélectionnées par une dénomination correcte de SEARCHNAME:

TREEWALK: dans SEARCHVALUE, l'id d'objet ou ARCPATH sont déposés auprès de l'objet de démarrage. Toute l'arborescence est parcourue et le ruleset est appelé pour chaque entrée avec le masque approprié.

WORKFLOW: tous les délais de processus de l'utilisateur ELOas sont lus et le ruleset pour chaque entrée avec le masque approprié est appelé. Dans le ruleset, il est possible de déclencher un transfert du processus.

MAILBOX_<Nom du profil>: une connexion au serveur mail est établie avec le nom de profil et le contenu d'une boîte de réception est entré et traité. Le ruleset est appelé pour chaque mail se trouvant dans la boîte de réception avec un document vide.

DIRECT: Ce ruleset peut être appelé avec `http-get` et livre directement un résultat. Les rulesets de ce type peuvent seulement être définis dans le classeur *Direct* et non pas dans *Rules*, étant donné qu'ils doivent être exécutés dans un autre thread.

TIMESTAMP: cet appel exécute une recherche de la dernière modification. En règle générale, l'on entre une section comme terme de recherche : "2012.01.01.00.0.00… 2012.01.31.23.59.59".

Rulesets

Bases

Les règles sont exécutées par le programme dans des intervalles réguliers, pour effectuer des tâches définies dans ELO.

Les règles sont écrites dans JavaScrpit et seront exécutées par le serveur d'indexation dans un laps de temps défini.

Traitement des règles

1. Le programme recherche tout d'abord des objets dans ELO, qui sont concernés par la définition de la règle.
2. La règle sera appliquée à tous les éléments trouvés dans ELO.

> **Information**
>
> Les réglages concernant les documents à rechercher seront définis dans les champs de recherche.

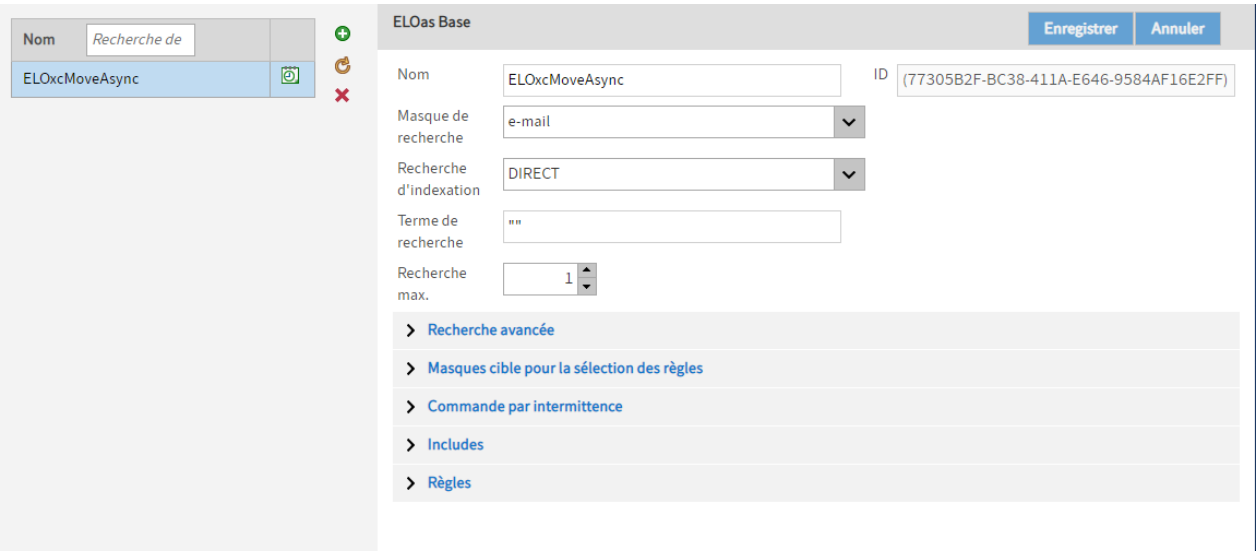
1. Si une règle n'est pas valable, elle ne peut ni être utilisée, ni appliquée. Un message d'erreur apparaît.

   Les règles définies sont déposées dans ELO dans la section *Administration* > *Rules*.

> **Information**
>
> Dans la console d'administration ELO, les règles sont divisées en deux parties. Sous *Règles*, vous trouverez la boîte *Règle 1*, avec une description des étapes à exécuter. Elle contient des informations à propos de ce qui doit être fait de chaque objet trouvé. Dans la boîte *Global Error Rule* est déterminé ce qui doit se passer lorsqu'une erreur survient.

Créer un ruleset via la console d'administration ELO

Vous pouvez créer de nouveaux rulesets dans la section *ELO Automation Services* de la console d'administration.



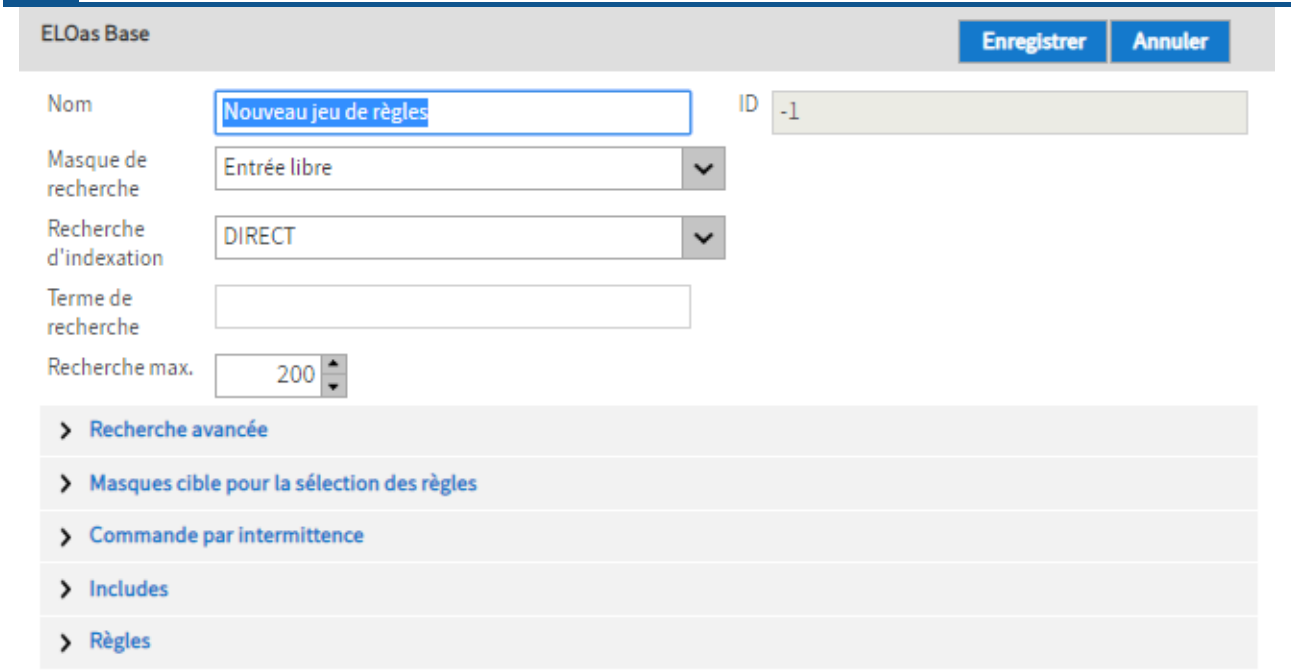
Illustr. : point de menu 'ELO Automation Services'

Ajouter (symbole + vert) : le bouton *Ajouter* vous permet de créer un nouveau jeu de règles.

> **Information**
>
> Ce n'est qu'après l'enregistrement que la règle de traitement nouvellement créée sera enregistrée dans ELO dans le classeur *Rules*. Vous pouvez configurer les règles de traitement dans la section *Règles*.

Récupérer à nouveau les données du serveur (symbole avec deux flèches rondes jaunes) : le bouton *Récupérer à nouveau les données du serveur* permet d'actualiser la section.

Supprimer (symbole X rouge) : le bouton *Supprimer* vous permet de supprimer le jeu de règles sélectionné.



Illustr. : nouveau ruleset

Nom : c'est ici que le nom de la règle de traitement doit être saisi, celui que vous aviez choisi lors de la création de la règle. Le nom pourra être modifié ultérieurement.

> **Remarque**
>
> Certains caractères ne sont pas permis. Veuillez prendre en compte la liste ci-dessous.

- ¶
- "
- /
- \
- :
- ;
- ,

Masque de recherche : sélectionnez le masque de recherche, qui doit être utilisé pour les documents à modifier.

Recherche d'indexation : sélectionnez le champ de groupe afin d'effectuer une recherche dans différents champs.

Terme de recherche : saisissez ici la chaîne de caractères qui doit être recherchée. Tous les documents dans ELO correspondants aux règles définies ici seront sélectionnés pour être modifiés selon les règles et les critères définis dans l'assistant. La chaîne de caractères devrait être saisie entre guillemets.

Recherche max : veuillez définir le nombre de résultats max.

Recherche avancée

Date de dépôt de...à : vous pouvez limiter la recherche à la sélection d'une date de dépôt précise ou à une période de dépôt précise.

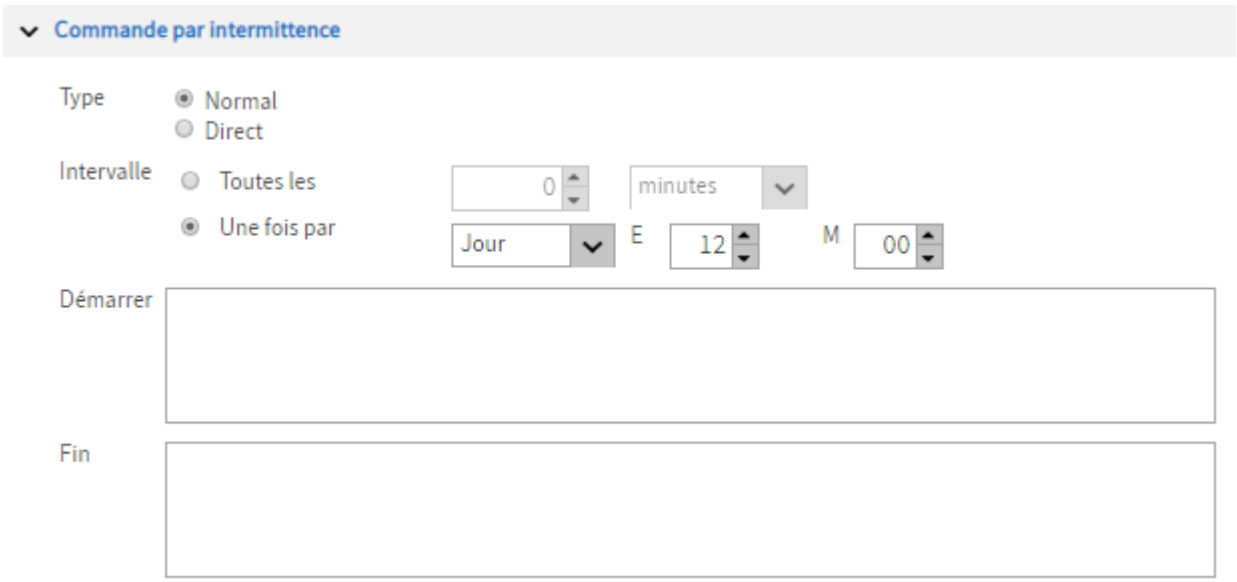
Date de...à : vous pouvez limiter la recherche à la sélection d'une date de dépôt précise ou à une période de dépôt précise.

Masques cibles pour la sélection des règles

Ajouter des masques cibles : ici vous pouvez sélectionner le masque pour le classeur cible, où les documents déplacés devront être archivés.

Commande par intermittence

Dans la sous-section *Commande par intermittence*, vous pouvez déterminer l'intervalle pour l'exécution de ELO Automation Services.



Illustr. : section 'Commande par intermittence'

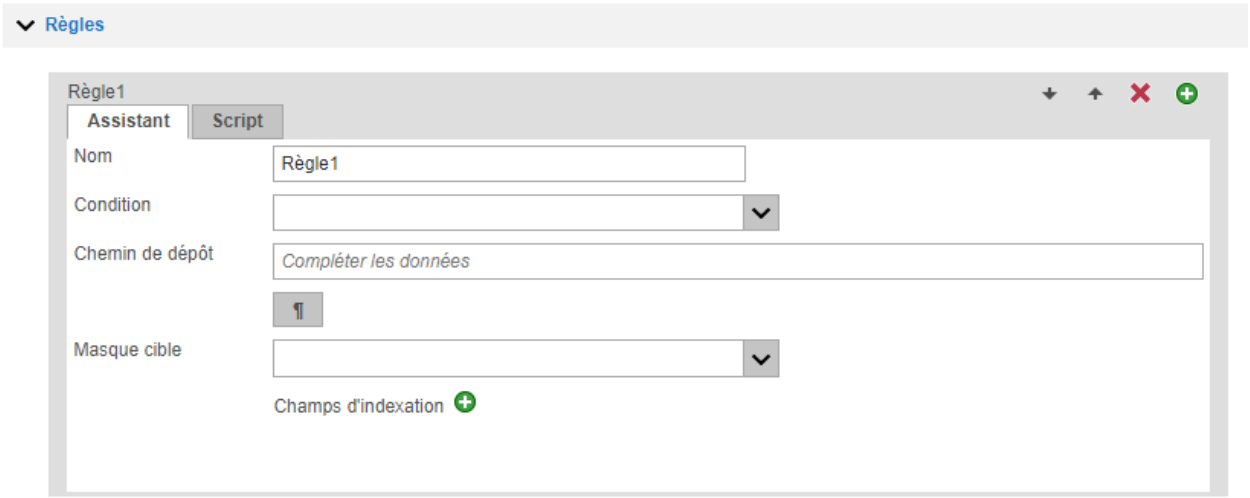
Démarrage : le champ de saisie contient le code de script qui sera exécuté avant l'exécution des règles de modification.

Fin : ce champ de saisie contient des codes de scripts qui seront exécutés après l'exécution des règles.

Includes

Ajouter une bibliothèque d'includes : le champ *Ajouter une bibliothèque d'includes* vous permet d'ajouter d'éventuelles bibliothèques de script requises à la règle ELOas.

Règles



Illustr. : section 'Règles'

Cette règle définie sera appliquée aux entrées trouvées dans ELO utilisant des réglages de base. Ici sera par ex. défini l'endroit où devra être déplacé le document.

Ajouter (symbole + vert) : vous pouvez ajouter une autre règle. Les règles sont travaillées du haut vers le bas.

Nom : ici, vous pouvez donner un nom à la règle.

Condition: ici, une règle d'interrogation sera définie pour, par exemple, vérifier le statut d'un champ.

Vous pouvez également appeler un script ici, avec lequel vous pouvez par exemple déplacer les fichiers dans le système de fichiers. Les scripts seront déposés dans ELO dans la section d'administration des ELO Automation Services.

> **Remarque**
>
> Cette configuration peut différer de celle de votre archive.

Chemin de dépôt : déterminez ici le classeur cible dans lequel le document doit être déplacé. En utilisant le bouton à la fin du champ de saisie, vous pouvez saisir le caractère de séparation pour les chemins dans ELO.

Masque cible : sélectionnez un masque pour le document.

Champs : ici vous pouvez remplacer le contenu des champs dans les métadonnées des documents cibles.

> **Information**
>
> Si vous basculez vers l'onglet *Script*, vous pouvez définir tous les réglages par le biais d'un script. Vous ne pourrez plus accéder à l'onglet *Rule*.

Traitement des erreurs

Dans le dialogue *Global Error Rule*, vous pouvez définir la manière dont doivent être traitées les erreurs par défaut. Cette règle est exécutée lorsqu'une erreur apparaît

Illustr. : définir le traitement des erreurs

Nom : le nom de la routine de traitement des erreurs est saisi ici.

Condition : les conditions concernant les règles d'erreurs sont définies ici.

Chemin de dépôt : le chemin de dépôt du rapport d'erreur dans l'archive est défini ici.

Masque cible : ici, vous définissez le masque pour le document avec le rapport d'erreur.

Champs : ici, vous pouvez définir des champs individuels, qui seront remplis avec une chaîne de caractères spécifique lors du dépôt.

> **Remarque**
>
> Si une règle de modification est invalide, elle ne pourra pas être enregistrée. La validité d'une règle de traitement est vérifiée lors de l'enregistrement.

Options et traitement des erreurs

Mettre les règles en mode pause

L'exécution de certaines règles (Rulesets) peut être arrêtée ou redémarrée grâce au lien Démarrer/Arrêter.

La page de statut de ELO Automation Service peut être trouvée par le biais du Manager du serveur d'application ELO correspondant ou par le biais de l'URL avec le schéma suivant :

```
http(s)://<nom de serveur>:<Port>/as->nom d'archive>/?cmd=statut
```

Illustr. : désactiver la règle de traitement par le biais du lien Stop.

Lorsque l'utilisateur clique sur *Stop*, alors le processus en cours s'arrête avec la règle correspondante. Cliquez sur le bouton *Démarrer* pour relancer le processus arrêté.

> **Remarque**
>
> L'affichage des statuts est mis à jour automatiquement toutes les 10 secondes. Cliquez sur *Start* ou sur *Stop* et cette commande sera renvoyée toutes les 10 secondes. Normalement, cela ne devrait pas créer de dysfonctionnement. Si plusieurs fenêtres de navigateur sont ouvertes en même temps pour la même page de statut, des réglages différents dans les différentes fenêtres peuvent aboutir à un comportement inattendu.

Message d'erreur "Invalid Ruleset"

Problème 1

Les services d'automation ELO (ELOas) fonctionnent, mais le message d'erreur est affiché dans le fichier de rapport: Invalid ruleset definition: Premature end of file.

Bien que ces règles soient définies correctement (orthographe correcte, pas d'erreur lors du parsing du document xml dans le navigateur), le fichier ne peut pas être lu par ELOas. Cela signifie que la demande a lieu, mais qu'un document vide est appelé (lignes d'erreur dans le fichier de rapport) et un message d'erreur apparaît. Vérifiez si les autorisations pour l'utilisateur ELOas sont suffisantes. Vérifiez si le serveur d'indexation et le gestionnaire de document de l'archive pertinente crée des messages d'erreur dans vos fichiers de rapport. Eventuellement, utilisez un autre client à des fins de test. Utilisez éventuellement un autre client à des fins de test, ou enregistrez le script à la place d'un fichier txt dans le classeur *Rules* comme propre classeur sous *Rules*, de telle sorte que le script puisse être utilisé comme entrée de base de données.

Une autre solution: Vérifiez que la version la plus récente de ELOas est utilisée, et que l'entrée suivante existe sous *<chemin d'installation?>config\as-<nom d'archive>* dans le fichier xml :

```
<entry key="tempdir"> ... </entry>
```

Ce répertoire temporaire doit exister, sans quoi le message d'erreur apparaît. L'utilisateur doit avoir écrit l'accès au niveau du système pour le répertoire temporaire.

Problème 2

Lors de la création d'une seconde règle, chaque champ de groupe n'est pas disponible dans le menu déroulant la première fois.

Dans ce cas, veuillez redémarrer la console d'administration ELO. Attention : ceci n'est pas valable pour la première règle créée !

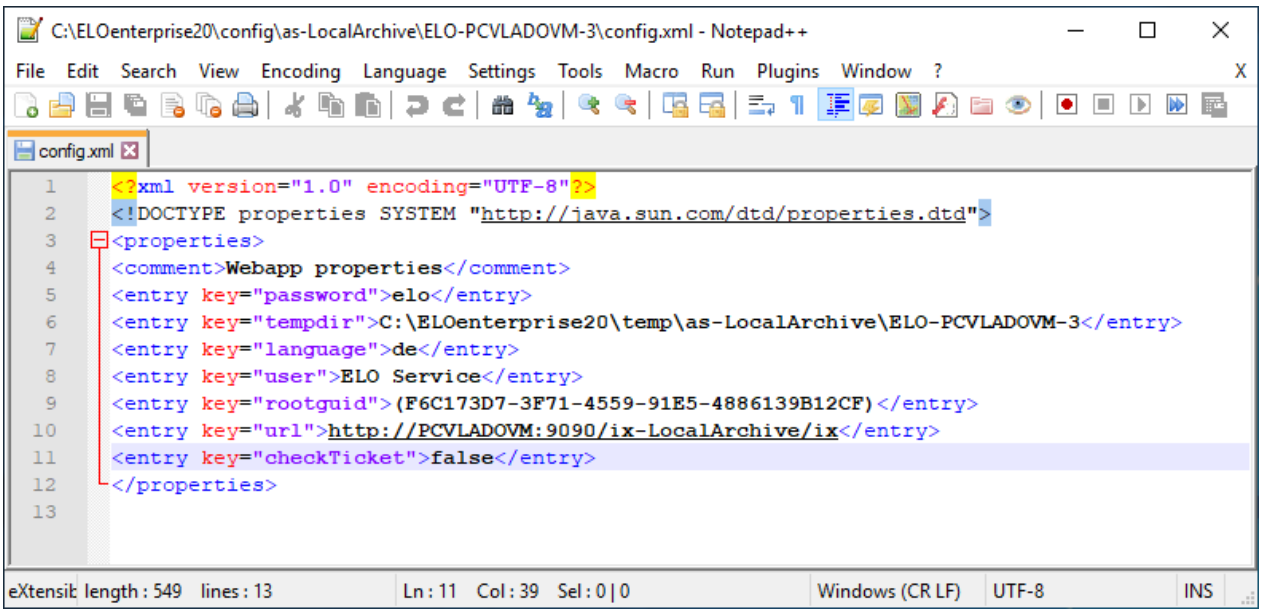
Démarrage manuel d'un ruleset

Normalement, ELOas exécute les rulesets définis dans des intervalles définis. Il existe également des processus qui sont si "volumineux" qu'ils ne peuvent pas être exécutés dans des intervalles courts, mais qui doivent être actifs rapidement après certaines modifications. Il existe la possibilité d'exécuter un ruleset manuellement par un URL (ou par script).

L'exécution de la règle par le biais d'une commande "HTTP-GET" ou "HTTP-RUN" est sécurisée par défaut dans ELOas 20 avec un ticket. Afin que la règle ELOas soit exécutée, un ticket valide doit être ajouté à l'URL ELOas correspondante.

http://localhost:9060/ELOas/actions?cmd=get&name=test&ticket=935666A2E27D8AB642C4C40AFAEAE2B9

La vérification de ticket interne peut être désactivée par le biais d'un nouveau paramètre de configuration ELOas du nom `checkTicket`.



Illustr. : config.xml

> **Attention**
>
> L'utilisation de ELOas avec une vérification de ticket désactivée au mode proxy présente un risque en termes de sécurité du système. Surtout, si le serveur d'indexation ELO est disponible sur Internet.

Exemple

L'exemple suivant montre comment l'on peut appeler un ruleset depuis un script client, qui pourra modifier certains objets.

Attention : étant donné que l'appel se fait par le biais d'un accès http, chaque utilisateur peut déclencher cette action avec un navigateur ou avec la commande d'un script. Vous devez donc

vous assurer que la fonction ne puisse pas être exécutée à mauvais escient (par exemple par le contrôle du numéro utilisateur ou par une directive interne fixe de l'objet ID).

Tout d'abord, il s'agit de prendre en considération le ruleset utilisé. En entrant un intervalle de 0 minutes (`<interval>0H</interval>`), ce ruleset est défini manuellement. L'appel n'est donc pas cyclique, il attend la réception d'une URL bien précise.

```
<ruleset>
    <base>
        <name>Expand Name</name>
        <search>
            <name>"OBJIDS"</name>
            <value></value>
            <mask>2</mask>
            <max>200</max>
        </search>
        <interval>0H</interval>
    </base>
<rule>
    <name>Expand Name</name>
    <condition></condition>
    <script>
        log.debug("Param1: " + EM_PARAM1);


        log.debug("UserId: " + EM_USERID);


        NAME = "Freigegeben: " + NAME;


        EM_WRITE_CHANGED = true;

    </script>
</rule>
<rule>
    <name>Global Error Rule</name>
    <condition>OnError</condition>
    <script></script>
</rule>
</ruleset>
```

La partie intéressante se trouve dans la section des scripts.

```
<script>
    log.debug("Param1: " + EM_PARAM1);


    log.debug("UserId: " + EM_USERID);
```

L'appel peut donner jusqu'à trois paramètres. Ceux-ci peuvent être demandés par les variables `EM_PARAM1`, `EM_PARAM2` et `EM_PARAM3`. Par ailleurs, le script peut donner le ticket de l'authentification actuelle pour la nouvelle authentification. Dans ce cas, la variable `EM_USERID` est remplie avec le numéro de l'utilisateur authentifié. S'il n'y a pas d'authentification, le numéro utilisateur est occupé avec -1. Dans le premier paramètre, l'on peut transmettre un ou deux identificateurs d'objets, ceux-ci écrasent alors la valeur Search provenant de la définition des rulesets. Dans ce cas, le nom du champ des métadonnées doit être "OBJIDS".

```
NAME = "Approuvé: " + NAME;
```

Dans notre exemple, le texte "autorisé" est placé devant la désignation des objets sélectionnés. Mais il est également possible d'effectuer d'autres modifications de l'objet SORD.

```
EM_WRITE_CHANGED = true;
```

Etant donné que l'objet a été modifié, il doit également être enregistré.

```
</script>
```

Activation du ruleset

Une fois ELOas démarré, ce ruleset est également démarré, mais il n'est pas encore actif. Il attend un déclencheur externe (visible au texte "trigger" dans le champ *Next run*).

Illustr. : page de statut ELOas

Le déclenchement peut être fait par l'appel d'une URL, ou encore depuis le client Windows par une commande de script (à partir de la version de client 7.00.056):

```
SendELOasRequest( <nom du serveur>, <numéro du port>, <nom de service>, <avec ticket>, <nom d
```

La commande `SendELOasRequest` effectue un appel asynchrone avec `run`. Ce ruleset est affiché dans la console d'administration ELO, non pas sous *Direct*, mais sous *Règles*.

Nom de serveur	Nom ou adresse IP du serveur ELOas.
Numéro de port	Numéro de port du serveur ELOas. Normalement, 8080, port http standard.
Nom de service	Nom de service du serveur ELOas. Dans une installation standard, il se compose du préfixe as- et du nom d'archive (par exemple as-ELO). Il faut absolument faire attention aux minuscules et majuscules, sinon, le serveur Tomcat annonce une erreur.
Avec ticket	0: ne pas envoyer les informations d'authentification

	1: envoyer le ticket actuel en tant qu'information d'authentification. Dans ce cas, ELOas vérifie le ticket et détermine le numéro utilisateur à partir de celui-ci. Cette information est mise à disposition au ruleset. Dans le ruleset, l'on peut décider si l'action est exécutée, et dans quelle mesure. actuellement, les informations d'authentification ne peuvent pas être évaluées lors de l'authentification SSO. Ce ne sera plus le cas dans la prochaine version du serveur d'indexation.
Nom du ruleset	Nom du ruleset à exécuter. Seuls les rulesets déclenchés peuvent être appelés. L'appel est ignoré pour les rulesets commandés par intervalle.
Parameter1	premier paramètre. Ce paramètre est utilisé comme terme de recherche pour l'exécution de ruleset , à condition qu'il ne soit pas vide.
Parameter2, Parameter3	autres paramètres en option Ceux-ci peuvent être demandés par le ruleset et diriger l'exécution.

Le script d'exemple complet pour un appel peut ressembler à ceci. Il appelle le ruleset *Expand Name* pour les objets avec l'ID d'objet 7944 et 7945.

```
Set Elo=CreateObject("Elo.Professional")
MsgBox Elo.SendELOasRequest("localhost", 8084, "/ELOmover/as" , 1, "Expand Name", "7944,7945"
```

Le ruleset peut être déclenché par l'appel d'une URL depuis d'autres applications:

```
http://localhost:8084/ELOmover/as?
cmd=run&amp;name=Expand%20Name&amp;param1=7944,7945&amp;param2=TestParam2
```

> **Remarque**
>
> Dans ce cas, nous ne pouvons pas donner d'informations d'authentification. Vous pouvez vous assurer dans votre ruleset que les données ne sont pas utilisées à mauvais escient.

Autres remarques

Déclenchement asynchrone du ruleset

Si un ruleset est déclenché par une adresse URL ou un appel de script, ELOas l'exécute de façon asynchrone. Si alors un autre ruleset est actif actuellement, l'exécution de script n'est pas retardée jusqu'à ce que ELOas soit disponible. Au lieu de cela, la commande d'activation est placée dans une file d'attente et effectuée à la prochaine occasion.

Cela a deux conséquences: le script client ne peut pas être sûr que l'opération a été exécutée si la commande a été traitée. Si c'est important pour le déroulement ultérieur du script, cela doit être vérifié dans le script en lui-même, puis intégré dans une file d'attente. Attention! Il se pourrait que ELOas effectue en ce moment même d'autres actions importantes. En général, un script ne devrait pas attendre la clôture d'une action ELOas.

Par ailleurs, il se pourrait qu'un utilisateur impatient active plusieurs fois le déclenchement. Dans ce cas, le ruleset est exécuté plusieurs fois. Il devrait donc être configuré de manière à ce

que ce déclenchement ne cause pas d'erreurs, par exemple lors d'une vérification de l'objet et de l'exécution qui s'est répétée.

Une autre conséquence découlant de l'exécution asynchrone: les erreurs apparues lors du traitement du ruleset ne peuvent pas être notées par le biais de l'appel.

Déclenchement synchrone du ruleset

Le ruleset est exécuté directement lors du déclenchement synchronisé et peut également rendre un résultat. Le déclenchement synchronisé est surtout utilisé par le processus ELO (créateur de formulaires). Lors de l'appel, `cmd=get` est requis au lieu de `cmd=run`. Par ailleurs, les rulesets pour l'appel synchronisé ne doivent pas se trouver dans le classeur *Rules*, mais dans le classeur *Direct*. Les rulesets synchrones sont exécutés dans un propre thread indépendamment des rulests asynchrones.

Vérification des autorisations

Lors de l'appel à partir du client Windows, le ticket client de l'authentification peut également être transféré en option. Dans ce cas, ELOas peut vérifier l'authentification et déterminer l'utilisateur actuel. En cas d'actions critiques, une vérification quant à un utilisateur autorisé devrait être faite dans le ruleset, et si l'authentification manque, l'exécution devrait être interrompue.

Un déclenchement anonyme peut être acceptable dans certains cas. Par exemple parce qu'un objet défini est traité. Dans ce cas, l'on doit veiller à ce que l'ID d'objet ne peut pas être modifié par l'appel. Cela peut se faire simplement dans l'événement `onstart`, la valeur est placée par `EM_SEARCHVALUE` dans le script. Dans ce cas, ce n'est pas le paramètre, mais la valeur pré-réglée du script ruleset qui est utilisée.

Ordre d'exécution

Un ruleset déclenché manuellement s'enclave normalement dans l'ordre d'exécution des rulesets. Si plusieurs déclencheurs existent pour un ruleset, tout d'abord, tous les déclencheurs sont exécutées, avant que le prochain ruleset soit traité.

Structure des règles

Ce document décrit la structure de la rule XML de ELOas. Normalement, cette structure est modifié par une interface utilisateur graphique. Si l'accès est manuel, cette description peut être utilisée en tant que référence. Cette description sert aussi de référence pour l'implémentation du GUI.

Structure générale

La structure intégrale est intégrée dans un tag <ruleset>. Celui-ci se compose de deux parties, une entrée <base> au début, suivie par un nombre illimité d'entrées <rule>.

L'entrée <base> contient les informations concernant la recherche des entrées à traiter. Il s'agit plus précisément de la ligne de recherche, du terme de recherche, des limitations de masque et de date.

Les entrées <rule> contiennent une directive de traitement. Chaque règle peut être dotée d'une condition, elle peut modifier la cible de dépôt ou ajuster les contenus des champs. Par ailleurs, une rule peut avoir un contenu JavaScript. Si celui-ci est défini, alors les autres entrées sont ignorées, mais elles peuvent rester avec des valeurs.

Si la condition d'une règle est "OnError", alors il s'agit d'une règle de traitement des erreurs. Après chaque règle peut être effectué un traitement d'erreurs, à la fin doit être entrée une règle de traitement d'erreurs. Cette règle d'erreurs finale est également appelée lorsqu'une erreur apparaît lors du déplacement ou de l'enregistrement. Si une erreur apparaît au sein d'une règle normale lors du traitement, la prochaine règle de traitement d'erreur est appelée le plus tôt possible, puis le traitement est interrompu.

Exemple pour un ruleset simple :

```
<ruleset>
    <base>
        <name>nom du ruleset</name>
        <search>
            <name>nom du champ de métadonnées dans le code JavaScript</name>
            <value>terme de recherche dans le code JavaScript </value>
            <mask>numéro du masque de document pour la recherche</mask>
        </search>
        <interval>5M</intervalle>
    </base>
    <rule>
        <name>nom de la règle</name>
        <destination mask="masque de classeur"> nouvelle cible dans le code JavaScript Code</
        <index>
            <name>nom du champ de métadonnées dans le code JavaScript</name>
            <value>nouveau contenu du champ de métadonnées dans le code JavaScript</value>
        </index>
    </rule>
    <rule>
```

```
        <name>nom de la règle de correction de l'erreur</name>
        <condition>OnError</condition>
    </rule>
</ruleset>
```

Toutes les entrées dans le paragraphe <base>

Tag	Fonction	Exemple
name	Nom du ruleset. Ce nom est affiché sur la page de statut, mais il n'est pas traité.	Traitement SAP
search	Paramètre pour la recherche des documents à traiter. Pour la description, voir le paragraphe suivant *Toutes les entrées dans le paragraphe '<search>*	
masks	Si vous souhaitez permuter vers un autre type de masque de dépôt dans le cadre du traitement, alors une liste de tous les numéros de masques cibles possible doit être affichée. Chaque numéro de masque est encadré par un tag <mask>.	<mask>3</mask><mask>4</mask>
interval	Intervalle de répétition pour le traitement de la recherche. Cet intervalle peut être saisi en minutes (5M) ou en heures (1H). Par ailleurs, il peut être exécuté une fois par jour à une heure précise (15:30), une fois par jour (17:20/SA) ou une fois par mois (22:00/31). Si une journée est entrée lors de l'exécution mensuelle, qui n'existe pas dans le mois actuel (par exemple le 31 au mois de février), alors c'est le dernier jour du mois qui est utilisé.	5M1H15:3017:20/SA22:00/31

Toutes les entrées dans '<search>'

Les entrées dans le paragraphe *<search>* déterminent quels documents doivent être traités. Au début de chaque passage, une recherche est effectuée avec ces paramètres. La liste des résultats est traitée selon les règles.

```
<search>
<name>nom du champ de métadonnées dans le code JavaScript Code</name>
<value>terme de recherche dans le code JavaScript</value>
<mask>numéro du masque de document pour la recherche</mask>
<max>nombre max. de documents par processus</max>
</search>
```

Tag	Fonction	Exemple
name	Nom du champ de métadonnées dans le code JavaScript Si le nom est fixe, un texte peut être directement entré entre guillemets. Mais il est également possible d'utiliser une autre expression JavaScript.	"ELOOUTL2"

Tag	Fonction	Exemple
value	Terme de recherche dans le code JavaScript. Si la valeur est fixe, un texte peut directement être entré entre guillemets. Mais il est également possible d'utiliser une autre expression JavaScript.	"ELO*"
mask	Numéro du masque de document pour la recherche Seul un masque de document et non pas un masque de recherche peut être utilisé, étant donné que tous les résultats sont sensés posséder la même définition de masque.	2
max	Nombre max. de document par passage lors de la demande au serveur d'indexation. S'ils existent plusieurs résultats, ceux-ci sont traités une fois que tous les autres rulesets ont été passés. Ceci est fait afin d'éviter qu'un ruleset volumineux empêche le traitement de tous les autres rulesets. Au plus 1000 documents par passage sont autorisés.	200
idatexdate	La liste des résultats peut être limitée par une plage de date dans la date de dépôt (idate) ou la date de document (xdate). Cette date peut être entrée en valeurs absolues au format de date ISO (YYYYMMTT) ou dans des valeurs relatives au tag actuel (-5). La section se compose d'une date de démarrage dans un tag <from> et d'une date de fin dans un tag <to>.	<idate><from>-5</from><to>+0</to></idate>

Toutes les entrées dans '<rule>'

Après le paragraphe <base>, suit un nombre illimité de paragraphes <rule\>. Lors du traitement, ceux-ci sont parcourus dans l'ordre de la définition.

Une <rule> peut exister en tant que règle normale et en tant que règle d'erreur. Dans l'exécution normale, les règles d'erreur sont tout simplement passées. En cas d'erreur, la prochaine règle d'erreurs disponible est appelée dans l'ordre, ensuite, le traitement est interrompu pour ce document, cela signifie qu'après une règle d'erreurs, d'autres règles ne sont pas traitées.

La dernière règle dans la chaîne <rule\> doit toujours être une règle d'erreur. Ainsi, il est assuré qu'un traitement des erreurs est disponible dans tous les cas. Par ailleurs, cette règle est également appelée si une erreur apparaît lors du déplacement ou de l'enregistrement.

Tag	Fonction	Exemple
name	Nom de la règle, est utilisé seulement pour une meilleure lisibilité.	Compléter l'index
condition	Condition d'exécution pour cette règle S'il s'agit d'une règle d'erreur, le texte fixe "OnError" est entré ici. Veuillez utiliser la bonne orthographe, sinon, la règle n'est pas reconnue comme règle d'erreur. La condition d'exécution se trouve en tant que CodeJavaScript. La règle n'est exécutée que si la condition est "true".	KDNR == "123"

Tag	Fonction	Exemple
destination	Nouvelle cible de dépôt du document en tant que chemin de dépôt. L'entrée est en option et peut rester vide, dans ce cas, le document reste à sa position d'origine. S'il existe plusieurs règles avec destination, la première cible est utilisée en tant que nouveau lieu de dépôt, toutes les autres cibles sont entrées en tant que référence en plus. Si une cible de dépôt n'existe pas encore, elle est créée automatiquement. Le tag destination peut contenir un attribut complémentaire "mask" qui contient le numéro du masque de classeur pour les nouveaux classeurs créés. Si cet attribut n'existe pas, la valeur "1" est utilisée par défaut, le numéro du masque de classeur dans une archive standard.	`<destination mask="1">¶ELO¶Mails¶" + ELOOUTL1</destination>`
mask	Nouveau masque de document Si cette entrée existe ou que le numéro de masque est -1, le masque d'origine reste. Lorsque le masque est modifié, toutes les entrées sont automatiquement adoptées avec le même nom de groupe. C'est également exécuté correctement, si la répartition des champs de métadonnées est différente. Si le masque de dépôt original contenait des champs que le nouveau masque ne possède pas, ces données sont rejetées automatiquement et sans message d'erreur. ELOas ne peut pas traiter de documents dont le masque de dépôt utilise le même nom de groupe pour plusieurs entrées, étant donné que le traitement interne et la structure des règles considèrent que l'attribution est univoque.	`<mask>20</mask>`
index	Dans une règle, il peut y avoir un nombre quelconque d'entrées d'indexation. Chaque entrée d'indexation contient le nom du champ concerné et une expression JavaScript avec la nouvelle version. En ce qui concerne les champs avec date ISO et les champs pour la date de dépôt et de document, l'entrée doit également être faite au format de date ISO. A côté des champs avec les noms de groupe du masque de recherche, sont disponibles tous les noms de groupes des masques de dépôt alternatifs ainsi que des "pseudo"-lignes d'indexation avec des valeurs standards de l'indexation. NAME : désignation	`<index><name>DOCDATE</name> <value>"20070930"</value></index>`

Tag	Fonction	Exemple
	DOCDATE : date de document ABLDATE : date de dépôt ARCHIVINGMODE: Dokumentenstatus 0, 1 oder 2 für "Keine Versionskontrolle", "Versionskontrolle eingeschaltet" oder "Keine Änderung möglich". ACL : adopter avec "PARENT" l'ACL de la nouvelle cible de dépôt. Avec `<droits>:<Name>`, vous pouvez définir les autorisations de groupe. OBJCOLOR : numéro de couleur de l'entrée OBJDESC : texte complémentaire OBJTYPE : Type de document ou du classeur de l'entrée. Attention: une attribution erronée peut causer des dysfonctionnements lors du prochain traitement. Les documents peuvent seulement posséder un OBJTYPE entre 254 et 286.	
script	une règle peut également contenir le code JavaScript à exécuter. Dans ce cas, tous les autres paramètres de cette règle sont ignorés, mais ils doivent également être contenus (par exemple à des fins de documentation).	

Modification des autorisations

Dans le champ de métadonnées pseudo ACL, une autorisation modifiée peut être déposée. Dans le cas le plus simple, l'on entre "PARENT", lors de l'enregistrement, l'autorisation du classeur cible pour cette entrée est adoptée. Il est également possible de déposer une liste d'autorisation complète. Cette liste se compose d'une suite d'autorisations individuelles, séparées par un pilcrow. Chaque autorisation se compose d'un masque d'autorisation (RWDELP – Read, Write, Delete, Edit, List, Permissions), suivi par un double-point et du nom de groupe. Pour les groupes ET, ce n'est pas le nom de groupe simple qui est entré, mais une suite de nom, séparé par un double-point.

```
R:Tous¶RW:Gestion opérationnelle¶RWDELP:Administration:Stuttgart:Management
```

Dans notre exemple, le groupe *Tous* reçoit un accès en lecture, le groupe *Gestion opérationnel*, un accès de lecture et d'écriture, et le groupe ET *Administration et Stuttgart et Management* un accès intégral au document.

Au cas où une autorisation doit être placée sur un utilisateur au lieu d'un groupe, ein "U" doit être adopté en plus dans la liste des autorisations.

```
UR:Administrator
```

Remarques

Lors de la création du code JavaScript, tous les noms de groupe du masque de recherche et des masques de dépôt alternatifs sont créés en tant que variable en majuscules. Ce procédé minimise le risque que les noms de groupe se recoupent avec les désignateurs standards du JavaScript ou de l'environnement runtime d'ELO. En principe, cela peut causer des problèmes, au cas où un des noms de groupe est identique à un désignateur standard ou une des listes de traduction.

```
var NAME;
var ARCDATE;
var DOCDATE;
var OBJCOLOR;
var OBJDESC;
var OBJTYPE;
var ARCHIVINGMODE;
var ACL;
var EM_PARENT_ID;
var EM_PARENT_ACL;
var EM_SEARCHNAME;
var EM_SEARCHVALUE;
var EM_SEARCHCOUNT;
var EM_SEARCHMASK;
var EM_IDATEFROM;
var EM_IDATETO;
var EM_XDATEFROM;
var EM_XDATETO;
var EM_FOLDERMASK = "1";
```

> **Information**
>
> Cette liste peut être élargie au cours de la progression du projet, elle peut aussi être complétée par des entrées personnalisées.

Le numéro du masque de dépôt du document actuel peut être modifié par une règle. S'il s'agit d'un numéro de masque ou d'un numéro qui ne se trouve pas dans la liste des masques cibles alternatifs, cela cause une erreur runtime lors de l'enregistrement du document, et non pas directement lors de l'attribution de masque.

Si une règle d'erreur est appelée en raison d'une erreur runtime, alors cela supprime toutes les cibles de dépôt prédéfinies des règles traitées auparavant. Si la règle d'erreur ne possède pas de propre `<destination>`, le document reste à sa position d'origine, sinon, c'est la cible de la règle d'erreur qui est utilisée.

Le déplacement et l'enregistrement des métadonnées modifiée se font seulement à la fin du traitement de la dernière règle. Si une erreur apparaît, alors la dernière règle d'erreur est appelée et non pas la règle d'erreur appartenant à la règle ayant déterminé la cible (mais ce qui est identique, s'il existe une seule règle d'erreur).

Exemple de structure

Voici une définition à titre d'exemple, le code créé est affiché: Cette information est faite en vue d'une orientation.

```
<ruleset>
    <base>
        <name>Mailmaske Thiele</name>
        <search>
            <name>"ELOOUTL2"</name>
            <value>"Thiele*"</value>
            <mask>2</mask>
            <max>2</max>
            <idate>
                <from>"-35"</from>

                <to>"+1"</to>
            </idate>
        </search>
        <masks>
            <mask>12</mask>
            <mask>13</mask>
            <mask>20</mask>
        </masks>
        <interval>1M</interval>
    </base>
    <rule>
        <name>Regel 1</name>
        <destination mask="5">"¶Thiele¶Mails¶" + ELOOUTL1</destination>
        <mask>20</mask>
        <index>
            <name>ADDENTRY</name>
            <value>getObjShort(2)</value>
        </index>
        <index>
            <name>ELOOUTL2</name>
            <value>"!!" + ELOOUTL2</value>
        </index>
        <index>
            <name>DOCDATE</name>
            <value>"20070930"</value>
        </index>
        <index>
            <name>ARCHIVINGMODE</name>
            <value>2</value>
        </index>
        <index>
            <name>ACL</name>
            <value>"PARENT"</value>
```

```
        </index>
    </rule>
    <rule>
        <name>Journal-Kopie</name>

        <destination mask="1">"¶Thiele¶Journale¶" + ELOOUTL1</destination>
    </rule>
    <rule>
        <name>Script rule</name>
        <script>
            moveTo(Sord, "¶Ablage¶Ziele1¶" + ELOOUTL1);
            moveTo(Sord, "¶Ablage¶Ziele2¶" + ELOOUTL2);
            moveTo(Sord, "¶Ablage¶Ziele3¶" + ELOOUTL3);
        </script>
    </rule>
    <rule>
        <name>Global Error Rule</name>
        <condition>OnError</condition>
        <destination>"¶Thiele¶Error"</destination>
        <index>
            <name>ELOOUTL2</name>
            <value>"!!" + ELOOUTL2</value>
        </index>
        <index>
            <name>ARCHIVINGMODE</name>
            <value>0</value>
        </index>
    </rule>
</ruleset>
```

Programmation

Programmer avec ELO Automation Services

Le chapitre "Programmer avec les services d'automation ELO (ELOas)" décrit la structure et l'utilisation de l'environnement JavaScript Runtime. Ces modules permettent d'élargir ELOas et d'exécuter des fonctions complémentaires qui n'existent pas dans la version de base.

Exécution de scripts

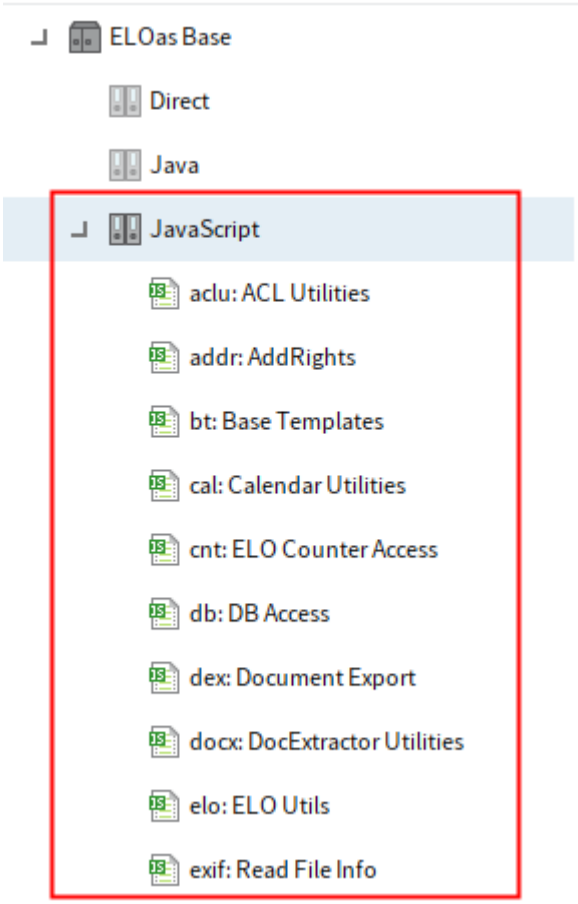
La configuration XML du ruleset n'est pas seulement interprétée par ELOas. Au lieu de cela, elle est traduite dans un programme JavaScript lors de la lecture, puis elle est liée avec les routines de base se trouvant également dans JavaScript. Ce script sera exécuté plus tard. Cela a plusieurs avantages:

Les attributions dans la configuration XML peuvent contenir des expressions JavaScript valables avec des appels de fonctions quelconques.

Dans la configuration XML, vous pouvez intégrer des extraits de code JavaScript avec des routines complexes.

Les routines de base peuvent être élargies avec toutes les fonctions imaginables. Celles-ci peuvent également être utilisées par des administrateurs qui n'ont pas de propres connaissances en matière de développement, il suffit d'appeler la fonction au sein de l'expression. Comme exemples, vous pouvez visualiser les modules DB Access et Document Export.

Les routines de base élargies peuvent utiliser des bibliothèques Java externes quelconques pour un élargissement des fonctions (par exemple le pilote JDBC ou bien le client IX pour une commande de serveur d'indexation directe).



Illustr. : répertoire 'JavaScript'

L'avantage des fonctions de base dans JavaScript est que ces fonctions peuvent être ajustées ou complétées, sans que ELOas en lui-même doive être modifié. Il est donc possible de travailler avec un programme standard, qui toutefois peut être adapté de façon très souple aux exigences.

ELOas apporte les fonctions de base nécessaires pour l'exécution de la recherche et le traitement des règles (Base Templates, Imports et ELO Utils). Normalement, cette partie doit rester inchangée, il n'est en règle générale pas sensé d'effectuer des modifications ici. Par ailleurs, il apporte deux modules pour l'accès à la base de données (DB Access) et l'exportation des fichiers de document (Document Export). D'autres modules sont prévus pour les versions ultérieures. Pour les partenaires busines, il est également prévu de créer en quelque sorte une bourse d'échange.

Afin que ces modules puissent être exploités sans conflits, un concept d'espace de noms est prévu afin que chaque module puisse être doté d'un propre espace de nom. Les espaces de nom doivent être écrits en minuscules, sinon, des conflits avec les noms de groupe de la définition de masque peuvent apparaître. Tous les noms d'espaces de nom à deux ou trois caractères sont réservés pour ELO et sont utilisés pour les modules standards et les extensions autorisées. Pour les propres modules, les partenaires peuvent utiliser des noms d'espaces de nom à quatre signes ou plus longs. Si vous créez un module qui sera mis en place dans seulement un projet, vous pouvez également utiliser des noms composés d'un seul signe. Le nom de module dans ELO doit commencer par le nom de l'espace de noms, suivi par un double point et une courte description (par exemple. `exd: exportation de document`). A l'interne, l'espace de nom doit être intégré de façon à ce qu'un objet JavaScript soit créé avec le nom de

l'espace de nom, et alors toutes les fonctions requises du module sont attribuées à cet objet. Remarque: les différentes fonctions sont séparées par une virgule et non pas par un point-virgule, étant donné qu'il s'agit d'une énumération.

```
var dex = new Object();
dex = {
    command1: function(x,y) {



        …
    },

    command2: function() {



        …
    }
}
```

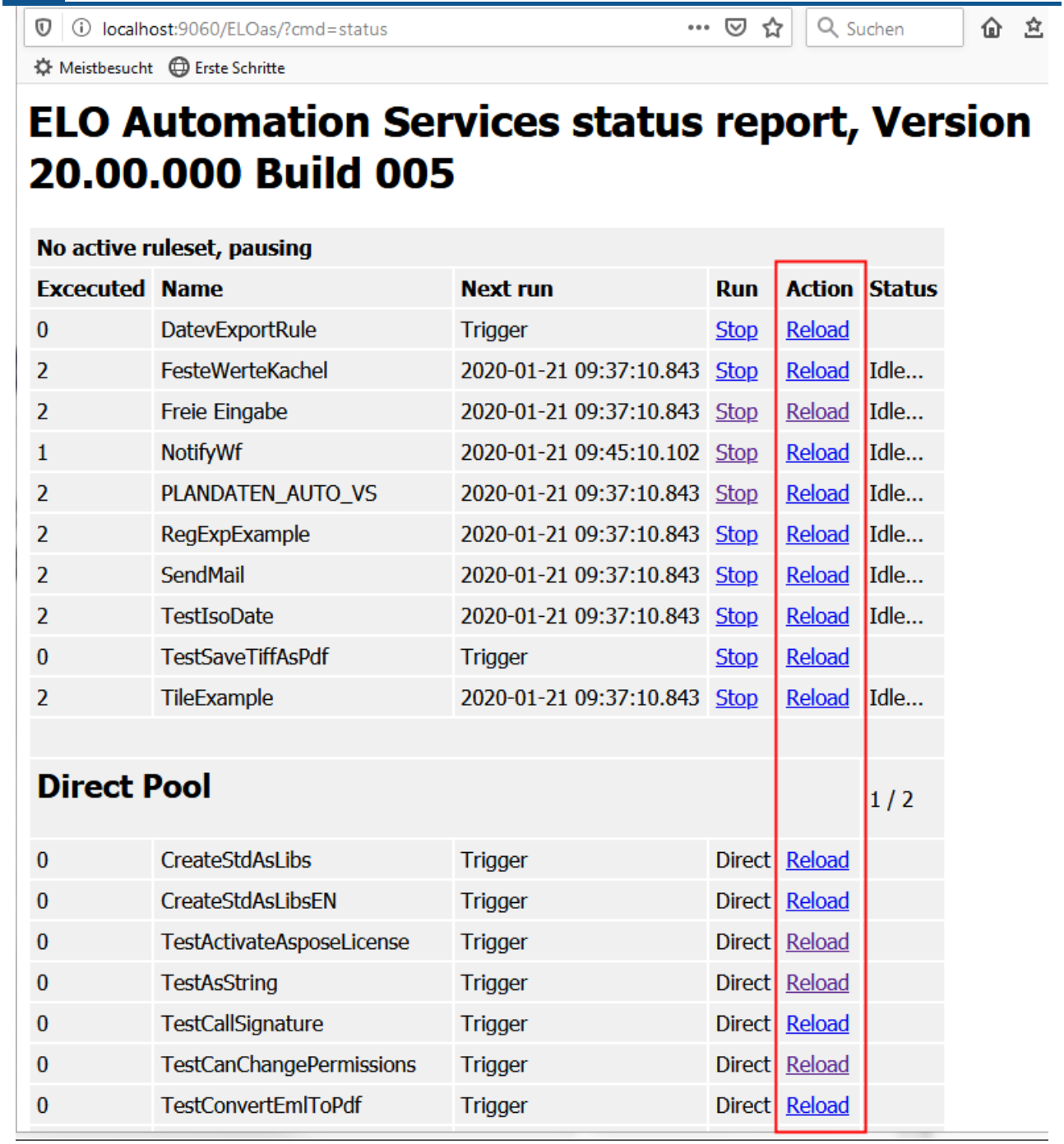
Ces fonctions peuvent être appelées par le code JavaScript Code avec `dex.command1(x,y)` ou `dex.command2()`. Chaque module possède une identification propre à lui et donc univalente, ce qui fait qu'ils peuvent être combinés, sans que des conflits de noms surviennent.

De tous les modules de base, le module *Imports* est le plus spécial. Dans le programme JavaScript créé, il apparaît toujours tout au début de la chaîne. C'est donc la place pour les importations nécessaires des bibliothèques Java. Par ailleurs, vous pouvez déposer des variables globales. Etant donné que ce module est un module global, il ne possède pas d'espace de nom.

La création de propres modules

Les modules nouveaux peuvent tout simplement être créés par l'administrateur, il suffit de créer un nouveau classeur avec le nom du module dans le classeur *ELOas\JavaScript*. Le code JavaScript à proprement parler est déposé dans le texte supplémentaire du classeur. La commande des autorisations ELO permet d'activer et de désactiver des modules, en plaçant une ACL qui permet ou non un accès de lecture au compte ELOas.

Dans tous les cas, les modules venant d'être créés ou autorisés sont seulement actifs une fois le service redémarré ou actualisé.



ELO Automation Services status report, Version 20.00.000 Build 005

No active ruleset, pausing

Excecuted	Name	Next run	Run	Action	Status
0	DatevExportRule	Trigger	Stop	Reload	
2	FesteWerteKachel	2020-01-21 09:37:10.843	Stop	Reload	Idle...
2	Freie Eingabe	2020-01-21 09:37:10.843	Stop	Reload	Idle...
1	NotifyWf	2020-01-21 09:45:10.102	Stop	Reload	Idle...
2	PLANDATEN_AUTO_VS	2020-01-21 09:37:10.843	Stop	Reload	Idle...
2	RegExpExample	2020-01-21 09:37:10.843	Stop	Reload	Idle...
2	SendMail	2020-01-21 09:37:10.843	Stop	Reload	Idle...
2	TestIsoDate	2020-01-21 09:37:10.843	Stop	Reload	Idle...
0	TestSaveTiffAsPdf	Trigger	Stop	Reload	
2	TileExample	2020-01-21 09:37:10.843	Stop	Reload	Idle...

Direct Pool

1 / 2

0	CreateStdAsLibs	Trigger	Direct	Reload	
0	CreateStdAsLibsEN	Trigger	Direct	Reload	
0	TestActivateAsposeLicense	Trigger	Direct	Reload	
0	TestAsString	Trigger	Direct	Reload	
0	TestCallSignature	Trigger	Direct	Reload	
0	TestCanChangePermissions	Trigger	Direct	Reload	
0	TestConvertEmlToPdf	Trigger	Direct	Reload	

Illustr. : page de statut ELOas

Le propre module peut apporter ses propres fonctions ou des variables globales. Etant donné que tous les modules sont exécutés ensemble dans un contexte JavaScript, il est tout de même important que l'on évite les conflits de noms. Malheureusement, ces conflits ne peuvent pas être reconnus comme erreurs par le JavaScript Interpreter, ce qui fait qu'ils ne peuvent pas être reconnus automatiquement.

Les objets du module ont une durée de vie illimitée. Une fois qu'ils ont été créés, ils restent actifs jusqu'à ce que le service soit terminé ou actualisé. Cela peut être très problématique dans certains cas, par exemple pour les connexions aux bases de données. Si une telle connexion est créée lors du démarrage du programme ou lors du premier run, et qu'elle reste active de façon illimitée, alors il se pourrait que les ressources soient utilisées sans que cela soit nécessaire (par exemple si le ruleset est actif seulement une fois par mois). De plus, il se

pourrait que la ressource deviennent invalide (par exemple en redémarrant le serveur de la base de données). Pour reconnaître un état invalide du service et démarrer une reconnexion automatique, il faut investir beaucoup de temps. Il est plus simple de veiller à ce que ces ressources sont connectées uniquement quand cela est nécessaires, et qu'elles soient autorisées automatiquement à la fin du ruleset (voir également le chapitre *Lazy Initialization*). Chaque module doit implémenter une fonction avec un nom spécial: `<Namespace>ExitRuleset` (par exemple `dexExitRuleset`). A la fin du traitement d'un ruleset, cette fonction spéciale est appelée pour chaque module. Dans cette fonction, il est possible de déposer les appels pour l'arrêt de la connexion.

Lazy Initialization

Si toutes les ressources externes doivent être connectées lors de chaque exécution du ruleset et qu'elles doivent être séparées à la fin, cela peut engendrer une utilisation de capacités inutile. Si un ruleset doit réagir rapidement et qu'il est exécuté une fois par minute, alors il se pourrait que souvent, il n'y ait pas de jeu de données actif devant être traité. Souvent, les connexions qui ne sont pas requises souvent sont créées et séparées. Pour cette raison, nous vous recommandons de connecter les ressources externes par le biais de "Lazy Initialization". Dans ce cas, la connexion n'est pas établie avec la recherche, mais seulement quand elle doit réellement être utilisée.

En pratique, ce schéma peut être implémenté facilement. Prenons l'exemple "Reader", nous souhaitons utiliser une ressource possédant les méthodes `Open()`, `Read()` et `Close()`. `Open()` doit seulement être exécuté lors du premier `Read()`, `Close()` seulement si un `Open()` a été exécuté. Le ruleset entre un nom utilisateur à partir de cette ressource, à l'aide de `readUser`. Le code JavaScript dans le Reader pourrait ressembler à ceci:

```
var readerInitialized = false;

var reader = new Object();
reader = {

function readUser() {
    If (!readerInitialized) {
        Open();
        readerInitialized = true;
    }
    return Read();
}

}

function readerExitRuleset() {
    if (readerInitialized) {
        Close();
    };
};
```

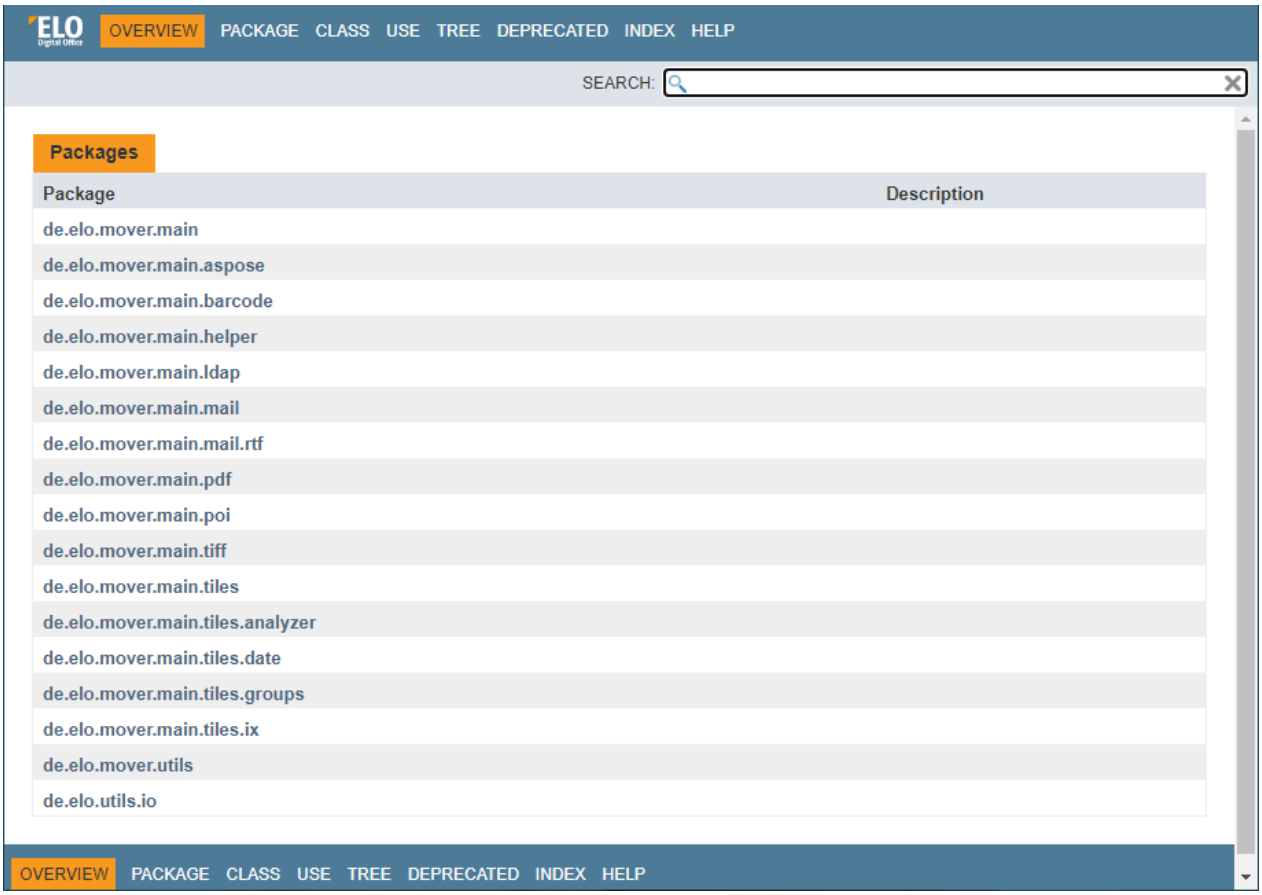
Grâce à la variable globale `readerInitialized`, le module retient si une connexion a été ouverte avec `Open()`. Celle-ci est placé sur false lors du démarrage du programme, il n'existe pas encore de contact.

Si une règle du ruleset doit déterminer le nom utilisateur, la fonction `readUser()` est appelée. Tout d'abord, l'on vérifie s'il existe déjà une connexion. Sinon, elle est ouverte avec `Open()`, et `readerInitialized` est placé sur true. Pour les appels suivants, `Open()` n'est pas à nouveau exécuté. Ensuite, la lecture est faite avec `Read()` dans la ressource.

Une fois le ruleset traité, la fonction *readerExitRuleset* est appelée pour le module `Reader`. Il est alors effectué une vérification quant à une connexion ouverte, puis, celle-ci est fermée avec `Close()` si nécessaire.

ELOas Javadoc

ELOas 21 met à disposition des classes et fonctions d'aide pour traiter les tâches récurrentes. La Javadoc de l'interface ELOas est disponible sous http://www.forum.elo.com/javadoc/as/21/. Par ailleurs, le master ELO actuel contient un rassemblement d'exemples de règles pour l'appel des fonctions de l'interface ELOas.

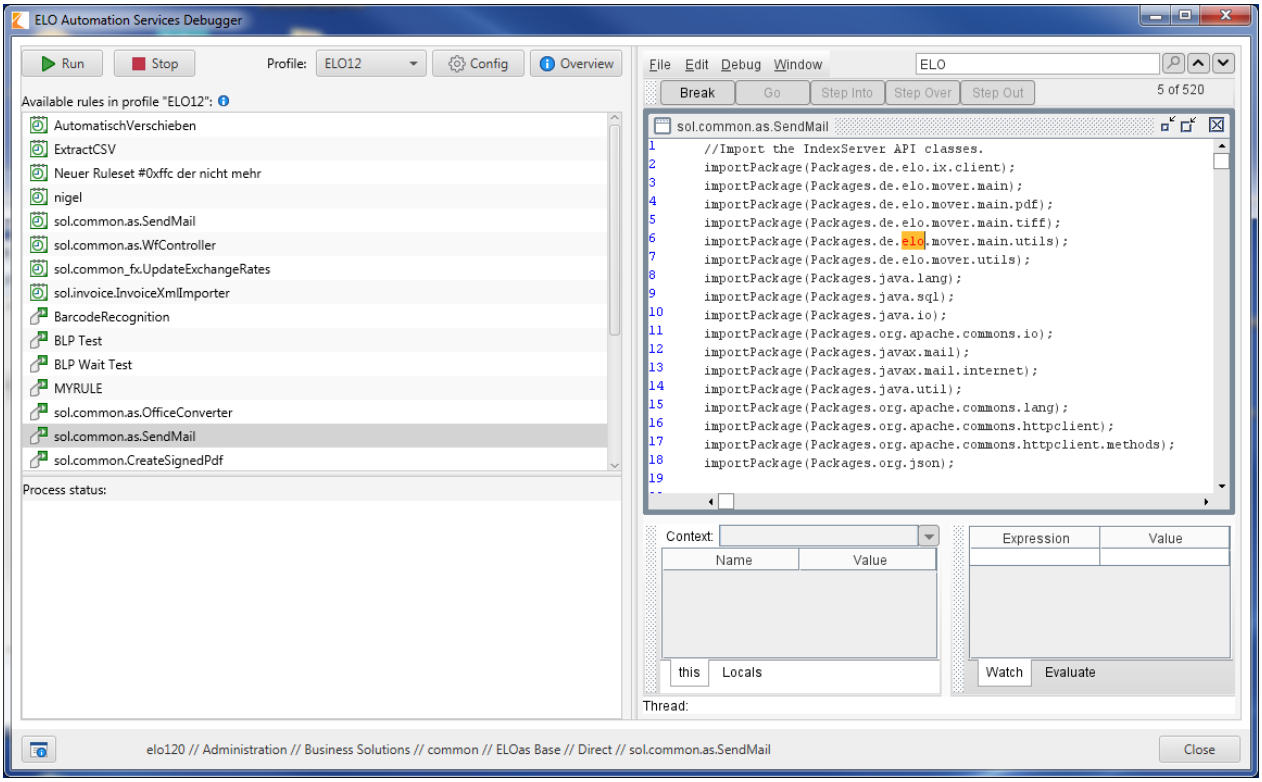


Illustr. : vue d’ensemble

Recherche d'erreurs

A partir de la version 7.00.024, il existe un débogueur pour ELOas. C'est la Debug Engine se trouvant dans la Rhino Engine qui est utilisée. Elle peut être activée par le biais d'un paramètre de configuration.

```
<entry key="debug">true</entry>
```

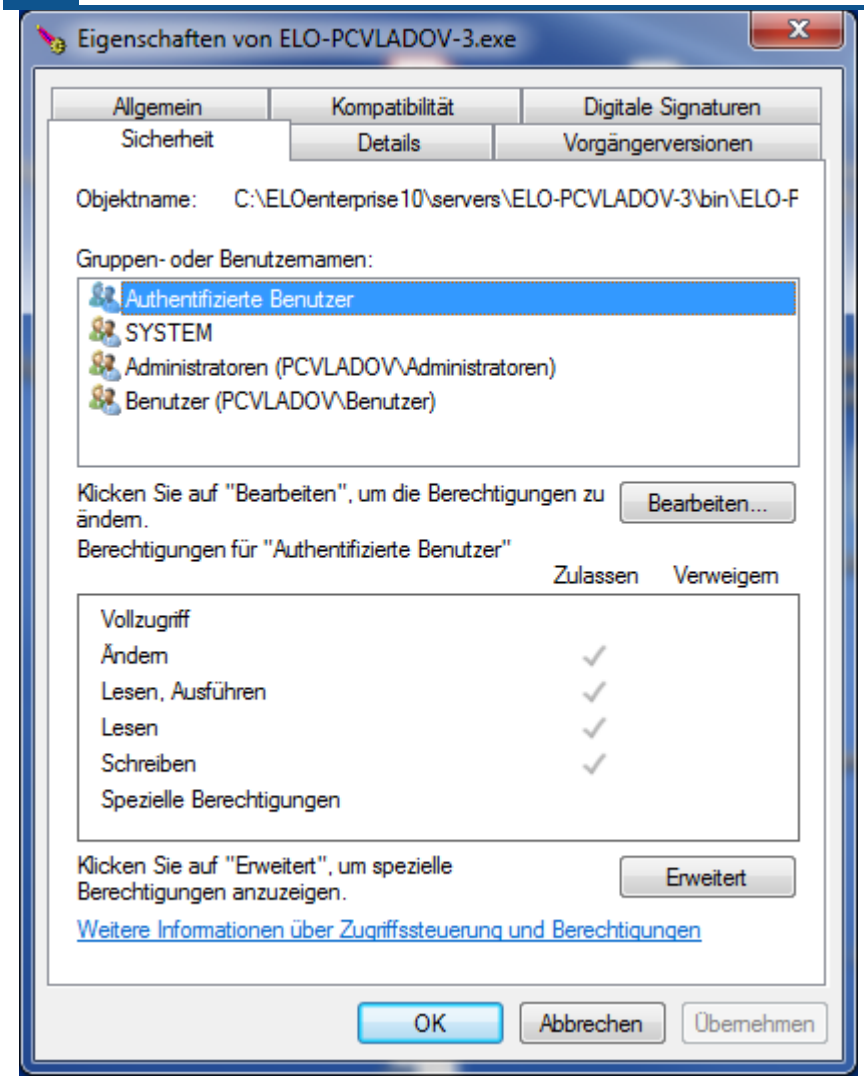


Illustr. : ELO Automation Services Debugger

Pour exploiter le débogueur, ELOas doit être exécuté de façon locale sur la machine de développement. Par ailleurs, il doit être démarré en tant que processus de console et non pas en tant que service Windows. Sinon, le débogueur ne fonctionne plus sous Windows Vista ou Windows 7.

Si vous avez plusieurs rulesets actifs, il existe une propre fenêtre de débogage pour chaque ruleset. Vous pouvez basculer entre les différents affichages par le biais de l'entrée de menu *Window*.

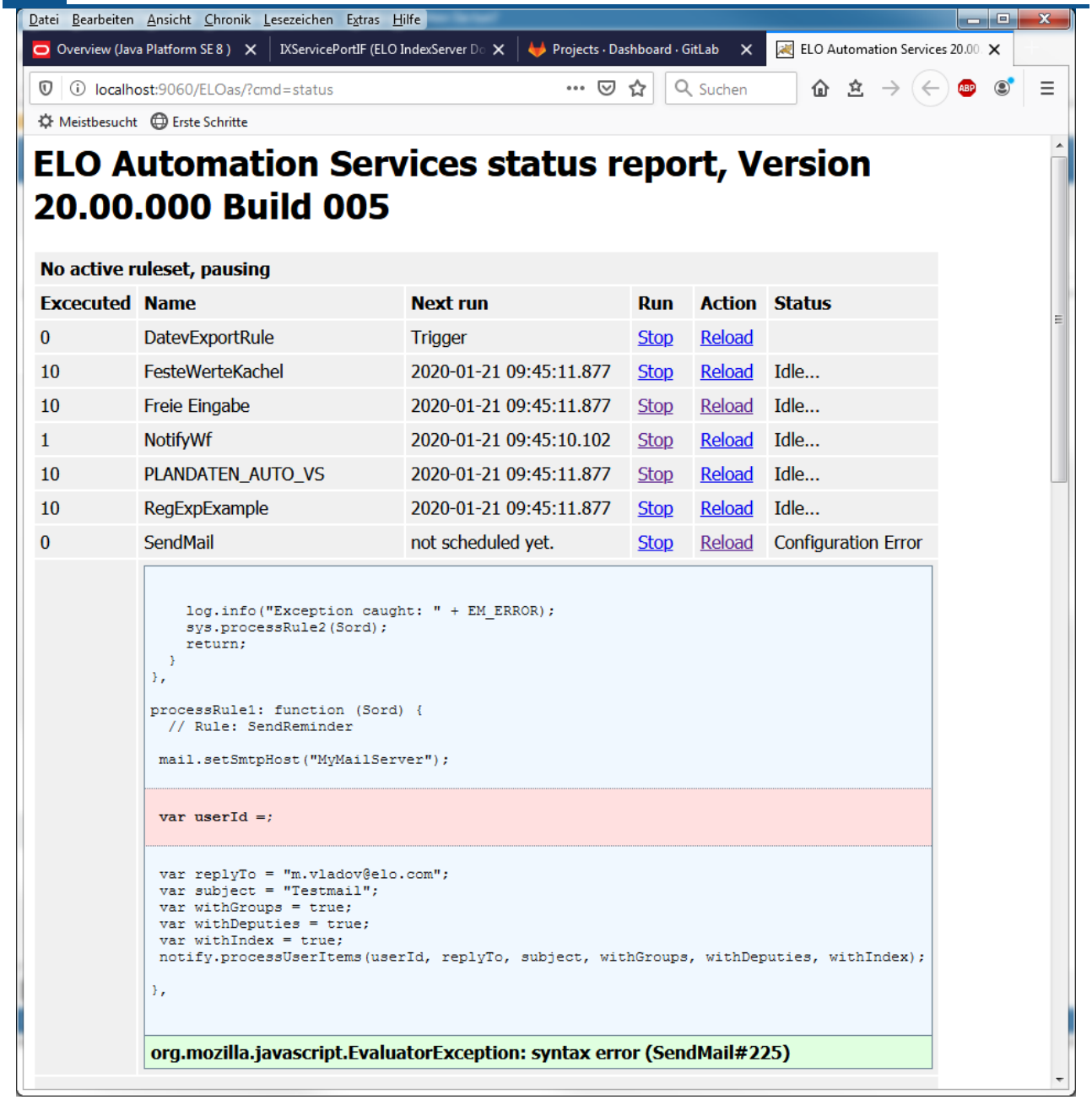
Dans le débogueur, vous pouvez placer les breakpoints sur les différentes fonctions et inspecter ou modifier les contenus des variables. Vous pouvez poursuivre l'exécution avec une étape individuelle ou dans un mode d'exécution.



Illustr. : propriétés de Apache Tomcat

Erreur de syntaxe dans le script

Si le script contient une erreur de syntaxe, alors le traitement JavaScript ne peut pas être démarré. Ces erreurs ont l'avantage d'être visibles directement lors du démarrage du programme et de pouvoir être affichées dans le dialogue de statut de ELOas.



Illustr. : erreur de syntaxe dans le script

Pour simplifier la recherche d'erreurs, le programme JavaScript complet est pris en compte dans le rapport ELOas, avec tous les modules intégrés. Le numéro d'erreur affiché sous forme de liste se réfère à ce paragraphe du rapport (à partir de "//Import the IndexServer API classes").

```
14:28:07,681 DEBUG (WorkingSet.java:368) - load JavaScript Templates,
        Parent GUID=(23594D10-4704-4FF9-938B-136792051D67)
14:28:07,744 DEBUG (WorkingSet.java:385) - Script file found: Base Templates
14:28:07,744 DEBUG (WorkingSet.java:385) - Script file found: Imports
14:28:07,744 DEBUG (WorkingSet.java:385) - Script file found: ELO Utils
14:28:07,759 DEBUG (WorkingSet.java:385) - Script file found: DB Access
14:28:07,759 DEBUG (WorkingSet.java:385) - Script file found: Document Export
14:28:07,759 DEBUG (WorkingSet.java:385) - Script file found: Dummy Modul mit
```

```
                                          Namenskonflikt
14:28:07,759 DEBUG (WorkingSet.java:276) - loadItems,
        Parent GUID=(9DAC7E8D-1467-4820-B53B-D27CCB5F06C0)
14:28:07,822 DEBUG (WorkingSet.java:286) - Number of Child entries: 1
14:28:07,822 DEBUG (WorkingSet.java:304) - Ruleset: MailRule1
14:28:08,025 DEBUG (WorkingSet.java:472) -
//Import the IndexServer API classes.
importPackage(Packages.de.elo.ix.client);
importPackage(Packages.java.lang);
importPackage(Packages.java.sql);
importPackage(Packages.sun.jdbc.odbc);
importPackage(Packages.java.io);

var NAME;
var ARCDATE;
var DOCDATE;
var OBJCOLOR;
var OBJDESC;
var OBJTYPE;
var ARCHIVINGMODE;
var ACL;

var EM_PARENT_ID;
var EM_PARENT_ACL;

var EM_NEW_DESTINATION = new Array();
var EM_FIND_RESULT = null;
…
```

Veillez à ce que cette sortie soit répétée à chaque redémarrage et lors du rechargement. Plusieurs listes peuvent se trouver dans un fichier de rapport. C'est toujours la dernière liste du rapport qui est actuelle.

Erreurs logiques et erreurs runtime

C'est un peu plus difficile lors des erreurs runtime. La seule possibilité est de limiter l'endroit où se trouve l'erreur en s'appuyant sur les sorties log. Cette sortie de log est beaucoup moins pratique qu'un débogueur interactif, mais présente de nombreux avantages lors d'un traitement important. Le Java Logger de ELOas est accessible sur la page JavaScript sous le nom *log*. Le code JavaScript peut être effectué avec les entrées `log.debug()`.

```
var cmd = "SELECT * FROM objekte where objid = 22"
var res = getLine(1, cmd)
log.debug(res.objshort)
log.debug(res.objidate)
log.debug(res.objguid)
```

Les sorties log du code JavaScript sont reconnaissables au nom de la classe manquante et au fait que le numéro de la ligne manque dans le rapport.

```
15:38:57,643 DEBUG (?:?) - Now init JDBC driver
15:38:57,659 DEBUG (?:?) - Get Connection
15:38:57,659 DEBUG (?:?) - Init done.
15:38:57,659 DEBUG (?:?) - createStatement
15:38:57,659 DEBUG (?:?) - executeQuery
15:38:57,659 DEBUG (?:?) - read result
15:38:57,659 DEBUG (?:?) - getLine done.
15:38:57,659 DEBUG (?:?) - Suchen geändert.
15:38:57,659 DEBUG (?:?) - 56666880
```

Modules standards

Cette documentation vous donne une description détaillée des modules standards suivants :

- cnt: ELO Counter Access
- db:DB Access
- dex: Document Export
- ix: Fonctions du serveur d'indexation
- wf: Workflow Utils
- mail: Mail Utils
- fu: File Utils
- run : Runtime Utilities.

Les modules standards *elo*, *tfer*, *addr*, *notify*, *exif*, *www* sont décrits dans Javadoc de l'interface ELOas interne sous http://www.forum.elo.com/javadoc/as/21/.

cnt: ELO Counter Access

Le module standard *cnt* met à disposition l'accès aux variables de compte du ELOam.

cnt : fonctions disponibles

Créer un compteur : la fonction `createCounter()` crée un nouveau compteur et l'initialise avec une valeur de démarrage. Si le compte existe déjà, il est ré-initialisé.

```
createCounter: function (counterName, initialValue) {


    var counterInfo = new CounterInfo();
    counterInfo.setName(counterName);
    counterInfo.setValue(initialValue);

    var info = new Array(1);
    info[0] = counterInfo;

    ixConnect.ix().checkinCounters(info, LockC.NO);
},
```

Déterminer la valeur de compteur : la fonction `getCounterValue()` permet de déterminer la valeur actuelle du compteur indiqué. Si le paramètre `autoIncrement` est `true`, la valeur du compte est comptée en plus automatiquement.

```
getCounterValue: function (counterName, autoIncrement) {


    var counterNames = new Array(1);
    counterNames[0] = counterName;
    var counterInfo = ixConnect.ix().checkoutCounters(counterNames, autoIncrement, LockC.NO);
```

```
    return counterInfo[0].getValue();
},
```

Créer un numéro de traçage à partir du compteur: si l'on requiert un numéro courant et pouvant être reconnu automatiquement, alors l'on peut utiliser la fonction `getTrackId()`. Elle lit tout d'abord la prochaine valeur du compte et code un chiffre avec un préfixe et un chiffre de vérification. Le string créé ressemble à ceci `<préfixe><numéro courant>C<chiffre de vérification>` ("ELO1234C0")

```
getTrackId: function (counterName, prefix) {


    var tid = cnt.getCounterValue(counterName, true);
    return cnt.calcTrackId(tid, prefix)
},
```

Créer un numéro de traçage: si l'on requiert un numéro courant et pouvant être reconnu automatiquement, alors l'on peut utiliser la fonction `calcTrackId()`. Elle code un chiffre avec un préfixe et un numéro de vérification. Le string créé ressemble à ceci `<préfixe><numéro courant>C<chiffre de vérification>` ("ELO1234C0")

```
calcTrackId: function (trackId, prefix) {


    var chk = 0;
    var tmp = trackId;

    while (tmp > 0) {
        chk = chk + (tmp % 10);
        tmp = Math.floor(tmp / 10);
    }

    return prefix + "" + trackId + "C" + (chk %10);
},
```

Recherche le numéro de traçage dans le texte : la fonction `findTrackId()` recherche un numéro de traçage dans un texte. Le préfixe et la longueur du chiffre peuvent être dirigés par un paramètre. Si le chiffre possède une longueur variable, le paramètre de longueur peut être placé sur 0. Si aucun résultat n'est trouvé dans le texte, c'est le -1 qui est livré en retour. Sinon, c'est la valeur du chiffre qui est livrée (et non pas l'ID du Track).

```
findTrackId: function (text, prefix, length) {
    text = " " + text + " ";

    var pattern = "\\s" + prefix + "\\d+C\\d\\s";
    if (length > 0) {
        pattern = "\\s" + prefix + "\\d{" +
```

```
                length + "}C\\d\\s";
    }

    var val = text.match(new RegExp(pattern, "g"));
    if (!val) {
        return -1;
    }

    for (var i = 0; i < val.length; i++) {
        var found = val[i];
        var number = found.substr(prefix.length + 1,
                                  found.length - prefix.length - 4);
        var checksum = found.substr(found.length - 2, 1);
        if (checkId(number, checksum)) {
            return number;
        }
    }

    return -1;
}
```

db : DB Access

Le module standard DB Access met à disposition un accès simple aux bases de données externes. De façon standard, les bases de données ODBC, ainsi que Microsoft SQL et Oracle SQL sont supportés. Si l'accès aux autres bases de données se fait avec un pilote JDBC natif, les fichiers JAR correspondants doivent être copiés dans le répertoire LIB du service, et les importations et paramètres d'accès doivent être placés dans le module *Imports*. L'ordre des définitions de la base de données dans le module Imports définit la valeur du paramètre *Numéro de connexion* dans les appels suivants.

db : fonctions disponibles

```
getColumn( Numéro de connexion, demande SQL);
```

Cet appel doit donner une demande SQL comme paramètre; celle-ci demande une colonne et livre seulement une ligne comme résultat.

Exemple :

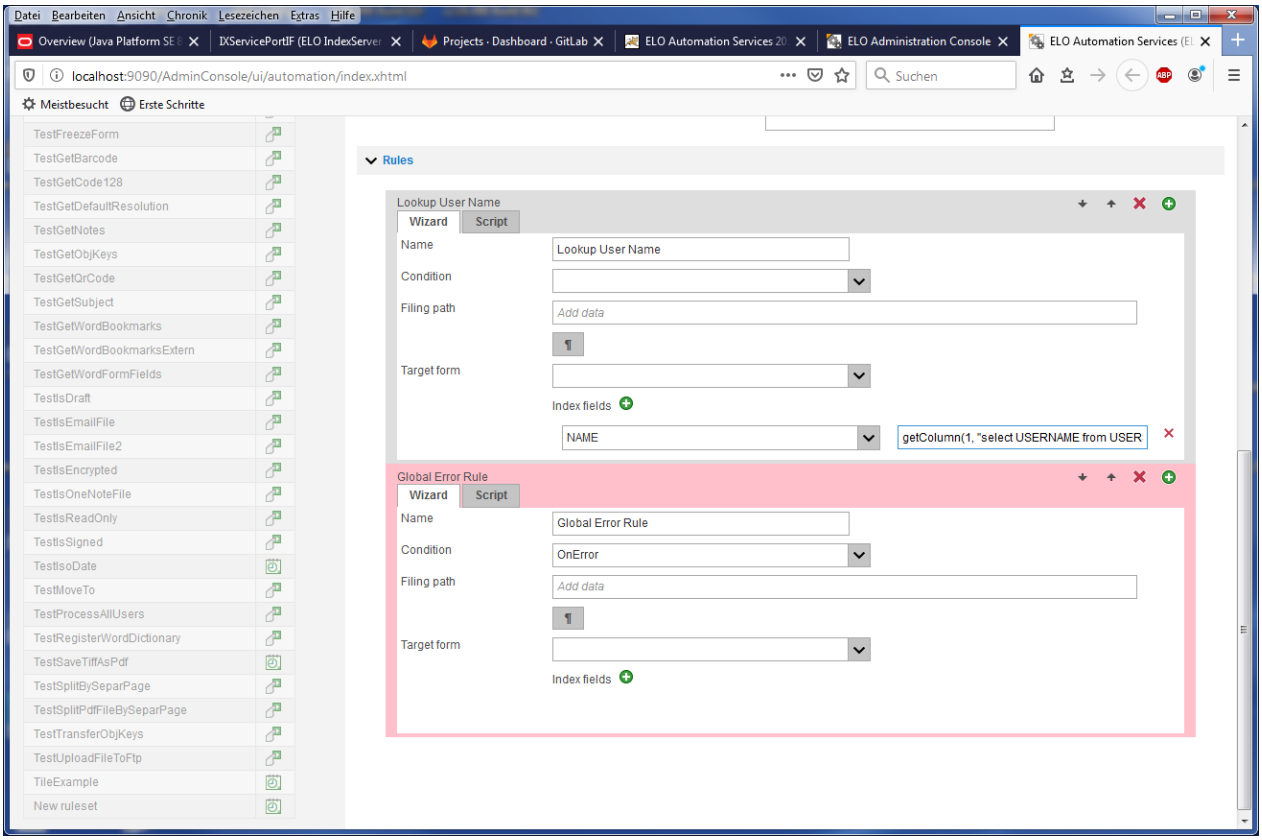
```
„select USERNAME from CUSTOMERS where USERID = 12345“
```

Par le biais du numéro de connexion, l'on détermine quelle connexion à la base de données est utilisée. La liste des connexions disponible est définie dans le module "Imports".

Exemple avec JavaScript Code:

```
var cmd = "select USERNAME from CUSTOMERS where USERID = 12345"
var res = getColumn(1, cmd)
log.debug(res)
```

Exemple dans le créateur GUI:



Illustr. : créateur GUI

Si la liste des résultats devait se composer de plusieurs lignes, seule la première valeur serait livrée. Tous les autres résultats sans ignorés sans message d'erreur.

```
getLine( Numéro de connexion, demande SQL );
```

En tant que résultat, cet appel donne en retour un objet JavaScript avec les valeurs de la première ligne de la demande SQL. La demande peut contenir un nombre illimité de colonnes, aussi un *. Les noms de colonnes doivent être univalents et être des désignateurs JavaScript valides. Attention aux minuscules et majuscules en ce qui concerne les désignateurs JavaScript.

Exemple :

```
"select USERNAME, STREET, CITY from CUSTOMERS where USERID = 12345"
```

Par le biais du numéro de connexion, l'on détermine quelle connexion à la base de données est utilisée. La liste des connexions disponible est définie dans le module "Imports".

Exemple avec JavaScript Code:

```
var cmd =
  "SELECT objshort, objidate, objguid FROM [elo20].[dbo].objekte where objid = 22"
var res = getLine(1, cmd)
log.debug(res.objshort)
log.debug(res.objidate)
log.debug(res.objguid)
```

Si la liste des résultats contient plusieurs lignes, seules les valeurs de la première ligne seront livrées en retour. Toutes les autres lignes sont ignorées sans message d'erreur.

```
getMultiLine(numéro de connexion, commande SQL, nombre max. de ligne)
```

Cette commande travaille comme l'appel `getLine`. Attention : ce n'est pas un objet, mais un array d'objets qui est rendu. Chaque ligne de la liste des résultats crée une entrée dans le Array. Afin que la mémoire ne "déborde" pas s'il s'agit d'une base de données volumineuses, l'on peut limiter le nombre max. de lignes. Vous pouvez ignorer d'autres résultats.

Exemple :

```
var obj = db.getMultiLine(1, "select objshort, objid from [elo80].[dbo].objekte where objid <

    for (var lg = 0; lg < obj.length; lg++) {
        log.debug(obj[lg].objid + " : " + obj[lg].objshort);
    }

doUpdate(numéro de connexion, commande SQL)
```

L'on ne peut pas utiliser les appels `getLine` ou `getColumn` pour exécuter une modification dans la base de données. Ces commandes utilisent la commande JDBC en interne `executeQuery` – et celui-ci permet seulement les demandes `SELECT`.

Pour modifier une entrée, l'on peut utiliser l'appel `doUpdate`. Celui-ci transfère la commande SQL entrée à la commande JDBC `executeUpdate` – grâce à ceci, l'on peut modifier les entrées existantes ou ajouter de nouvelles entrées.

> **Information**
>
> Etant donné que tous les paramètres doivent être transférés sous forme de texte, l'on doit faire attention que d'éventuels guillemets soient codés correctement. Sinon, des messages d'erreur peuvent apparaître, au pire, une SQL Injection sur le serveur SQL.

Importation

Le type et le volume des importations requises dépendent de la base de données; veuillez consulter la documentation correspondante pour tout complément d'information. Si nécessaire, les fichiers JAR utilisés doivent être copiés dans le répertoire du service ELOas.

Voici un exemple pour les importations nécessaires de la bridge JDBC-ODBC:

```
importPackage(Packages.sun.jdbc.odbc);
```

Dans le module `Imports` de ELOas Libraries 12, un système standard a été introduit. Le sélecteur du système standard a utilisé la valeur standard `SordC.mbLean` pour des raisons de performance et est utilisé pour le traitement des règles ELOas standards.

```
const EM_SYS_STDSEL = SordC.mbLean;
```

Par ailleurs, un sélecteur système du nom `EM_SYS_SELECTOR` a été introduit dans le module `Imports`. Le sélecteur système est configuré conformément à la valeur du sélecteur standard dans le module `bt`. Dans l'événement `onStart`, le sélecteur système peut utiliser et traiter les autres propriétés de l'entrée, mis à part l'ID et le nom.

```
EM_SYS_SELECTOR=SordC.mbAll;
```

De même, les constantes du processus ont été élargies en terme de sélecteur de proccessus du nom `EM_WF_SELECTOR` :

```
var EM_WF_SELECTOR = SordC.mbLean;
```

Paramètres de connexion

Les paramètres de connexion à la base de données sont déposés dans le module Imports. Il existe une liste de connexions, qui peuvent être appelées ultérieurement avec leur numéro de connexion (commençant par 0).

```
var EM_connections = [
    {
        driver: 'sun.jdbc.odbc.JdbcOdbcDriver',
        url: 'jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\\Temp\\EMDemo.mdb',
        user: '',
        password: '',
        initdone: false,
        classloaded: false,
        dbcn: null
    },
    {
        driver: 'com.microsoft.sqlserver.jdbc.SQLServerDriver',
        url: 'jdbc:sqlserver://srvt02:1433',
        user: 'elodb',
        password: 'elodb',
        initdone: false,
        classloaded: false,
        dbcn: null
    }
];
```

Les informations suivantes doivent être déposées pour chaque connexion:

driver	Nom de classe JDBC pour la connexion à la base de données Vous obtiendrez cette information par le fabricant du pilote JDBC ou par le créateur de la base de données.
url	URL d'accès à la base de données. C'est ici que sont déposés les paramètres de connexion dépendants de la base de données, par exemple les chemins de fichiers pour les bases de données Access ou les noms de serveur et ports pour les bases de données SQL. Ces paramètres de connexion dépendent du fabricant et doivent être lus dans la documentation correspondante.
user	Nom d'authentification pour l'accès à la base de données. Ce paramètre n'est pas utilisé par toutes les bases de données (par exemple par des bases de données Access qui ne sont pas protégées). Dans ce cas, le paramètre peut rester vide.
password	Mot de passe pour l'authentification à la base de données.
initdone	Variable interne pour la “lazy initialization”.
classloaded	Variable interne permettant de contrôler si le fichier de classe a déjà été chargé.
dbcn	Variable interne d'enregistrement de l'objet de connexion à la base de données.

JavaScript-Code

La routine `dbInit` est seulement appelée à l'intérieur du module. Elle est appelée avant chaque accès à la base de données et vérifie si une connexion a déjà été établie; et la crée si nécessaire.

```
function dbInit(connectId) {
  if (EM_connections[connectId].initdone == true) {
```

```
    return
  }

  log.debug("Now init JDBC driver")
  var driverName = EM_connections[connectId].driver
  var dbUrl = EM_connections[connectId].url
  var dbUser = EM_connections[connectId].user
  var dbPassword = EM_connections[connectId].password
  try {
    if (!EM_connections[connectId].classloaded) {
      Class.forName(driverName).newInstance()

      log.debug("Register driver ODBC")
      DriverManager.registerDriver(new JdbcOdbcDriver())
      EM_connections[connectId].classloaded = true
    }

    log.debug("Get Connection")
    EM_connections[connectId].dbcn = DriverManager.getConnection(
      dbUrl,
      dbUser,
      dbPassword
    )

    log.debug("Init done.")
  } catch (e) {
    log.debug("ODBC Exception: " + e)
  }

  EM_connections[connectId].initdone = true
}
```

La fonction `exitRuleset_DB_Access()` est appelée automatiquement après la terminaison du traitement Ruleset. Elle vérifie si une connexion existe, puis la referme. Ce contrôle doit exister pour toutes les bases de données configurées.

```
function exitRuleset_DB_Access() {
  log.debug("dbExit")

  for (i = 0; i < EM_connections.length; i++) {
    if (EM_connections[i].initdone) {
      if (EM_connections[i].dbcn) {
        try {
          EM_connections[i].dbcn.close()
          EM_connections[i].initdone = false
          log.debug("Connection closed: " + i)
        } catch (e) {
          log.info("Error closing database " + i + ": " + e)
        }
```

```
      }
    }
  }
}
```

La fonction `getLine()` lit une ligne dans la base de données avec des colonnes quelconques et place les résultats dans un objet JavaScript. Cet objet contient, pour chaque colonne, une variable Member avec le nom de colonne.

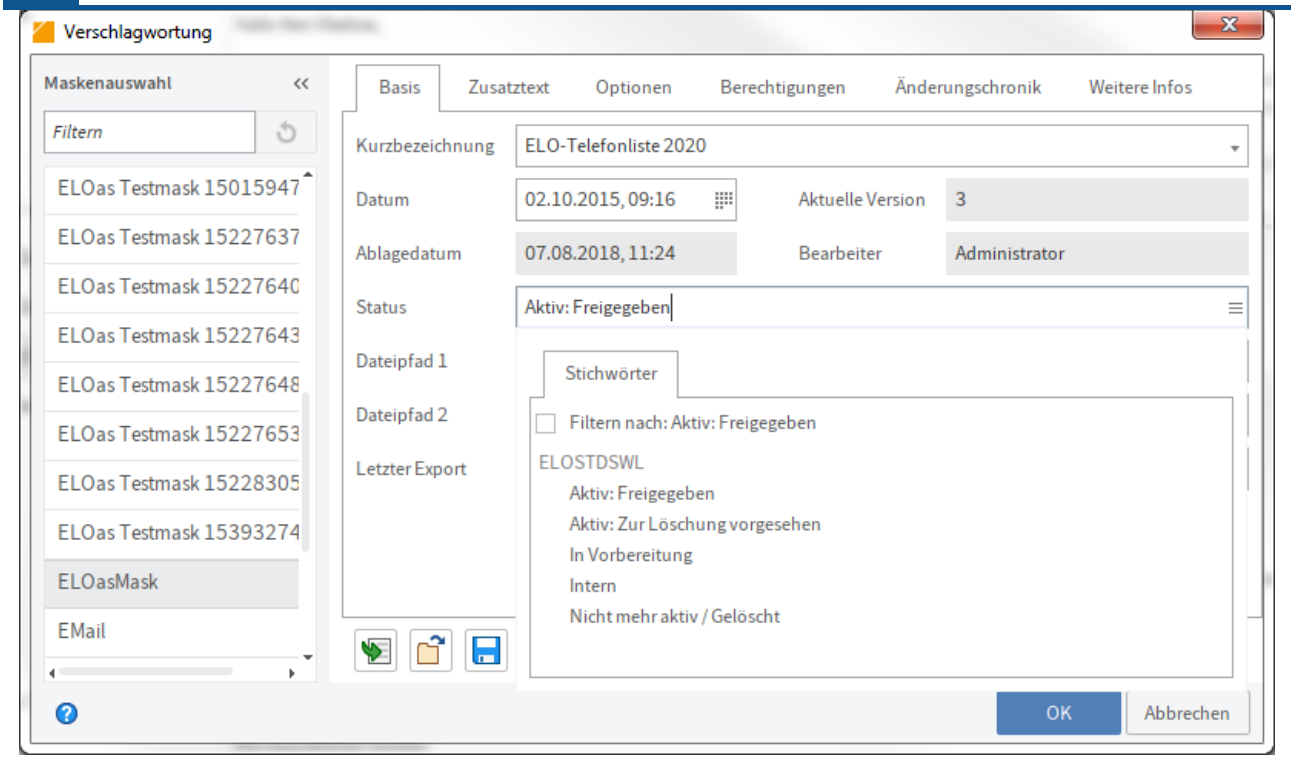
```
function getLine(connection, qry) {
  // la sous-fonction crée un objet JavaScript avec

  // le contenu de la base de données ayant été enregistré
  function dbResult(connection, qry) {
    // Tout d'abord établir la connexion
    dbInit(connection)

    // puis créer un objet SQL Statement
    var p = EM_connections[connection].dbcn.createStatement()

    // et exécuter la requête
    var rss = p.executeQuery(qry)

    // rss contient la liste des résultats, la première
    // ligne est enregistrée
    if (rss.next()) {
      // le nombre de colonnes est déterminé via les métadonnées
      var metaData = rss.getMetaData()
      var cnt = metaData.getColumnCount()

      // Une member variable est créée pour chaque colonne
      // Celle-ci porte le nom du nom de la colonne SQL, et en tant que valeur

      // le contenu de la base de données qui a été enregistré.
      // La première colonne peut toujours être gérée via le nom
      // first.
      for (i = 1; i <= cnt; i++) {
        var name = metaData.getColumnName(i)
        var value = rss.getString(i)

        this[name] = value
        if (i == 1) {
          this.first = value
        }
      }
    }

    // Ensuite, la liste des résultats et le statement
    // SQL sont fermés.
```

```
    rss.close()
    p.close()
  }

  // voici le démarrage de la fonction. Un
  // objet JavaScript avec le contenu de la base de données
  // est exigé.
  var res = new dbResult(connection, qry)

  return res
}

// La fonction getColumn est uen variante spéciale de
// l'appel getLine. La requête SQL ne doit
// montrer qu'une colonne en tant que résultat. S'il existe
// d'autres colonnes, celles-ci sont ignorées,
// tout comme les lignes supplémentaires.
function getColumn(connection, qry) {
  var res = getLine(connection, qry)
  return res.first
}
```

dex: Document Export

Le module *Document Export* peut exporter des documents dans le système de fichiers depuis l'archive. Cette exportation n'est pas un processus unique – si une nouvelle version de document est créée, le module écrit automatiquement un fichier actualisé. Les fichiers déjà publiés peuvent être supprimés. Pour des raisons de sécurité, les fichiers se trouvent seulement dans un chemin pré-configuré.

Un masque de dépôt doit être défini, qui contient le statut de document et une ou plusieurs cibles de dépôt dans le système de fichiers. En plus, le numéro de document de la dernière exportation est enregistré dans le masque.



Illustr. : champ de statut dans le masque

Le champ de statut détermine les actions devant être exécutées. Avec *Actif: autorisé*, le fichier est authentifié à l'exportation. *Actif : prévu pour la suppression* fait que le fichier est supprimé du système de fichiers et que le statut est placé sur *Plus actif / supprimé*. Tous les autres réglages de statut n'engendrent pas d'action de ELOas et sont prévus pour les documents internes ou les documents n'ayant pas encore été autorisés. Etant donné que cette valeur est demandée par le traitement interne, il est sensé de remplir cette ligne seulement à partir d'une liste de mots-clés pré-configurée.

Les champs Chemin de fichier 1..5 contiennent le chemin et le nom de fichier du document dans le système de fichiers. Il s'agit d'un chemin relatif, la partie de démarrage est définie dans le module JavaScript comme `dexRoot` et peut y être ajustée. Cette partie fixe est prévue par sécurité, sinon, des fichiers quelconques peuvent être écrasés par des saisies erronées.

Le champ *Dernière exportation* contient le numéro de document de la dernière version de fichier exportée. Si une nouvelle version de fichier a été créée après un traitement, le module reconnaît ceci et une copie du fichier est écrite dans le système de fichiers. Ensuite, ce champ est actualisé.

Si une erreur est apparue lors du traitement, le texte "ERROR" est déposé dans le champ *Dernière exportation*, grâce à la règle d'erreurs. L'on peut donc créer un registre dynamique dans ELO, celui-ci vérifie le champ quant à la valeur ERROR et peut afficher une liste actuelle de tous les documents qui ne peuvent pas être exportés.

Exemple pour un registre dynamique si le masque possède l'ID 22:

```
!+ , objkeys where objmask = 22 and objid = parentid and okeyname ='PDEXPORT
        and okeydata ='ERROR'
```

dex : fonctions disponibles

Le module met à disposition seulement une fonction : `processDoc`. Celui reçoit l'objet SORD du serveur d'indexation en tant que paramètre et vérifie selon le statut si le fichier doit être exporté ou supprimé; puis il exécute l'action correspondante. Le nouvel identificateur de document est transféré en tant que valeur de retour. L'objet SORD actuel est disponible au sein d'un traitement de règle dans la variable JavaScript `Sord`.

Exemple dans le code XML Ruleset:

```
<rule>
    <name>Regel 1</name>
    <condition>(PDEXPORT != Sord.getDoc()) &amp;&amp; (PDEXPORT != "ERROR") ||
               (PDSTATUS == "Actif : suppression") </condition>
    <index>
    <name>PDEXPORT</name>
    <value>dex.processDoc(Sord)</value>
    </index>
</rule>
```

dex : code JavaScript

En premier, le chemin de base `docRoot` est déterminé pour le dépôt de document. Le chemin cible est toujours déterminé à partir de ce réglage et à partir de l'entrée utilisateur dans le masque de dépôt. Il serait possible de laisser ce chemin de base vide, de façon à ce que l'utilisateur puisse entrer des chemins quelconques. Ce procédé risquerait d'engendre des problèmes de sécurité, étant donné que chaque utilisateur est en mesure d'écraser des fichiers quelconques dans la zone d'accès de ELOas.

```
var dexRoot = "c:\\temp\\"
```

La fonction `processDoc` est appelée à partir de la définition des règles. Ici, est vérifié le statut de l'objet SORD du serveur d'indexation et la fonction requise est appelée.

```
function processDoc(Sord) {
  log.debug("Statut: " + PDSTATUS + ", Name: " + NAME)

  if (PDSTATUS == "Actif : prévu pour la suppression") {
    return dex.deleteDoc(Sord)
  } else if (PDSTATUS == "actif : validé") {
    return dex.exportDoc(Sord)
  }

  return ""
}
```

Si le statut était placé sur "supprimer", la suppression des fichiers est déclenchée dans la fonction `deleteDoc`, et le statut est permuté en "supprimé".

```
function deleteDoc(Sord) {
  dex.deleteFile(PDPATH1)
  dex.deleteFile(PDPATH2)
  dex.deleteFile(PDPATH3)
  dex.deleteFile(PDPATH4)
  dex.deleteFile(PDPATH5)

  PDSTATUS = "Plus actif / supprimé"
  return Sord.getDoc()
}
```

La fonction `deleteFile` exécute la suppression. Elle vérifie tout d'abord si un nom de fichier est configuré et si le fichier existe, et le supprime dans le système de fichiers.

```
function deleteFile(destPath) {
  if (destPath == "") {
    return
  }

  var file = new File(docRoot + destPath)
  if (file.exists()) {
    log.debug("Delete expired version: " + docRoot + destPath)

    file["delete"]()
  }
}
```

S'il s'agit d'écrire une nouvelle version de fichier, la fonction interne exportDoc est appelée. Le fichier est cherché par le gestionnaire de documents et copié dans le classeur cible.

```
function exportDoc(Sord) {
  var editInfo = ixConnect
    .ix()
    .checkoutDoc(Sord.getId(), null, EditInfoC.mbSordDoc, LockC.NO)
  var url = editInfo.document.docs[0].getUrl()
  dex.copyFile(url, PDPATH1)
  dex.copyFile(url, PDPATH2)
  dex.copyFile(url, PDPATH3)
  dex.copyFile(url, PDPATH4)
  dex.copyFile(url, PDPATH5)

  return Sord.getDoc()
}
```

La fonction `copyFile` exécute le processus de copie dans le classeur cible. Il est tout d'abord vérifié s'il existe un nom du fichier cible et si une version ancienne éventuellement existante

doit être supprimée. Ensuite, la nouvelle version est cherchée par le gestionnaire de documents et enregistrée dans le classeur cible.

```
function copyFile(url, destPath) {
    if (destPath == "") {
        return;
    }

    log.debug("Path: " + docRoot + destPath);


    var file = new File(docRoot + destPath);
    if (file.exists()) {
        log.debug("Delete old version.");
        file["delete"](#ELODOC-D50FBC7EA85D4A709D2C12762E1B9F300);
}
```

ix : Fonctions du serveur d'indexation

Le module ix contient un regroupement de plusieurs fonctions du serveur d'indexation, qui sont souvent requises dans les scripts. Il s'agit de simples wrapper placés autour de la commande du serveur d'indexation, il ne s'agit pas d'une nouvelle fonctionnalité complexe.

ix : fonctions disponibles

Suppression d'une entrée SORD : la fonction `deleteSord()` est dotée de paramètre, à savoir les ID d'objet de l'entrée SORD qu'il s'agit de supprimer, ainsi que de l'entrée parent.

```
deleteSord: function (parentId, objId) {


    log.info("Delete SORD: ParentId = " + parentId + ", ObjectId = " + objId);


    return ixConnect.ix().deleteSord(parentId, objId, LockC.NO, null);
},
```

Rechercher une entrée : la fonction `lookupIndex()` détermine l'ID d'objet d'une entrée qui est trouvée via le chemin de dépôt. Le paramètre `archivePath` doit commencer par un tiret.

```
lookupIndex: function (archivePath) {


    log.info("Lookup Index: " + archivePath);


    var editInfo = ixConnect.ix().checkoutSord("ARCPATH:" + archivePath, EditInfoC.mbOnlyId,
    if (editInfo) {
```

```
        return editInfo.getSord().getId();
    }   else {
        return 0;
    }
}
```

Rechercher une entrée: la fonction `lookupIndexByLine()` détermine l'ID d'objet d'une entrée selon la recherche d'une ligne d'indexation. Si le paramètre Mask IDest transféré avec un string vide, alors une recherche sur plusieurs masques est effectuée. Le nom de groupe et le terme de recherche doivent être transférés.

```
lookupIndexByLine : function(maskId, groupName, value) {
    var findInfo = new FindInfo();
    var findByIndex = new FindByIndex();
    if (maskId != "") {
        findByIndex.maskId = maskId;
    }

    var objKey = new ObjKey();
    var keyData = new Array(1);
    keyData[0] = value;
    objKey.setName(groupName);
    objKey.setData(keyData);

    var objKeys = new Array(1);
    objKeys[0] = objKey;

    findByIndex.setObjKeys(objKeys);
    findInfo.setFindByIndex(findByIndex);

    var findResult = ixConnect.ix().findFirstSords(findInfo, 1, SordC.mbMin);
    ixConnect.ix().findClose(findResult.getSearchId());

    if (findResult.sords.length == 0) {
        return 0;
    }

    return findResult.sords[0].id;
},
```

Lecture des informations plein texte : la fonction `getFulltext()` livre l'information plein texte actuelle pour un document. Le plein texte est rendu en tant que string.

Remarque

Il n'est pas possible de reconnaître s'il n'existe pas de plein texte, ou si le traitement du plein texte est complètement clos ou si l'action a été annulée avec une erreur. C'est le texte existant au moment de la demande qui est livré (s'il n'existe pas d'informations plein texte, il s'agit alors d'un string vide).

```
getFulltext: function(objId) {
    var editInfo = ixConnect.ix().checkoutDoc(objId, null,
                                              EditInfoC.mbSordDoc, LockC.NO);
    var url = editInfo.document.docs[0].fulltextContent.url
    var ext = "." + editInfo.document.docs[0].fulltextContent.ext
    var name = fu.clearSpecialChars(editInfo.sord.name);

    var temp = File.createTempFile(name, ext);
    log.debug("Temp file: " + temp.getAbsolutePath());

    ixConnect.download(url, temp);
    var text = FileUtils.readFileToString(temp, "UTF-8");
    temp["delete"]();

    return text;

}
```

Création d'une liste de classeur: la fonction `createSubPath()` vérifie si le chemin de classeur existe dans l'archive et crée automatiquement les parties manquantes si nécessaire.

```
createSubPath: function (startId, destPath, folderMask) {
    log.debug("createPath: " + destPath);

    try {
        var editInfo = ixConnect.ix().checkoutSord("ARCPATH:" + destPath,
                                                   EditInfoC.mbOnlyId, LockC.NO);
        log.debug("Path found, GUID: " + editInfo.getSord().getGuid() +
                  " ID: " + editInfo.getSord().getId());
        return editInfo.getSord().getId();;
    }   catch (e) {
        log.debug("Path not found, create new: " + destPath +
                  ", use foldermask: " + folderMask);
    }

    items = destPath.split("¶");
    var sordList = new Array(items.length - 1);
    for (var i = 1; i < items.length; i++) {
    log.debug("Split " + i + " : " + items[i]);
    var sord = new Sord();
    sord.setMask(folderMask);
```

```
    sord.setName(items[i]);

    sordList[i - 1] = sord;
    }

    log.debug("now checkinSordPath");
    var ids = ixConnect.ix().checkinSordPath(startId, sordList,
            new SordZ(SordC.mbName | SordC.mbMask));
    log.debug("checkin done: id: " + ids[ids.length - 1]);

    return ids[ids.length - 1];
}
```

wf: Workflow Utils

Le module *wf* contient des accès plus faciles aux données de processus. Il existe deux groupes de fonctions.

Les fonctions 'high level' `changeNodeUser` et `readActiveWorkflow` sont à utiliser pour l'accès simple d'un traitement de processus en cours et travaillent avec le processus actuellement actif. Elles sont très simples à utiliser, mais elles exécutent seulement une fonction simple.

Les fonctions low leven `readWorkflow`, `writeWorkflow`, `unlockWorkflow` et `getNodeByName` peuvent être utilisées à partir de n'importe où. Si plusieurs modifications doivent être effectuées dans le même processus, l'on peut s'assurer que le processus est seulement lu et écrit une fois, et non pas x-fois pour chaque opération.

wf : Fonctions disponibles

Modifier le nom utilisateur d'un noeud de personne : la fonction `changeNodeUser()`: permute, dans le processus actuel dans le noeud de processus du nom `nodeName` l'utilisateur avec un nouvel utilisateur `nodeUserName`

Etant donné que cet appel lit toujours le processus intégral, le modifie et le ré-écrit automatiquement, cet appel simple devrait n'être utilisé que si un seul noeud doit être modifié. Si plusieurs modifications sont nécessaires, les fonctions décrites ci-dessous devraient être utilisées pour lire, modifier et enregistrer un processus.Etant donné que cette fonction détermine l'ID du processus dans le processus actuellement actif, il peut seulement être appelé à partir de la recherche „WORKFLOW“. Lors de l'utilisation dans un TREEWALK ou une recherche normale, un ID de processus définie au hasard est utilisée.

```
changeNodeUser: function(nodeName, nodeUserName) {
    var diag = wf.readActiveWorkflow(true);
    var node = wf.getNodeByName(diag, nodeName);
    if (node) {
        node.setUserName(nodeUserName);
        wf.writeWorkflow(diag);
    }   else {
        wf.unlockWorkflow(diag);
```

```
    }
}
```

Copier le nom utilisateur d'un noeud: la fonction 'copyNodeUser()' est semblable à 'changeNodeUser', mais elle copie le nom utilisateur d'un noeud dans un autre noeud.

```
copyNodeUser: function(sourceNodeName, destinationNodeName) {
    var diag = wf.readActiveWorkflow(true);
    var sourceNode = wf.getNodeByName(diag, sourceNodeName);
    var destNode = wf.getNodeByName(diag, destinationNodeName);

    if (sourceNode && destNode) {
        var user = sourceNode.getUserName();
        destNode.setUserName(user);
        wf.writeWorkflow(diag);

        return user;
    }   else {
        wf.unlockWorkflow(diag);
        return null;
    }
}
```

Importer le processus actuel : la fonction `readActiveWorkflow()` permet d'importer le processus actif actuellement dans une variable locale, en vue d'un traitement. A la fin, il peut être écrit avec `writeWorkflow`, ou bien le verrouillage peut être autorisé avec `unlockWorkflow`.

```
readActiveWorkflow: function(withLock) {
    var flowId = EM_WF_NODE.getFlowId();
    return wf.readWorkflow(flowId, withLock);
    },
```

Importer le processus : la fonction `readWorkflow()` importe un processus dans une variable locale. Celui-ci peut être évalué et modifié. Si les modifications doivent être enregistrées, alors celles-ci peuvent être ré-écrites par `writeWorkflow`. Si le processus a été lu avec Lock, mais que les modifications ne doivent pas être enregistrées, le verrouillage peut être retiré avec `unlockWorkflow`.

```
readWorkflow: function(workflowId, withLock) {
    log.debug("Read Workflow Diagram, WorkflowId = " + workflowId);
    return ixConnect.ix().checkoutWorkFlow(String(workflowId),
                                           WFTypeC.ACTIVE,
                                           WFDiagramC.mbAll,
                                           (withLock) ? LockC.YES : LockC.NO);
},
```

Exporter le processus : la fonction `writeWorkflow()` exporte le processus depuis une variable locale dans la base de données. Un verrouillage d'écriture éventuellement existant est ré-initialisé automatiquement.

```
writeWorkflow: function(wfDiagram) {
    ixConnect.ix().checkinWorkFlow(wfDiagram, WFDiagramC.mbAll, LockC.YES);
},
```

Annuler le verrouillage de lecture : fonction `unlockWorkflow()`. Si un processus avec verrouillage de lecture a été lu, mais qu'il ne doit pas être modifié, l'on peut annuler le verrouillage grâce à `unlockWorkflow`.

```
unlockWorkflow: function(wfDiagram) {
    ixConnect.ix().checkinWorkflow(wfDiagram, WFDiagramC.mbOnlyLock, LockC.YES);
},
```

Rechercher un noeud de processus : la fonction 'getNodeByName()' recherche le noeud de processus pour un nom de noeud. Le nom doit être univalent, sinon, c'est le premier noeud trouvé qui est livré.

```
getNodeByName: function(wfDiagram, nodeName) {
    var nodes = wfDiagram.getNodes();
    for (var i = 0; i < nodes.length; i++) {
        var node = nodes[i];
        if (node.getName() == nodeName) {
            return node;
        }
    }

    return null;
},
```

Démarrer un processus depuis le modèle : la fonction `startWorkflow()` démarre un nouveau processus vers und ID d'objet ELO depuis un modèle de processus.

```
startWorkflow: function(templateName, flowName, objectId) {
    return ixConnect.ix().startWorkFlow(templateName, flowName, objectId);
}
```

mail: Mail Utils

Ce module sert à envoyer des e-mails. A ces fins, il requiert un hôte SMTP par le biais duquel les mails peuvent être envoyés. Celui-ci doit être indiqué avant le premier envoi de mail par le biais de la fonction `setSmtpHost`. Ensuite, l'on peut envoyer des messages par `SendMail` ou `SendMailWithAttachment`. Le module se compose de deux parties, pour envoyer des messages et pour lire les boîtes de réception de messagerie.

mail : Fonctions disponibles pour lire une boîte de réception

Dans le ruleset, vous pouvez définir que l'action de base n'est pas une recherche dans l'archive ELO ou dans la liste des tâches ELO, mais que c'est une boîte de réception qui doit être parcourue. Pour chaque type de boîte de réception, une routine d'authentification doit être déposée dans le module mail. Dans cette fonction, le serveur mail doit être contacté, le classeur Mail souhaité doit être recherché et la liste des messages lue. Ensuite, le traitement ELOas normal prend les commandes. Pour chaque mail, un document est préparé dans le classeur, défini dans SEARCHVALUE, puis le ruleset est exécuté (l'objet du message est automatiquement adopté dans la désignation). Si l'entrée n'est pas enregistrée à la fin, alors on ne trouve rien dans l'archive en conséquence. Seuls les mails enregistrés sont transférés dans l'archive.

```
<search>
<name>"MAILBOX_GMAIL"</name>
<value>"ARCPATH:¶ELOas¶IMAP"</value>
<mask>2</mask>
```

Dans le ruleset, le nom doit être défini en tant que nom `MAILBOX_<nom de la connexion>` et en tant que valeur, le chemin d'archive ou le numéro du classeur cible. Par ailleurs, le masque à utiliser pour les nouveaux documents doit être défini.

Le mail est traité dans le script du ruleset. Ici aussi, le module *mail* propose quelques routines qui vous simplifient la vie. Dans l'exemple suivant, le corps du message est transféré dans le texte supplémentaire, l'expéditeur, le destinataire et l'ID du mail sont transférés dans les champs correspondants du masque e-mail :

```
OBJDESC = mail.getBodyText(MAIL_MESSAGE);
ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
ELOOUTL3 = msgId;
EM_WRITE_CHANGED = true;
```

Si des valeurs ou informations complémentaires doivent être utilisées, un objet JavaMailMimeMessage Message est disponible dans la variable `MAIL_MESSAGE`.

Afin que les mails déjà traités ne soient pas transférés deux fois dans l'archive, une recherche de l'ID du mail doit être effectuée avant le traitement. Si le mail est déjà dans l'archive, la variable `MAIL_ALLOW_DELETE` est placée sur true, sinon, le mail est traité. En plaçant le flag de suppression, le mail est supprimé de la boîte de réception ou marqué comme traité.

```
var msgId = MAIL_MESSAGE.messageID;
if (ix.lookupIndexByLine(EM_SEARCHMASK, "ELOOUTL3", msgId) != 0) {
    log.debug("Le mail existe déjà dans l'archive, ignorer ou supprimer");
    MAIL_ALLOW_DELETE = true;
}   else {
    OBJDESC = mail.getBodyText(MAIL_MESSAGE);
    ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
```

```
    ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
    ELOOUTL3 = msgId;
    EM_WRITE_CHANGED = true;
}
```

Ce procédé lit un message deux fois (une fois pour le traitement normal, et une fois pour la suppression), mais cette méthode a l'avantage considérable que le mail est seulement supprimé de la boîte de réception s'il existe réellement dans l'archive.

Si vous souhaitez utiliser une boîte de réception pour la surveillance, les quatre fonctions suivantes sont requises dans la librairie JavaScript :

établir une connexion, ouvrir les répertoires de la boîte de réception : `connectImap_<nom de la connexion>`

Prochain message de la liste devant être traité : `nextImap_<nom de la connexion>`

Supprimer le message ou le marquer comme étant traité : `finalizeImap_<om de la connexion>`

Fermer la connexion : `closeImap_<nom de la connexion>`

De ces quatre fonctions, seule une doit être implémentée dans les cas simples : établir la connexion – `connectImap_<nom de la connexion>`. Etant donné qu'une multitude d'actions spécifiques aux projets existent (paramètres d'authentification, rechercher le classeur cible), il n'existe pas d'intégration standard. Les trois autres fonctions existent déjà avec une fonction standard dans le système. Elles doivent seulement être intégrées si l'on souhaite exécuter des fonctions complémentaires.

Connecter à IMAP Server : la fonction `connectImap_<nom de la connexion>()` doit établir une connexion au serveur de messagerie et extraire les informations dans la boîte de réception souhaitée. Les messages existants sont déposés dans la variable `MAIL_MESSAGES`. Le store Mail doit être enregistré dans les variables `MAIL_STORE` et le classeur lu dans la variable `MAIL_INBOX`. Ces deux variables sont requises à la fin du traitement pour fermer la connexion. La variable `MAIL_DELETE_ARCHIVED` détermine si l'on peut effectuer des suppressions dans la boîte de réception. Si elle est placée sur 'false', les exigences de suppression du ruleset sont ignorées. Cette fonction n'est pas directement appelée par le biais d'un script, elle est activée dans ELOas (pour la recherche MAILBOX, dans l'exemple `MAILBOX_GMAIL`).

```
connectImap_GMAIL: function() {
    var props = new Properties();
    props.setProperty("mail.imap.host", "imap.gmail.com");
    props.setProperty("mail.imap.port", "993");
    props.setProperty("mail.imap.connectiontimeout", "5000");
    props.setProperty("mail.imap.timeout", "5000");
    props.setProperty("mail.imap.socketFactory.class",
                      "javax.net.ssl.SSLSocketFactory");
    props.setProperty("mail.imap.socketFactory.fallback", "false");
    props.setProperty("mail.store.protocol", "imaps");

    var session = Session.getDefaultInstance(props);
    MAIL_STORE = session.getStore("imaps");
```

```
    MAIL_STORE.connect("imap.gmail.com", "<<<USERNAME>>>@gmail.com",
                       "<<<PASSWORT>>>");
    var folder = MAIL_STORE.getDefaultFolder();
    MAIL_INBOX = folder.getFolder("INBOX");
    MAIL_INBOX.open(Folder.READ_WRITE);
    MAIL_MESSAGES = MAIL_INBOX.getMessages();
    MAIL_POINTER = 0;
    MAIL_DELETE_ARCHIVED = false;
},
```

Fermer la connexion : la fonction `closeImap_<nom de la connexion'>` est en option et permet de fermer la connexion actuelle vers le serveur IMAP. S'il n'existe pas de tâches spécifiques lors de la fermeture, vous ne devez pas implémenter cette fonction. Au lieu de cela, l'implémentation standard 'cloImap()' de la library est utilisée. Celle-ci ferme le classeur et le store.

```
closeImap_GMAIL: function() {
    // ici, vous pouvez exécuter vos propres actions avant la fermeture

    // action standard, refermer le classeur et le store.
    MAIL_INBOX.close(true);
    MAIL_STORE.close();
},
```

Marquer le message comme traité ou le supprimer : la fonction `finalizeImap_<nom de la connexion>()` est en option et supprime le message actuel ou le marque comme déjà traité d'une autre manière. Si elle n'est pas intégrée, ELOam utilise l'intégration standard qui supprime un mail traité dans le classeur.

L'exemple ne supprime pas l'e-mail, mais le marque comme "lu".

```
finalizeImap_GMAIL: function() {
    if (MAIL_DELETE_ARCHIVED && MAIL_ALLOW_DELETE) {
        message.setFlag(Flags.Flag.SEEN, true);
    }
},
```

Traiter le prochain message de la liste : la fonction `nextImap_<nom de connexion>` : cette fonction est en option et livre au ruleset le prochain message de la boîte de réception sélectionnée en vue d'un traitement. Si la fonction n'est pas intégrée, ELOas utilise l'intégration standard qui donne chaque document en traitement.

L'exemple montre une intégration qui traite seulement les mails non-lus. Elle peut être utilisée dans l'intégration finalizeImap nommée ci-dessus, elle ne supprime pas les mails traités, mais qui les marque comme étant lus.

> **Remarque**

> Si vous travaillez avec cette méthode, vous devez vous assurer que la boîte de réception ne soit pas trop volumineuse (par exemple par une suppression automatique après une durée précise).

```
nextImap_GMAIL: function() {
    if (MAIL_POINTER > 0) {
        mail.finalizePreviousMessage(MAIL_MESSAGE);
    }

    for (;;) {
        if (MAIL_POINTER >= MAIL_MESSAGES.length) {
            return false;
        }

        MAIL_MESSAGE = MAIL_MESSAGES[MAIL_POINTER];

        var flags = MAIL_MESSAGE.getFlags();
        if (flags.contains(Flags.Flag.SEEN)) {
            MAIL_POINTER++;
            continue;
        }

        MAIL_ALLOW_DELETE = false;
        MAIL_POINTER++;
        return true;
    }

    return false;
},
```

Lire le texte du corps du message : la fonction `getBodyText()` reçoit le message en tant que paramètre (disponible dans le script par le biais de la variable MAIL_MESSAGE) et livre le corps du mail en tant que paramètre de retour. Le premier MIME Part du type TEXT/PLAIN est recherché. S'il n'existe pas de partie correspondante, un string vide est livré.

```
getBodyText: function(message) {
    var content = message.content;
    if (content instanceof String) {
        return content;
    }   else if (content instanceof Multipart) {
        var cnt = content.getCount();
        for (var i = 0; i < cnt; i++) {
            var part = content.getBodyPart(i);
            var ct = part.contentType;
            if (ct.match("^TEXT/PLAIN") == "TEXT/PLAIN") {
                return part.content;
            }
        }
```

```
    }

    return "";
},
```

Déterminer l'expéditeur : la fonction 'getSender()' livre l'adresse e-mail de l'expéditeur.

```
getSender: function(message) {
    var adress = message.sender;
    return adress.toString();
},
```

Déterminer le destinataire : la fonction `getRecipients()` livre une liste de tous les destinataires (TO et CC). S'il existe plus d'un destinataire, la liste est livrée au format de l'index de colonne, si l'on transfère le symbole de séparation ELO ¶ dans le paramètre `delimiter`.

```
getRecipients: function(message, delimiter) {
    var adresses = message.allRecipients;

    var cnt = 0;
    if (adresses) { cnt = adresses.length; }
    var hasMany = cnt > 1;

    var result = "";
    for (var i = 0; i < cnt; i++) {
        if (hasMany) { result = result + delimiter; }
        result = result + adresses[i].toString();
    }

    return result;
}
```

Fonctions disponibles pour l'envoi de mails

La fonction envoyer n'est pas utilisée directement par ELOas. Il s'agit de fonctions permettant de simplifier la programmation de script pour masquer la complexité de l'API JavaMail devant le développeur de script.

Authentifier le serveur SMTP : la fonction `setSmtpHost()` indique à la librairie l'hôte SMTP devant être utilisé. Il est utilisé pour l'envoi mail. Cette fonction doit être activée avant le premier appel `sendMail`.

```
setSmtpHost: function(smtpHost) {
    if (MAIL_SMTP_HOST != smtpHost) {
        MAIL_SMTP_HOST = smtpHost;
        MAIL_SESSION = undefined;
    }
},
```

Envoyer un mail : la fonction sendMail() envoie un e-mail. En tant que paramètre, les adresses des expéditeurs et destinataires sont livrés, ainsi que l'objet et le texte du mail.

```
sendMail: function(addrFrom, addrTo, subject, body) {
    mail.startSession();
    var msg = new MimeMessage(MAIL_SESSION);
    var inetFrom = new InternetAddress(addrFrom);
    var inetTo = new InternetAddress(addrTo);
    msg.setFrom(inetFrom);
    msg.addRecipient(Message.RecipientType.TO, inetTo);
    msg.setSubject(subject);
    msg.setText(body);
    Transport.send(msg);
},
```

Envoyer un mail avec pièce-jointe : la fonction `sendMailWithAttachment()` envoie un mail. En tant que paramètre, l'adresse de l'expéditeur et du destinataire sont livrés, ainsi que l'objet, le texte du mail et l'ID d'objet pour la pièce de l'archive ELO. La pièce-jointe est enregistrée en tant que fichier temporaire dans le chemin temp, il doit donc y avoir assez d'espace.

```
sendMailWithAttachment: function(addrFrom, addrTo, subject, body, attachId) {
    mail.startSession();
    var temp = fu.getTempFile(attachId);
    var msg = new MimeMessage(MAIL_SESSION);
    var inetFrom = new InternetAddress(addrFrom);
    var inetTo = new InternetAddress(addrTo);
    msg.setFrom(inetFrom);
    msg.addRecipient(Message.RecipientType.TO, inetTo);
    msg.setSubject(subject);

    var textPart = new MimeBodyPart();
    textPart.setContent(body, "text/plain");

    var attachFilePart = new MimeBodyPart();
    attachFilePart.attachFile(temp);

    var mp = new MimeMultipart();
    mp.addBodyPart(textPart);
    mp.addBodyPart(attachFilePart);
    msg.setContent(mp);
```

```
    Transport.send(msg);

    temp["delete"]();
}
```

fu: File Utils

Les fonctions de la section File Utils assistent l'utilisateur ELOas lors d'opérations de fichier.

Fonctions disponibles

Purger le nom de fichier : lorsqu'un nom de fichier doit être créé à partir de la désignation, alors celui-ci peut contenir des caractères qui peuvent engendrer des problèmes dans le répertoire des fichiers (par exemple :,\, &). Voilà pourquoi la fonction 'clearSpecialChars' remplace tous les caractères par un underscore, sauf les chiffres et les lettres).

```
clearSpecialChars: function(fileName) {
    var newFileName = fileName.replaceAll("\\W", "_");
    return newFileName;
},
```

Charger le fichier document : la fonction `getTempFile()` charge le fichier document de l'objet ELO indiqué dans le système de fichiers local (dans le classeur temp de ELOas). Si le fichier n'est plus requis, il doit être supprimé par le développeur de script par le biais de la fonction `deleteFile`. Sinon, il reste sur le disque dur.

> **Remarque**
>
> Il ne s'agit pas d'un nom de fichier, mais d'un objet Java File qui est retourné.

```
getTempFile: function(sordId) {
    var editInfo = ixConnect.ix().checkoutDoc(sordId, null,
                                              EditInfoC.mbSordDoc, LockC.NO);
    var url = editInfo.document.docs[0].url;
    var ext = "." + editInfo.document.docs[0].ext;
    var name = fu.clearSpecialChars(editInfo.sord.name);

    var temp = File.createTempFile(name, ext);
    log.debug("Temp file: " + temp.getAbsolutePath());

    ixConnect.download(url, temp);

    return temp;
},
```

Supprimer le fichier : la fonction `deleteFile()` attend un objet Java File comme paramètre (pas de string) et supprime ce fichier.

```
deleteFile: function(delFile) {
    delFile["delete"]();
}
```

run: Runtime Utilities

Ce module contient des routines permettant d'accéder à Java Runtime. A partir de là, vous pouvez démarrer des processus externes ou demander des informations sur l'état actuel de la mémoire.

Démarrer le processus :: la commande `execute(command)` permet de démarrer un processus externe. ELOas attend la clôture de cet appel et poursuit ensuite le traitement. Ainsi, vous pouvez évaluer les actions de ce processus.

```
log.debug("Process: " + NAME );
run.execute("C:\\ Tools\\BAT\\dirlist.bat");
log.debug("Read Result");
var txt = dex.asString("dirlist.txt");
```

Demander une mémoire libre et disponible : les commandes `freeMemory()` et `maxMemory()` servent à afficher la mémoire actuellement disponible et la mémoire disponible au maximum.

```
log.debug "freeMemory: " + run.freeMemory() +
        ", maxMemory: " + run.maxMemory());
```

Exemples

Exemple - Déplacer un document

Un document doit être déplacé dans ELO.

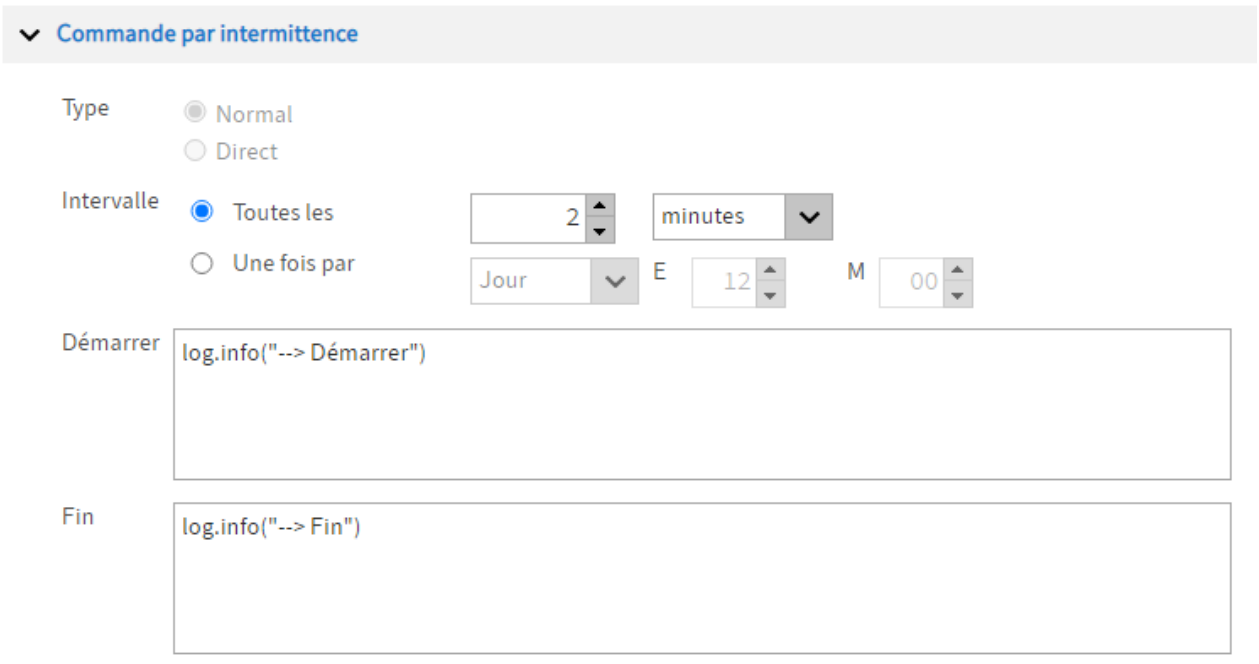
1. Ouvrez la section des *services d'automatisation ELO* dans la console d'administration ELO.
2. Cliquez sur le bouton *Ajouter*.
3. Saisissez un nouveau nom, tel que *Déplacer la newsletter* pour la règle.

   La nouvelle règle est créée, mais pas encore enregistrée.
4. Sélectionnez un masque de recherche.

   Cet exemple utilise un masque du nom *Marketing*.
5. Dans le champ *Indexation*, veuillez sélectionner un champ du masque que vous souhaitez sélectionner pour la sélection des documents.

   Cet exemple utilise le champ *Statut*. Si la valeur *Envoyé* se trouve dans le champ `Statut`, le document doit être déplacé. Les documents ayant d'autres valeurs ne doivent pas être déplacés.
6. En tant que terme de recherche, entrez `"envoyé"`.

> **Remarque**
>
> Si des guillemets sont utilisés dans cet exemple, cela signifie qu'ils sont nécessaires. S'il manque un guillemet ou même les deux, cela entraine une erreur.



Illustr. : commande par intermittence pour une règle

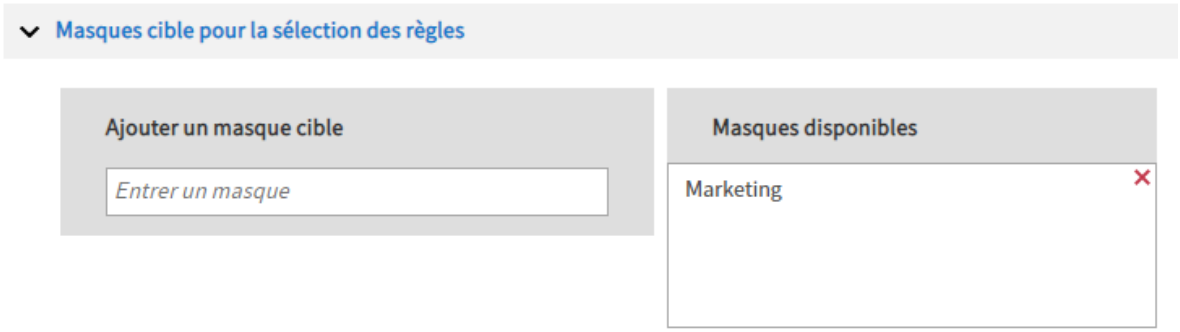
1. Déterminez l'intervalle dans lequel la règle doit à nouveau être exécutée.

   En option : dans les champs *Démarrage* et *Fin*, veuillez entrer les commandes de script si nécessaire qui doivent être exécutées au début de l'exécution ou après l'exécution de la règle.

   Cet exemple utilise la commande `log.info("<texte quelconque>")`, pour marquer le début et la fin de l'exécution de la règle dans le fichier log des ELO Automation Services. Cela peut être d'une grande aide pour l'analyse des erreurs.

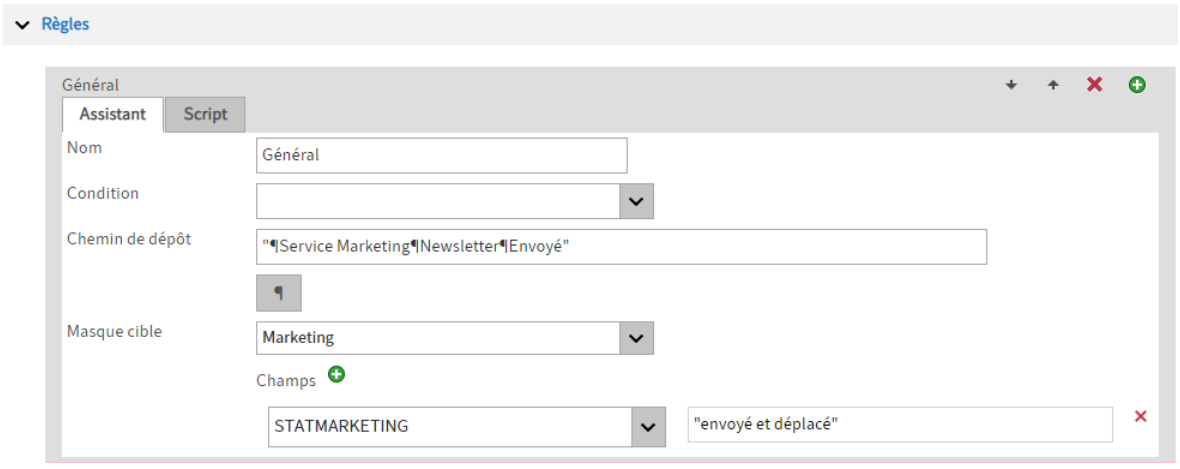
   Le fichier log se trouve dans le chemin suivant :

   `<Chemin d'installation>\logs\<Nom de l'instance de serveur>\as-<Nom du répertoire>.log`



   Illustr. : sélection des masques cibles disponibles

2. Sous *Masques cibles pour la sélection des règles*, entrez le masque utilisé.



   Illustr. : réglages pour la règle

3. Entrez un nom pour la première règle sur l'onglet *Assistant* sous *Règles*.

   Cet exemple utilise le nom *Dépôt*.

4. Dans le champ *Chemin de dépôt*, entrez le chemin cible souhaité.

   Cet exemple utilise le chemin suivant :

`"¶Service Marketing¶Newsletter¶Envoyé"`

5. Sous *Masque cible*, sélectionnez le masque sélectionné en haut.
6. Cliquez sur *Ajouter un champ* (symbole plus vert) et sélectionnez le champ utilisé en-haut.
7. Dans le champ d'entrée, entrez une valeur qui doit être écrite dans le champ du masque. Cela empêche un traitement infini par la règle.

   Cet exemple utilise la valeur : `"envoyé et déplacé"`.

Illustr. : règle pour les erreurs

   En option : veuillez définir une règle pour le cas d'erreurs par le biais de la section *Global Error Rule*.

   Cet exemple utilise le chemin suivant :

   `"¶Service Marketing¶Newsletter¶Erreur"`

   De plus, dans cet exemple, la valeur `"Erreur lors du déplacement"` est entrée dans le champ de statut, pour empêcher une boucle illimitée.

8. Veuillez enregistrer le ruleset.

Illustr. : page de statut de ELO Automation Services, reload

9. Basculez vers la page de statut de ELO Automation Services et cliquez sur *Reload* pour la règle correspondante.

La page de statut de ELO Automation Service peut être trouvée par le biais du Manager du serveur d'application ELO correspondant ou par le biais de l'URL avec le schéma suivant :

```
http(s)://<nom de serveur>:<port>/as-<nom de l'archive>/?cmd=status
```

La règle déplace les documents comportant la chaîne de caractères "envoyé", dans le classeur *Envoyé*.

Exemple - surveiller une boîte à lettres mail

La librairie JavaScript des services d'automation ELO contient un module permettant d'envoyer et de recevoir des e-mails. Voici comment utiliser ELOas pour la surveillance d'une boîte de réception.

> **Information**
>
> Cet exemple n'est pas sensé simuler un archivage mail. Mais nous avons d'autres modules adéquats dans la liste des produits. Il doit servir de base pour „Autoresponder“, donc les programmes qui engendrent une action automatiquement suite à un mail (par exemple un utilisateur envoie un mail d'enregistrement, après quoi son compte est autorisé).

Démarche générale

Avant qu'un ruleset puisse être créé pour le traitement des boîtes de réception, une connexion à la boîte de réception doit être créée dans le module *mail*. Etant donné qu'il existe beaucoup de différences et de possibilités, on ne peut pas travailler avec une liste de configuration simple. Il est nécessaire qu'une méthode `connect` soit créée pour chaque connexion à la boîte de réception. Celle-ci doit établir la connexion au serveur mail, sélectionner la boîte de réception et lire la liste des messages.

Chaque connexion à la boîte de réception contient un nom simple et court, par exemple GMAIL. Ce nom est requis à différents endroits et doit être compatible à l'identificateur, cela signifie qu'il doit commencer par une lettre et peut contenir d'autres lettres ou chiffres. Ce nom est requis à différents endroits dans le ruleset et dans l'implémentation JavaScript.

Etablissement de la connexion

La librairie JavaScript apporte déjà une définition dans l'installation standard, pour une connexion avec le nom GMAIL. Nous allons l'utiliser pour cet exemple. Etant donné que le nom de connexion se retrouve dans les fonctions spéciales, vous pouvez définir plusieurs connexions en parallèle et les utiliser dans différents rulesets.

La fonction standard pour l'établissement de la connexion GMAIL ressemble à ceci:

```
connectImap_GMAIL: function() {


    var props = new Properties();
    props.setProperty("mail.imap.host", "imap.gmail.com");
    props.setProperty("mail.imap.port", "993");
    props.setProperty("mail.imap.connectiontimeout", "5000");
    props.setProperty("mail.imap.timeout", "5000");
    props.setProperty("mail.imap.socketFactory.class",
                      "javax.net.ssl.SSLSocketFactory");
    props.setProperty("mail.imap.socketFactory.fallback", "false");
    props.setProperty("mail.store.protocol", "imaps");
```

```
    var session = Session.getDefaultInstance(props);
    MAIL_STORE = session.getStore("imaps");
    MAIL_STORE.connect("imap.gmail.com",
                       "<BENUTZER>@gmail.com",
                       "<PASSWORT>");
    var folder = MAIL_STORE.getDefaultFolder();
    MAIL_INBOX = folder.getFolder("INBOX");
    MAIL_INBOX.open(Folder.READ_WRITE);
    MAIL_MESSAGES = MAIL_INBOX.getMessages();
    MAIL_DELETE_ARCHIVED = false;
},
```

L'exemple se connecte au serveur Googlemail "imap.gmail.com" sur le port "993" par le biais d'une connexion codée (`mail.store.protocol` – `imaps`). Ces informations sont entrées dans un Property Object. Votre propre serveur mail requiert peut-être d'autres valeurs, celles-ci se trouvent dans les explicatifs concernant le serveur de messagerie.

> **Information**
>
> Si vous créez un compte google mail, vous devez tout d'abord l'autoriser pour un accès IMAP. Cela se fait sous *Réglages* > *Transfert et POP/IMAP* > *Activer IMAP*.

Cette authentification est effectuée par le biais de la commande `MAIL_STORE.connect`. Le nom de serveur doit être entré ainsi que l'utilisateur de la boîte de réception avec mot de passe.

Après l'authentification, c'est le classeur *Boîte de réception* qui est parcouru. Il est bien sûr possible de surveiller d'autres classeurs, par exemple *Envoyé* :

```
MAIL_INBOX = folder.getFolder("[Google Mail]/envoyé")
```

La commande `MAIL_INBOX.getMessages()` permet finalement de lire tous les mails du classeurs et de les lire dans la liste des messages internes. Cette liste est traitée ultérieurement, le ruleset est appelé une fois pour chaque entrée de cette liste.

La variable `MAIL_DELETE_ARCHIVED` détermine si le ruleset doit supprimer le message ou le marqué comme traité, après le traitement. Si son statut est "false" (comme dans le pré-réglage), le statut du message n'est pas modifié. C'est très pratique dans la phase de test, afin que l'on ne doivent pas sans cesse créer de nouveaux mails. Normalement, cette entrée devrait avoir la valeur "true".

Créer un ruleset

Un simple ruleset permet de traiter le contenu de la boîte de réception se compose de deux parties: la définition de la recherche et le script permettant de traiter le mail.

La recherche est définie de la manière suivante :

```
<search>
<name>"MAILBOX_GMAIL"</name>
<value>"ARCPATH:¶IMAP"</value>
<mask>2</mask>
<max>200</max>
</search>
```

Le nom de recherche "MAILBOX_GMAIL" signale qu'il ne s'agit pas d'une recherche dans l'archive normale, mais d'une boîte de réception du nom de connexion GMAIL. Les documents ELO créés sont déposés dans l'armoire "IMAP" (par le biais de `ARCPATH:¶IMAP`) et sont créés avec le masque 2 (e-mail dans une archive ELO standard). Le nombre de résultats n'est pas pertinent en règle générale, mais il devrait être entré afin qu'il n'y ait pas de message d'erreur dans le créateur.

Le script d'exécution est déterminé par la fonction requise. Un cadre simple pourrait ressembler à ceci :

```
<script>
    log.debug("Process Mailbox: " + NAME);


    OBJDESC = mail.getBodyText(MAIL_MESSAGE);
    ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
    ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
    EM_WRITE_CHANGED = true;
    MAIL_ALLOW_DELETE = true;
</script>
```

Lors de l'exécution de script, le message est disponible dans les variables `MAIL_MESSAGE`. A partir de là, il est possible de lire des valeurs standards, par exemple le text du message, l'expéditeur et le destinataire. Afin que cela soit plus simple, le module mail met à disposition les routines d'aide `getBodyText`, `getSender` et `getRecipients`.

L'objet est adopté automatiquement dans la désignation (NAME). Le corps du message est écrit dans le texte supplémentaire et l'expéditeur et le destinataire sont transférés dans les champs de métadonnées correspondants. En dernier, le message est marqué comme traité ou supprimé par `MAIL_ALLOW_DELETE`.

L'exemple complet ressemble à ceci :

```
<ruleset>
<base>
<name>Mailbox</name>
<search>
<name>"MAILBOX_GMAIL"</name>
<value>"ARCPATH:¶IMAP"</value>
<mask>2</mask>
<max>200</max>
```

```
</search>
<interval>10M</interval>
</base>
<rule>
<name>List</name>
<condition></condition>
<script>
    log.debug("Process Mailbox: " + NAME);


    OBJDESC = mail.getBodyText(MAIL_MESSAGE);
    ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
    ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
    EM_WRITE_CHANGED = true;
    MAIL_ALLOW_DELETE = true;
</script>
</rule>
<rule>
<name>Global Error Rule</name>
<condition>OnError</condition>
<script></script>
</rule>
</ruleset>
```

Traitement surveillé

L'exemple simple a un désavantage prépondérant: si un mail a déjà été marqué comme étant "traité" ou qu'il a été supprimé et que le processus est interrompu, avant que les données ont pu être enregistrées dans l'archive, un jeu de données reste tel quel, sans avoir été traité. Ce problème peut être évité complètement, si l'on travaille à deux niveaux: ce nouveau message est tout d'abord seulement enregistré dans ELO, mais il n'est pas encore supprimé. Si un mail se trouvant dans ELO est trouvé plus tard, il est supprimé.

Deux conditions sont importantes pour ce procédé: le mail doit être reconnaissable de façon univoque et une vérification doit être effectuée afin de savoir si elle existe déjà dans l'archive. Le premier point est très simple: chaque mail possède un ID de mail interne. Celui-ci peut être enregistré dans ELO dans un champs de métadonnées (par exemple dans le masque mail standard dans le champ ELOOUTL3, réservé pour l'ID du mail). Le deuxième point peut être traité très facilement grâce à une routine d'aide depuis le module ELOix : `ix.lookupIndexByLine`.

Le script modifié ressemble à ceci :

```
<script>
    log.debug("Process Mailbox: " + NAME);


    //   // lorsque le message est déjà dans l'archive: alors supprimer.
```

```
    var msgId = MAIL_MESSAGE.messageID;
    if (ix.lookupIndexByLine(EM_SEARCHMASK, "ELOOUTL3", msgId) != 0) {
        log.debug("le mail existe déjà dans l'archive,
                  ignorer ou supprimer");
        MAIL_ALLOW_DELETE = true;
    }   else {
        OBJDESC = mail.getBodyText(MAIL_MESSAGE);
        ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
        ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
        ELOOUTL3 = msgId;
        EM_WRITE_CHANGED = true;
    }
</script>
```

Marquer au lieu de supprimer

Dans l'implémentation standard, un mail traité est supprimé de la boîte de réception. Ce n'est pas souhaité dans quelques cas. Il est également possible d'effectuer un marquage. Un candidat possible est le drapeau "lu". Un mail traité est placé sur "lu" par ELOas et se différencie ainsi d'un nouveau mail. Dans ce cas, d'autres méthodes doivent être définies dans la mail JavaScript Library, en plus de la méthode `connectImap` :

nextImap_GMAIL(): cette fonction permute vers le prochain message. Dans cet exemple, elle doit vérifier si un mail a déjà été marqué comme lu, et le passer si nécessaire.

finalizeImap_GMAIL(): cette fonction marque le message traité. Dans l'implémentation standard, le message est supprimé. Dans notre exemple, il doit seulement être marqué comme *lu*.

nextImap_GMAIL

Cette fonction permute vers le prochain message. Elle passe de façon séquentielle par le biais de la liste des messages, la position actuelle est enregistrée dans la variable `MAIL_POINTER`. Si un message a déjà été marqué comme lu, elle est passée. Celle-ci est activée lors du premier message non-lu (il est copié dans la variable `MAIL_MESSAGE`) et la valeur true est livrée. S'il n'existe pas d'autres messages, un false est livré. ELOas termine alors le traitement de ce ruleset et bascule vers le suivant.

```
nextImap_GMAIL: function() {


    for (;;) {
        if (MAIL_POINTER >= MAIL_MESSAGES.length) {
            return false;
        }

        MAIL_MESSAGE = MAIL_MESSAGES[MAIL_POINTER];

        var flags = MAIL_MESSAGE.getFlags();
```

```
        if (flags.contains(Flags.Flag.SEEN)) {
            MAIL_POINTER++;
            continue;
        }

        MAIL_ALLOW_DELETE = false;
        MAIL_POINTER++;
        return true;
    }

    return false;
},
```

Une initialisation est effectuée, la variable `MAIL_ALLOW_DELETE` est placée sur false. Si un traitement est effectué dans le ruleset, celui-ci devrait placer la variable sur true. Dans ce cas, le mail est marqué dans la méthode finaliseImap en tant que traité.

finalizeImap_GMAIL

La fonction `finalizeImap_GMAIL` doit marqué un mail comme traité, cela se fait en plaçant le drapeau `SEEN`. Elle peut seulement être placée si la méthode `Connect` le permet (`MAIL_DELETE_ARCHIVED`) et que le ruleset a placé le mail actuel comme étant archivé (`MAIL_ALLOW_DELETE`).

```
finalizeImap_GMAIL: function() {


    if (MAIL_DELETE_ARCHIVED &amp;amp; MAIL_ALLOW_DELETE) {
        message.setFlag(Flags.Flag.SEEN, true);
    }
},
```

Exemple - Migration d'une base de données de documents

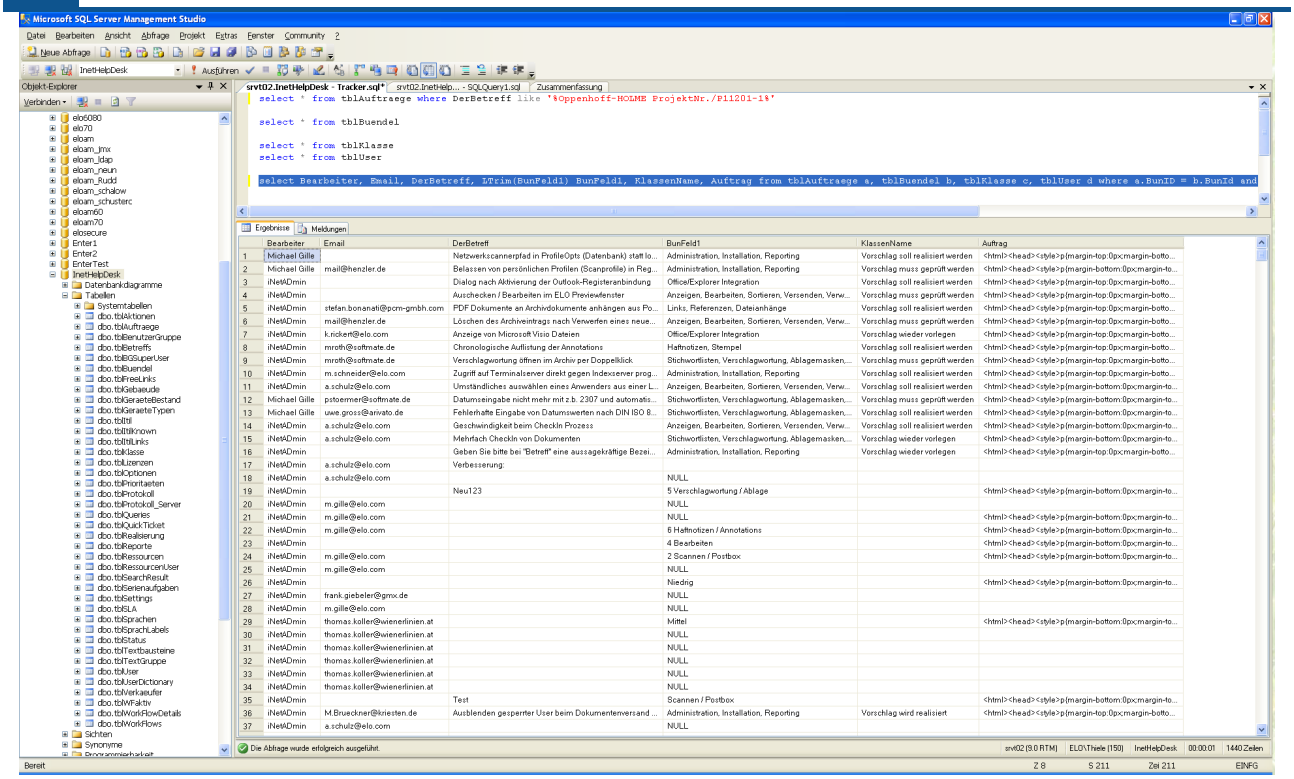
A ces fins, une base de données d'environ 1400 entrées doit être migrée dans ELO. Dans cette base de données, se trouvent les métadonnées et les documents. Du côté d'ELO, un classeur doit être créé à partir de chaque entrée des métadonnées. Ce classeur contient alors le document comme sous-entrée. Pour la migration, ELOas a été sélectionné comme outil.

Etant donné que ELOas ne peut pas créer de documents actuellement, une entrée dummy doit avoir été traitée pour chaque classeur dans une étape de pré-traitement. Heureusement, les entrées de la base de données ont un numéro courant commençant par 1, jusqu'à 1440. Il est très simple de créer les classeurs Dummy par le biais d'un script vbs. Tous les classeurs sont créés dans une armoire avec l'ID d'objet 274312.

```
Set ELO = CreateObject("ELO.professional")
Elo.CheckUpdate 0

for i=1 to 1440
    call Elo.PrepareObjectEx( 0, 4, 337 )
    Elo.ObjShort="TrackId " &amp; i
    Elo.ObjIndex="#274312"
    call Elo.SetObjAttrib(2, i)
    call Elo.SetObjAttrib(0, "GilleM")
    call Elo.SetObjAttrib(3, "Amélioration du produit")
    Elo.UpdateObject
next
Elo.CheckUpdate 1
```

Ensuite, le ELOas est mis en place. Les données sont prises en charge depuis une base de données SQL.

```
"select responsable, e-mail, objet, LTrim(BunFeld1) BunFeld1,
        nom de la classe, ordre
                de [InetHelpDesk].[dbo].tblordres a,
                   [InetHelpDesk].[dbo].tblBuendel b,
                   [InetHelpDesk].[dbo].tblclasse c,
                   [InetHelpDesk].[dbo].tblUser d
                       where a.BunID = b.BunId
                               and a.KlaID = c.KlaID
                               and a.UsrID = d.UsrID
                               and AufID = " + ETS_COUNT
```



Illustr. : base de données SQL

Il s'agit d'un statement SELECT, mais qui ne propose rien de spécial à part cela. Un point important: dans la liste select, il existe une colonne *LTrim(BunFeld1) BunFeld1*. Dans le champ de la base de données *BunFeld1*, les données se trouvent avec des guillemets qui ne sont pas souhaités. Ceux-ci sont supprimés avec LTrim. Mais alors la colonne n'aurait plus de nom, voilà pourquoi le nom de la colonne est à nouveau indiqué comme étant *BunFeld1*. Cette technique doit toujours être appliquée si l'on veut travailler avec des valeurs calculées dans la liste de sélection.

Le ruleset complet ressemble à ceci :

```
<ruleset>
    <base>
        <name>ImportTracker</name>
        <search>
            <name>"ETS_COUNT"</name>
            <value>"*"</value>
            <mask>337</mask>
            <max>200</max>
        </search>
        <interval>1H</interval>
    </base>
<rule>
    <name>Rule1</name>
    <condition></condition>
    <script>

        /* ici sont enregistrées les données au sujet du classeur actuel
```

```
   depuis la base de données */

var item = db.getLine(1, "select utilisateur, e-mail, objet,
        LTrim(BunFeld1) BunFeld1, Nom de la classe, offre
                from [InetHelpDesk].[dbo].tblAuftraege a,
                     [InetHelpDesk].[dbo].tblBuendel b,
                     [InetHelpDesk].[dbo].tblKlasse c,
                     [InetHelpDesk].[dbo].tblUser d
                        where a.BunID = b.BunId
                                and a.KlaID = c.KlaID
                                and a.UsrID = d.UsrID
                                and AufID = " + ETS_COUNT);

/* ETS_COUNT contient le numéro record,
   il est vidé une fois le traitement effectué.
ETS_COUNT = "";

   /* Le champ de la désignation est rempli depuis la base de données, attention à la

NAME = item.l'objet;
if (NAME == "") { NAME = "unknown"; }
if (NAME.length() > 127) { NAME = NAME.substring(0, 126); }

// l'initiateur est rempli depuis la base de données.
ETS_MAIL = item.Email;

/* Dans la base de données, le champ thème était rempli avec d'autres mots-clés que d
   C'est pourquoi il existe un tableau de traduction.
   ELO travaille avec un index de colonnes. */
var thème = item.BunFeld1;
if (thème == "Administration, Installation, Reporting") {
        thème = "Administration¶Installation¶Reporting"; }
if (thème == "Afficher, trier, modifier, envoyer, gérer, rechercher") {
        thème = "Traitement de document¶Visionneuse¶Traitement de la structure¶Recher
if (thème == "Afficher, modifier, trier, envoyer, gérer, rechercher") {
        thème = "Traitement de document¶Visionneuse¶Traitement de la structure¶Recher
if (thème == "Interface utilisateur, design, menus, navigation") {
        thème = "Usability¶Interface"; }
if (thème == "Notes, tampons") {
        thème = "Annotations"; }
if (thème == "Office / Explorer Integration") {
        thème = "Office Integration¶OS Integration"; }
if (thème == "Disponibilité hors ligne") {
        thème = "hors ligne"; }
if (thème == "Liens, références, pièces-jointes") {
        thème = "Liens¶Références"; }
if (thème == "Numériser, boîtes de réception, convertir, imprimer") {
        thème = "Numériser¶Boîte de réception¶Convertir¶Imprimer"; }
if (thème == "Sécurité, authentification, cryptage, droits utilisateur") {
```

```
        thème = "Droits utilisateurs"; }
if (thème == "Listes de mots-clés, métadonnées, masques de dépôt, versioning") {
        thème = "Métadonnées¶Dépôt de document"; }
if (thème == "Processus, Tâches") {
            thème = "Processus¶Tâches"; }
if (thème == "Interfaces, scripts") {
        thème = "Scripting¶Interfaces"; }
ETS_THEMA = thème;

ETS_USER = "Gestion du produit";
ETS_STATUS_INT = item.KlassenName;

EM_WRITE_CHANGED = true;

L'information de la base de données est entrée maintenant. Le document manque.

/* Celui-ci est créé sous forme d'un fichier de commande XML pour ELO XML Importer. T

var id = Sord.getId();
var dataFile = new File("d:\\temp\\trk\\" + id + ".htm");
Utils.stringToFile(item.ordre, dataFile, "ISO-8859-15");


/* puis le flux de données XML est créé.
   Etant donné que l'entrée des métadonnées est liée
   au classeur, seules les métadonnées rudimentaires sont disponibles. */
var xmlDesc = NAME.replace("\"", "'").
                   replace("&amp;", "&amp;amp;").
                   replace("<", "&amp;lt;").
                   replace(">", "&amp;gt;");
var xmlFile = new File("d:\\temp\\trk\\" + id + ".xml");
var xmlText = "<?xml version=\"1.0\" ?><eloobjlist
               ver=\"1.0\"><obj><desc value=\"";
xmlText = xmlText + xmlDesc;
xmlText = xmlText +
          "\"/><type value=\"0\"/><destlist><destination
                  type=\"1\" value=\"#";
xmlText = xmlText + id;
xmlText = xmlText + "\"/></destlist><docfile name=\"";
xmlText = xmlText + id;
xmlText = xmlText + ".htm\"/></obj></eloobjlist>";

// En dernier, est écrit le fichier XML.
                   replace(">", "&amp;gt;");
var xmlFile = new File("d:\\temp\\trk\\" + id + ".xml");
var xmlText = "<?xml version=\"1.0\" ?><eloobjlist
               ver=\"1.0\"><obj><desc value=\"";
xmlText = xmlText + xmlDesc;
xmlText = xmlText +
```

```
                    "\"/><type value=\"0\"/><destlist><destination
                            type=\"1\" value=\"#";
            xmlText = xmlText + id;
            xmlText = xmlText + "\"/></destlist><docfile name=\"";
            xmlText = xmlText + id;
            xmlText = xmlText + ".htm\"/></obj></eloobjlist>";

            // le fichier XML est créé en dernier.
            Utils.stringToFile(xmlText, xmlFile, "UTF-8");


    </script>
</rule>
<rule>
    <name>Global Error Rule</name>
    <condition>OnError</condition>
    <script></script>
</rule>
</ruleset>
```

Une fois que ELOas a complété l'entrée des métadonnées depuis la base de données et qu'il a créé les fichiers document XML, l'importateur ELO XML entre en jeu. Il importe les fichiers HTML dans le classeur correspondant. Le processus de migration est terminé. Ce projet prend environ 4 heures.

Exemple - Treewalk pour ELOas

Une fonction de navigation dans l'arborescence est disponible pour le traitement des documents dans les services d'automation ELO. Non seulement des sections de recherche peuvent être traitées, mais aussi des structures d'arborescence intégrales.

Introduction

Normalement, ELOas effectue une recherche de champ d'indexation, afin de déterminer la liste des documents à traiter. En alternative, il est possible d'exécuter également un tree walk. Ce treewalk vous permet parcourir des branches d'archive ou l'archive intégrale. Chaque entrée est parcourue deux fois: une fois lors de l'entrée, ensuite, toutes les sous-entrées sont parcourues, puis, lors de la sortie.

Exemple: il existe une armoire avec les classeurs 1 et 2. Le classeur 1 contient le registre 1.1. Il en découle le déroulement suivant:

```
Armoire (entrer)
Classeur 1 (entrer)
Registre 1.1 (entrer)
Registre 1.1 (quitter)
Classeur 1 (quitter)
Classeur 2 (entrer)
Classeur 2 (quitter)
Armoire (quitter)
```

Un script peut vérifier à l'appui des variables `EM_TREE_STATE` si le ruleset est appelé dans la branche croissante (entrée) ou encore dans la branche décroissante (sortie). Celui contient 0 lors de l'entrée et 1 lors de la sortie. L'enregistrement se fait seulement lors de la sortie. Les modifications qui sont exécutées lors de l'entrée restent telles quelles jusqu'à la sortie, même si d'autres objets ont été traités entre-temps.

Pour initier un treewalk, il suffit d'entrer la valeur "TREEWALK" dans le nom de groupe de l'index de recherche, et en tant que terme de recherche, le numéro du noeud de démarrage. Attention! Sur le noeud de démarrage, aucune règle n'est appelée. Cette règle n'est exécutée que pour les sous-entrées.

Exemple

L'exemple d'application suivant parcourt une branche d'archive et place un identificateur interne (TrackID) dans tous les objets du type de masque 6 (Track Item). Le classeur de démarrage possède l'ID 3352.

Dans cet exemple, nous n'avons pas prévu de traitement des erreurs, c'est pourquoi la règle correspondante est vide.

```
<ruleset>
    <base>
        <name>Create TrackId</name>
```

```
        <search>
            <name>"TREEWALK"</name>
            <value>3352</value>
            <mask>6</mask>
            <max>200</max>
        </search>
        <interval>10M</interval>
    </base>

<rule>
    <name>CreateId</name>
    <script>
        if ((EM_TREE_STATE == 1) &amp;amp;&amp;amp; (EM_ACT_SORD.getMask() == 6)) {
            // nur TrackItems bearbeiten
            //cnt.createCounter("ETSTrackId", 10000);

            if (ETS_TICK == "") {
                log.debug("Create new TrackId: " + NAME);



                ETS_TICK = cnt.getTrackId("ETSTrackId", "V");
                EM_WRITE_CHANGED = true;
            }
        }
    </script>
</rule>

<rule>
    <name>Global Error Rule</name>
    <condition>OnError</condition>
    <script>
    </script>
</rule>
</ruleset>
```

La partie intéressante du ruleset se trouve dans la section de script, voici une liste individuelle:

```
if ((EM_TREE_STATE == 1) &amp;amp;&amp;amp; (EM_ACT_SORD.getMask() == 6)) {
```

Le script doit seulement être exécuté lors de la sortie (`EM_TREE_STATE == 1`) et seulement sur les objets du type `TrackItem` (`EM_ACT_SORD.getMask() == 6`).

```
// modifier seulement TrackItems
//cnt.createCounter("ETSTrackId", 10000);
```

L'exemple utilise un counter qui doit être créé ultérieurement, par exemple par la commande nommée ci-dessous. Il peut seulement être créé une fois, sinon, le Track ID est ré-initialisé à chaque fois.

```
if (ETS_TICK == "") {
```

S'il n'existe pas encore de Track ID (en d'autres termes, si le champs de métadonnées ETS_TICK est vide), alors il s'agit d'en créer un.

```
log.debug("Create new TrackId: " + NAME)

ETS_TICK = cnt.getTrackId("ETSTrackId", "V")
```

Pour créer des tracks ID, il existe une méthode très pratique dans le module counter `cnt: getTrackId(<CounterName>, <Prefix> )`. Cette méthode prend une nouvelle valeur counter et la complète avec le préfixe et une somme de contrôle. Dans notre exemple, la valeur Counter 10001 devient le Track ID V10001C2.

```
EM_WRITE_CHANGED = true
```

L'objet doit seulement être enregistré si un nouveau ID de traçage a été créé.

```
}
}
```

Le ruleset est exécuté toutes les 10 minutes et parcourt l'intégralité du classeur Track Item. Toutes les entrées sans track ID sont complétées automatiquement, peu importe avec quel client elles ont été créées.

Variables de Runtime Environment

Si le ruleset est exécuté, il existe de nombreuses variables en plus de la valeur `EM_TREE_STATUS` qui peuvent être prises en compte.

Nom	Contenu
`EM_TREE_STATUS`	Définit si le ruleset est exécuté dans la branche croissante (0) ou décroissante (1).
`EM_ACT_SORD`	Contient l'objet SORD avec les données d'objet actuelles.
`EM_PARENT_SORD`	Contient l'objet SORD avec les données du noeud parent. Ces données peuvent être modifiées par principe. Il faut veiller à ce que ces modifications soient enregistrées. A ces fins, la modification doit être reconnue dans la branche décroissante, et le drapeau `EM\_WRITE\_CHANGED` doit être placé sur `true`.

Nom	Contenu
EM_ROOT_SORD	Contient l'objet SORD avec le noeud de démarrage. Etant donné que le ruleset n'est pas appliqué sur cette entrée, un enregistrement manuel doit être effectué lors de modifications. Cela peut se faire par exemple en plaçant la variable EM_SAVE_TREE_ROOT.
EM_INDEX_LOADED	A l'opposé d'un traitement après une recherche, l'on ne peut pas s'assurer qu'un objet SORD chargé possède un masque bien précis dans le tree walk. En principe, chaque masque peut apparaître. Les variables d'indexation prédéfinies peuvent seulement être créées et remplies à partir des champs de métadonnées pour les masques qui ont été authentifiés dans la définition sous <mask>et sous <masks>. Dans ce cas, la variable EM_INDEX_LOADED est placée sur true. S'il existe un masque inconnu, l'accès aux champs de métadonnées est seulement possible par le biais de l'objet EM_ACT_SORD, EM_INDEX_LOADED a le statut false. Remarque: si les variables d'indexation sont remplies, l'on ne doit pas modifier directement les champs de métadonnées dans EM_ACT_SORD. Ces modifications sont perdues avant l'enregistrement, quand les variables d'indexation sont écrites.
EM_TREE_LEVEL	Cette variable permet de voir à quel niveau du tree walk l'on se trouve. Les sous-entrées du noeud de démarrage se trouvent au niveau 0 (les règles ne sont pas appelées pour le noeud de démarrage).
EM_TREE_MAX_LEVEL	L'on peut déterminer la profondeur maximale par le biais de cette règle. Les sous-entrées encore plus profondes sont ignorées. Normalement, la valeur est 32. S'il doit être modifié, il peut être placé sur la valeur souhaitée avant le traitement dans la routine onstart.
EM_SAVE_TREE_ROOT	Aucune règle n'est appelée pour le noeud de démarrage du treewalk. Si celui-ci a été modifié par l'accès EM_TREE_ROOT ou EM_PARENT_SORD, un enregistrement de ces modifications doit être authentifié par le placement des variables EM_SAVE_TREE_ROOT. <onend>var result = …var oldstate = …EM_SAVE_TREE_ROOT = result != oldstate;log.debug("now save root: " + EM_SAVE_TREE_ROOT);</onend>
EM_TREE_EVAL_CHILDREN	Si le logiciel remarque qu'une sous-section est exclue du traitement, la variable EM_TREE_CHILDREN est placée sur false. Cette valeur est évaluée seulement pour la branche croissante (il serait trop tard pour la branche décroissante, étant donné que la sous-section a déjà été passée) et il est initialisé avec true pour chaque objet (le comportement standard est que l'intégralité de la sous-section est passée).
EM_TREE_ABORT_WALK	Si une passation doit être complètement annulée, l'on peut placer le drapeau EM_TREE_ABORT_WALK à n'importe quel moment. Dans ce cas, aucune autre sous-entrée n'est passée. Les autres entrées du même niveau, qui n'ont pas encore été traitées, restent ainsi. L'on peut placer ce flag afin d'annuler le travail après une erreur grave.

Nom	Contenu
	Remarque : dans la routine `onstart`, les contrôles de runtime peuvent être exécutés afin de vérifier si le treewalk peut être effectué. Sinon, vous pouvez interrompre le runtime par le biais de ce drapeau.

Exemple - Traitement de processus

Il existe une extension pour traiter les rendez-vous de processus. Dans le processus, l'on peut créer des noeuds de personne pour le compte ELOas. Si un processus active ce noeud, il est possible de déterminer et de traiter une liste des rendez-vous actifs des processus par le biais d'une recherche ELOas "WORKFLOW". Les métadonnées peuvent être complétées et le processus transféré.

Les rulesets nécessaires doivent être créés au niveau XML.

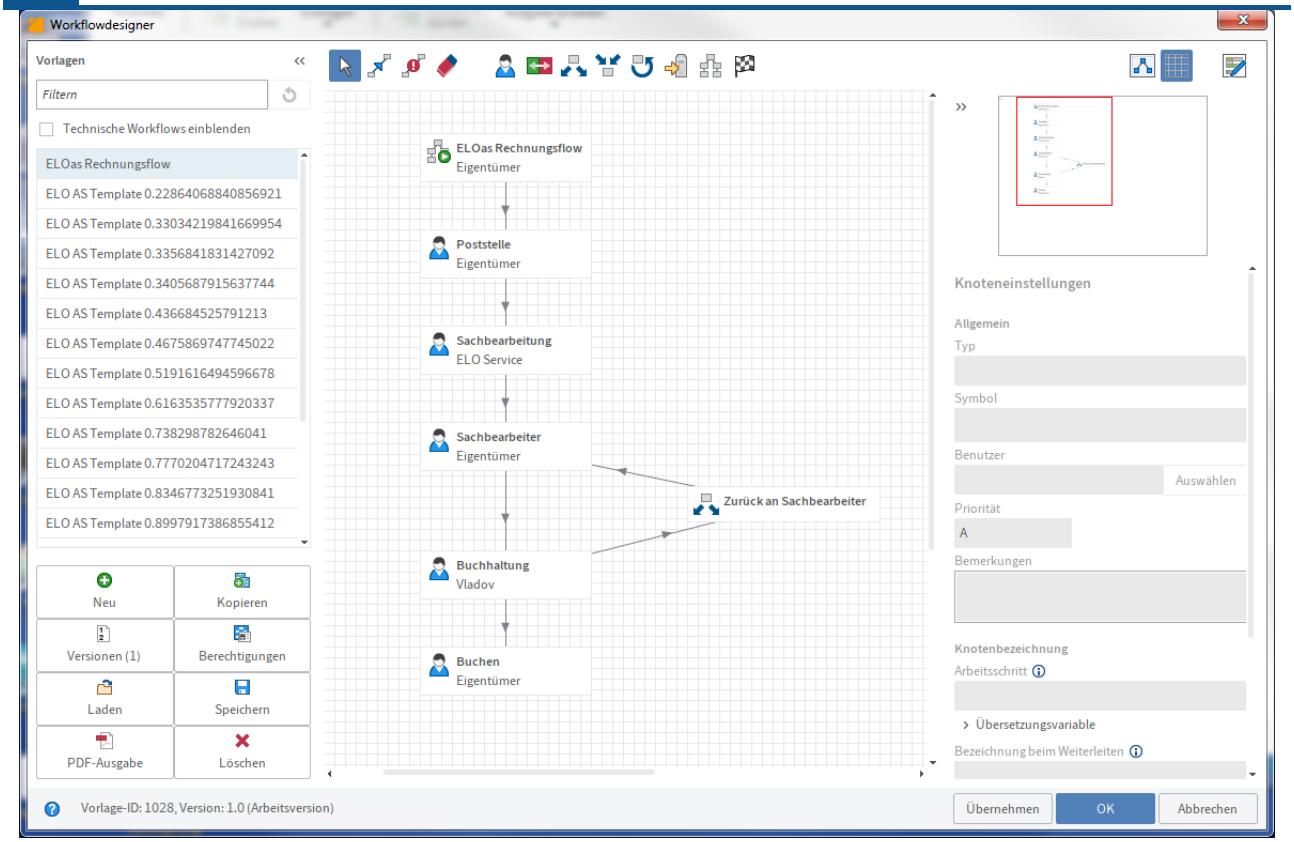
Le rassemblement de la liste des rendez-vous de processus ressemble en gros à une recherche normale. En tant que nom pour le champ de métadonnés, est entré "WORKFLOW", le terme de recherche en lui-même est ignoré et devrait rester vide.

```
<base>
    <name>Workflow2</name>
    <search>
        <name>"WORKFLOW"</name>
        <value></value>
        <mask>13</mask>
        <max>1000</max>
    </search>
    <interval>1M</interval>
</base>
```

Même si aucun masque de recherche n'est requis pour le rassemblement de la liste, un masque de recherche doit tout de même être indiqué. Seuls les processus de la liste des dates possédant ce masque sont traités. C'est nécessaire afin que les données d'indexation puissent être chargées dans les variables JavaScript locales. Si les processus sont possibles pour plusieurs masques, le ruleset doit être créé plusieurs fois.

> **Information**
>
> Dans la liste des rendez-vous, aucun „FindFirst – FindNext“ n'est exécuté. S'il existe beaucoup d'échéances, qui ne sont pas traitées, il se pourrait que les nouvelles échéances ne soient pas trouvées pour un traitement.

Lors du traitement des processus, il existe deux activités en plus de la modification des métadonnées: un transfert ciblé et une modification du processus. L'exemple suivant doit montrer comment l'on peut influencer le processus selon les métadonnées actuelles. A ces fins nous allons présenter un processus d'approbation simple, pour lequel le responsable n'a pas encore été défini au début. Au cours du processus, le service d'entrée de courrier est entré dans un champ d'indexation *RESPONSABLE*. Dans le modèle, le noeud responsable est initialisé avec le champ d'indexation *propriétaire*, la valeur en elle-même est lue par ELOas dans le champ d'indexation *RESPONSABLE* et entrée dans le noeud. ELOas tourne sous le nom ELO *elowf* et a un noeud de personne entre le poste de l'entrée de courrier et le responsable.



Illustr. : créateur de processus, modèle de processus avec ELOas

Une fois que le processus arrive auprès d'ELOas, le service de l'entrée de courrier a déterminé le responsable. ELOas lit le champ de métadonnées RESPONSABLE et entre la valeur dans le noeud suivant *RESPONSABLE*. Cela se fait par la simple règle suivante:

```
<rule>
    <name>Expand Name</name>
    <condition></condition>
    <script>
        log.debug("Process WF: " + NAME);


        wf.changeNodeUser("Responsable", RESPONSABLE);

        EM_WF_NEXT = "0";
    </script>
</rule>
```

La modification du nom utilisateur ELO est effectuée par la commande `wf.changeNodeUser`. Le premier paramètre est le nom du noeud de processus et le second paramètre est le nom utilisateur ELO qui doit être entré. La librairie *wf* (verrouiller le processus, lire, rechercher un noeud, actualiser un utilisateur, enregistrer le processus, annuler le verrouillage) s'occupe du reste.

Une fois que le nom utilisateur a été placé, le processus doit être transféré. Cela se fait en plaçant les variables `EM_WF_NEXT`. Si celle-ci reste vide, rien n'est transféré. La date reste (cela

ne devrait pas être le cas indéfiniment, sinon, la liste sera trop longue tôt ou tard). Si toutes les conditions préliminaires pour un transfert ont été rassemblées, alors l'on peut entrer le numéro de connexion ou le nom du noeud suivant. S'il existe seulement un successeur, alors il faut entrer le numéro de connexion: `EM_WF_NEXT = "0";`.

S'il existe plusieurs successeurs, il vaut mieux indiquer le nom du noeud suivant. Admettons que le processus soit comptabilisé automatiquement après le traitement, cela signifie que le noeud Comptabilité est transféré à ELOas. Celui-ci exécute un script qui vérifie les données. Si tout est en ordre, alors la fonction `ERPverify()` livre un true et le processus doit être transféré au noeud *Comptabiliser*. S'il existe une erreur, alors le processus retourne vers l'utilisateur responsable. Le script pourrait ressembler à ceci:

```
If (ERPverify()) {
    EM_WF_NEXT = "Comptabiliser";
}   else {
    EM_WF_NEXT = "Responsable";
}
```

Dépôt via ELO Dropzone

Dépôt ELOas par le biais des tuiles ELO Dropzone

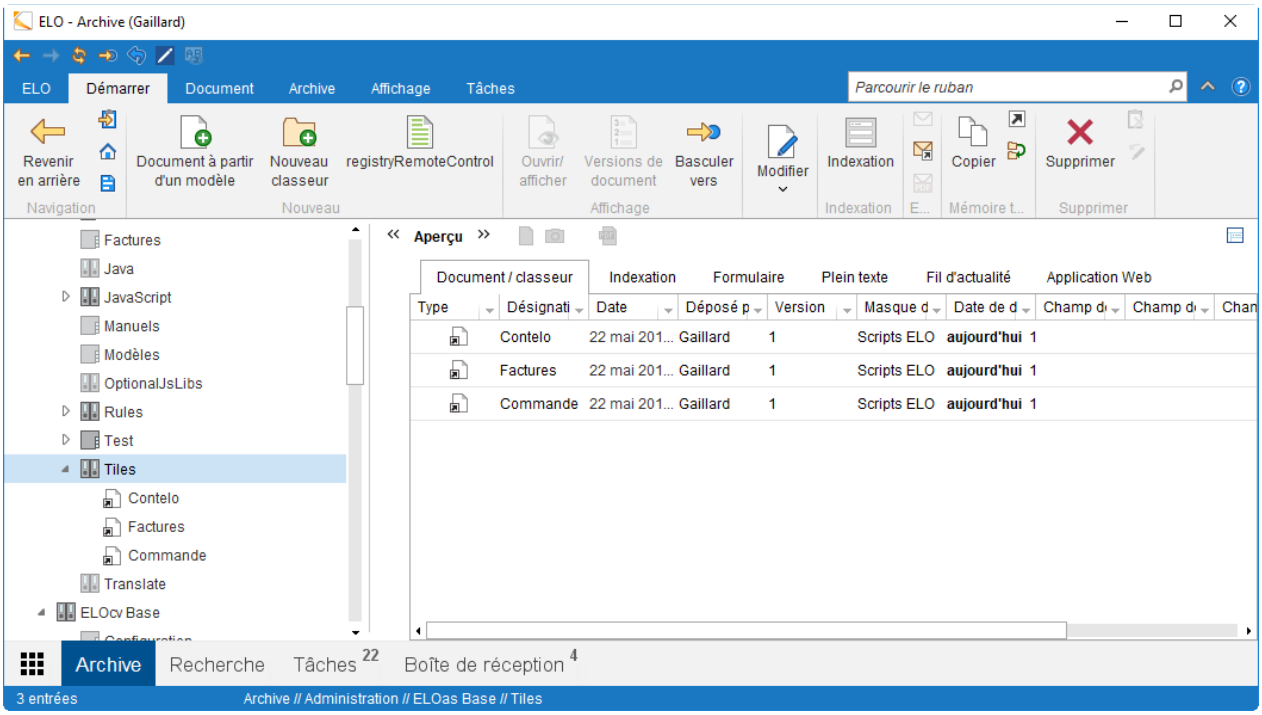
ELOas 20.0 permet le dépôt automatique par le biais des tuiles de ELO Dropzone. Vous pouvez définir les métadonnées pour les nouveaux documents dans une tuile d'ELO Dropzone. Procédez de la manière suivante pour configurer le dépôt automatique.

> **Remarque**
>
> Les librairies standard ELOas sont nécessaires pour le dépôt automatique par le biais de ELO Dropzone. Vous les trouverez sur le site de téléchargement officiel *http://www.forum.elo.com/script/20/eloinst.html*. De plus, il vous faut une licence pour ELO XML Importer pour le dépôt automatique de documents via une tuile Dropzone.

Pas à pas

1. Veuillez créer les tuiles ELO Dropzone avec le module ELO Dropzone. Les tuiles sont enregistrées dans le classeur *// Administration // Dropzone*. Les définitions de tuile sont enregistrées dans le texte supplémentaire de l'entrée.

2. Le classeur "ELOas Base" contient le sous-classeur "Tiles". Dans ce classeur sont référencés les tuiles ELO Dropzone requises. ELOas ne fait pas de différenciation entre les tuiles personnelles et les tuiles globales.

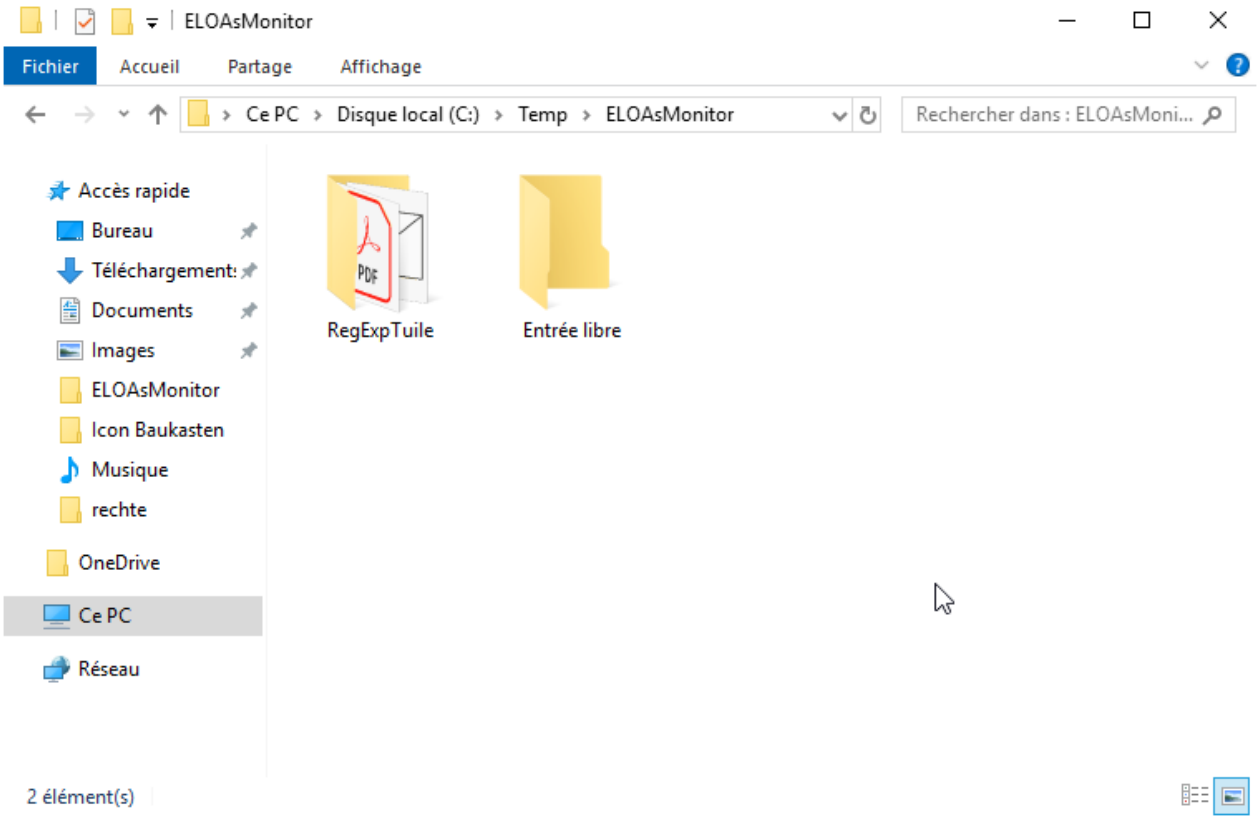


Illustr. : classeur avec tuiles ELO Dropzone référencées

1. Un répertoire surveillé est défini dans le fichier de configuration ELOas "config.xml" :

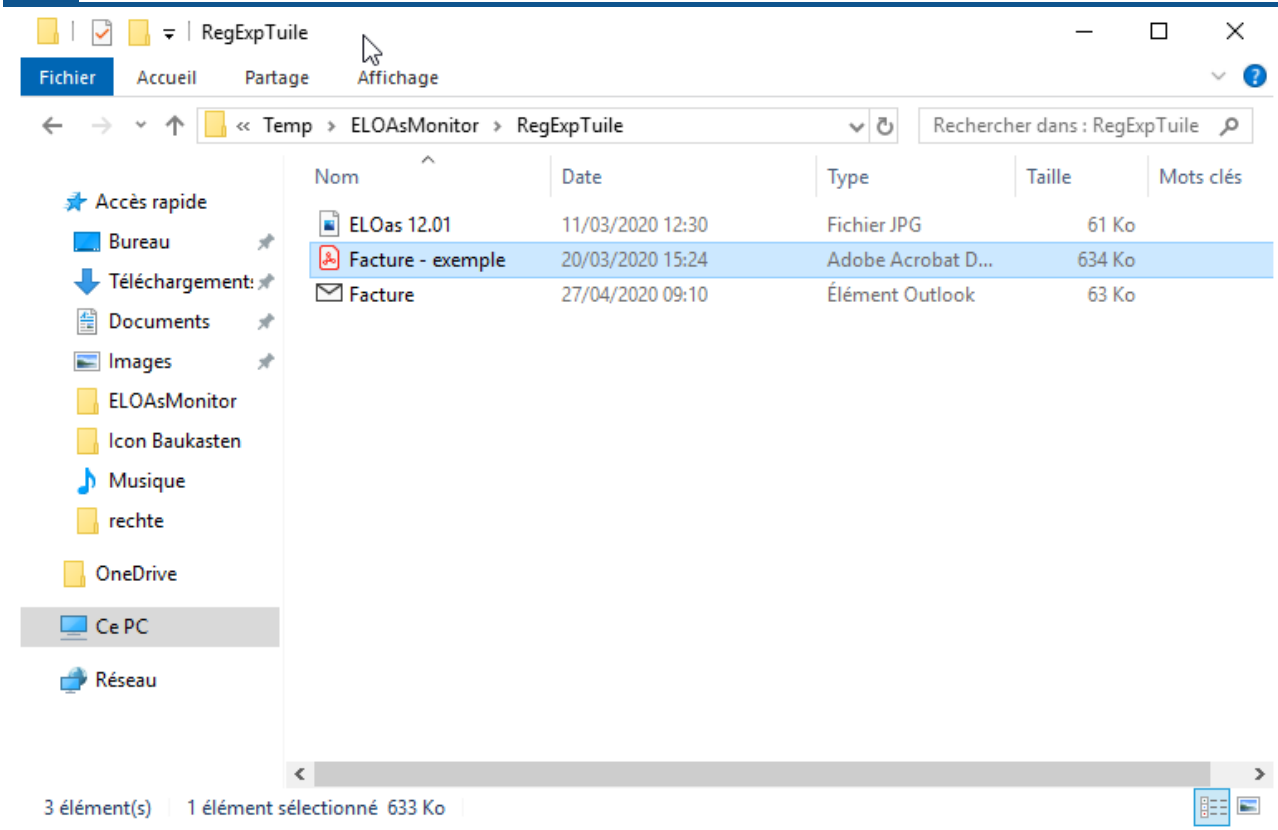
```
<entry key="monitordir">C:\temp\ELOAsMonitor</entry>
```

Un sous-répertoire est attendu pour chaque tuile. Les tuiles existantes doivent avoir une désignation univalente afin que le dépôt automatique fonctionne.



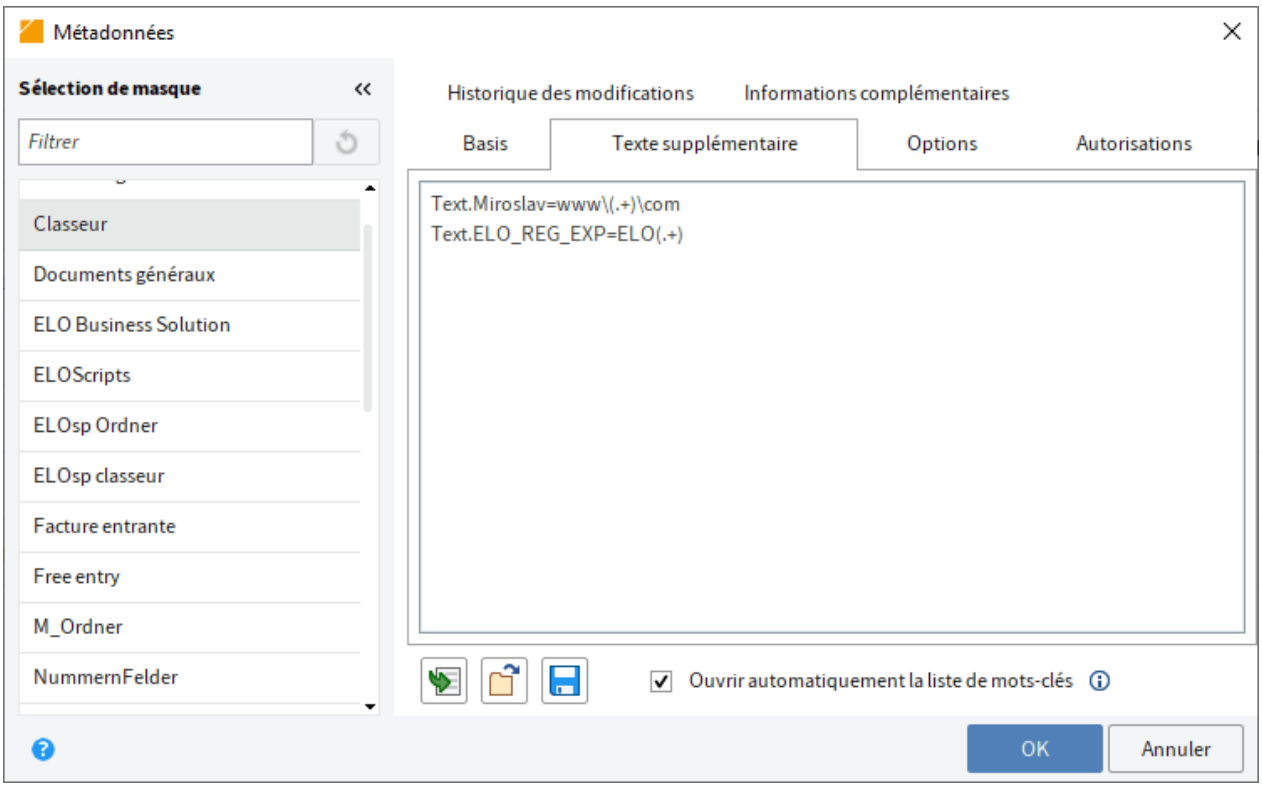
Illustr. : répertoire surveillé

Les fichiers qui seront transmis automatiquement dans l'archive par ELOas et les définitions de tuiles sont enregistrés dans les différents sous-répertoires.



Illustr. : différents documents dans le sous-répertoire, avant le dépôt automatique

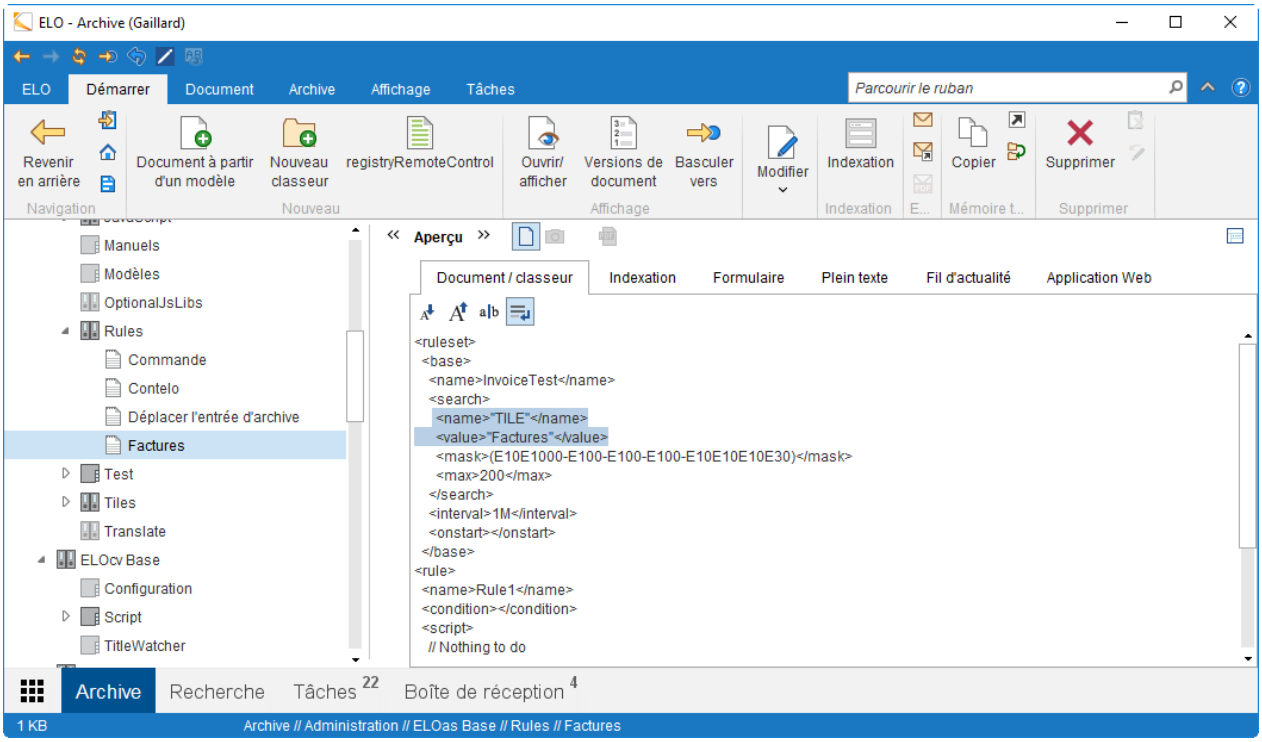
1. La configuration des expressions régulières est effectuée dans les métadonnées du classeur "Tiles". Ces expressions régulières seront disponibles pour toutes les tuiles.



Illustr. : expression régulière dans les métadonnées du classeur 'Tiles'

1.

Avec ELOas 11.0, il existe un nouveau type de règle pour le dépôt par le biais d'une tuile ELO Dropzone. Dans le paragraphe "<name>" se trouve la valeur "TILE" et dans le paragraphe "<value>", la désignation de la tuile ELO Dropzone référencée.



Illustr. : règle ELOas pour le dépôt par le biais de ELO Dropzone

Dans un intervalle temporel défini, ELOas vérifie dans le répertoire des tuiles surveillées s'il existe de nouveaux fichiers à archiver, et les archive le cas échéant. Les classeurs existants ne sont pas pris en compte. Dans une tuile, on peut définir que les fichiers locaux soient supprimés après le dépôt. S'il existe des erreurs lors d'un processus de dépôt, les fichiers problématiques sont déplacés dans un sous-répertoire du nom "Errors", afin que celui-ci ne soient pas toujours traités par ELOas.

Code-barres

Introduction

Dans les services d'automation ELO, il existe la classe d'aide `ELOAsUtils` avec des fonctions permettant de lire et d'écrire des codes-barres. Vous pouvez utiliser la librairie Softek et la librairie ZXing pour la fonctionnalité des codes-barres.

> **Information**
>
> Vous trouverez d'autres informations au sujet de la librairie Softek sous : *http://www.bardecode.com/en1/app/barcode-reader-toolkit-for-windows/*.
>
> Vous trouverez d'autres informations sur la librairie ZXING sous : *http://zxing.github.io/zxing/apidocs*.

Lire un code-barres avec la librairie Softek

Depuis la version 10 de ELOas, la lecture de codes-barres est possible par le biais de la librairie Softek. En fonction du système d'exploitation (32/64 bits), la reconnaissance des codes-barres se fait par le biais des DLL "SoftekBarcodeDLL.dll" ou "SoftekBarcode64DLL.dll". La reconnaissance des codes-barres est utilisée de la manière suivante dans un script ELOas :

```
var barcodeReader = Packages.de.elo.mover.utils.ELOAsUtils.
                    createBarcodeReader2(emConnect);
var barcodeFile = new File("C://temp//BarcodeFile.tif");
var barcodeCount = barcodeReader.ScanBarCode(barcodeFile.getPath());
log.info("barcodeCount=" + barcodeCount);
var barcodeDescr = barcodeReader.GetBarString(barcodeCount);
log.info("barcodeDescr=" + barcodeDescr);
```

Formats codes-barres de la librairie Softek

Les formats de codes-barres supportés se trouvent dans la documentation officielle de la librairie Softek. Les format suivants sont pris en charge :

- Codabar 1D
- Code 128 1D
- Code 2 of 5 Datalogic 1D
- Code 2 of 5 Iata1 1D
- Code 2 of 5 Iata2 1D
- Code 2 of 5 Industrial 1D
- Code 2 of 5 Interleaved 1D
- Code 2 of 5 Matrix 1D
- Code 3 of 9 1 D
- Code 3 of 9 Extended 1D
- Code 93 1D
- EAN-8 1D
- EAN-13 1D
- GS1-128, UCC-128, EAN-128 1D
- GS1-Databar 2D
- Patch Code Symbols 1D
- UPC-A 1D
- UPC-E 1D
- QR-Code 2D
- Data Matrix ECC200 2D
- Micro-PDF-417 2D
- PDF-417

Exemples pour la lecture d'un QR-code

La lecture de QR-codes est activée par la directive ELOas suivante :

```
"barcodeReader.setReadQrCode(1);"
```

> **Information**
>
> Les méthodes de la classe `BarcodeReader` sont décrites dans la *ELOas Javadoc* officielle.

Lire un code-barres avec la librairie ZXing

La lecture des codes-barres se fait par le biais de la méthode statique `getBarcode` de la classe ELOas `ELOAsUtils`. Dans cette métrhode, le fichier, la page avec le code-barres et la configuration des codes-barres sont transmis en tant que paramètres.

```
String barcode = ELOAsUtils.getBarcode(IXconnect ixConnect, File file, int page, String barco
```

Les différents réglages de la configuration sont séparés à l'aide du symbole pipe. Un exemple de configuration pourrait ressembler à ceci :

Exemple

```
String barcodeConfig = "POSSIBLE_FORMATS:CODE_128,QR_CODE|
                        CHARACTER_SET:UTF8|ALLOWED_EAN_EXTENSIONS:2,5|
                        PURE_BARCODE:TRUE|RETURN_CODABAR_START_END:TRUE|
                        ASSUME_CODE_39_CHECK_DIGIT:TRUE|TRY_HARDER:TRUE";
```

De plus, il existe dans la classe `ELOAsUtils` la méthode `getBarcodeResult` qui livre le résultat intégral des codes-barres.

Exemple

```
Result barcodeResult = ELOAsUtils.getBarcodeResult(IXconnect ixConnect, File file, int page,
```

Formats codes-barres de la librairie ZXing

Les formats de codes-barres supportés se trouvent dans la documentation officielle de la librairie ZXING. Les différents formats sont affichés dans la classe "ZXing" `BarcodeFormat`. Vous trouverez plus d'informations sous : *http://zxing.github.io/zxing/apidocs/*.

Les format suivants sont pris en charge :

- Aztec 2D
- CODABAR 1D
- Code 128 1D
- Code 39 1D
- Code 93 1D
- Data Matrix 2D
- EAN-13 1D
- EAN-8 1D
- ITF (Interleaved Two of Five) 1D
- MaxiCode 2D
- PDF417
- QR Code 2D
- RSS 14
- RSS EXPANDED
-

UPC-A 1D
- UPC-E 1D
- extension UPC/EAN

Créer un code-barres avec la librairie ZXing

La création de codes-barres dans une page de document se fait par le biais de la méthode `writeBarcode`. Dans cette méthode, le fichier cible, le texte de code-barres, la taille du code-barres et la configuration du code-barres sont transmis en tant que paramètres.

```
ELOasUtils.writeBarcode(IXconnect ixConnect, File targetFile, String barcodeText, int width,
```

Exemple de configuration

```
String barcodeConfig = "AZTEC_LAYERS:13|CHARACTER_SET:UTF8|
                        DATA_MATRIX_SHAPE:FORCE_RECTANGLE|
                        ERROR_CORRECTION:M|MARGIN:20|PDF417_COMPACT:TRUE|
                        PDF417_COMPACTION:NUMERIC|PDF417_DIMENSIONS:5,10,5,10";
```

L'appel d'une méthode AS statique dans une règle ELOas ressemble à la chose suivante :

```
var result = Packages.de.elo.mover.utils.ELOAsUtils.
                getBarcode(emConnect,barcodeFile, 1, barcodeConfig);
```

Ainsi, l'utilisation des informations des codes-barres est également possible dans les services d'automation ELO.

Préposé au débogage

Préposé au débogage ELOas

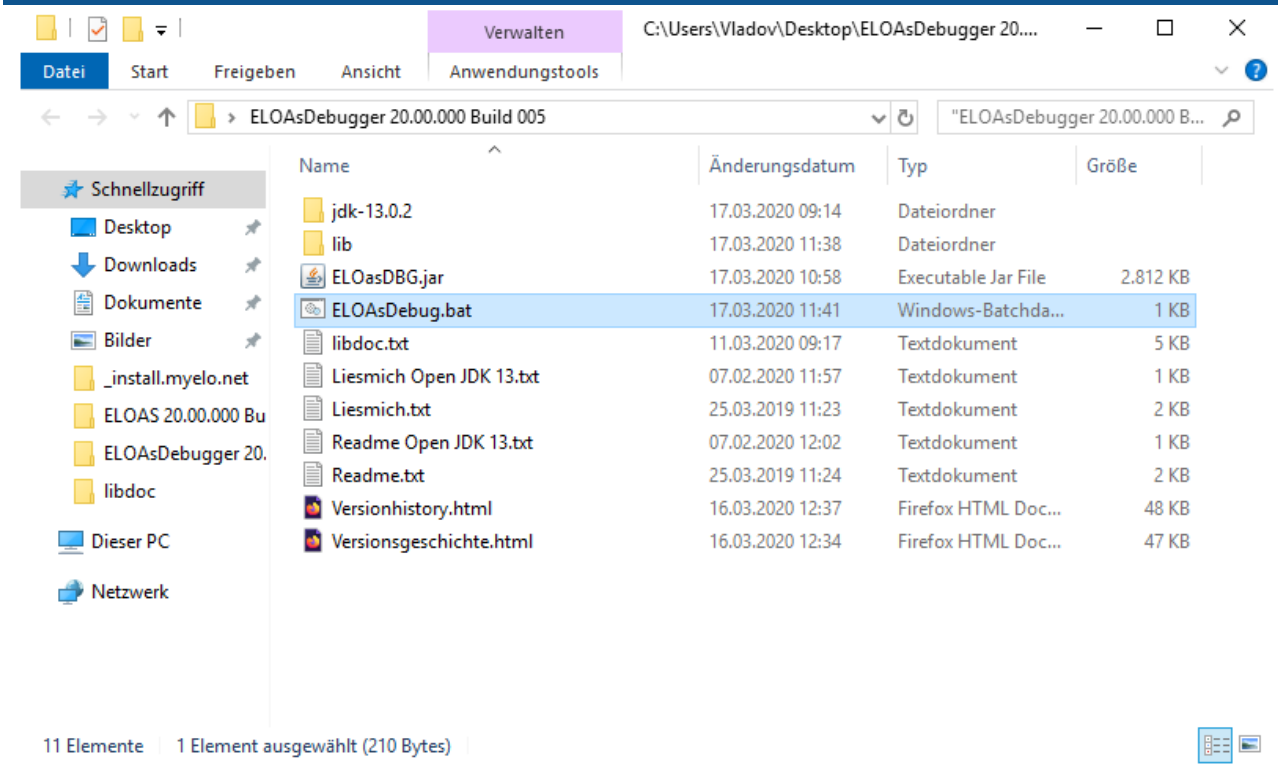
La recherche d'erreurs peut être très compliquée dans un ruleset important. Dans chaque passage, JavaScript doit être ajusté. *A ces fins, vous devez soumettre le document au check-out, le modifier, le soumettre au check-in, puis cliquer sur "Reload".* Par ailleurs, l'exploitation du Rhino Debuggers sous Apache Tomcat est problématique dans un environnement Windows 7. Ce point et le processus de check-out/check-in peuvent être épargnés grâce à la mise en place du préposé au débogage ELOas.

> **Information**
>
> Pour plus d'informations sur le débogueur ELOas, vous pouvez lire la documentation distincte „ELO Automation Services Debugger (Java FX)“.

Ouvrir le programme

ELOas Debugger 20 est livré avec OpenJDK 13. Le débogueur utilise l'environnement Runtime Java et est démarré avec le contenu suivant par le biais du fichier "ELOAsDebug.bat" :

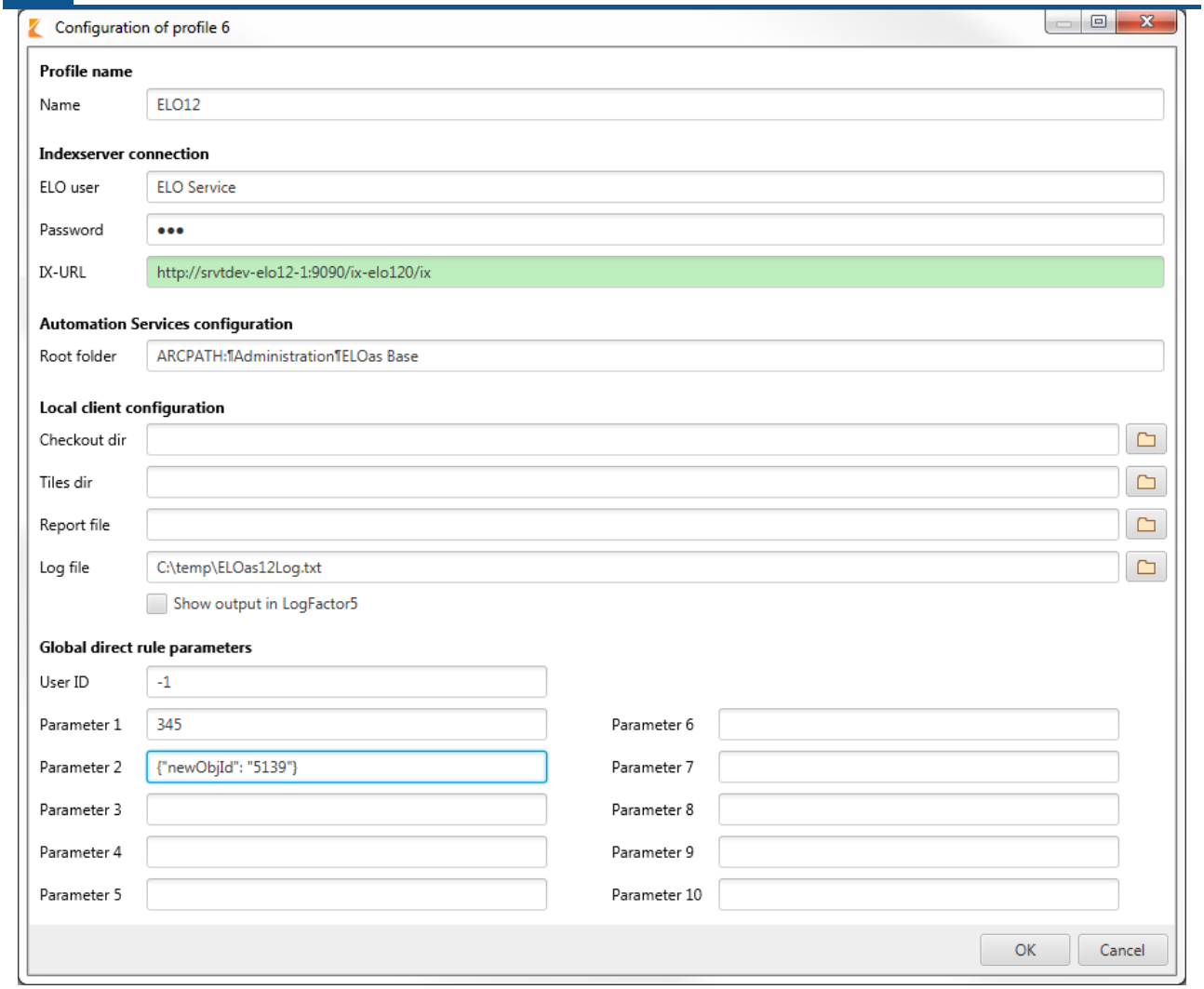
```
.\jdk-13.0.2\bin\java.exe -Xmx1000m --module-path=.\lib\modules

        --add-modules javafx.controls,javafx.base,javafx.graphics,

                javafx.web,javafx.swing
                        -classpath ".\*;.\lib\*"

                                de.elo.mover.eloasdbg.javafx.StartEloAs
```



Illustr. : fichier 'ELOAsDebug.bat' dans le système de fichiers Windows

Configuration

Cliquez sur le bouton *Config*. C'est par ici que vous pouvez atteindre le dialogue de configuration.



Illustr. : configuration du débogueur

L'ID du profil de débogueur ELOas qu'il s'agit de modifier s'affiche dans le titre. Le mot de passe utilisateur est maintenant masqué dans le champ de texte pour le mot de passe utilisateur.

Nom : le nom de profil doit se composer d'au plus 15 caractères.

ELO user : le nom de l'utilisateur ELO.

Password : le mot de passe de la connexion au serveur d'indexation.

IX-URL : l'URL du serveur d'indexation. IX-URL : le champ de texte passe au vert lorsqu'une connexion est possible.

Root folder : le chemin d'enregistrement de la configuration ELOas.

Checkout dir : le bouton derrière le champ *Checkout dir* permet la sélection du répertoire de check-out du client Java ELO.

Tiles dir : le bouton placé derrière le champ *Tiles dir* permet la sélection du répertoire surveille pour les tuiles de ELO Dropzone référencées.

Report file : le bouton placé derrière le champ *Report file* permet de sélectionner un fichier de rapport pour le débogueur ELOas.

Log file : le fichier derrière le champ *Log file* permet de sélectionner le fichier de journalisation.

Global direct rule parameters : ici, vous pouvez définir les paramètres globaux pour les règles ELOas directes. Vous pouvez modifier l'ID utilisateur ainsi que 10 paramètres.

Le dialogue de configuration a une barre de défilement qui s'affiche à partir lors rapetissement du dialogue à partir d'une taille de dialogue précise.

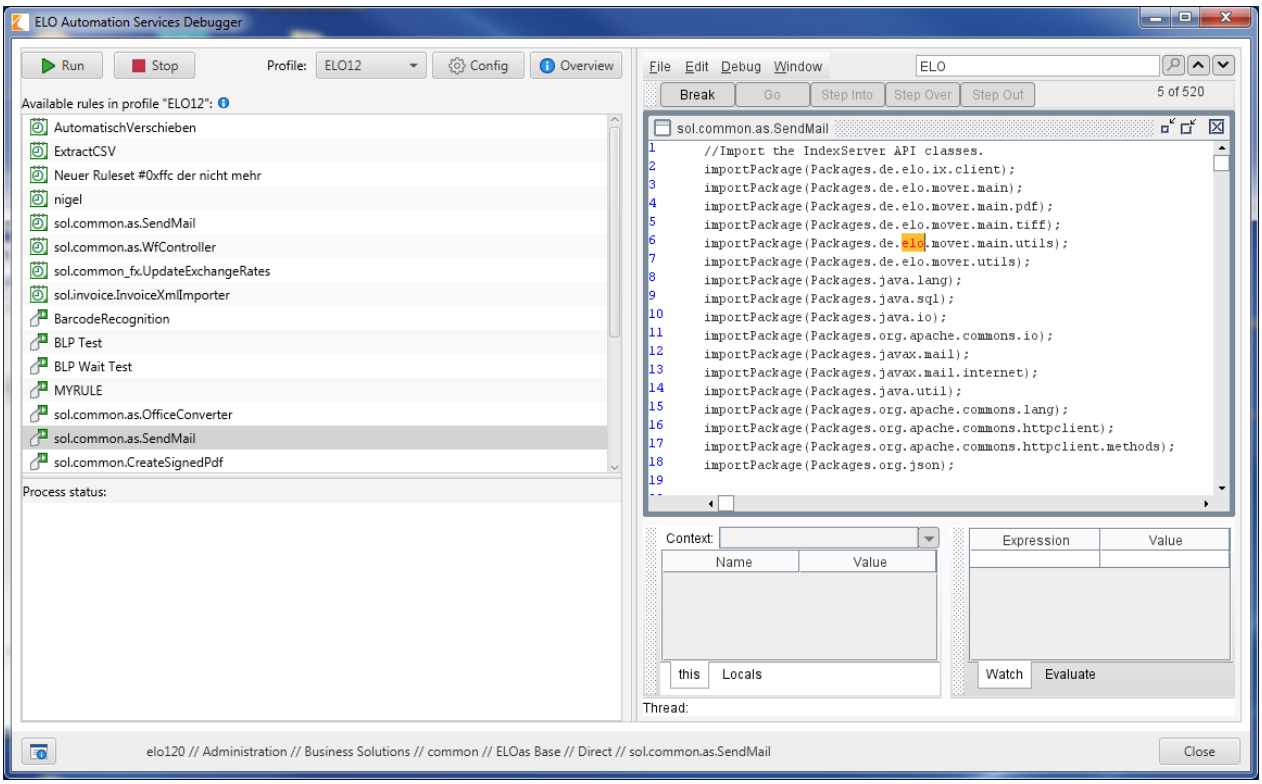
Une fois que vous avez appuyé sur le bouton OK, les réglages de profil effectués sont enregistrés dans l'éditeur du registre. Les réglages pour le profil de débogage ELOas actuel (Id: 1) sont enregistrés à la position suivante dans l'éditeur du registre :

"HKEY_CURRENT_USER\Software\JavaSoft\Prefs\elo digital office\eloas.1".

Le bouton *Annuler* vous permet de rejeter les modifications actuelles et de fermer le dialogue. Le dialogue permettant de modifier un profil de débogage ELOas peut également être refermé par le biais du bouton échap. Ce dialogue a une taille minimale adéquate. Lorsque vous agrandissez le dialogue, les différents composants de dialogue sont également agrandis de façon adéquate. Ainsi, l'utilisateur est en mesure de visualiser de longues entrées de profil.

Modifier le ruleset

Après le redémarrage, tous les rulesets de la configuration actuelle sont chargés. Ils ne sont pas encore effectués, afin que vous ayez la possibilité de placer des points de break dans le code JavaScript.



Illustr. : ELOas Automation Services, modifier le ruletset

Si vous avez plusieurs rulesets, nous vous prions de sélectionner le bon dans la fenêtre Rhino Debugger sous *Window*. Maintenant, vous pouvez placer des points de break et démarrer le ruleset.

Le démarrage d'un ruleset est exécuté par un clic de l'entrée du ruleset en question, puis avec un clic sur *Run*. Maintenant, le ruleset est activé directement; toutefois, il reste soumis à la commande par intervalles. Si vous avez réglé un démarrage du ruleset pour minuit, alors il ne sera pas actif dans le debugger avant minuit. A des fins de débogage, le réglage est "1M" – donc, une fois par minute – pour des exécutions récurrentes et "10H" – donc, toutes les 10 heures – pour des rulesets exécutés une fois.

Si vous souhaitez modifier un ruleset ou un fichier JavaScript, alors vous pouvez effectuer un check-out ou appeler directement le fichier soumis au check-out avec un éditeur approprié. Effectuez les modifications et enregistrez les données. Tant que l'éditeur n'ouvre pas exclusivement le fichier (cela est rarement le cas pour les éditeurs), vous devez refermer l'éditeur. Dans le préposé au débogage, cliquez une nouvelle fois sur *Run*. Le ruleset est rechargé et redémarré automatiquement depuis l'archive et le répertoire de check-out. Le fichier log est lui aussi créé à nouveau, de manière à ce que vous ne deviez pas travailler avec des anciennes versions de log.

Débogueur (Java FX)

Ouvrir le programme

Le débogueur ELOas est démarré par le biais du fichier "EloAsDebug.exe“ et requiert au moins JRE 1.7 sur votre ordinateur.

Le débogueur ELOas peut également être démarré avec la commande suivante :

```
"C:\Program Files\Java\jre1.8.0_152\bin\javaw.exe" -classpath ".\*;lib\*" de.elo.mover.eloasd
```

Utilisation avec OpenJDK

Le débogueur ELOas peut également être utilisé avec OpenJDK. ELOas Debugger 20 est livré avec OpenJDK 13 comme package intégral. Une installation Java distincte n'est plus nécessaire pour cette raison.

Pour ceci, cliquez sur le fichier "ELOasDebug.bat" dans le répertoire de programmes du débogueur ELOas.

Représentation sur des écrans haute résolution

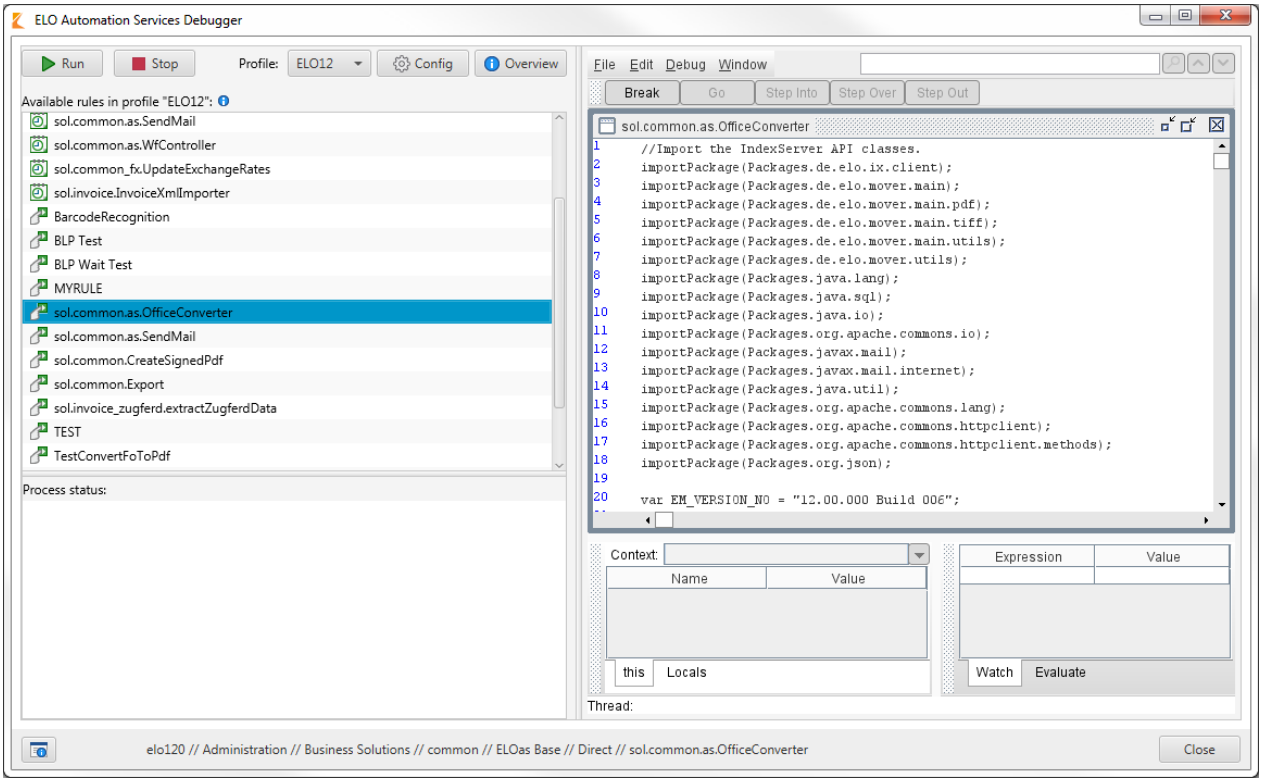
Afin que le débogueur rhino (qui est intégré dans le débogueur ELOas) soit affiché correctement sur un écran haute résolution avec une résolution horizontale de 4000 pixels à partir de la version 1703 de Windows, vous devez bien configurer l'option pour la scalabilité de l'application. L'option s'enregistre de la manière suivante :

1. Avec la touche droite de la souris, cliquez sur le fichier "EloAsDebug.exe" et sélectionnez le point de menu "Propriétés".
2. Ouvrez l'onglet "Comptabilité".
3. Cliquez sur le bouton "Modifier les réglages PPP élevés".
4. Activez la case "Overwrite behavior for high DPI scaling".
5. Dans le menu déroulant, sélectionnez "Système".
6. Cliquez sur *OK* pour enregistrer les modifications.

ELOas Debugger 20 contient la Rhino-Skripting-Engine "rhino-1.7.12.jar".

Interface utilisateur

L'interface du débogueur ELO ressemble à ceci :



Illustr. : interface utilisateur du débogueur ELOas

Une fois le débogueur ELOas démarré, c'est toujours la première règle qui est sélectionnée. Le contenu de la règle est affiché à droite.

> **Information**
>
> Le débogueur ELOas n'est actuellement que disponible en anglais.

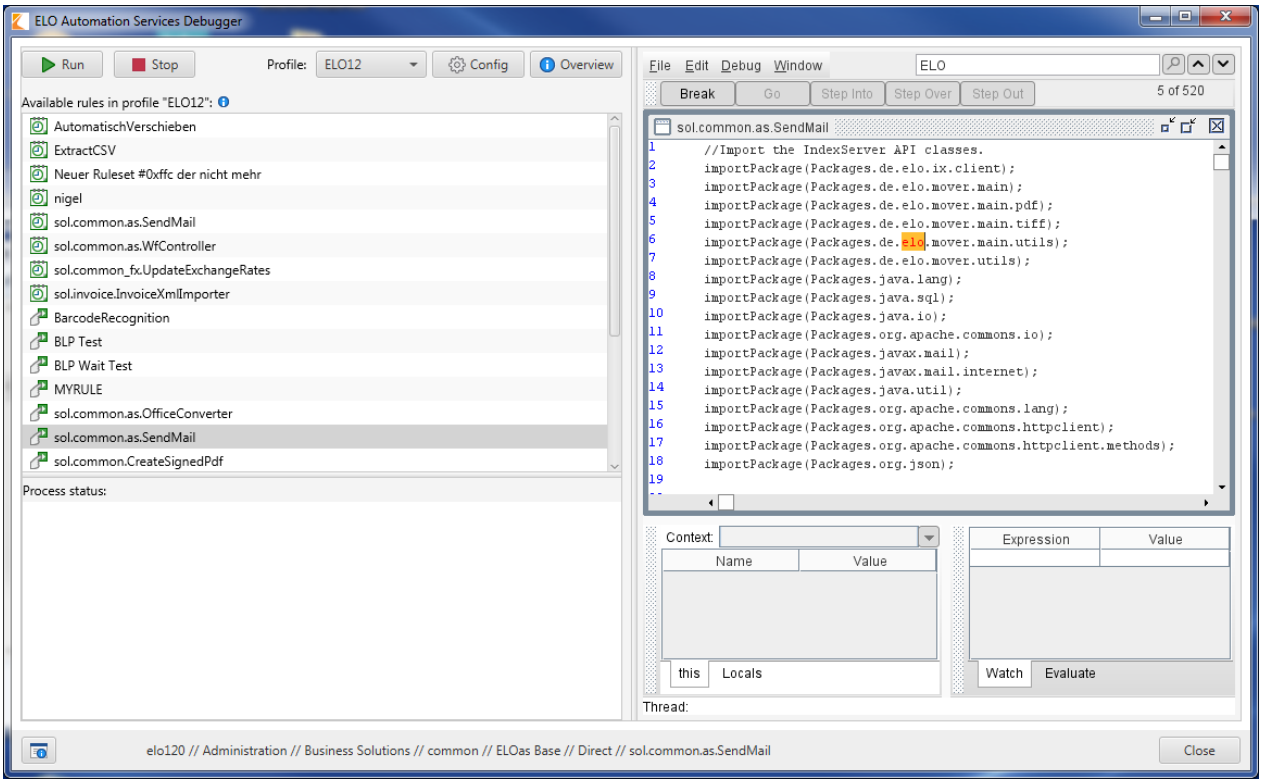
La taille et la position des dialogues de programmes sont enregistrés dans le registre et restaurés lors du prochain démarrage de programme. Le dialogue principal du débogueur ELOas est concerné, ainsi que le dialogue du programme *LogFactor5*, le dialogue pour le débogueur Rhino, le dialogue pour l'aperçu du profil, la configuration de profils et de paramètres et le dialogue *A props du programme*.

La disposition des colonnes dans le dialogue *LogFactor 5* est également enregistrée dans le registre et restaurée lors du prochain démarrage de programme.

Le débogueur ELOas contient une séparation entre la liste contenant les règles existantes et la zone de statut. La position du fractionnement est enregistrée dans le registre système et restaurée lors du prochain démarrage.

Parcourir le contenu de la règle

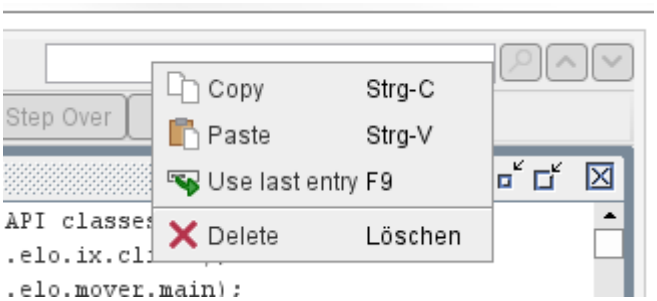
Le contenu de la règle s'affiche à droite de l'interface du débogueur ELOas. Vous pouvez parcourir le contenu grâce au champ de recherche.



Illustr. : parcourir le contenu de la règle

Si vous entrez le terme de recherche souhaité dans le champ de recherche, il est affiché en couleur dans le contenu de la règle.

Le menu contextuel du champ de recherche vous propose d'autres options.



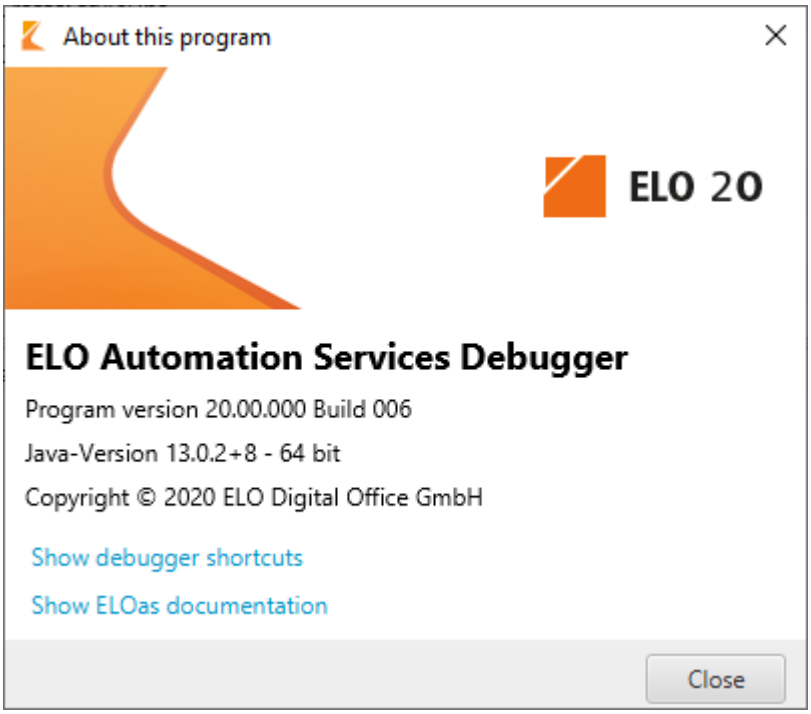
Illustr. : menu contextuel du champ de recherche

Copy vous permet de copier le terme de recherche, *Paste* vous permet d'ajouter un texte provenant de la mémoire temporaire. Avec *Use last entry,* vous entrez le dernier terme de recherche dans le champ de recherche. Avec *Delete,* vous supprimez le terme de recherche.

Messages de statut

Dans la fenêtre principale du débogueur ELOas, se trouve un champ de texte pour les messages d'état du processus d'arrière-plan. Ce champ ne peut pas être édité, mais son contenu peut être copié dans la mémoire temporaire.

Informations de programme



Illustr. : dialogue ‚A propos du programme'

Dans la partie inférieure gauche du dialogue, se trouve un bouton pour les informations sur le programme. Lorsque vous cliquez sur ce bouton, s'affiche un dialogue avec des informations sur le programme. Vous voyez de quelle version de programme et de quelle version Java il s'agit. Par ailleurs, vous trouverez des liens vers les raccourcis clavier du débogueur ELOas et vers la documentation de ELOas. Ce dialogue peut être refermé par le biais du bouton *Close*.

Démarrer une règle ELOas

Grâce au bouton *Run* pour démarrer le processus de débogage d'une règle as. Le processus de débogage est effectué dans le débogueur rhino intégré. Si l'option pour la sortie LF5 est activée dans le profil de débogage ELOas actuel, ce programme démarre dans une fenêtre distincte. Dans cette fenêtre s'affichent les sorties du débogueur ELOas. Les différents niveaux de journalisation peuvent être marqués en couleur si nécessaire. Grâce au bouton *Stop*, vous pouvez interrompre le processus de débogage d'une règle AS. Les boutons pour démarrer et interrompre une règle sont toujours actifs dans la version actuelle du débogueur ELOas.

Si aucune règle n'a été sélectionnée et que vous appuyez sur le bouton Run, l'utilisateur obtient la remarque suivante.

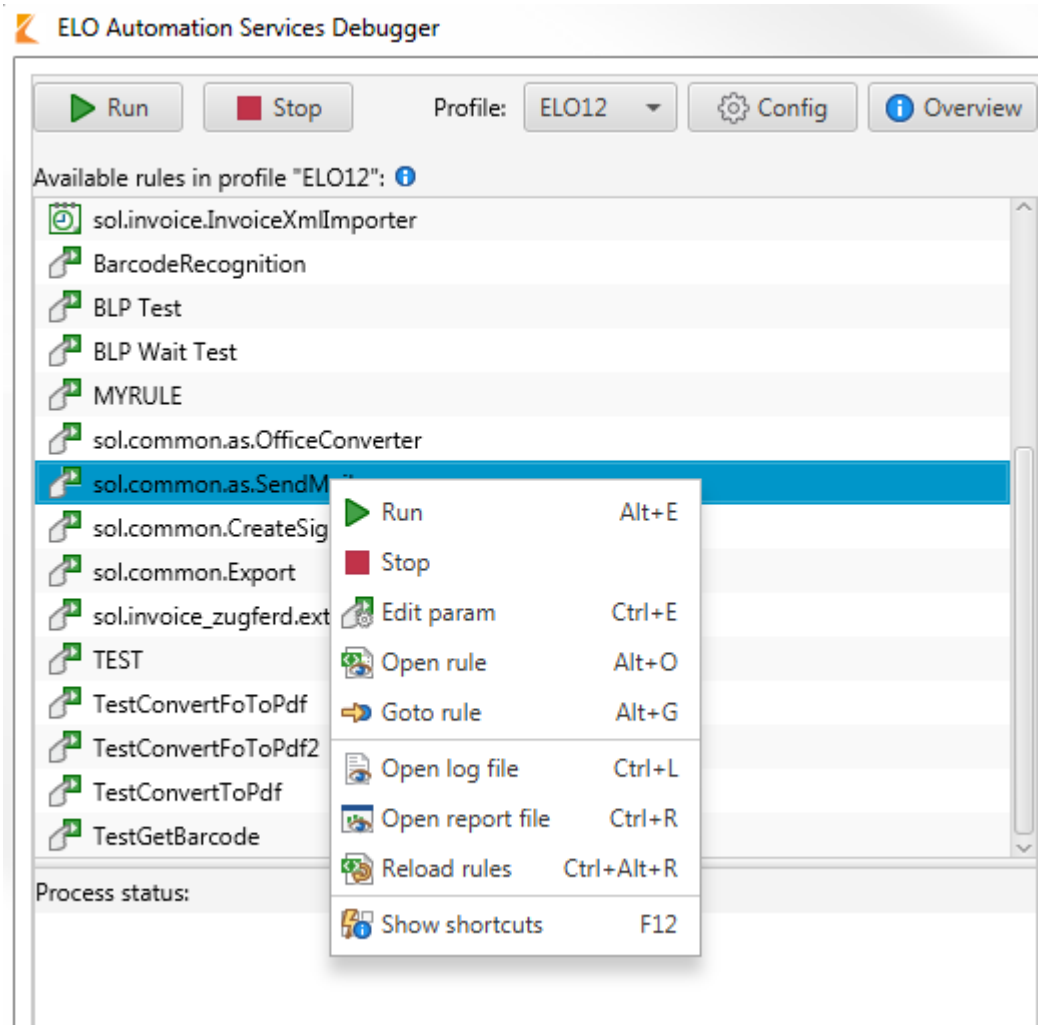
Illustr. : message d'erreur lors du démarrage, sans règle AS sélectionnée

Profils

Utiliser les profils du débogueur ELOas

Dans la fenêtre supérieure à droite du dialogue principal du débogueur ELOas, existe un menu déroulant avec les profils du débogueur ELOas. Ce menu déroulant contient 10 profils. Chaque profil peut être identifié de manière univalente en fonction du nom de profil.

Les composants graphiques correspondants sont actualisés lors de la sélection d'un profil. La liste avec les règles existante est également actualisée lors de la modification du profil. Dans cette liste s'affichent les règles du profil du débogueur ELOas actuellement utilisé. Effectuez un double-clic sur une règle pour faire démarrer le processus de débogage pour cette règle.



Illustr. : menu contextuel dans le débogueur ELOas

Cliquez sur la touche droite de la souris pour faire afficher le menu contextuel de la liste. L'utilisateur peut sélectionner l'action suivante dans le menu contextuel affiché :

Run : démarrer le processus de débogage pour la règle AS sélectionnée.

Stop : arrêter le processus de débogage.

Edit param : ajuster les paramètres de la règle. Les paramètres spécifiques pour la règle directe sont utilisés au lieu de la règle globale.

Open rule : ouvrez la règle en tant que fichier texte.

Goto rule : ouvrez l'emplacement de dépôt de la règle dans ELO.

Open log file : ouvre le fichier de journalisation du débogueur ELOas.

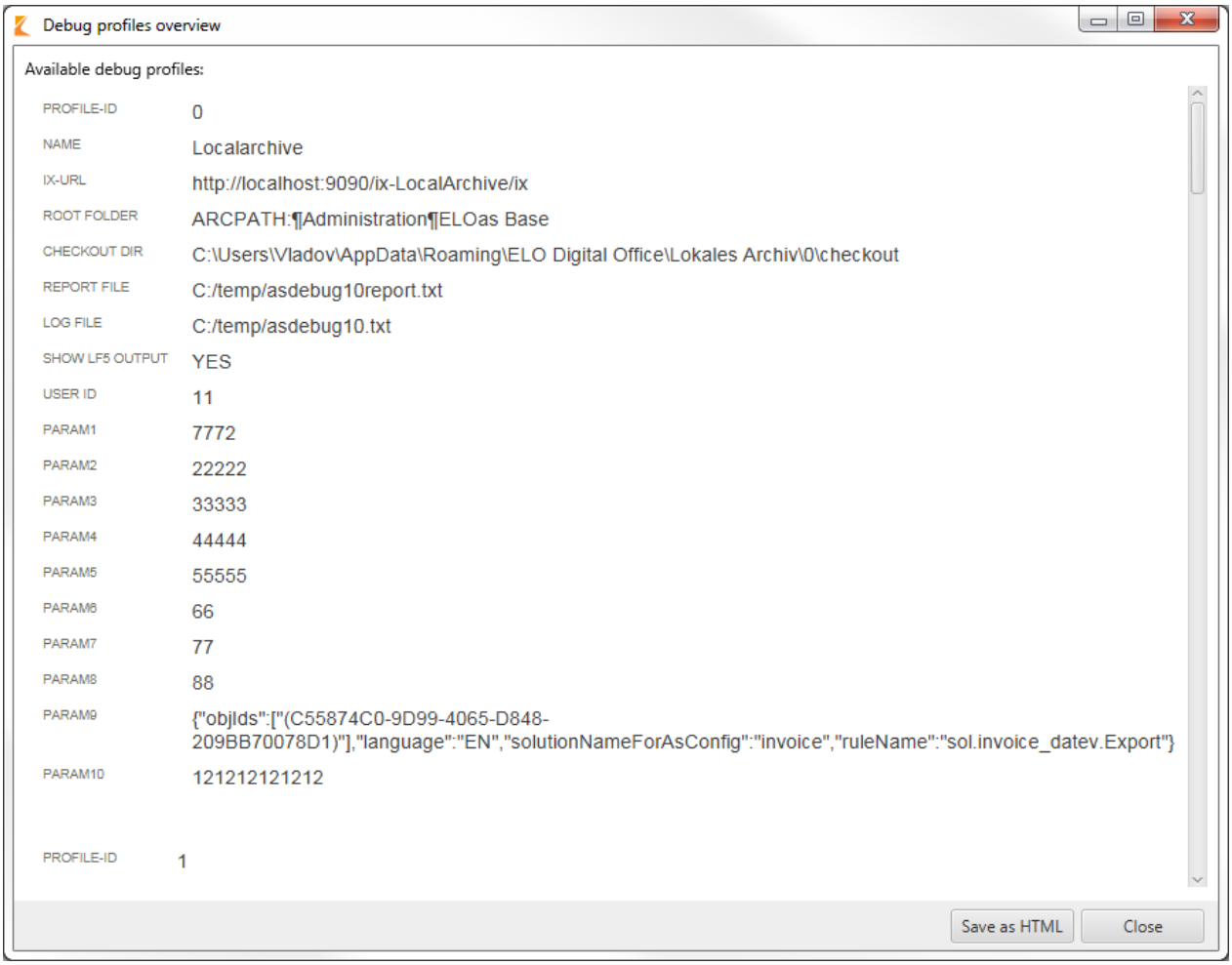
Open report file : ouvre le fichier de rapport du débogueur ELOas.

Reload rules : permet de recharger les règles ELOas existantes.

Show shortcuts : affiche les raccourcis clavier dans le débogueur ELOas.

Les règles disponibles sont affichées dans la liste. Les différentes règles se différencient par leur symbole. Le symbole de type de règle contient un texte descriptif.

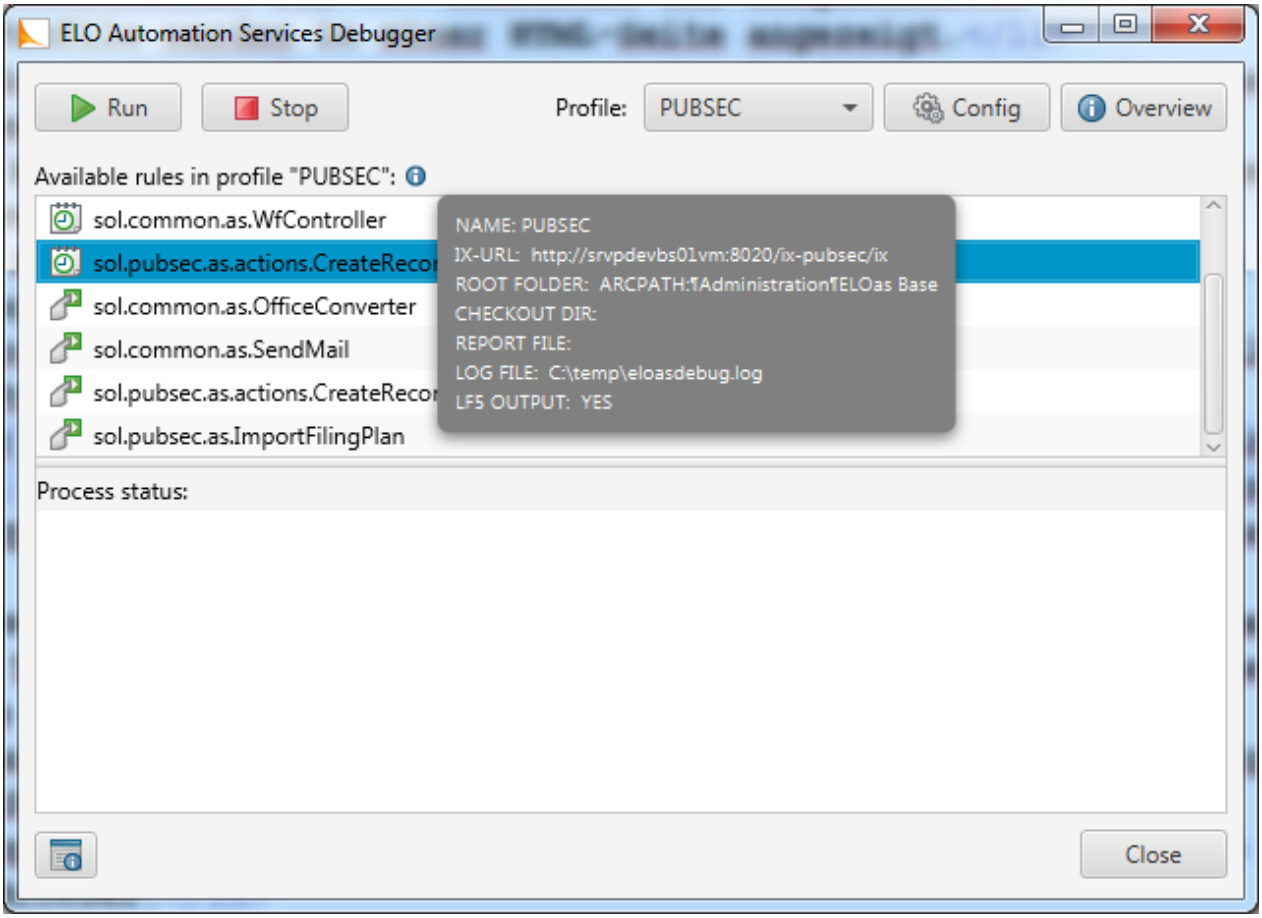
Aperçu de profil



Illustr. : aperçu de profil du débogueur ELOas

Le bouton *Aperçu* ouvre l'aperçu des profils de débogueur ELOas existants. Les profils existants sont affichés au format HTML. L'aperçu affiche les propriétés importantes d'un profil. L'aperçu peut être enregistré dans le système de fichiers local sous forme d'un fichier HTML (bouton *Save).*

Informations sur le profil sélectionné

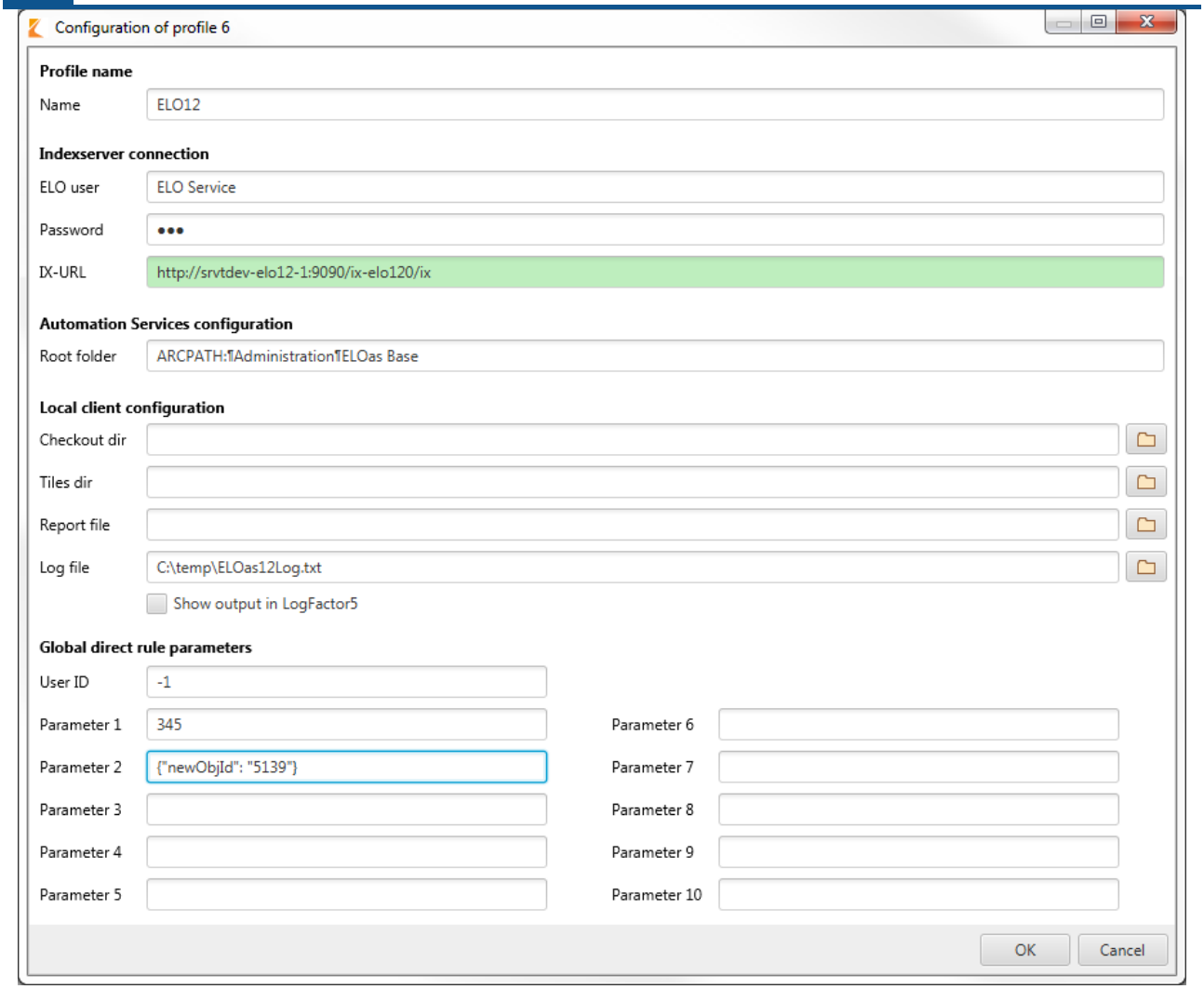


Illustr. : informations sur le profil sélectionné

Dans le dialogue principal du débogueur ELOas, est maintenant visible un symbole d'info affichant les données les plus importantes du profil sélectionné actuellement.

Modifier les profils

Vous pouvez modifier le profil de débogueur ELOas actif par le biais du bouton *Config*.



Illustr. : dialogue de configuration d'un profil

L'ID du profil de débogueur ELOas qu'il s'agit de modifier s'affiche dans le titre. Le mot de passe utilisateur est maintenant masqué dans le champ de texte pour le mot de passe utilisateur.

Nom : le nom de profil doit se composer d'au plus 15 caractères.

ELO user : le nom de l'utilisateur ELO.

Password : le mot de passe de la connexion au serveur d'indexation.

IX-URL : l'URL du serveur d'indexation. IX-URL : le champ de texte passe au vert lorsqu'une connexion est possible.

Root folder : le chemin d'enregistrement de la configuration ELOas.

Checkout dir : le bouton derrière le champ *Checkout dir* permet la sélection du répertoire de check-out du client Java ELO.

Tiles dir : le bouton placé derrière le champ *Tiles dir* permet la sélection du répertoire surveille pour les tuiles de ELO Dropzone référencées.

Report file : le bouton placé derrière le champ *Report file* permet de sélectionner un fichier de rapport pour le débogueur ELOas.

Log file : le fichier derrière le champ *Log file* permet de sélectionner le fichier de journalisation.

Global direct rule parameters : ici, vous pouvez définir les paramètres globaux pour les règles ELOas directes. Vous pouvez modifier l'ID utilisateur ainsi que 10 paramètres.

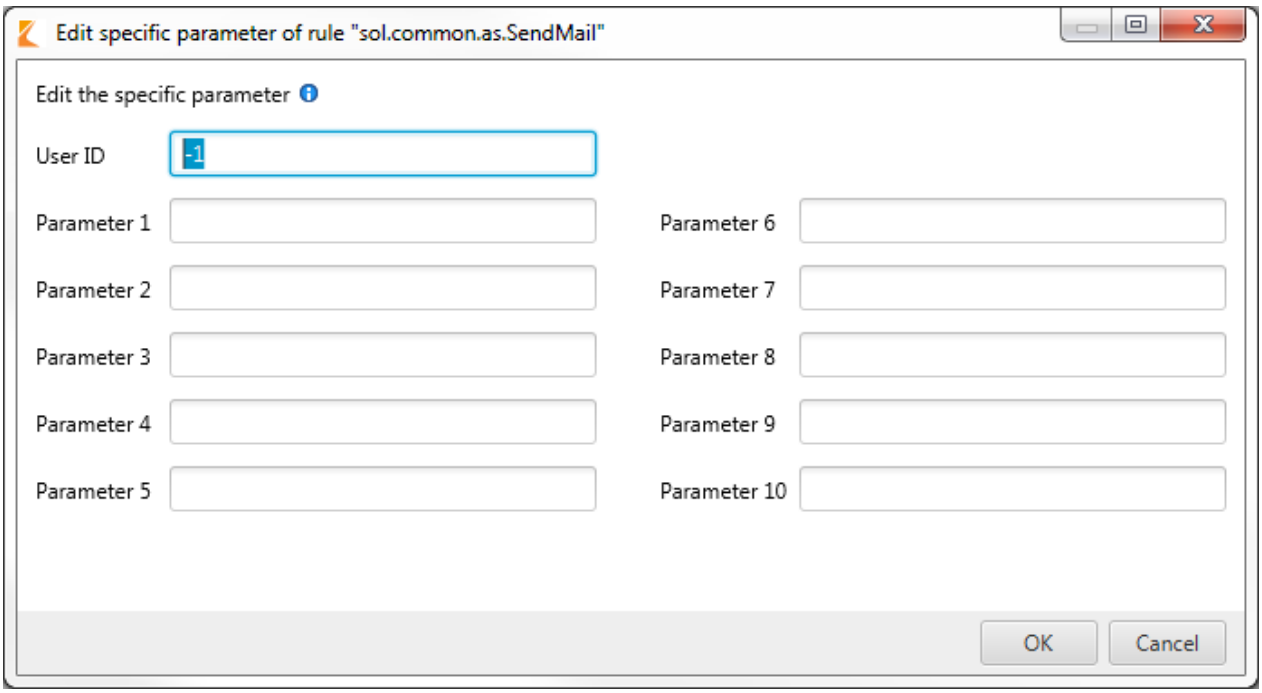
Le dialogue de configuration a une barre de défilement qui s'affiche à partir lors rapetissement du dialogue à partir d'une taille de dialogue précise.

Une fois que vous avez appuyé sur le bouton OK, les réglages de profil effectués sont enregistrés dans l'éditeur du registre. Les réglages pour le profil de débogage ELOas actuel (Id: 1) sont enregistrés à la position suivante dans l'éditeur du registre :

"HKEY*CURRENT*USER\Software\JavaSoft\Prefs\elo digital office\eloas.1".

Le bouton *Annuler* vous permet de rejeter les modifications actuelles et de fermer le dialogue. Le dialogue permettant de modifier un profil de débogage ELOas peut également être refermé par le biais du bouton échap. Ce dialogue a une taille minimale adéquate. Lorsque vous agrandissez le dialogue, les différents composants de dialogue sont également agrandis de façon adéquate. Ainsi, l'utilisateur est en mesure de visualiser de longues entrées de profil.

Traiter des règles directes

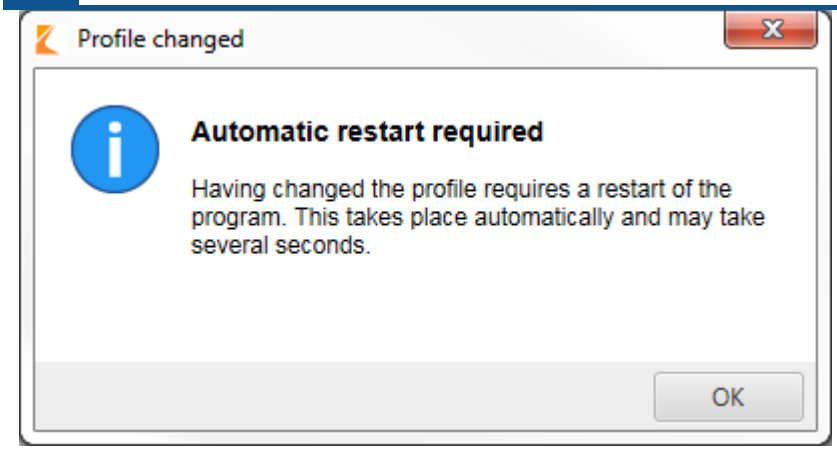


Illustr. : dialogue de traitement de règles directes

Ce dialogue vous permet de modifier les paramètres spécifiques d'une règle directe. Pour basculer dans ce dialogue, il vous suffit de sélectionner une règle dans la liste des règles existantes du menu contextuel. Dans le dialogue affiché, il est possible de modifier l'ID utilisateur et les dix paramètres prévus.

Changement de profil

Si le profil a été modifié et que vous souhaitez basculer dans un autre profil, le débogueur ELOas requiert un redémarrage.



Illustr. : dialogue lors du redémarrage automatique après un changement de profil

Raccourcis clavier

Les différents dialogues de débogueur ELOas peuvent également être lancés par le biais de raccourcis clavier.

`F1` affiche le dialogue *A propos du programme*.

`F2` ouvre l'aperçu des profils.

`F3` bascule vers le prochain résultat de recherche.

`UMSCHALT + F3` bascule vers le dernier résultat de recherche.

`F9` utilise la dernière entrée.

`F12` ouvre l'aperçu des raccourcis clavier.

`Ctrl + P` affiche le dialogue pour la configuration de profil.

`Ctrl + O` ouvre l'aperçu de profils.

`Ctrl + E` affiche le dialogue pour traiter les paramètres.

`Ctrl + I` affiche la version de programme du débogueur ELOas.

`Ctrl+S` affiche le contenu de script à l'avant-plan.

`Ctrl+ G` ouvre la fonction de filtrage.

`STRG + L` ouvre le fichier de journalisation.

`Ctrl+R` ouvre et affiche le fichier rapport défini.

`STRG + F` permet de mettre à l'avant la fenêtre "LogFactor5".

`STRG + J` ouvre le dialogue de sélection des labels.

`Ctrl + W` évalue le texte marqué.

`STRG + ALT + L` ouvre le dialogue "Go to line". Entrez la ligne vers laquelle vous souhaitez naviguer.

`STRG + ALT + R` recharge les règles.

`Alt+R` ouvre et affiche le fichier rapport défini.

`ALT + O` ouvre la règle sélectionnée.

`ALT + G` permet de basculer vers le chemin d'enregistrement de la règle dans ELO.

Bibliothèques Java

Le chapitre suivant contient une liste des bibliothèques Java contenus dans le débogueur ELOas.

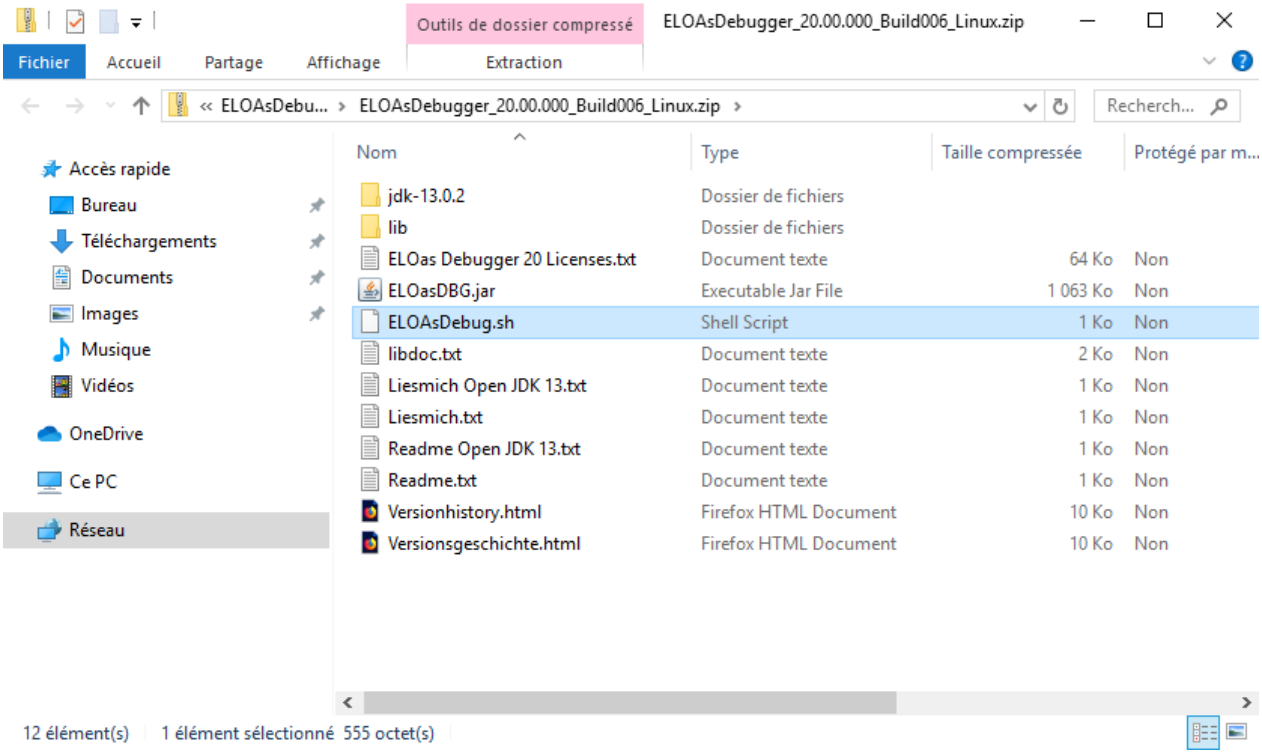
Numéro	Bibliothèque	Description
1.	EloixClient.jar	Bibliothèque permettant d'accéder au serveur d'indexation (interface du serveur d'indexation)
2.	eloserverutils.jar	Classes générales d'aide du serveur ELO
3.	commons-lang-2.6.jar	Méthodes d'aide pour les opérations String fréquentes, opération de sérialisation et Objekt Reflection.
4.	commons-lang3-3.9.jar	Méthodes d'aide pour les opérations String fréquentes, opération de sérialisation et Objekt Reflection.
5.	aspose-cad-19.7.jar	Bibliothèque permettant de créer et de gérer les documents AutoCAD
6.	aspose-cells-19.8.jar	Bibliothèque permettant de créer et de gérer les documents Microsoft Excel.
7.	aspose-diagram-19.8-jdk16.jar	Bibliothèque permettant de créer et de gérer les documents Microsoft Visio.
8.	aspose-email-19.8-jdk16.jar	Bibliothèque pour la lecture de messages d'e-mails
9.	aspose-words-19.9-jdk17.jar	Bibliothèque permettant de créer et de gérer les documents Microsoft Word.
10.	aspose-slides-19.9-jdk16.jar	Bibliothèque permettant de créer et de gérer les documents Microsoft PowerPoint.
11.	aspose.pdf-19.8.jar	Bibliothèque permettant de créer et de gérer les fichiers PDF
12.	aspose-barcode-19.8.jar	Bibliothèque permettant de créer et de gérer les codes-barres
13.	bcprov-jdk15on-1.52.jar	Bibliothèque pour l'accès aux documents cryptés
14.	httpclient-4.4.jar	Bibliothèque pour l'envoi de demandes http
15.	httpcore-4.4.jar	Bibliothèque pour l'envoi de demandes http
16.	jai_codec.jar	Bibliothèque pour le traitement d'images dans Java
17.	jai_core.jar	Bibliothèque pour le traitement d'images dans Java
18.	jai_imageio patch.jar	Bibliothèque pour le traitement d'images dans Java
19.	log4j-1.2.17.jar	Bibiliothèque pour les sorties log dans les applications Java
20.	slf4j-log4j12-1.7.25.jar	Bibiliothèque pour les sorties log
21.	slf4j-api-1.7.25.jar	Interface de SLF4J-Logger
22.	jcl-over-slf4j-1.7.25.jar	Bibliothèque pour la migration vers SLF4J Logger.
23.	rhino-1.7.12.jar	Bibliothèque pour l'exécution des scripts JavaScript
24.	commons-io-2.7.jar	Bibliothèque avec des méthodes d'aide utiles pour les opérations de fichiers fréquentes

Numéro	Bibliothèque	Description
25.	bcpkix-jdk15on-1.59.jar	Bibliothèque pour le traitement de documents Microsoft Office cryptés
26.	bcmail-jdk15on-1.59.jar	Bibliothèque pour le traitement d'e-mails cryptés
27.	bcprov-jdk15on-1.59.jar	Bibliothèque pour le traitement de documents Microsoft Office cryptés
28.	mlibwrapper_jai.jar	Autre bibliothèque pour le traitement d'images dans Java
29.	jna.jar	Bibliothèque pour l'accès aux ressources système
30.	platform.jar	Autre bibliothèque pour l'accès aux ressources système
31.	forms-1.1.0.jar	Bibliothèque pour la création de layouts des composants graphiques
32.	commons-codec-1.9.jar	Bibliothèque avec des classes de codage et de décodage générales pour Base 64, Hex et URLs.
33.	pdfbox-2.0.18.jar	Bibliothèque pour l'accès aux fichiers PDF
34.	fontbox-2.0.18.jar	Bibliothèque pour la police de fichiers PDF
35.	xmpbox-2.0.18.jar	Autre bibliothèque pour le travail avec les documents PDF
36.	javax.mail-1.6.2.jar	Bibliothèque pour l'envoi des e-mails
37.	activation-1.1.1.jar	Bibliothèque d'aide pour l'envoi des e-mails
38.	metadata-extractor-2.13.0.jar	Bibliothèque pour l'extraction de métadonnées des fichiers image
39.	xmpcore-6.1.10.jar	Bibliothèque pour le traitement, l'impression et la conversion de documents
40.	db2jcc4.jar	Bibliothèque pour le pilote de la base de données DB2
41.	json-20190722.jar	Bibliothèque pour la création de Strins JSON
42.	gson-2.8.6.jar	Autre bibliothèque pour la création de Strings JSON
43.	sqljdbc4.jar	Bibliothèque pour le pilote de la base de données Microsoft SQL Server.
44.	ojdbc6.jar	Bibliothèque pour le pilote de la base de données Oracle
45.	imgscalr-lib-4.2.jar	Bibliothèque pour l'ajustement
46.	poi-4.1.0.jar	Bibliothèque pour l'accès aux documents Microsoft Office
47.	poi-scratchpad-4.1.0.jar	Autre bibliothèque pour l'accès aux documents Microsoft Office
48.	poi-ooxml-4.1.0.jar	Autre bibliothèque pour l'accès aux documents Microsoft Office
49.	poi-ooxml-schemas-4.1.0.jar	Autre bibliothèque pour l'accès aux documents Microsoft Office
50.	xmlbeans-3.0.1.jar	Autre bibliothèque pour l'accès aux documents Microsoft Office
51.	commons-compress-1.18.jar	Bibliothèque d'aide pour l'accès aux documents Microsoft Office
52.	commons-collections4-4.3.jar	Bibliothèque d'aide pour l'accès aux documents Microsoft Office
53.	core-3.4.0.jar	Bibliothèque pour la reconnaissance des codes-barres

Numéro	Bibliothèque	Description
54.	javase-3.4.0.jar	Autre bibliothèque pour la reconnaissance des codes-barres
55.	fop.jar	Bibliothèque pour la conversion des fichiers XML au format PDF
56.	xmlgraphics-commons-2.3.jar	Bibliothèque pour le traitement de fichiers XML
57.	batik-all-1.10.jar	Bibliothèque pour les applications qui utilisent des images au format SVG
58.	avalon-framework-impl-4.3.1.jar	Bibliothèque pour la création et la configuration de composants
59.	avalon-framework-api-4.3.1.jar	Interface de la bibliothèque pour la création et la configuration de composants
60.	serializer-2.7.2.jar	Bibliothèque pour la sérialisation
61.	xalan-2.7.2.jar	Bibliothèque pour la conversion de documents XML vers HTML
62.	xercesImpl-2.9.1.jar	Bibliothèque pour un analyseur XML
63.	xml-apis-1.3.04.jar	API Java pour les opérations XML
64.	xml-apis-ext-1.3.04.jar	Bibliothèque pour une interface DOM, SAX, et JAXP
65.	jsch-0.1.55.jar	Bibliothèque pour l'implémentation Java de SSH2
66.	jacob.jar	Bibliothèque pour l'accès aux objets COM à partir d'un environnement Java
67.	jacob-1.19-Lib.jar	Bibliothèque avec les DLL Jacob pour l'accès aux objets COM à partir d'une application Java
68.	postgresqljdbc4.jar	Pilote JDBC pour la base de données PostgreSQL
69.	quartz-2.3.0.jar	Bibliothèque pour l'exécution des processus devant être effectués à un moment précis
70.	quartz-jobs-2.3.0.jar	Autre bibliothèque pour l'exécution des processus devant être effectués à un moment précis

ELOas Debugger sous Linux

ELOas Debugger 20 pour Linux est également livré en tant que package intégral qui est copié à l'emplacement requis dans le système de fichiers. Le débogueur est démarré par le biais du fichier "ELOasDebug.sh".



Illustr. : package du programme de ELOas Debugger pour Linux

Autres thèmes

Installation manuelle de ELOas

Ce document décrit l'installation manuelle des ELO Automation Services (ELOas). Sous ELOprofessional, le module est créé automatiquement par l'installation serveur. Dans le cas d'une installation ultérieure ou dans un environnement dispersé, il faut effectuer une installation manuelle.

Comme presque tous les modules provenant de la ligne de serveur ELOenterprise, ELOas est programmé comme servlet et requiert un environnement runtime Java ainsi qu'un serveur d'application, par exemple Tomcat 9.0. Pour cela, au minimum la version 11 de Java est requise.

La configuration est déposée dans un fichier XML *config.xml* dans le répertoire de configuration standard de ELO. Vous pouvez ainsi effectuer les mises à jour sans aucun problème et la configuration subsiste. La langue standard pour ELOas est déterminée en fonction de la langue d'installation sélectionnée lors de ELO Server Setup; par ailleurs, elle est entrée dans le fichier *config.xml* dans le paramètre `language` (dans notre exemple, nous avons entré "en" pour l'anglais) :

```
<entry key="language">en</entry>
```

Les directives d'exécution de ELOas avec les règles, les listes de traductions et les scripts de base se trouvent dans un classeur dans l'archive. Dans la configuration, seul l'accès au serveur d'indexation ELO et ce classeur de base doivent être définis.

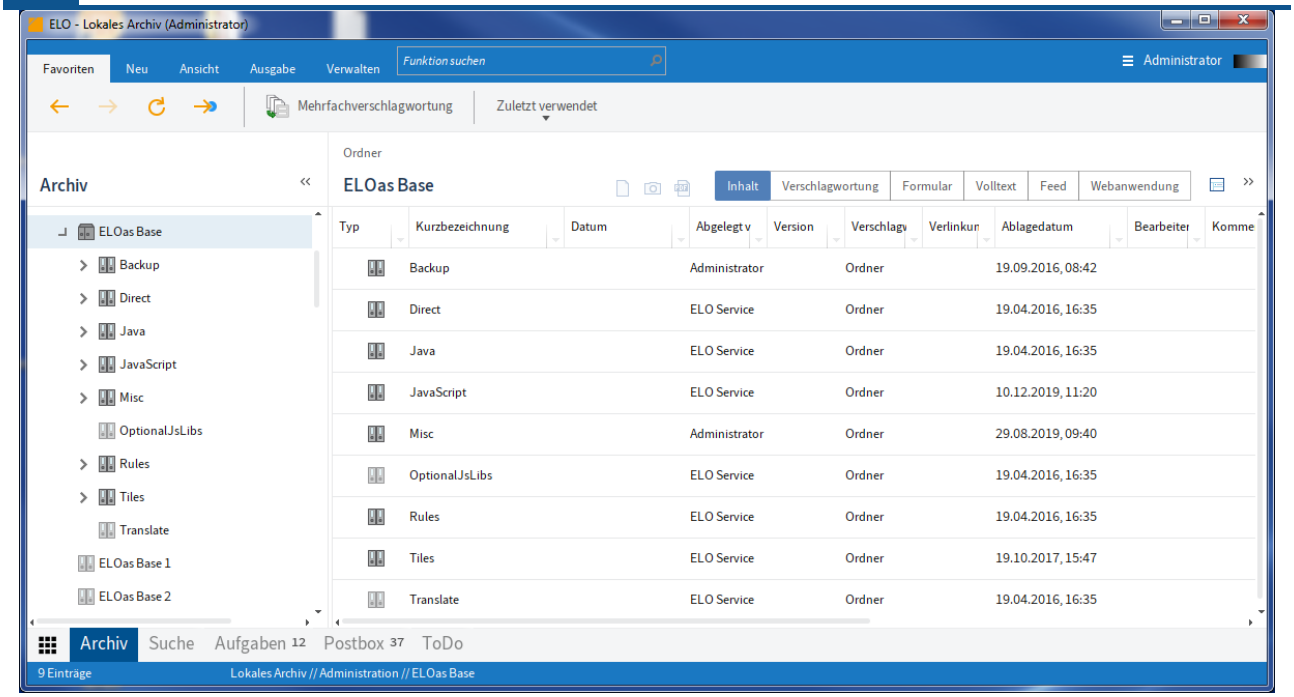
Fichiers requis

Dans l'archive ZIP destinée à l'installation manuelle, vous trouverez les fichiers suivants:

ELOas.war
ELOas.xml
logback.xml
config.xml
ELO Automation Services Konfiguration.zip
Installation.pdf
JavaScriptCode.pdf
Regeldefinition.pdf

Préparations pour l'installation

Pour l'exploitation ELOas, les librairies ELOas standards doivent se trouver dans le classeur "JavaScript". Les nouvelles versions de ELOas installer automatiquement les libraries ELOas standards, si celles-ci ne sont pas disponibles. Les librairies ELOas standards actuelles peuvent être téléchargées et installées à tout moment à partir du site officiel de *scripting**.



Illustr. : classeur pour les rules dans ELO

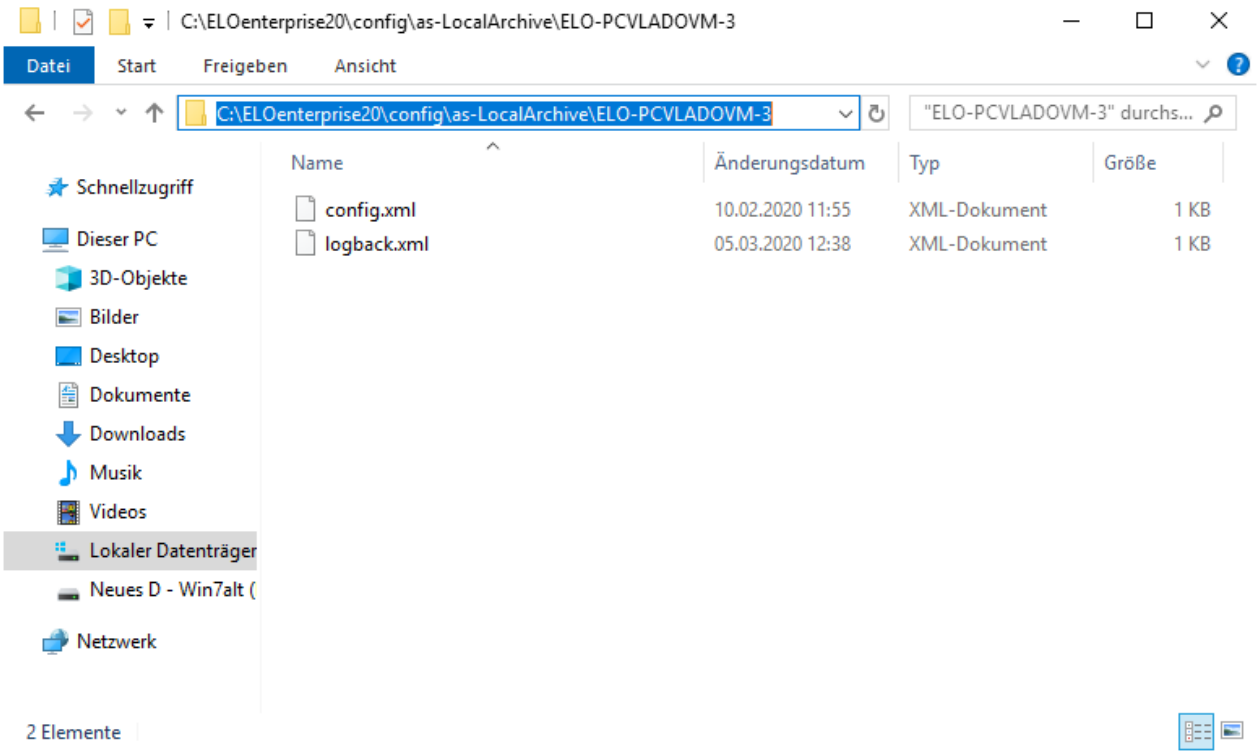
Le sous-classeur *Rules* contient les règles personnalisées, voici un exemple qui peut être utilisé comme modèle pour vos propres solutions.

Les fichiers *ELOas.war* et *ELOas.xml* doivent être renommés selon le nom d'archive et la convention standard ELO pour les noms de service: *as-<nom de l'archive>.war* ou *as-<nom de l'archive>.xml.* Pour l'archive "elo20", il s'agirait alors de "as-elo20.war" ou de "as-elo20.xml". Veuillez respecter les minuscules et majuscules, étant donné que celles-ci sont importantes pour un accès ultérieur. Ces deux fichiers sont alors copiés dans un répertoire temporaire sur votre ordinateur, sur lequel tourne le serveur d'application (par exemple C:\TEMP).

Dans le fichier *ELOas.xml*, le chemin pour le répertoire de configuration de votre environnement ELO doit être entré :

```
<?xml version='1.0' encoding='UTF-8'?>
<Context path="/as-elo20">
    <Environment name="webappconfigdir"
                 value="G:\ELOprofessional\config\as-elo20"
                 type="java.lang.String" override="false"/>
</Context>
```

Un sous-répertoire est créé pour cette configuration ELOas pour les fichiers *clogback.xml* et *config.xml*, et ces deux fichiers sont copiés à cet endroit.



Illustr. : répertoire de configuration pour ELOas

Le nom du répertoire de configuration devrait commencer par "as-", puis contenir le nom de l'archive. Pour l'archive "elo20", il doit donc porter le nom "as-elo20". Dans le fichier *logback.xml,* le chemin pour le répertoire du sortie doit être adapté à l'installation locale.

<file>C:/Programme/Tomcat 9.0/logs/as-elo20.log</file>

Dans le fichier *config.xml*, les paramètres d'accès au serveur d'indexation doivent être adaptés.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>parameters for this web application</comment>
<entry key="url">http://testserver:8080/ix-elo20/ix</entry>
<entry key="user">Services</entry>
<entry key="password">130-167-2-31-129-121-203-174-234-167-21-87-88-80-78-122</entry>
<entry key="rootguid">(F6C173D7-3F71-4559-91E5-4886139B12CF)</entry>
</properties>
```

La clé `url` contient le chemin d'accès au serveur d'indexation ELO. Soyez attentif aux majuscules et minuscules, en cas d'erreur, le serveur d'indexation sera introuvable.

La clé `user` contient le nom d'authentification de ELOas au serveur d'indexation. Normalement, vous devriez créer un propre compte pour des services additionnels, qui ne peut pas être utilisé par d'autres utilisateurs.

La clé `password` contient le mot de passe ELO. A des fins de test, vous pouvez effectuer cette entrée dans le texte clair. Après le démarrage du service, le rapport contient une remarque

concernant le codage. Vous pouvez copier ce texte dans la configuration avec Copier&Coller à partir du rapport.

La clé `rootguid` contient le GUID du classeur de base de ELOas, le préréglage est le GUID du classeur d'exemple à partir du jeu de données d'importation. Si vous avez créé un propre registre pour ces données, vous pouvez alors simplement définir le GUID en utilisant le script suivant (fichier GetGuid.vsb dans l'archive ZIP) dans le client Windows ELO :

```
Set Elo=CreateObject("ELO.professional")
if Elo.SelectView(0)=1 then
    Id=Elo.GetEntryId(-1)

    if Id>1 then
        if Elo.PrepareObjectEx( Id, 0, 0 ) > 0 then
            call Elo.ToClipboard(Elo.ObjGuid)
            MsgBox Elo.ObjGuid
        end if
    end if
end if
```

Ce script détermine le GUID de l'entrée sélectionnée actuellement et le copie dans le presse-papiers Windows. A partir de là, vous pouvez l'adopter dans la configuration à partir de l'éditeur par Ctrl-V.

La clé `rootguid` est également utilisée pour la configuration de plusieurs instances ELOas. Vous pouvez exploiter jusqu'à dix instances ELOas. Vous pouvez créer un classeur "ELOas Base" sous *<nom de l'archive> // Administration*. Dans le fichier *config.xml*, créez des paramètres `rootguid` distincts pour chaque classeur "ELOas Base"-comme dans l'exemple suivant :

```
<entry key="rootguid">(F6C173D7-3F71-4559-91E5-4886139B12CF)</entry>
<entry key="rootguid1">(D6EF1F0B-ADE4-C3E2-74F9-3658ED55449A)</entry>
<entry key="rootguid2">(2CFDEA54-3DA9-E567-F335-6F3D223C9BAF)</entry>
```

Les règles ELOas des différents classeurs "ELOas Base" sont effectuées indépendamment les unes des autres. Même s'il existe plusieurs instances, les logs sont créés dans un fichier journal (log4j.properties). Le chemin du fichier log est défini dans le fichier de configuration *logback.xml*.

La clé `tempdir` contient en option un répertoire pour le téléchargement temporaire des fichier texte, si les données XML et JavaScript doivent se trouver dans des fichiers texte, à la place du texte supplémentaire. Si `tempdir` est vide ou qu'il n'existe pas, la version du texte supplémentaire est utilisée automatiquement, sinon, c'est la version du fichier texte.

```
<entry key="tempdir">C:\Temp\ELOas</entry>
```

Remarque

> Lors de la création d'un nouvel utilisateur pour ce service, le serveur d'indexation réagit à cette modification avec un délai. Par sécurité, vous pouvez supprimer le cache utilisateur sur la page de statut du serveur d'indexation ELO, afin de forcer une actualisation immédiate.

Déployement des fichiers

Maintenant, vous pouvez entrer les paramètres pour l'installation dans le serveur d'application. Le chemin de contexte (n'est pas en option, même si c'est noté ainsi dans la console d'administration du Tomcat) contient le nom de l'application Web, les deux chemins de fichiers pointent sur la configuration et le fichier de programme. Il suffit de cliquer sur *Installer* pour installer l'application.

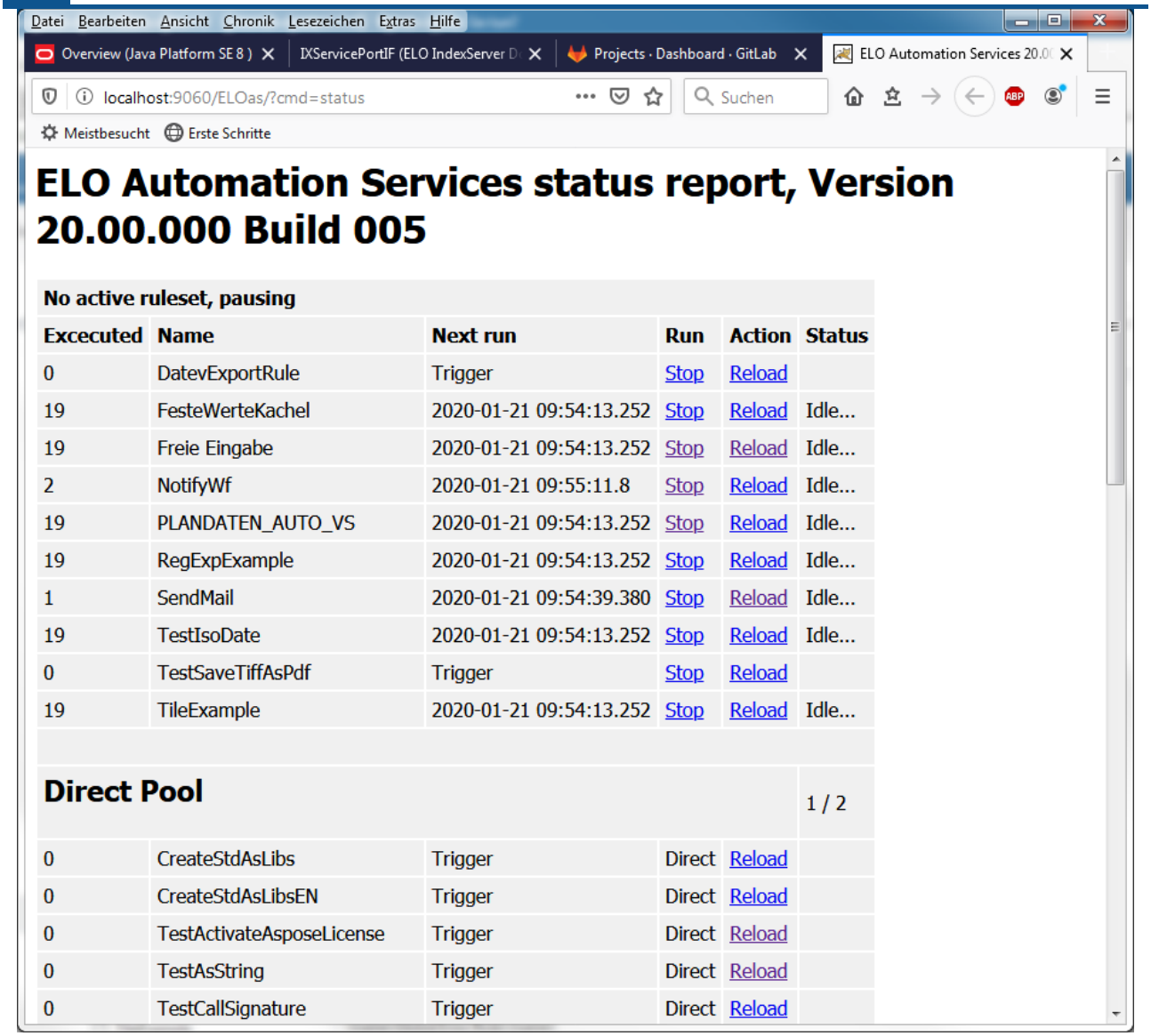
Illustr. : entrer les paramètres pour le déploiement

Le fichier "ELOas.war" dans la version actuelle d'ELOas contient un fichier de texte du nom "version.txt". Ce fichier contient les extensions des différentes versions ELOas.

Afficher la page des statuts

ELOas dispose d'une propre page de statut, celle-ci peut être atteinte par l'adresse URL suivante:

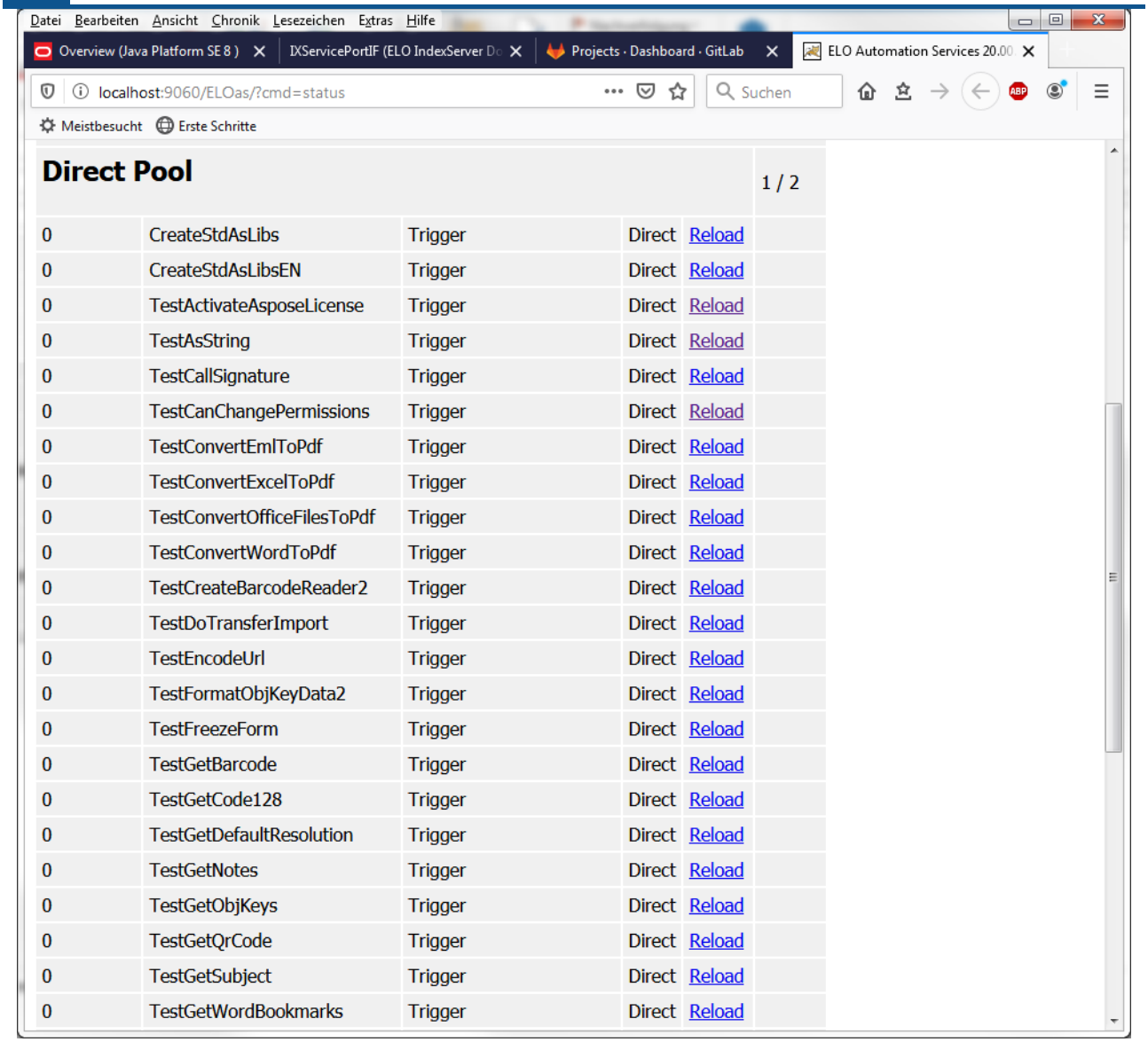
http://<SERVERNAME>:9070/as-<NOM DE L'ARCHIVE>/as?cmd=status



Illustr. : page de statut ELOas avec règles actives

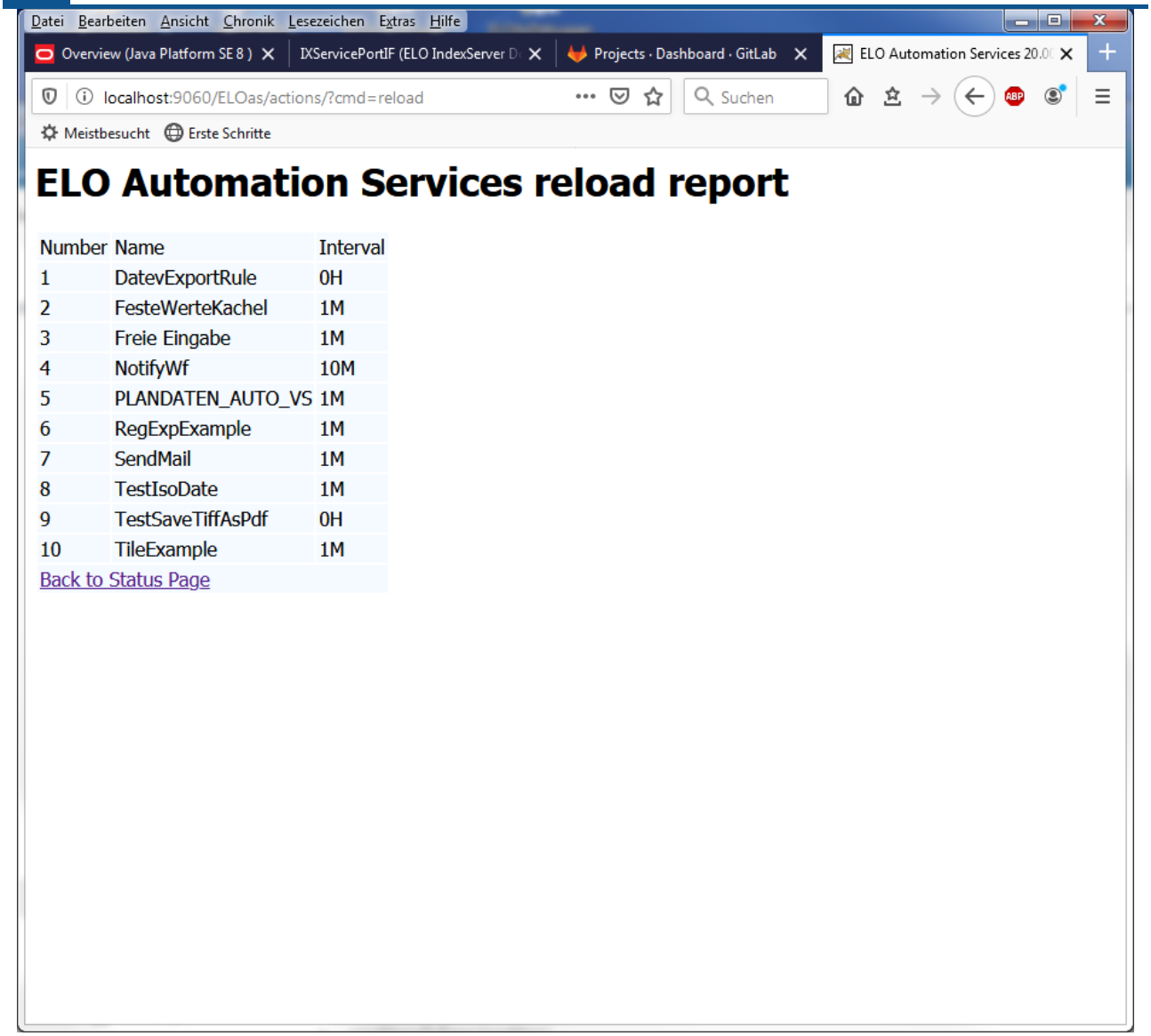
Les rulesets actifs sont affichés sous forme de liste sur la page de statut, avec l'information, combien de fois ils ont déjà été exécutés et quand la prochaine exécution est prévue.

En cas d'erreur JavaScript, celle-ci est affichée sur la page de statut, afin le numéro de ligne de l'erreur et du code de programme dans cette section.



Illustr. : appliquer les modifications avec Reload

Les modifications des règles et des scripts dans l'archive peuvent être prises en compte par un clic sur *Reload* sans redémarrer le serveur.



Illustr. : ELO Automation Services reload report

Back to Status Page vous permet de revenir à l'affichage normal de l'état.

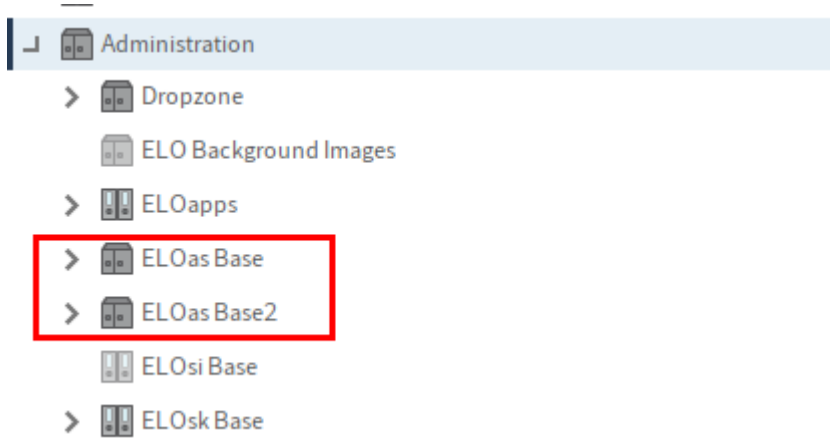
Sur l'onglet *Ajouter*, les catalogues contiennent des éléments, qui devraient être coordonnés avec la mise en page générale du document. A l'aide de ces catalogues, vous pouvez ajouter de tableaux, des en-têtes, des notes de fin de page, des listes etc.

Installer plusieurs instances ELOas

Il est possible d'installer plusieurs instances des ELO Automation Services pour la même archive (ELO Document Manager) dans les environnements ELOenterprise. Etant donné que ELOas doit utiliser un GUID fixe pour son classeur ruleset dans l'archive, le programme ELO Server Setup ne peut pas être utilisé pour l'installation de plusieurs instances de ELOas dans la même archive. Plusieurs instances ELOas ne peuvent pas se partager le même classeur de base.

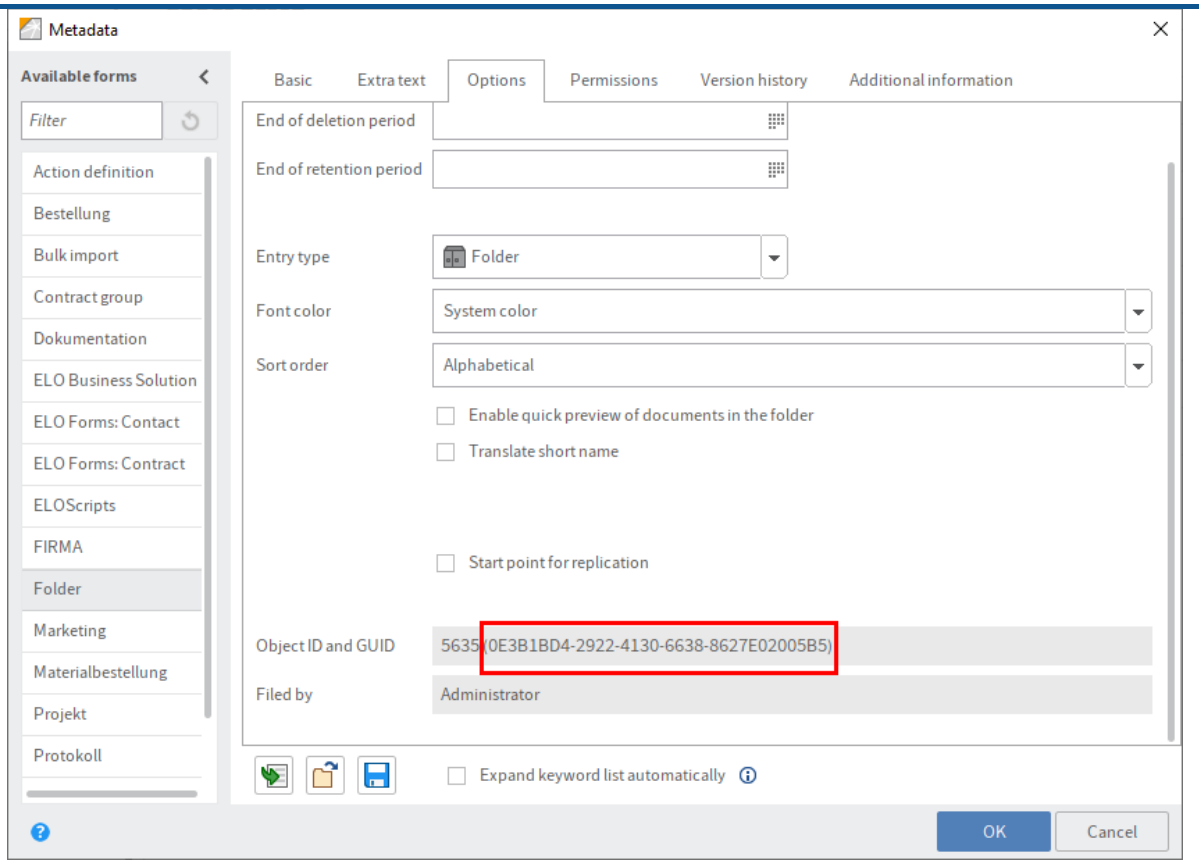
Tout d'abord, installez des instances supplémentaires de ELO Automation Services pour une archive de la manière suivante.

1. Créez une copie du classeur *ELOas Base*.



Illustr. : deuxième classeur ELOas Base

2. Copiez le GUID du nouveau classeur *ELOas Base* dans un éditeur de texte.



Illustr. : GUID du deuxième classeur ELOas Base

3. Stoppez l'instance Tomcat à l'emplacement auquel vous souhaitez installer la nouvelle instance ELOas.

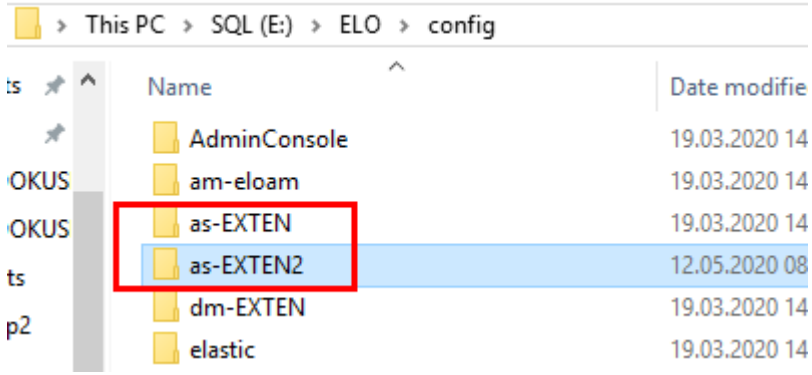
4. Basculez vers le répertoire de configuration de l'application Web (`<tomcat install>\conf\Catalina\localhost`), et copiez le fichier XML de l'application. Dans cet exemple, `as-EXTEN01.xml` est copié vers `as-EXTEN02.xml`.

> **Information**
>
> Si nécessaire, vous pouvez également copier le fichier de configuration ELOas sur un autre serveur Tomcat.

1. Ouvrez le fichier XML copié dans un éditeur de texte, et ajustez les entrées pour `webappconfigdir` et `Context path` à la nouvelle instance de ELOas. Dans notre exemple :

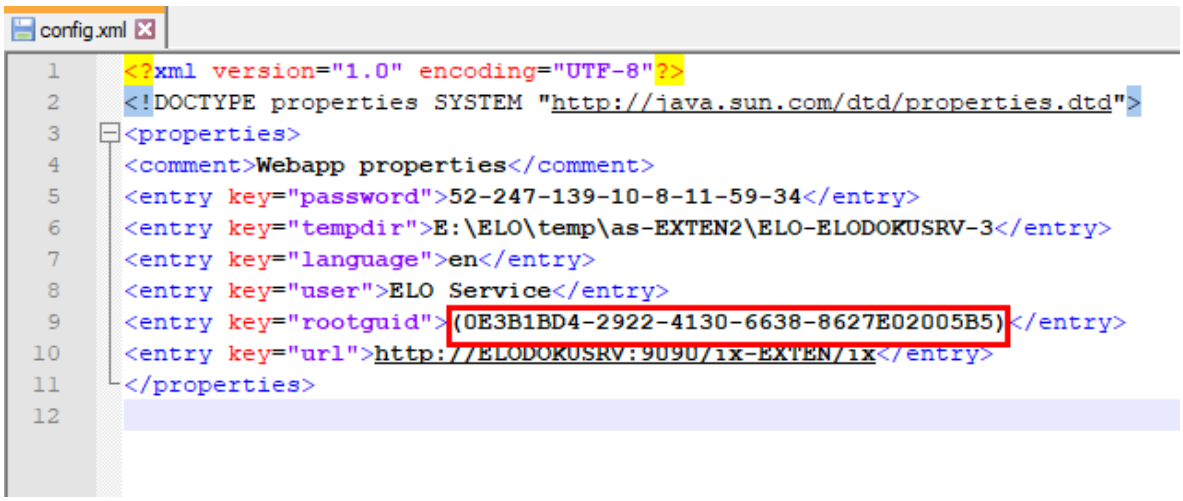
```
<Context docBase="E:\ELO\prog\webapps\as.war" path="/as-EXTEN2" unpackWAR="true">
<Environment name="webappconfigdir" override="false" type="java.lang.String" value="E:
</Context>
```

2. Naviguez vers le répertoire *config*, comme nous l'avons décrit dans l'étape précédente. Copiez le répertoire de configuration pour ELOas, pour créer un nouveau répertoire de configuration pour la nouvelle instance.



Illustr. : répertoire copié et renommé

3. Ouvrez le nouveau répertoire, puis ouvrez le fichier *config.xml* dans un éditeur de texte.

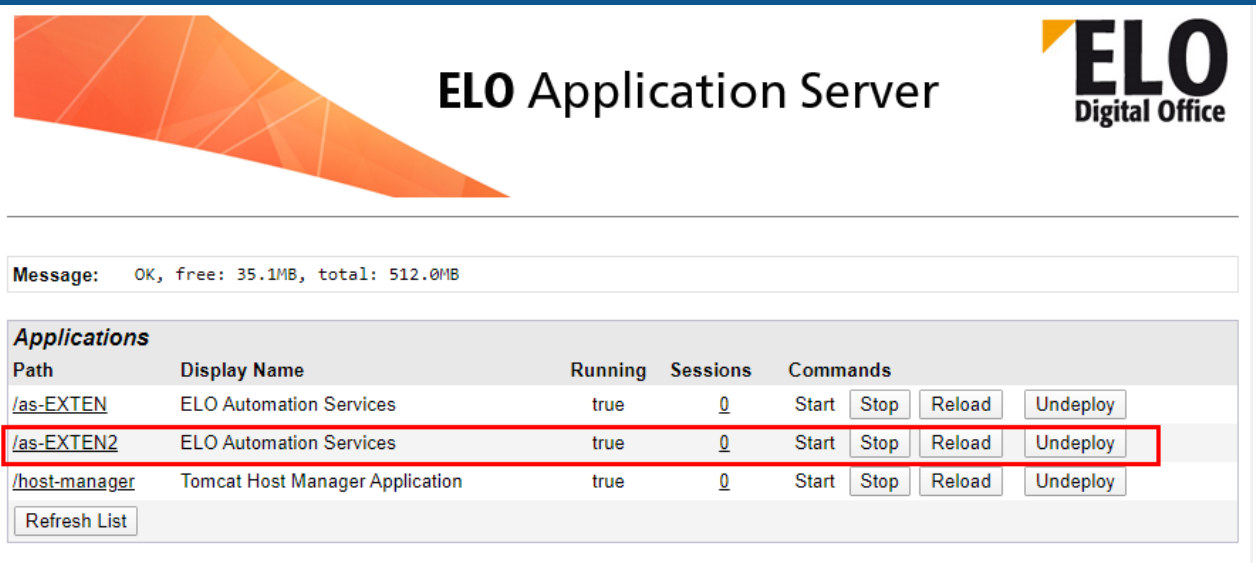


Illustr. : GUID adapté pour la deuxième instance de ELOas

4. Ajustez l'entrée `rootguid` de manière à ce que le GIUD soit identique à celui du classeur d'archive de l'étape 2 :

5. Ouvrez le fichier *logback.xml* dans le même répertoire et donnez un autre nom au fichier de rapport.

6. Démarrez ELO Application Server (Tomcat).



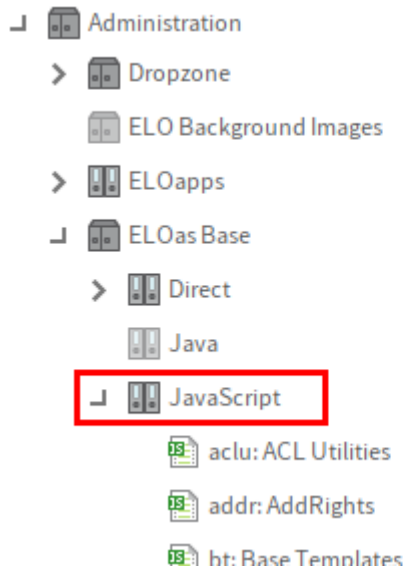
Illustr. : deuxième instance ELOas sur le serveur

1. Dans Tomcat Server Manager, vérifiez si la nouvelle instance ELOas fonctionne.

Installer les librairies ELOas

Dans la configuration standard, ELO Automation Services contient quelques librairies. Toutefois, nous vous recommandons d'installer plusieurs librairies JavaScript pour permettre une fonctionalité maximale. Ces librairies sont disponibles à part et sont actualisées régulièrement.

Tout d'abord, vous devez les importer dans l'archive.



Illustr. : classeur ELOas Base dans l'arborescence

En premier, nous vous recommandons de vérifier si ces fichiers JavaScript n'ont pas déjà été enregistrés sous :

```
Administration // ELOas Base // JavaScript
```

Si ce classeur contient déjà des fichiers JavaScript, veuillez tout d'abord vous assurer qu'ils n'ont pas été adaptés pour votre environnement. Sinon, vous devez les supprimer avant d'effectuer la mise à jour.

Installez les librairies ELO à partir de ELO SupportWeb sous : http://www.forum.elo.com/script/20/eloinst.html.