Table des matières

[Fonctionner avec Github 2](#_Toc519724713)

[Installer Git et Mvn 2](#_Toc519724714)

[Récupérer la répository sur github : 2](#_Toc519724715)

[Le projet 2](#_Toc519724716)

[La structure du projet : 2](#_Toc519724717)

[Eclipse Oxygen 3.0 3](#_Toc519724718)

[Build avec Maven 3](#_Toc519724719)

[Premiers pas avec Docker 4](#_Toc519724720)

[General guidelines and recommendations 4](#_Toc519724721)

[Prérequis 4](#_Toc519724722)

[Etapes en séquence : 5](#_Toc519724723)

[Construction de l’image et du container : 5](#_Toc519724724)

[Démarrage du container : 5](#_Toc519724725)

[Afficher les logs : 5](#_Toc519724726)

[Ouvrir une console sur un container running : 5](#_Toc519724727)

[Lister les containers running : 6](#_Toc519724728)

[Lister les containers présents : 6](#_Toc519724729)

[Supprimer les containers présents : 6](#_Toc519724730)

[Lister les images 6](#_Toc519724731)

[Supprimer les images 6](#_Toc519724732)

[Nifi 6](#_Toc519724733)

[Lancement 6](#_Toc519724734)

[Processeurs « customs » tesseract-ocr 8](#_Toc519724735)

[PdfSplit 8](#_Toc519724736)

[TiffConverter 8](#_Toc519724737)

[Tess4JOcr 8](#_Toc519724738)

[Enhancements 11](#_Toc519724739)

[Optimiser le runtime de nifi 11](#_Toc519724740)

[Processeurs : 11](#_Toc519724741)

[Template : 11](#_Toc519724742)

# Fonctionner avec Github

## Installer Git et Mvn

<https://git-scm.com/downloads>  
$ git config --global user.name "John Doe"

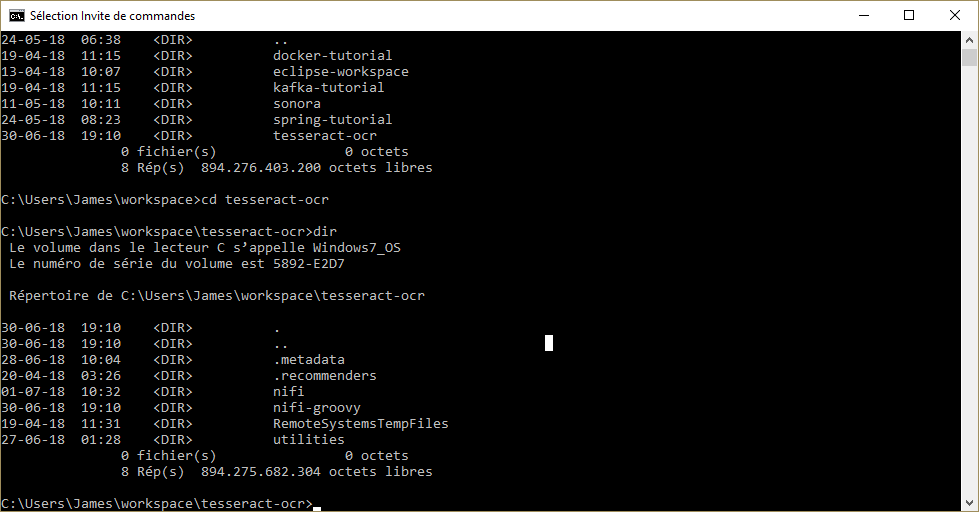
$ git config --global user.email johndoe@example.com

<https://maven.apache.org/download.cgi>

## Récupérer la répository sur github :

En ligne de commande se positionner sur le répertoire devant accueillir le workspace :

* git clone <https://github.com/hurtrobotic/nifi.git>



Vous devriez avoir le répertoire nifi.

Le projet

Le projet vise à la réalisation de processeurs «customs » fonctionnant sous Apache nifi. Une première version inclue plusieurs processeurs en rapport avec la solution « tesseract-ocr ».  
Il contient également tout le nécessaire à la définition, la création d’un container Docker. Le container englobe la solution (processeurs et groovy scripts).

La structure du projet :   
Rentrer sous le folder nifi. C’est notre folder de travail.  
Il faut exécuter les commandes docker depuis le folder contenant ce fichier.  
Les commandes mvn seront également à exécuter depuis ce fichier.

Le fichier **Docker-commands.txt** situé à la racine, reprend les principales commandes docker.  
Il est souvent difficile de connaître par cœur ces commandes avec les paramètres qui vont bien ….  
Nous reviendrons plus tard sur l’utilisation proprement dite de Docker.

Le fichier **Dockerfile** est l’élément de base de notre packaging Docker.

Le folder **nifi-ext-libs** contient essentiellement les libs pour l’exécution correcte du processeur « ExecuteScript » avec Groovy. Contient également un jar : tess4JBox-0.0.1.jar reprenant les librairies nécessaires à l’utilisation de PdfBox sous la runtime Groovy.  
Ce folder sera monté dynamiquement et mappé vers le volume du container **/opt/nifi/modules**.  
Il est ainsi possible d’en ajouter sans avoir à reconstruire le container.

Le folder **nifi-nar-bundles** : contient la structure projet nécessaire au build des processeurs Apache Nifi contenant les dépendances nécessaires à la construction d’une archive un peu spéciale au format « nar ».

Toujours dans ce folder, les sources du processeur se retrouvent dans le folder nifi-tess4J-bundle/nifi-tess4J-processors. Dans le folder nifi-tess4J-bundle/nifi-nar-processor/target le « nar file » construit et copié ensuite vers le container dans les libs de nifi lors de l’exécution du Dockerfile.

Le folder « **nifi-resources** » contient d’une part les properties et les profiles nécéssaires à la détection de langue. Ils sont copiés vers le container lors de l’exécution du Dockerfile.

Le folder **« nifi-scripts »** contient les scripts groovy pouvant être appelés via le processeur « ExcuteScript ». Ce folder sera monté dynamiquement et mappé vers le volume du container **/opt/nifi/nifi-1.6.0/importscripts**

Le folder « **nifi-template** » contient les templates xml pouvant être uploadés via l’interface web de nifi. Notre solution de « capture ocr » se retrouve ainsi dans ces templates et peut-être déployé directement sur un docker « from scratch ». Merci de vérifier toutefois les volumes mappés sur votre environnement local.   
Le fichier d’aide avec les commandes « Docker-commands.txt » devra être adapté pour les mapping locaux vers les **environnements** linux et mac (dernière commande « docker run … ».

Le folder « **sh** » contient les « shell scripts » utilisés (démarrage, clean des logs, etc).   
Ils sont copiés vers le container lors de l’exécution du Dockerfile.

# Eclipse Oxygen 3.0

J’utilise pour l’heure Eclipse Oxygen.3a Release (4.7.3a).  
J’envisage de passer prochainement sur IntelliJ pour une meilleure intégration avec le framework spring.   
Quoi qu’il en soit, la structure de projet déterminée ici est indépendante de l’éditeur.   
En effet, les différents modules sont des projets de type « Maven Module Project».  
J’utilise maven en mode de commande pour construire les modules (nar files).

# Build avec Maven

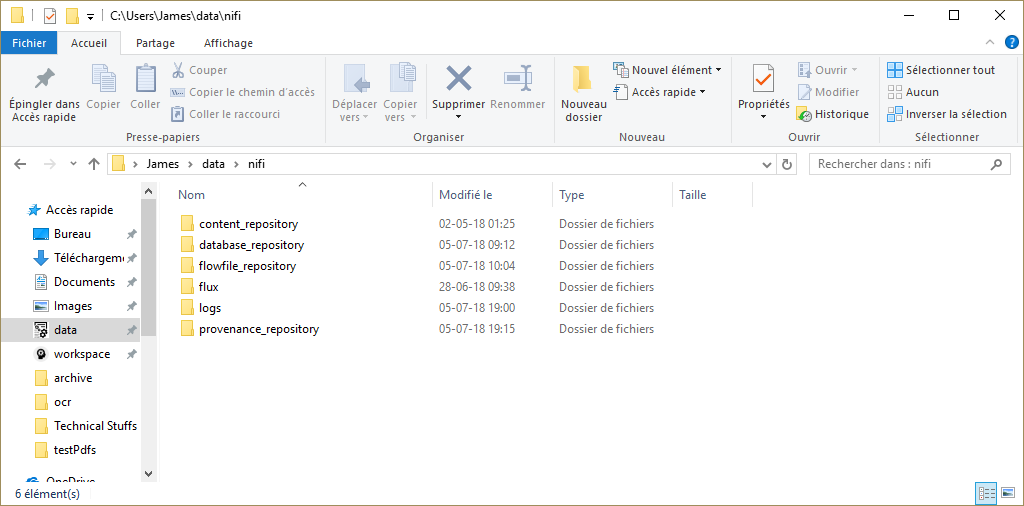
Pour effectuer le build des Processeurs « customs », il faut se placer au niveau du folder nifi et lancer la commande :

* mvn clean package

# Premiers pas avec Docker

General guidelines and recommendations  
Containers should be ephemeral  
 *The container produced by the image your Dockerfile defines should be as ephemeral as possible. By “ephemeral,” we mean that it can be stopped and destroyed and a new one built and put in place with an absolute minimum of set-up and configuration. You may want to take a look at the Processes section of the 12 Factor app methodology to get a feel for the motivations of running containers in such a stateless fashion.*

## Prérequis

1. Build maven des processeurs.
2. Définition des répertoires partagés avec le container (captures de fichier, etc) : en fonction de l’environnement de travail, reprendre le fichier /sh/build-folder-structure.bat ou build-folder-structure.sh et le placer dans le répertoire cible.  
   Sous mon environnement windows, le répertoire ce situe ici : C:\Users\James\data\nifi   
     
   Ces folders seront définis au niveau de docker comme « Volumes ».  
   Actuellement le mapping des volumes se fait lors de l’appel (nous le verrons plus tard) à la commande « docker run …. ».  
   Cela peut-être également paramétrable dans un « docker-compose » ou via l’utilisation d’une application comme KiteMatic.  
     
   D’autres volumes, ne concernant pas les données ou la persistance, sont définis directement au niveau du workspace, pour des facilités de développement :   
   - nifi-ext-libs  
   - nifi-scripts

## Etapes en séquence :

Cfr : Docker-commands.txt  
Cfr : doc\ DockerCheatSheet.pdf

### Construction de l’image et du container :

docker build -t hurtrobotic/nifi-ocr .

La commande va créer l’image initiale et instancier un container.  
Cette image une fois construite peut être sauvegarder dans une répository, sur le docker hub par exemple : docker push hurtrobotic/nifi-ocr

### Démarrage du container :

docker run --name nifi-ocr -it -p 8080:8080 -p 8181:8181   
-v //host\_mnt/c/Users/James/data/nifi/content\_repository:/opt/nifi/nifi-1.6.0/content\_repository

-v //host\_mnt/c/Users/James/data/nifi/database\_repository:/opt/nifi/nifi-1.6.0/database\_repository -v //host\_mnt/c/Users/James/data/nifi/flowfile\_repository:/opt/nifi/nifi-1.6.0/flowfile\_repository   
-v //host\_mnt/c/Users/James/workspace/tesseract-ocr/nifi/nifi-scripts:/opt/nifi/nifi-1.6.0/importscripts   
-v //host\_mnt/c/Users/James/data/nifi/logs:/opt/nifi/nifi-1.6.0/logs   
-v //host\_mnt/c/Users/James/workspace/tesseract-ocr/nifi/nifi-ext-libs:/opt/nifi/modules   
-v //host\_mnt/c/Users/James/data/nifi/provenance\_repository:/opt/nifi/nifi-1.6.0/provenance\_repository   
-v //host\_mnt/c/Users/James/data/nifi/flux:/opt/nifi/flux hurtrobotic/nifi-ocr

La commande ici renseigne :

* Le nom du container
* Le mapping de port (-p)
* Le mapping des volumes (-v)

Le container démarre avec la console en intéractif.  
Une fois le container lancé une fois, il apparait dans la liste des containers présents.

Si le container est présent, il est dès lors possible de démarrer en lançant la commande :   
docker start nifi-ocr  
et de l’arrêter avec :   
docker stop nifi-ocr  
🡺 sans avoir à remettre tous les paramètres de mapping, etc.

Afficher les logs :   
docker container logs -f nifi-ocr

Ouvrir une console sur un container running :   
docker exec -it --user root nifi-ocr bash

Lister les containers running :   
docker ps

Lister les containers présents :   
docker ps -a

Supprimer les containers présents :   
Win : FOR /f "tokens=\*" %i IN ('docker ps -a -q') DO docker rm %i  
Linux : docker rm $(docker ps -a -q)

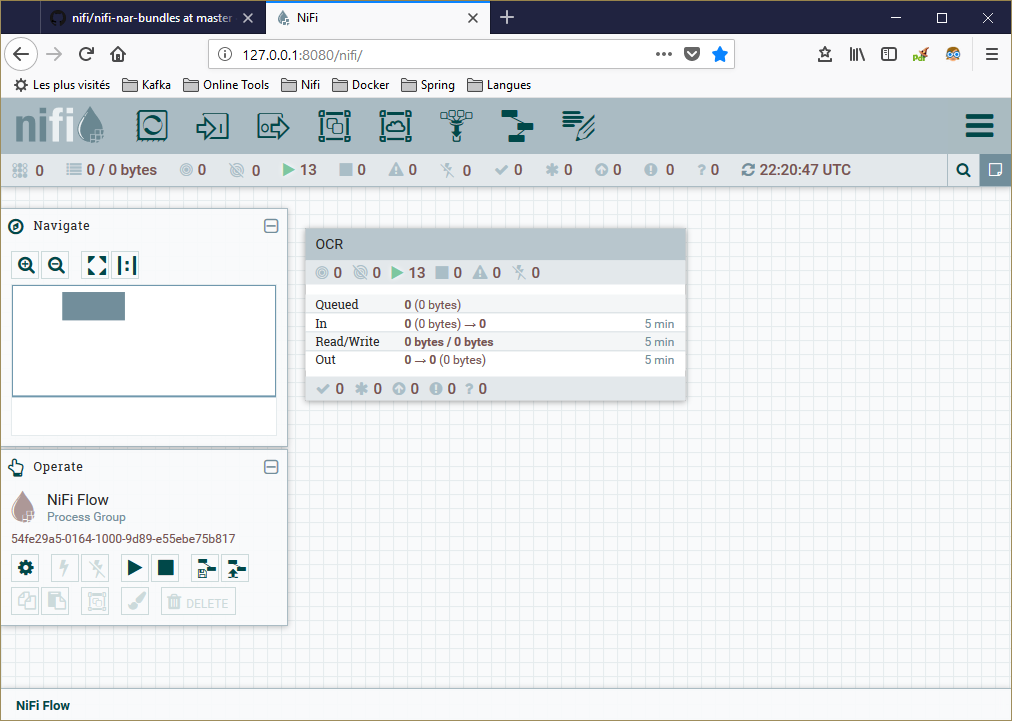
Lister les images   
docker images

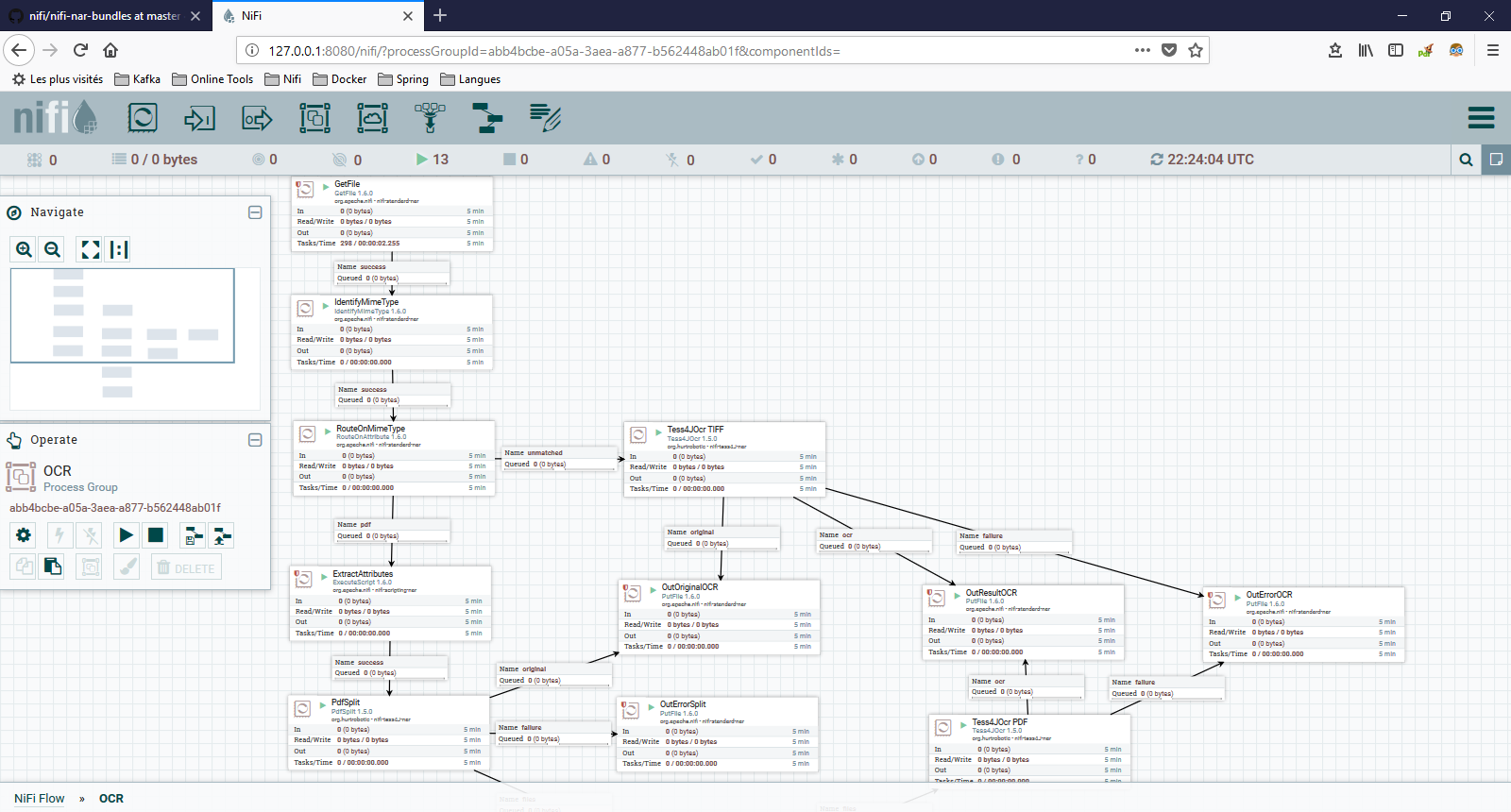
Supprimer les images  
Win : FOR /f "tokens=\*" %i IN ('docker images -q -f "dangling=true"') DO docker rmi %i  
Linux : docker rmi $(docker images -q -f dangling=true)

# Nifi

## Lancement

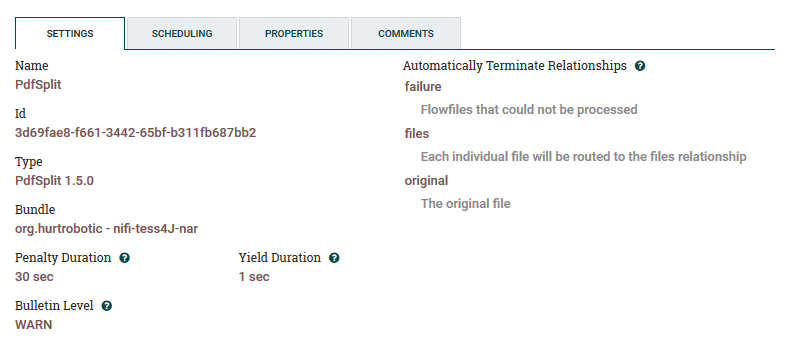
docker start nifi-ocr





## Processeurs « customs » tesseract-ocr

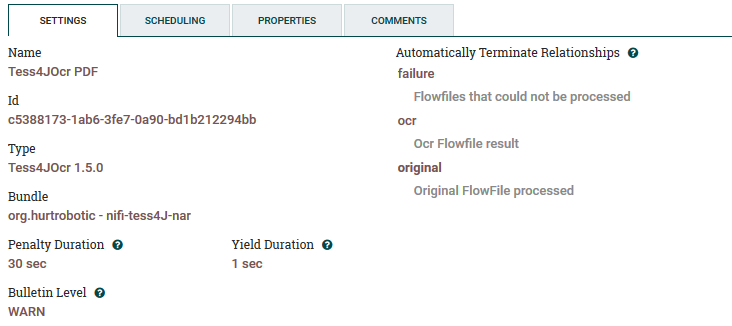
### PdfSplit

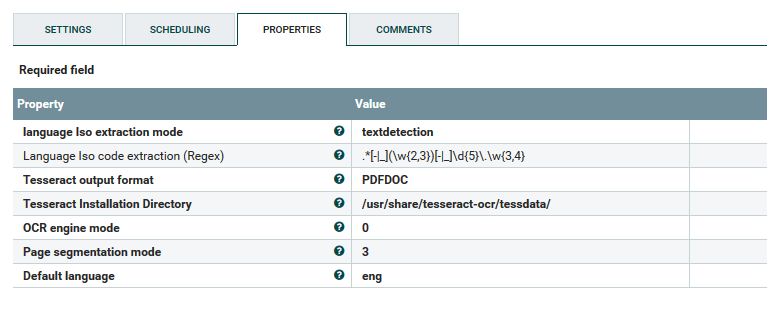


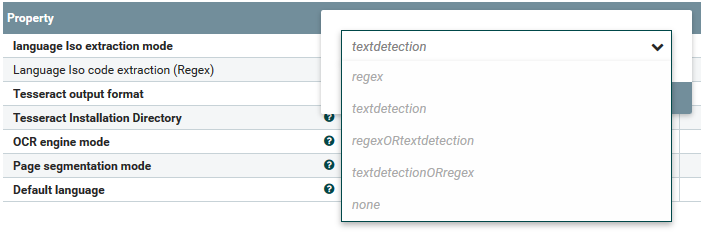
### TiffConverter

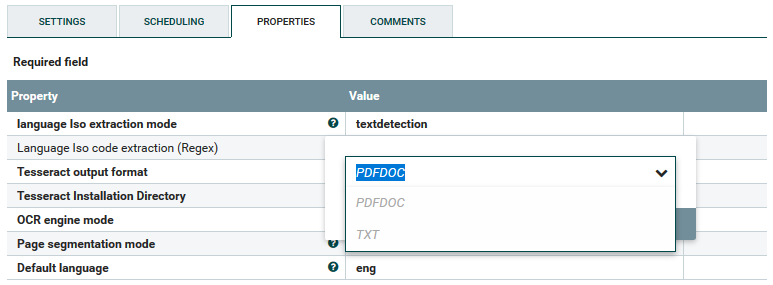


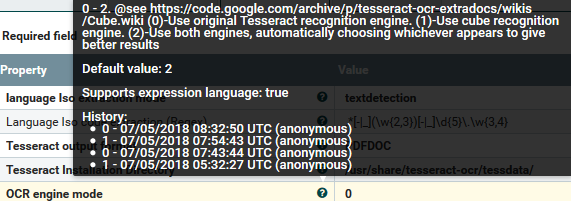
### Tess4JOcr

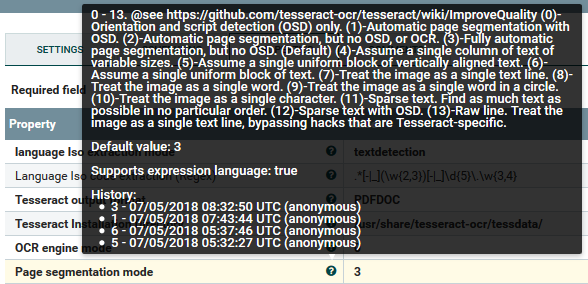


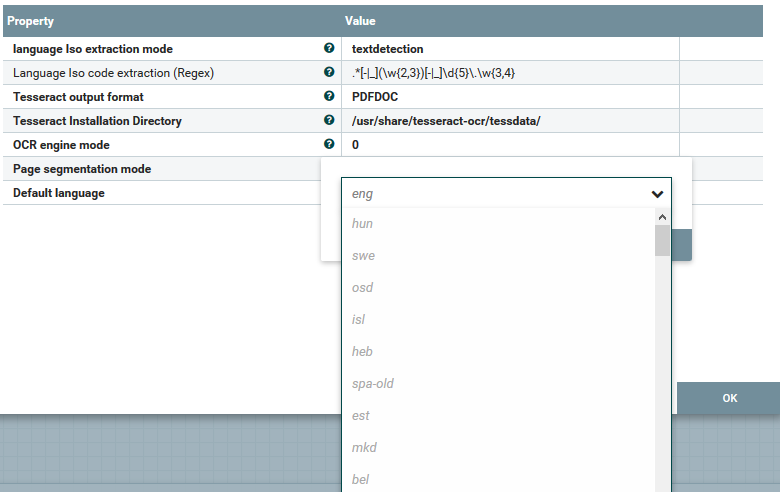












# Enhancements

## Optimiser le runtime de nifi

<https://community.hortonworks.com/articles/7882/hdfnifi-best-practices-for-setting-up-a-high-perfo.html>

## Processeurs :

* Implémenter un thread de timeout..
* Embed ocr in pdf file output (now just text output)

## Template :

* Intégration Elastic Search