Content for your Business

Guide

ES-Capture - Import Management

[CAPTURE]

TERMS AND CONDITIONS

TABLE OF CONTENTS

Conventions followed in this manual

Information
Additional Information about the current topic

Caution
Special warning on a particular point of usage which could lead to a serious malfunction.

In case of difficulty…
Technical solution to a problem.

Reference
Reference to another EverSuite guide.

See
Link to an another paragraph

INTRODUCTION

EverSuite's ES-Capture service is used when importing data to prepare it in a standard format which EverSuite can understand: the **pivot format**.

The goal is to transform a dataset in any format into a dataset in a known format, which can be injected straight into EverSuite using the **Compliance** solution. The import may be the result of a series of intermediate processes.

The ES-Capture service makes use of the ES-DTS service, the data transformation service, which uses the **CSDTS_JOB** and **CSDTS_RUN** tables to manage processing and **CSRIGHT_CAPTURE** to manage the rights on these jobs. ES-DTS is also responsible for checking setup, initializing and activating all the sub-modules needed for the ES-Capture service to work.

A complete **process** (job) is a series **of steps**. Each step has:

- an input dataset,
- a Data Transformation Unit
- an output dataset.

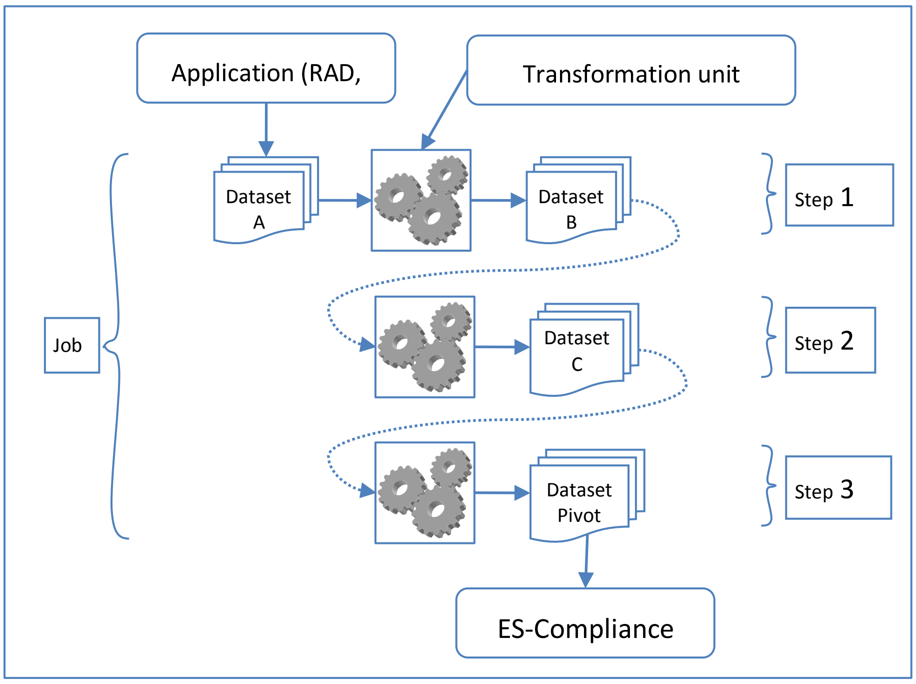


Figure 1: Example of a series of steps in a job

The process is scheduled based on:

- the order of the steps fixed when the job was created
- the job's starting priority

- the execution priority of the task associated with the job (Java thread)

The actions in each processing step are archived by the ES-BIS service (Activity statistics). Their progress can be followed in real time in a graphical interface. At the same time these actions add to the data elements' history, stored with the dataset.

A step executes a program (`DTU` = Data Transformation Unit) which reads an input `Dataset` and creates an output `Dataset`in a different format.

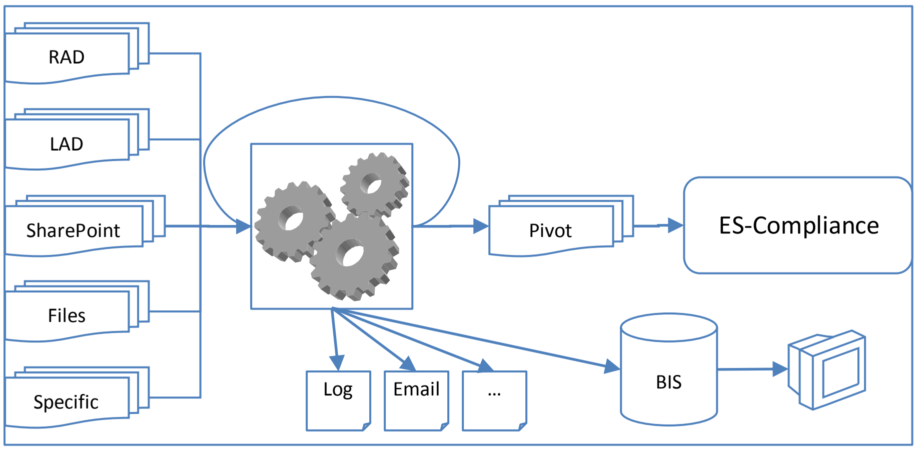


Figure 2: Processing elements

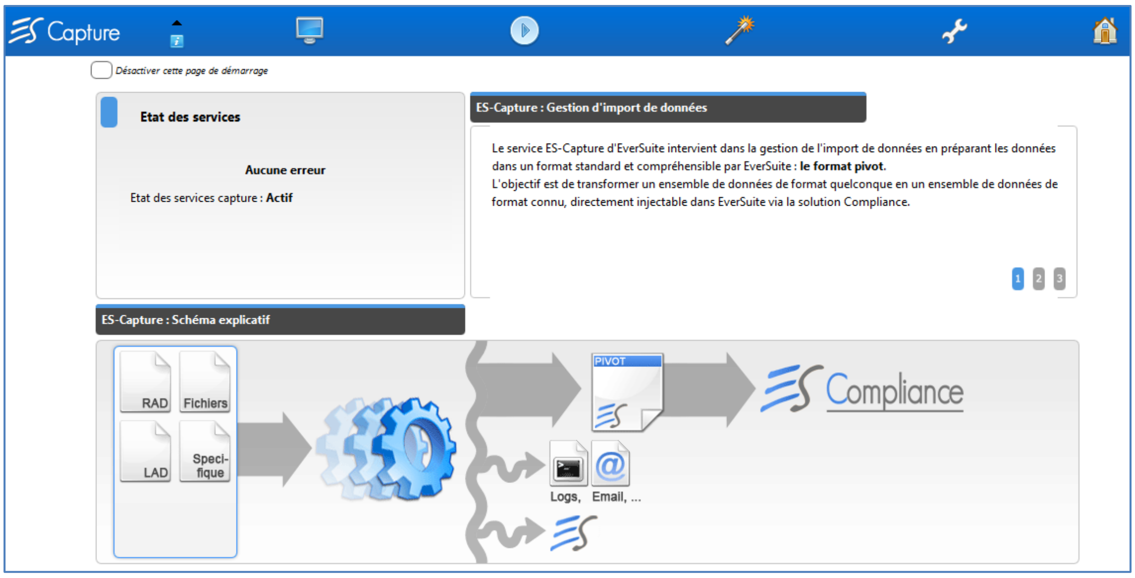
The final data format is known as the **pivot format**. This format can be inserted straight into an EverSuite application. It groups the data - the documents themselves - and the XML files which contain the metadata, the history, the format version used, the import rules, etc. EverSuite can handle two types of pivot format:

- The "pivot light" format is for a quick import of data exposed in a simple format.
- The "pivot" format supports data with a more complex structure (a more complicated hierarchical structure).

The user interface allows you to create data transformation processes, step by step, to save them, and to add them to the list of jobs to run automatically. The user interface depends on the general Java API for the ES-Capture service: `APIDispatcher`. It uses dedicated classes: `JobBuilder` to create the job, or `JobManager` to submit and execute it.

1 USER FUNCTIONS

1.1 OVERVIEW



Etat des services

Aucune erreur

Etat des services capture : **Actif**

ES-Capture : Gestion d'import de données

Le service ES-Capture d'EverSuite intervient dans la gestion de l'import de données en préparant les données dans un format standard et compréhensible par EverSuite : **le format pivot.**
L'objectif est de transformer un ensemble de données de format quelconque en un ensemble de données de format connu, directement injectable dans EverSuite via la solution Compliance.

Figure 3: ES-Capture home page

The [Job Management] menu allows you to create, update or delete, list and search processes. These processes may then be submitted for execution.

The [Executing Jobs] menu lists processes which are being executed, waiting to be executed, or which have finished.

The [Predefined Jobs] menu allows you to create or delete predefined import jobs of data stored in a simple format (pivot light). It also allows automatic deployment of the standard datasets provided by EVER TEAM.

The [Configuration] menu allows you to migrate to a new version of EverSuite, provides an editor for the pivot format and gives access to the locks console.

1.2 JOB MANAGEMENT

The ES-Capture > Job management menu is for preparing jobs in the job definition table (**CSDTS_JOB**):

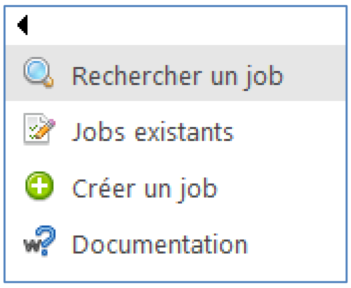


Figure 4: Job Management menu

The menus shown are limited to the operations authorized on these jobs for the current user: viewing, updating and executing the job, multiple updates and all the standard print and export functions.

1.2.1 Create a job

Creating a job involves describing input and output datasets for each step as well as the processing to be carried out, i.e. the transformation unit.

The ES-Capture > Job Management > Create a job menu uses a graphical interface, but it is also possible to create jobs directly from the pivot light data format in the ES-Capture > Predefined Jobs > Standard Import menu.

The screen gives an overview of the job. The interface guides you through the definition of each step. When creating, the graphical elements in the first step are grayed out.

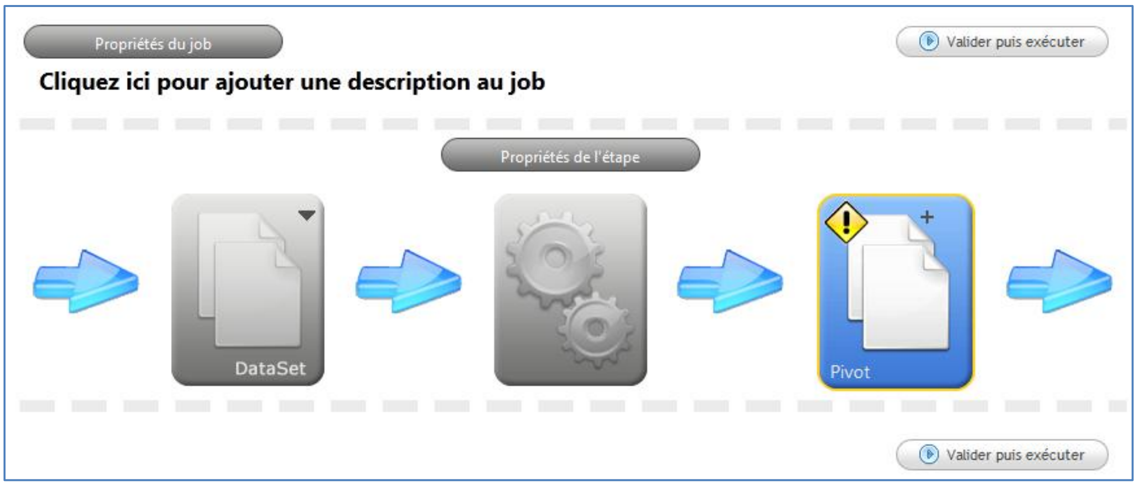


Figure 5: Job overview

There are two types of graphical elements:

DataSet element: class which recognizes a certain data format

dtu element: data transformation unit compatible with a Dataset element

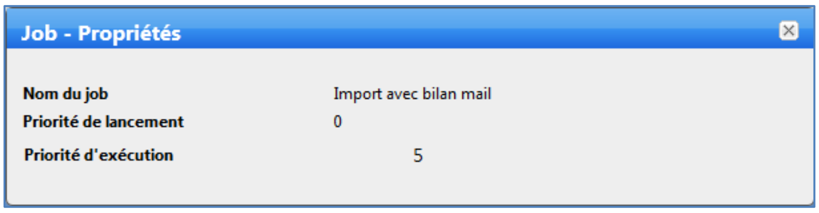
Elements are configured from left to right. Elements which have not yet been configured are grayed out.

When the format of the last Dataset element at the bottom on the right has been configured, and if it is not in pivot format, a new grayed out line is created automatically.

Declare names

Job properties

Click on [Job properties] to define its name. A job name may contain any alphanumeric character or special character. Once it has been confirmed, the job name is displayed in place of [Job properties].


Job - Propriétés

Nom du job	Import avec bilan mail
Priorité de lancement	0
Priorité d'exécution	5

Figure 6: Declaring a name / Job name

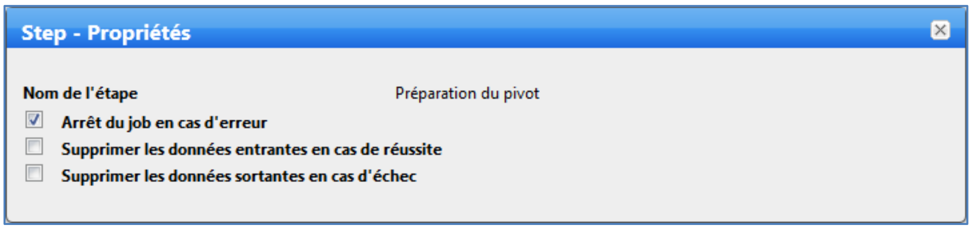
Description

A click on the Click here to add a description to the job area allows you to give the job a more detailed description.

Step properties

Click on [Step properties] to define its name. A step name may contain any alphanumeric character or special character. Once it has been confirmed, the job name is displayed in place of [Job properties]

For each step, you can opt to stop the job if there is an error and decide what happens to input and output data.


Step - Propriétés

Nom de l'étape: Préparation du pivot
- [✓] Arrêt du job en cas d'erreur
- [ ] Supprimer les données entrantes en cas de réussite
- [ ] Supprimer les données sortantes en cas d'échec

Figure 7: Declaring a name / Step name

Dclare the first step

Reference
For more information on defining and using Datasets and DTUs, use the ES-Capture > Job Management > Documentation menu: it lists the Datasets which can be used as input and output for each DTU.

For the first step, three elements are inserted automatically:

Input data (Dataset)

A click on ▼ [Change the Dataset type] allows you to choose the Dataset to process from the list of possible types in EverSuite. These data are associated with a standard **dataset** class.

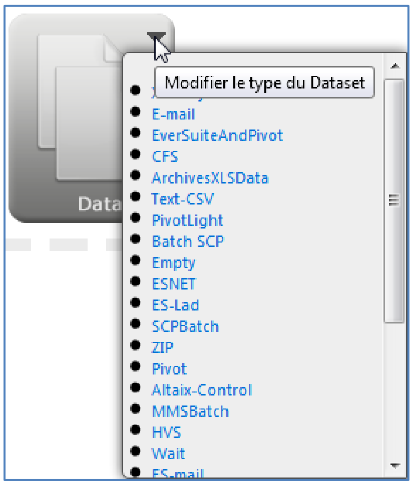
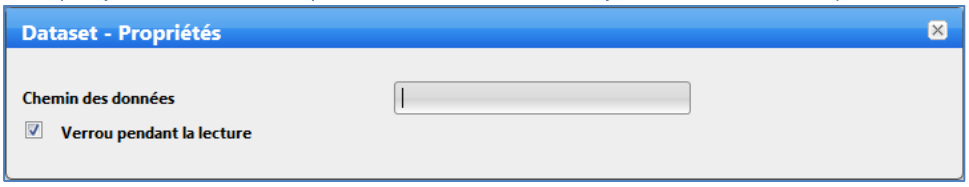


Figure 8: Declaring the first step / input data

Next click on the + icon associated with the Dataset to specify the properties (or attributes) for the selected class.

Each Dataset or DTU element has its own set of values, which can be updated in the Properties window, for example: name, encryption type for an export, decryption key for an import, directory path, etc. The values of these parameters are stored in the element attributes.

Example: for CFS data, ES-Capture needs to know which folder contains the input data.


Dataset - Propriétés

Chemin des données

☑ Verrou pendant la lecture

Figure 9: CFS input data properties

Transformation unit (DTU)

Select the transformation unit (DTU) corresponding to the input data declared by clicking on ▼ [Change the type of DTU]. The list of DTUs is entirely dependent on the type of data selected (they must be able to process the format of these data).

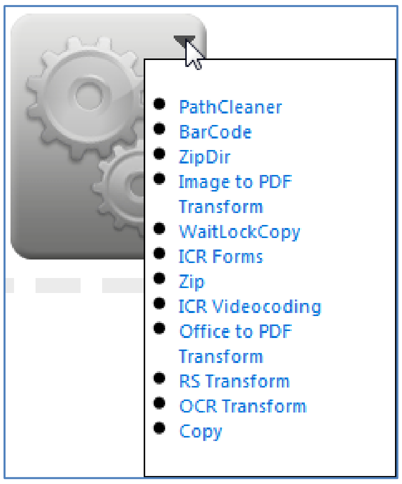
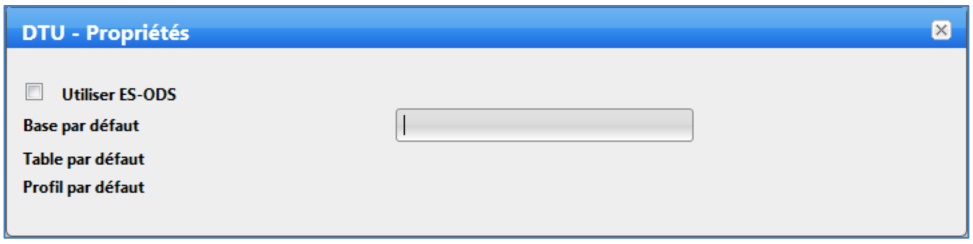


Figure 10: List of DTUs associated with the PivotLight dataset

Next click on the ✚ icon associated with the DTU to specify its properties (or attributes).

Example: for the Copytransformation unit, ES-Capture needs to know the name of the table the data will be imported into.



DTU - Propriétés

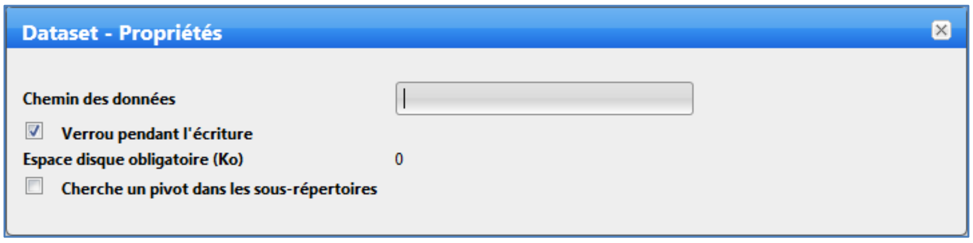
☐ Utiliser ES-ODS
Base par défaut
Table par défaut
Profil par défaut

Figure 11: Properties for the Copy transformation unit

Output data (Dataset)

The output Dataset is initialized by default for the `Pivot` format type. A click on ▼ [Change the Dataset type] automatically displays the list of data formats handled by the transformation unit selected previously.

Depending on the format of the output data, you must specify the properties (or attributes) of the corresponding class, by clicking on the ✚ icon. For example, for the pivot format, ES-Capture needs to know which folder to store transformed data in.



Dataset - Propriétés

Chemin des données
☑ Verrou pendant l'écriture
Espace disque obligatoire (Ko) 0
☐ Cherche un pivot dans les sous-répertoires

Figure 12: Properties for the pivot format

The ⚠ icon indicates that the data selected or the transformation unit are not compatible with the other choices in this step, or that the element properties have not been filled out correctly. By simply placing the mouse on the graphical element, you can view the message giving the origin of the problem.

Declare another step

To declare an additional step, click on one of the arrows [Click her to insert a step].

ES-Capture displays a line with two more graphical elements, depending on the context:

- a transformation unit and some output data
- output data and a transformation unit.

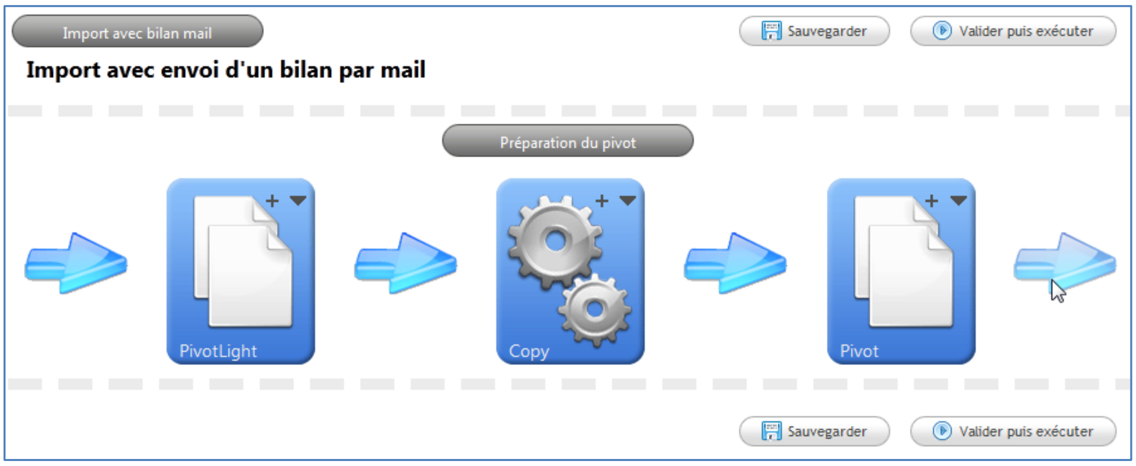


Figure 13: Adding a second step

A click on the icon [Click here to delete next step] deletes the following line.

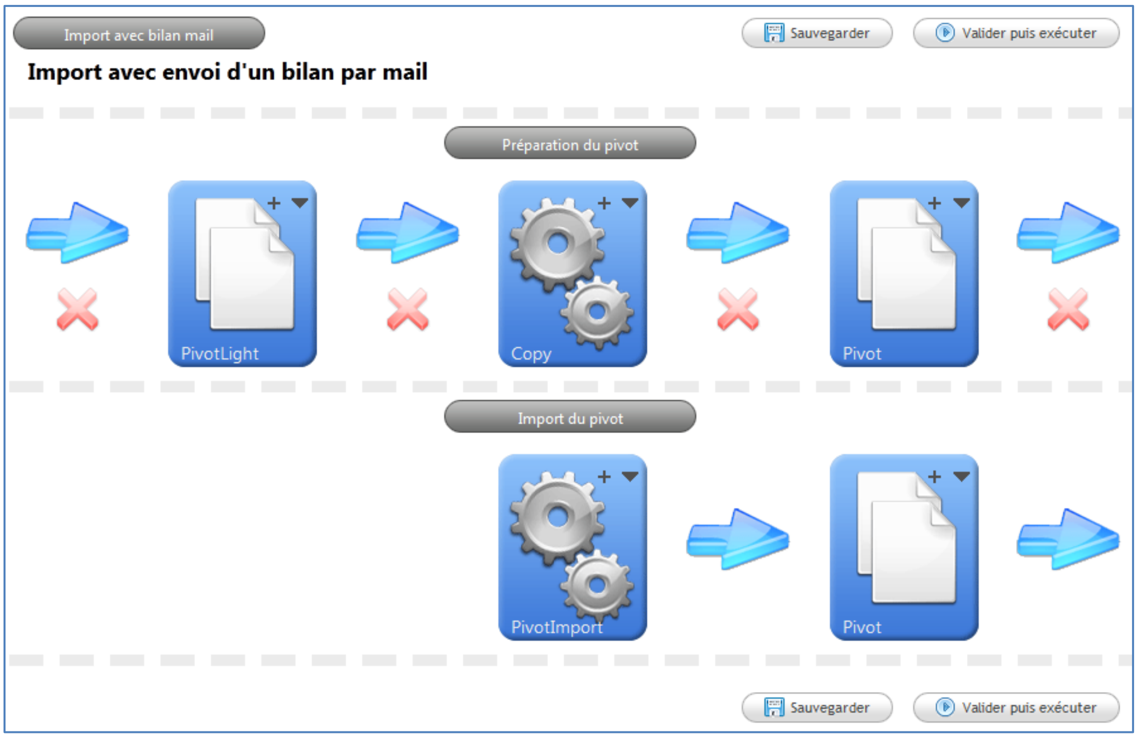


Figure 14: Example of an ES-Rad data process and data compression in a zip file

Verifications

As each step's input is the Dataset from the previous step, and a DTU does not work with all *Datasets*, it is not always possible to delete a step or change the type of a Dataset without invalidating subsequent elements. To show that they need to be changed, these elements' icon is given a yellow border. By simply placing the mouse on the graphical element, you can view the message giving the origin of the problem.

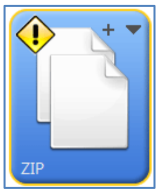


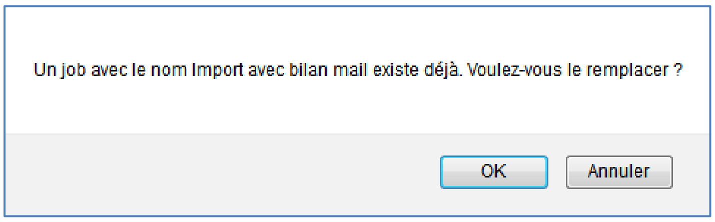
Figure 15: Incompatible graphical element

The last process (DTU) in the job must be set up to produce its output in `Pivot` format. Otherwise the job is not valid.

Save the job

The [Save] button allows you to save the job in its current state, valid or not. You can come back to it later to update it. Once it has been saved, the [Save] button is masked until the next time a change is made to the job.

The name of the process is tested for uniqueness in the database:



Un job avec le nom Import avec bilan mail existe déjà. Voulez-vous le remplacer ?

Figure 16: Message after testing the record for uniqueness

This procedure for saving makes it easy to duplicate a job, so that jobs can be set up rapidly.

Execute the job

The general validation button,[Validate then executet2/>],runs more checks on the steps before placing the job in the queue.

The job's progress is displayed automatically in the ES-Capture > Executing jobs > Jobs in progress menu. When processing is complete, the job appears in the ES-Capture > Executing jobs > Finished jobs menu.

Example: DTU / ABBYY Recognition Server

EverSuite side

Here is an example of an ES-Capture job defined in XML, which uses ABBYY Recognition Server 2.0: `Job ABBYY RS2.xml`. EverSuite is also compatible with ABBYY RS version 3.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<Job>
  <Properties>
    <Name>Unnamed Job</Name>
    <Description>Click to add a description...</Description>
    <StartPriority>0</StartPriority>
    <ThreadPriority>5</ThreadPriority>
  </Properties>
  <Steps>
    <Step>
      <Index>0</Index>
      <Properties>
        <Name>Step with no name</Name>
        <StopOnError>true</StopOnError>
        <DeleteInputOnSuccess>false</DeleteInputOnSuccess>
        <DeleteOutputOnError>false</DeleteOutputOnError>
      </Properties>
      <Input>
        <Dataset>
          <Name>Excel</Name>
          <ClassName>es.cls.dts.dataset.impl.DSExcelSimple</ClassName>
          <Properties>
            <URI>D:\data\pivot_in_excel</URI>
            <LockedRead>false</LockedRead>
          </Properties>
        </Dataset>
      </Input>
      <DTU>
        <Name>RS Transform</Name>
        <ClassName>es.cls.dts.dtu.impl.DTUTransOCRRS</ClassName>
        <Properties>
          <FileRegEx>.*(\.bmp$|\.tif$|\.jpg$|\.pcx$|\.png$|\.dcx$|\.pdf$)</FileRegEx>
          <RSWorkflowDir>\\axa_abbyy\Workflows\ES-Capture Test1</RSWorkflowDir>
          <RSResultTimeout>120</RSResultTimeout>
          <RSSleepDuration>5</RSSleepDuration>
        </Properties>
      </DTU>
      <Output>
        <Dataset>
          <Name>Pivot</Name>
          <ClassName>es.cls.dts.dataset.impl.DSPivot</ClassName>
          <Properties>
            <URI>D:\data\pivot_out_pivot</URI>
            <LockedWrite>false</LockedWrite>
          </Properties>
        </Dataset>
      </Output>
    </Step>
  </Steps>
</Job>
```

Figure 17: Job ABBYY RS2.xml

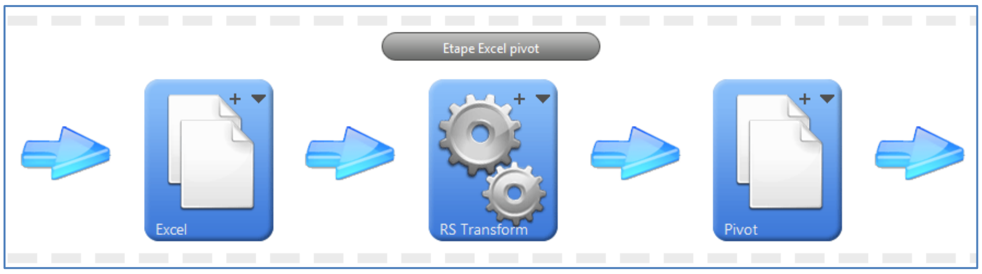
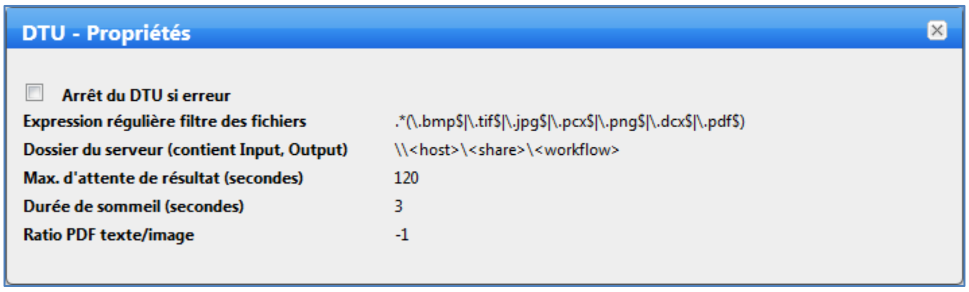


Figure 18: Job ABBYY RS2 in the ES-Capture editor



DTU - Propriétés

☐ Arrêt du DTU si erreur	
Expression régulière filtre des fichiers	.*(\.bmp$|\.tif$|\.jpg$|\.pcx$|\.png$|\.dcx$|\.pdf$)
Dossier du serveur (contient Input, Output)	\\<host>\<share>\<workflow>
Max. d'attente de résultat (secondes)	120
Durée de sommeil (secondes)	3
Ratio PDF texte/image	-1

Figure 19: Property of DTU RS Transform

`Stop DTU if error`: check this box to stop the operation in case of error.

`Regular expression file filter`: regular expression for the input Dataset files to be transformed.

`Server folder (contains Input, Output)`: ABBYY server shared folder, used by the DTU as the parent of three sub-folders: **`Input`**, **`Output`** and **`Exceptions`**. These sub-folders must be set up in ABBYY RS for**`Input`**, conversion results (**`Output`**) and errors (**`Exceptions`**).

`Maxseconds to wait for result`: by default, the same value as the default for an ABBYY workflow, 2 minutes.

`Sleep duration (seconds)`: waiting time between checks for a return (between sending a request for processing to the server and the return of one or more file(s)).

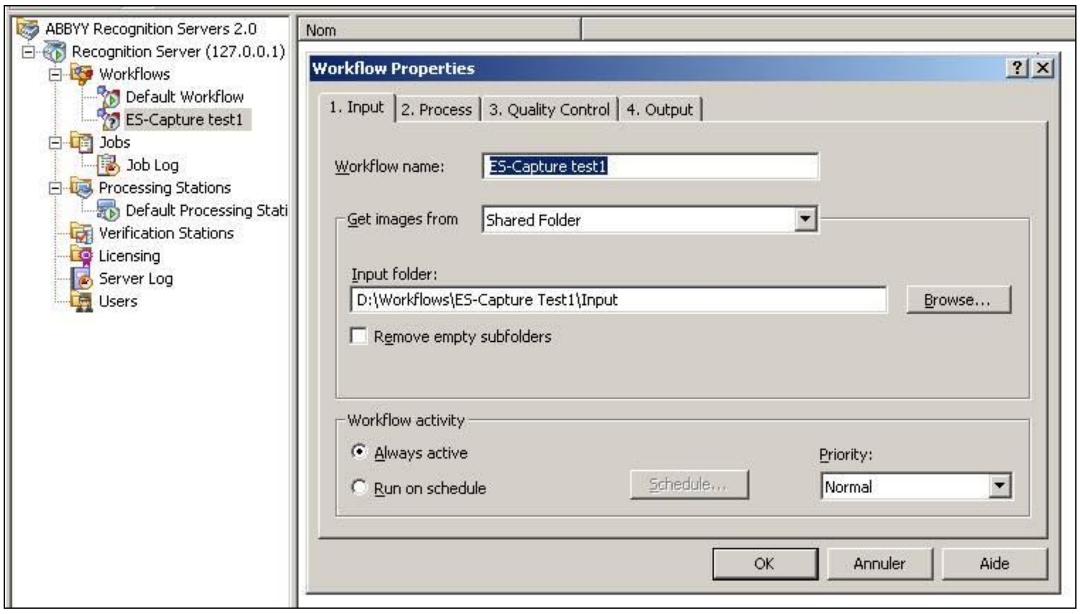
`Text/image PDF ratio`: ratio for optimizing processes and distinguishing text or image PDFs (as a number of characters).

ABBYY RS side

In ABBYY RS, a "workflow" was created; for ABBYY, this means a "set of processing parameters".

1. Open the **`Remote Administration Console`** program.
2. Right click on **`Workflows`**, then on [New Workflow] (here: *`ES-Capture Test1`*)
3. Create a shared directory on the server, for example *`Workflows`*
4. Create a sub-directory with the same name as the workflow just created (*`ES-Capture Test1`*). The ES-Capture job must be given the path to this folder (\\host_abbyy\Workflows\ES-Capture Test1).
5. Create three sub-sub-directories: **`Input`**, **`Output`** and **`Exceptions`**. The ES-Capture DTU uses these subs-sub-directories automatically.

Complete paths must be specified to the four places shown in the following screenshots:


Workflow Properties

1. Input | 2. Process | 3. Quality Control | 4. Output

Workflow name: ES-Capture test1

Get images from: Shared Folder

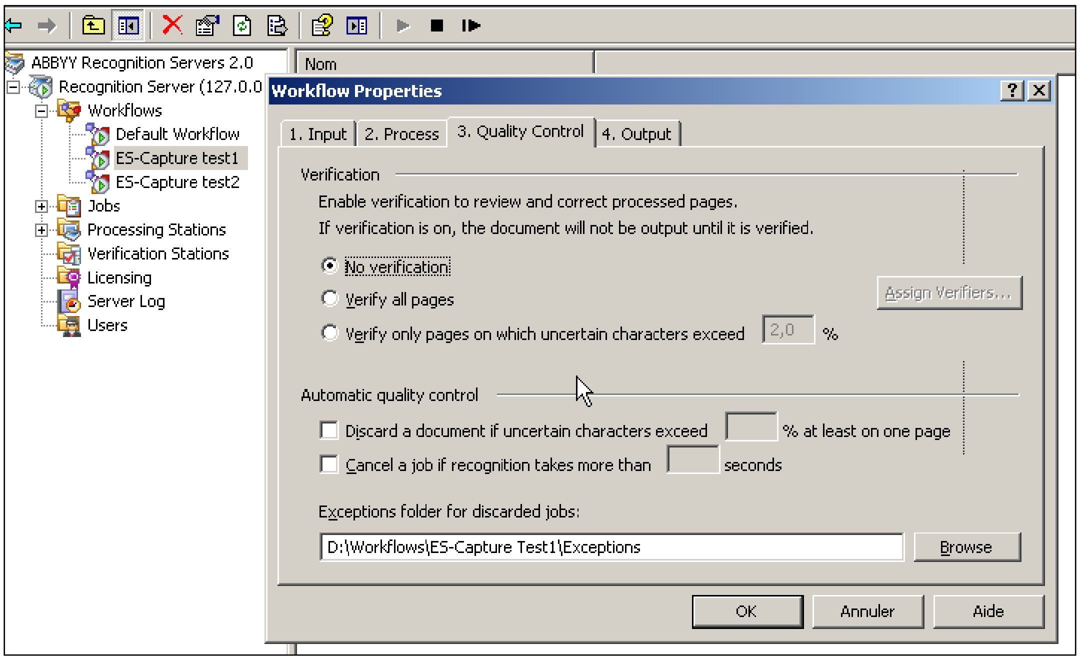
Input folder: D:\Workflows\ES-Capture Test1\Input

☐ Remove empty subfolders

Workflow activity: (●) Always active ( ) Run on schedule

Priority: Normal

Figure 20: Creating the ABBYY workflow / Input directory


Workflow Properties

1. Input | 2. Process | 3. Quality Control | 4. Output

Verification

Enable verification to review and correct processed pages.
If verification is on, the document will not be output until it is verified.

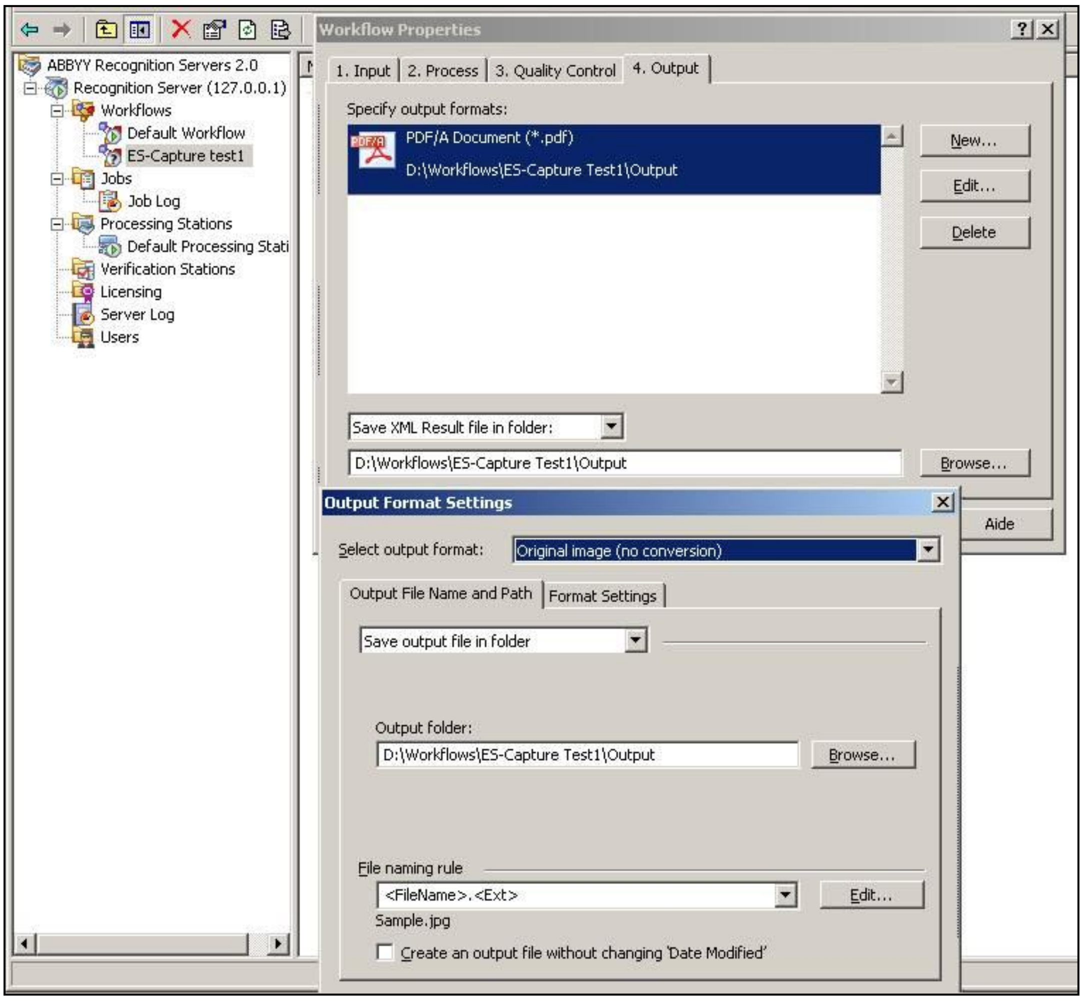
(●) No verification
( ) Verify all pages
( ) Verify only pages on which uncertain characters exceed 2,0 %

Automatic quality control

☐ Discard a document if uncertain characters exceed % at least on one page
☐ Cancel a job if recognition takes more than seconds

Exceptions folder for discarded jobs: D:\Workflows\ES-Capture Test1\Exceptions

Figure 21: Creating the ABBYY workflow / Exceptions directory



Specify output formats:
PDF/A Document (*.pdf)
D:\Workflows\ES-Capture Test1\Output

Save XML Result file in folder:
D:\Workflows\ES-Capture Test1\Output

Select output format: Original image (no conversion)

Save output file in folder

Output folder: D:\Workflows\ES-Capture Test1\Output

File naming rule
<FileName>.<Ext>
Sample.jpg

☐ Create an output file without changing 'Date Modified'

Figure 22: Creating the ABBYY workflow / Output directory

Example: converting documents

The ES-Capture service also converts documents attached to imported data.

An ES-Capture job can import documents into an application via the pivot format. The job is a series of predefined processes (DTU) which always finish by producing a file in the pivot format. Two of the possible processes involve converting attached documents to PDF or PDF/A format:

- *Image to PDF transform*
- *Office to PDF Transform*

The example is importing requests into the Case Management solution, using the **demands.xml** file (\apps\cfs\import).

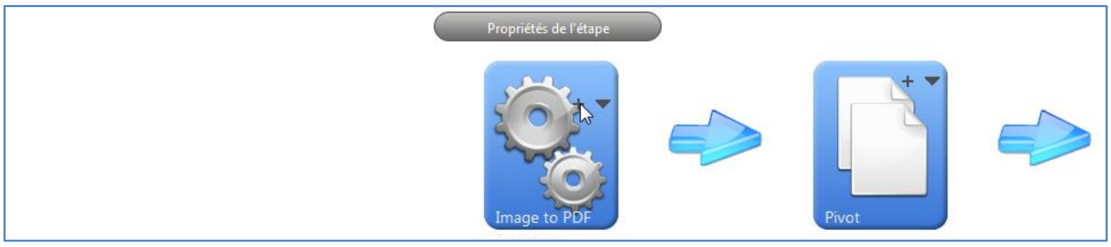
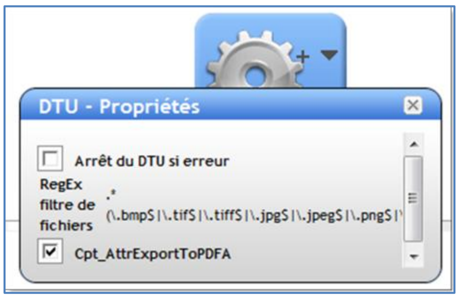


Figure 23: Adding a step with the Image to PDF DTU

Check the input filters (with the + icon).



[ ] Arrêt du DTU si erreur
RegEx filtre de fichiers .*(\.bmp$|\.tif$|\.tiff$|\.jpg$|\.jpeg$|\.png$|
[✓] Cpt_AttrExportToPDFA

Figure 24: Checking filters

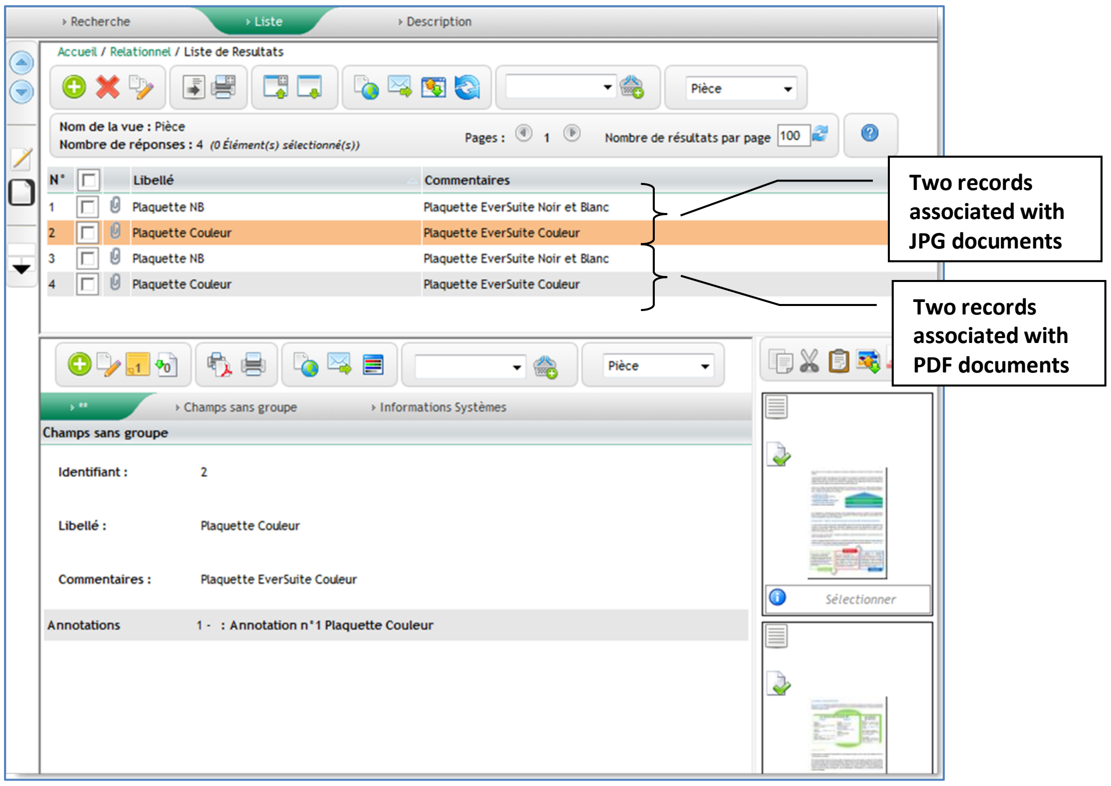
Save the change made to the job with the button [Save]. Execute the job with the button [Validate then execute].

```
<Step>
<Index>1</Index>
<Properties>
<Name>Step with no name</Name>
<StopOnError>true</StopOnError>
<DeleteInputOnSuccess>false</DeleteInputOnSuccess>
<DeleteOutputOnError>false</DeleteOutputOnError>
</Properties>
<Input />
<DTU>
<Name>Image to PDF Transform</Name>
<ClassName>es.cls.dts.dtu.impl.DTUTransPDF</ClassName>
<Properties>
<DTUStopOnError>false</DTUStopOnError>
<FileRegEx>.*(\.bmp$|\.tif$|\.tiff$|\.jpg$|\.jpeg$|\.png$|\.pdf$)</FileRegEx>
<ExportToPDFA>true</ExportToPDFA>
</Properties>
</DTU>
<Output>
<Dataset>
<Name>Pivot</Name>
<ClassName>es.cls.dts.dataset.impl.DSPivot</ClassName>
<Properties>
<URI></URI>
<LockedWrite>true</LockedWrite>
<MandatoryFreeSpace>0</MandatoryFreeSpace>
<PivotIntoSubDir>false</PivotIntoSubDir>
</Properties>
</Dataset>
</Output>
```

```
</Step>
<Step>
```
Figure 25: XML stream for the step

To see the result of importing the **demands.xml** file taken as an example, you must go to the Case Management solution or directly to the **ITEMDEMO** table (document table).

If it is the first time the job was run (before adding the conversion process), one sees that two records have been created, each associated with two image documents (jpg), black and white or color. After adding conversion to PDF, one sees that two records have been created, each associated with two PDF documents, in black and white or color.



Accueil / Relationnel / Liste de Resultats

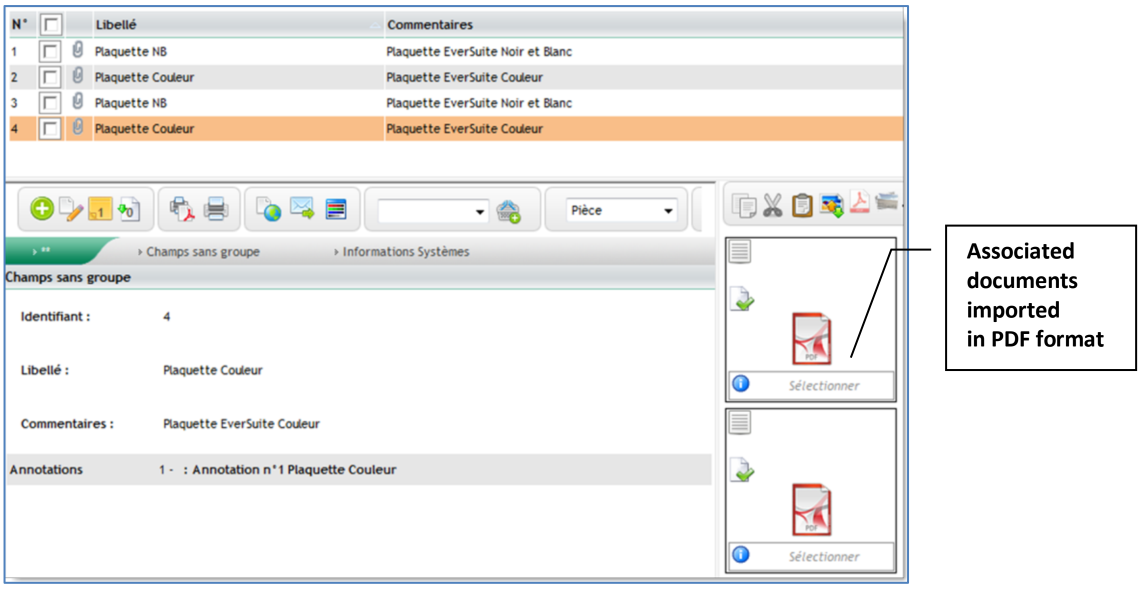
Nom de la vue : Pièce
Nombre de réponses : 4 (0 Élément(s) sélectionné(s))
Pages : 1
Nombre de résultats par page 100

N°	Libellé	Commentaires
1	Plaquette NB	Plaquette EverSuite Noir et Blanc
2	Plaquette Couleur	Plaquette EverSuite Couleur
3	Plaquette NB	Plaquette EverSuite Noir et Blanc
4	Plaquette Couleur	Plaquette EverSuite Couleur

Champs sans groupe

Identifiant :	2
Libellé :	Plaquette Couleur
Commentaires :	Plaquette EverSuite Couleur
Annotations	1 - : Annotation n°1 Plaquette Couleur

Figure 26: Creating records associated with JPG and PDF documents



N°		Libellé	Commentaires
1		Plaquette NB	Plaquette EverSuite Noir et Blanc
2		Plaquette Couleur	Plaquette EverSuite Couleur
3		Plaquette NB	Plaquette EverSuite Noir et Blanc
4		Plaquette Couleur	Plaquette EverSuite Couleur

Champs sans groupe

Identifiant : 4

Libellé : Plaquette Couleur

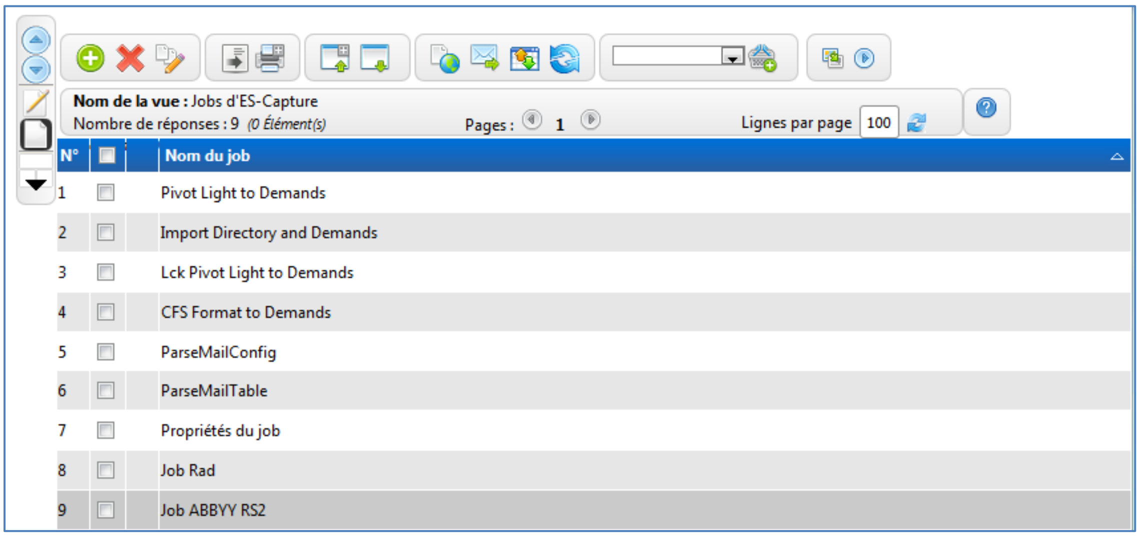
Commentaires : Plaquette EverSuite Couleur

Annotations 1 - : Annotation n°1 Plaquette Couleur

Figure 27: Import of associated documents in PDF format

1.2.2 List existing jobs

The list of processes which have been set up is displayed in the standard EverSuite interface, in the ES-Capture > Job Management > Existing jobs menu.



Nom de la vue : Jobs d'ES-Capture
Nombre de réponses : 9 *(0 Élément(s)*
Pages : 1
Lignes par page 100

N°		Nom du job
1		Pivot Light to Demands
2		Import Directory and Demands
3		Lck Pivot Light to Demands
4		CFS Format to Demands
5		ParseMailConfig
6		ParseMailTable
7		Propriétés du job
8		Job Rad
9		Job ABBYY RS2

Figure 28: Job list

The toolbar contains the standard EverSuite tools, as well as two tools specific to job management: update the job in graphical view and execute the job.

Update job in graphical view

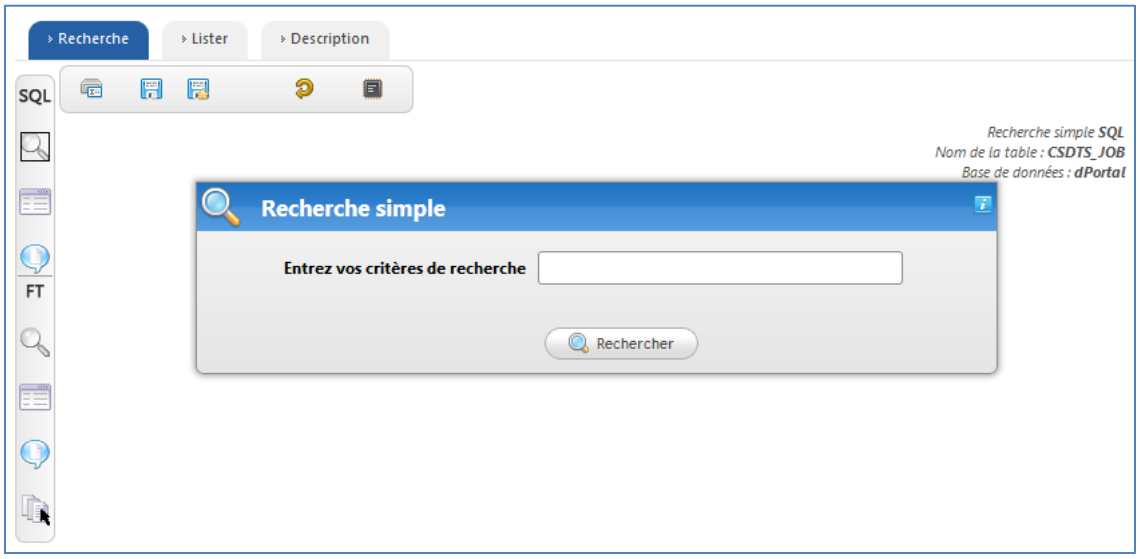
The icon [Update job in graphical view] opens the job editor with the job currently selected: the job definition may be updated.

Execute the job

The icon [Execute selected job] allows you to run the data transformation, as described in the record.

1.2.3 Search for a job

Search uses the standard EverSuite interface (search modes, query management toolbars). The various search modes on offer are the ones set up by the administrator. They are either SQL or Full Text (FT).



Recherche simple SQL
Nom de la table : CSDTS_JOB
Base de données : dPortal

Recherche simple

Entrez vos critères de recherche

Figure 29: ES-Capture search window

1.3 EXECUTING JOBS

The ES-Capture >Executing Jobs menu includes various menus for tracking jobs being executed, waiting to be executed, or which have finished.

These menus allow you to follow the progress of the processes, and to zoom in on their properties and their history. These jobs are stored in the table of executing jobs (**CSDTS_RUN**). a single job may give rise to several successive executions. There is a record in the **CSDTS_RUN** table for every execution of a job:

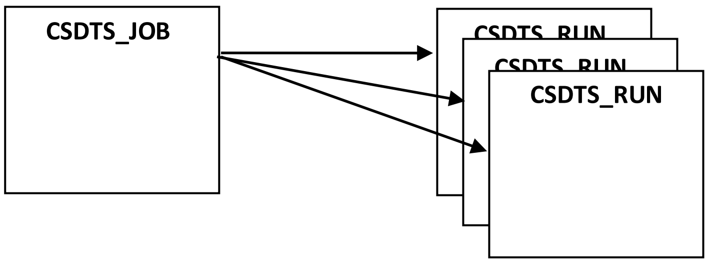


Figure 30: Jobs and executing jobs

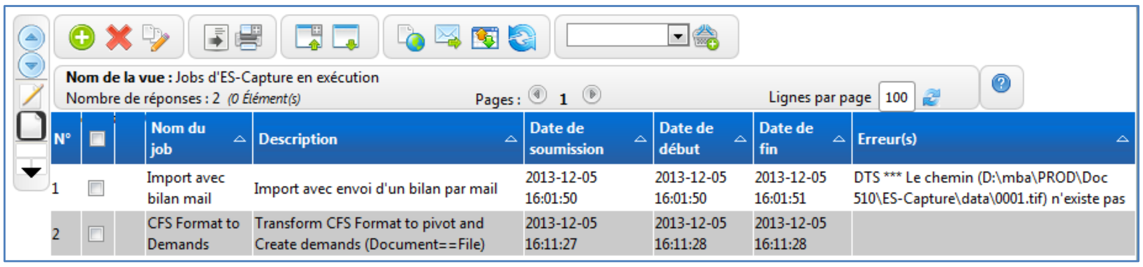
The interface uses ES-Capture Java APIs: the APIs generate their result as JDOM used to enrich the FreeMarker templates.

The interface separates visually jobs pending execution from the jobs in progress or completed. Jobs in progress may have a text giving information on the number of files processed, the current batch, etc. Completed jobs are just kept temporarily so that they can be consulted. It is possible to delete pending or completed jobs, to stop current jobs, and to change their priority.

1.3.1 Job list

The job list includes all jobs which have been executed at least once (in the **CSDTS_RUN** table). The list shows:

- date and time of the record
- date and time execution started
- date and time execution finished
- an error message if there was a problem when executing the job.



Nom de la vue : Jobs d'ES-Capture en exécution
Nombre de réponses : 2 *(0 Élément(s)*
Pages : 1
Lignes par page 100

N°	☐		Nom du job	Description	Date de soumission	Date de début	Date de fin	Erreur(s)
1	☐		Import avec bilan mail	Import avec envoi d'un bilan par mail	2013-12-05 16:01:50	2013-12-05 16:01:50	2013-12-05 16:01:51	DTS *** Le chemin (D:\mba\PROD\Doc 510\ES-Capture\data\0001.tif) n'existe pas
2	☐		CFS Format to Demands	Transform CFS Format to pivot and Create demands (Document==File)	2013-12-05 16:11:27	2013-12-05 16:11:28	2013-12-05 16:11:28	

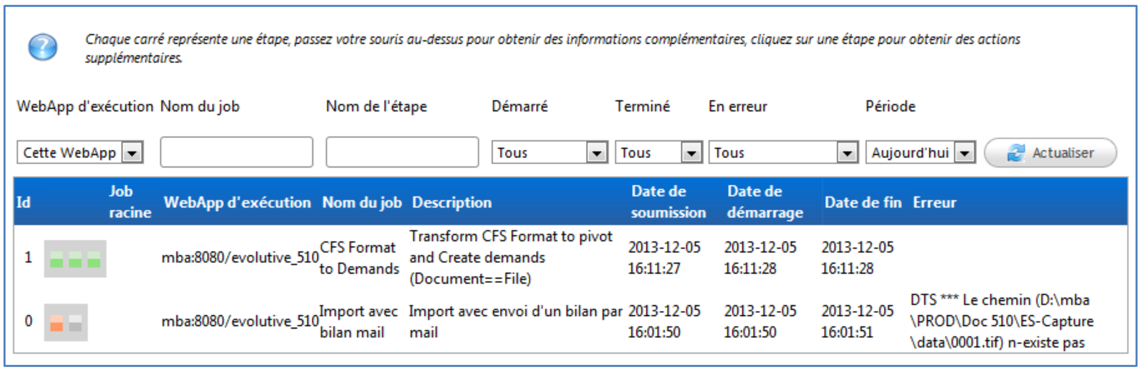
Figure 31: list of executing jobs

A job is complete if it has a start date and an end date. The job completed with no errors if the `Error(s)` field is empty.

A job can have one of the following statuses:

- *0*: submitted but not started
- *1*: started, running (no end date)
- *2*: started and completed with no errors (has a start date and an end date)
- *3*: error occurred, noted in the `Error(s)` field
- *4*: stopped by the user
- *5*: stopped cleanly when the server was shut down normally
- *6*: status set when the webapp was restarted, after an unplanned shutdown.

1.3.2 Job list (graphical interface)



Chaque carré représente une étape, passez votre souris au-dessus pour obtenir des informations complémentaires, cliquez sur une étape pour obtenir des actions supplémentaires.

WebApp d'exécution	Nom du job	Nom de l'étape	Démarré	Terminé	En erreur	Période
Cette WebApp			Tous	Tous	Tous	Aujourd'hui

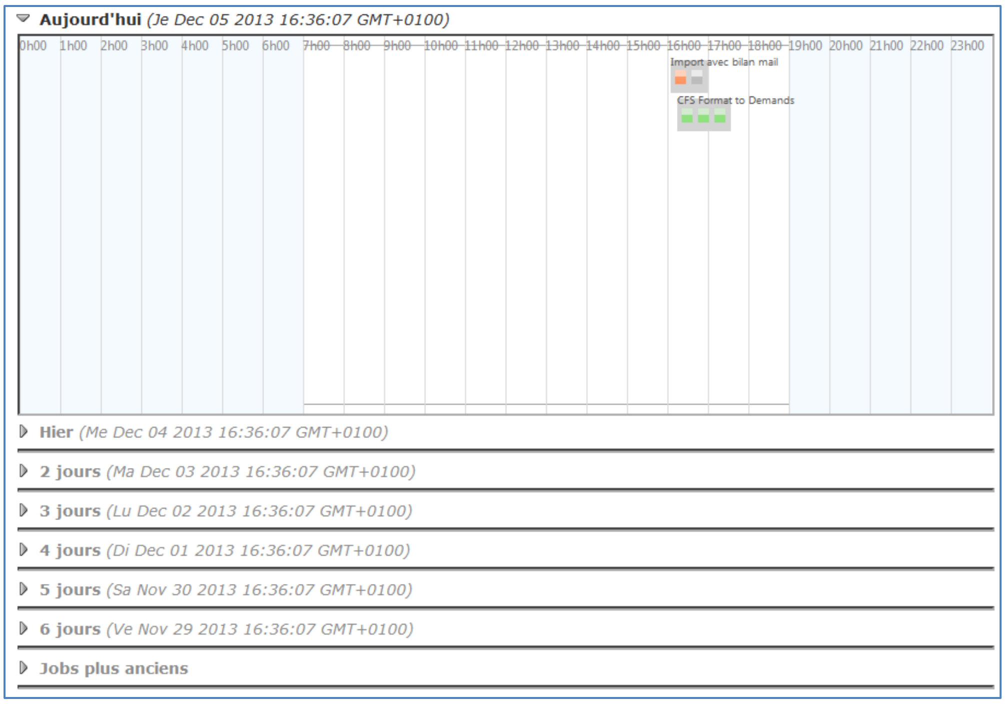
Actualiser

Id	Job racine	WebApp d'exécution	Nom du job	Description	Date de soumission	Date de démarrage	Date de fin	Erreur
1	🟩🟩🟩	mba:8080/evolutive_510	CFS Format to Demands	Transform CFS Format to pivot and Create demands (Document==File)	2013-12-05 16:11:27	2013-12-05 16:11:28	2013-12-05 16:11:28	
0	🟥▨	mba:8080/evolutive_510	Import avec bilan mail	Import avec envoi d'un bilan par mail	2013-12-05 16:01:50	2013-12-05 16:01:50	2013-12-05 16:01:51	DTS *** Le chemin (D:\mba \PROD\Doc 510\ES-Capture \data\0001.tif) n-existe pas

Figure 32: Job list (graphical interface)

1.3.3 Jobs pending, in progress, completed

The list of jobs pending, in progress or completed are shown in a timeline, broken down to hours for the day, like this:



Aujourd'hui *(Je Dec 05 2013 16:36:07 GMT+0100)*
Hier *(Me Dec 04 2013 16:36:07 GMT+0100)*
2 jours *(Ma Dec 03 2013 16:36:07 GMT+0100)*
3 jours *(Lu Dec 02 2013 16:36:07 GMT+0100)*
4 jours *(Di Dec 01 2013 16:36:07 GMT+0100)*
5 jours *(Sa Nov 30 2013 16:36:07 GMT+0100)*
6 jours *(Ve Nov 29 2013 16:36:07 GMT+0100)*
Jobs plus anciens

Figure 33: Default window for the "Executing jobs" tool

Pending jobs

Jobs are added to the queue when they are submitted for execution, either manually or by an external program. As soon as they are liable to be executed, they appear in the ES-Capture > Executing jobs > Jobs in progress menu.

The number of jobs executing is limited by the administrator's setup.

Jobs in progress

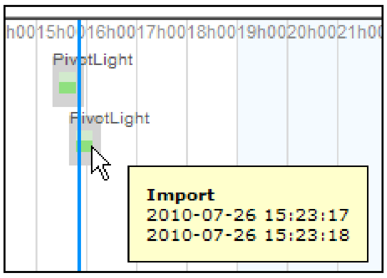
The execution of the job is represented by a colored bar starting at the time the job started in the timeline.

When the job has finished executing, the end date and time are recorded in the record and the job is displayed in the ES-Capture > Executing jobs > Completed jobs menu.

Completed jobs

Jobs executed during the day appear on the timeline. Its frame begins at the time the job *started* and finishes at the time it completed.

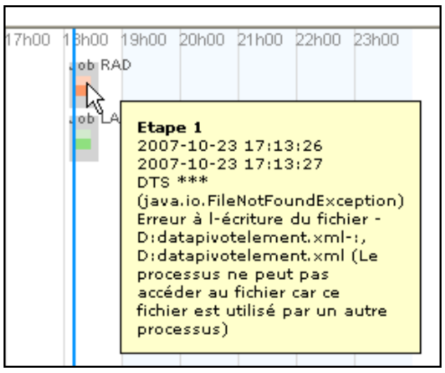
- Jobs which executed successfully appear in green. If you place the mouse on the frame, the start and end date and time are displayed.



Import
2010-07-26 15:23:17
2010-07-26 15:23:18

Figure 34: Job which executed correctly

- Jobs which did not execute successfully appear in red. If you place the mouse on the frame, the start and end date and time, and the error message, are displayed.

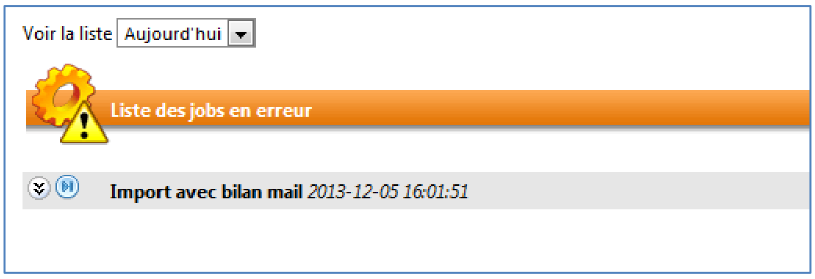


Etape 1
2007-10-23 17:13:26
2007-10-23 17:13:27
DTS ***
(java.io.FileNotFoundException)
Erreur à l-écriture du fichier -
D:datapivotelement.xml-:,
D:datapivotelement.xml (Le
processus ne peut pas
accéder au fichier car ce
fichier est utilisé par un autre
processus)

Figure 35: A job with an error

1.3.4 Jobs with errors

ES-Capture displays the jobs which could not be completed and offers to rerun them.


Voir la liste Aujourd'hui

Liste des jobs en erreur

Import avec bilan mail 2013-12-05 16:01:51

Figure 36: Jobs with errors

The icon [More details] gives access to more information on the error.

The icon [Restart job from this error] restarts the job starting at the step which had an error, after confirmation:

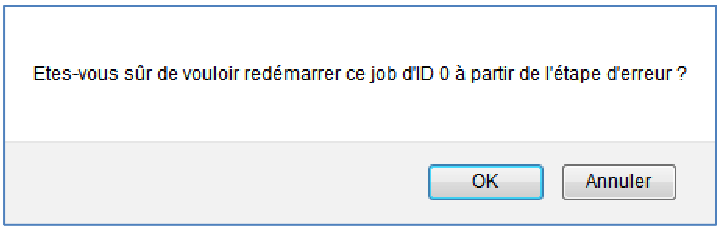

Etes-vous sûr de vouloir redémarrer ce job d'ID 0 à partir de l'étape d'erreur ?

Figure 37: Request for confirmation before rerunning jobs with errors

2 CONFIGURATION

2.1 MIGRATION

The Configuration > Migration menu allows you to create the tables required for applications which did not use the ES-Capture service in a previous version.

2.2 PIVOT EDITOR

2.2.1 Pivot format

Type

- Universal format: can store all specific formats.
- Portable format: can be stored in different forms: directory, OpenOffice zip, etc.
- Customization: is sufficiently accessible to be created by an administrator or partner who must supply his data in this format.
- Version management: always handles all data processed by its version (version of the format).
- Import/Export EverSuite : peut servir à exporter et importer une solution EverSuite.
- Multiple files: is represented by several XML files next to the data and attached document files. These files contain the elements which describe the data, their metadata, history, the import policy (import rules), etc. Having many files having an enormous file to read sequentially as a stream. The order of the format elements sets priorities for processing the data. Having many files also allows the administrator or partner to change his data easily and extend a part of the format without changing the overall format.
- Relative paths: in an XML file, file access paths are relative to the folder containing this XML file, so that a branch can be moved. However, for special requirements or to avoid copying a huge mass of data, absolute paths are possible.

Components

Transport

Some types of transport may require different packaging, for example:

- for internet: a compressed and/or encrypted archive
- for a CD-ROM: directories and files.

The type of transport and packaging may be specified or deduced from the nature of the data delivered.

Version

The overall version of the format is given in the first tag, when sequential processing is chosen. The version applies to all the files making up the format: if one of the XML files changes format, the overall version is updated.

Security

Just as the whole thing, each element may be signed and/or encrypted with different algorithms. The algorithm and the length of the key must be specified in the format.

If elements are signed, the certificates needed for verifying all the signatures must be included at the start of the format.

Example of encryption methods:

- **Asymmetric key**: encryption with the public key of the data recipient, who is the only one who can decrypt with his private key
- **Secret key**: single key, already known to the sender and the recipient, or transmitted to the recipient.

When a signature is attached to an element, this element is immediately verified:

- by rereading all the signed data,
- and using a certificate specified at the start of the format, in the basic element (element.xml).

Typing

A type of data is a set type, for example TIFF and XML files output from ES-RAD, a solution exported from EverSuite, an ES-Cold stream, etc.

The pivot format can encapsulate several data types, each one of which will be processed by the software component capable of importing it.

The entity represented by a dataset requires the choice of the software component which knows its structure and its contents, and can read then use these data, by importing them, for example.

Logical representation

Item

An **Element** is the basic object of the format. It describes a piece of data, which may be enriched with metadata (deduced or added), and which has a history.

An element is described with a series of attribute. One can obtain its type, name, identifier, data location, etc.

An element may correspond to a physical element, and point to a piece of data. The data may be a whole file, part of a global file, a directory or a list of child elements.

Tree

The **Element** object tree means you can describe batches of scanned files and data tables from an EverSuite solution.

Physical representation

Item

Several XML files are used to define an element. The minimum file is called **element.xml**. It contains the description of the element itself. For a more complex tree, each XML file may contain the sections described below. Here are the attributes which can be used in the pivot format:

- **`elt_name`**: element name
- **`type`**: element type (`root`, `file`, `record`, `deposit`, etc.)
- **`fromid`**: old ID for the element in the previous Dataset
- **`parentid`**: ID of the parent element
- **`datatype`**: data type
- **`datapath`**: absolute path to the data
- **`dataoffset`**: range - beginning of the data in the content
- **`datalength`**: range - data length
- **`rangeunit`**: unit for dataoffset and datalength (byte or page)
- **`rulename`**: name of a rule defined in the base element Policy.

These various sections in an XML file may be placed in separate files with the name of the tag (but in lowercase):

- **`attributes.xml`**: list of attributes for the type, data, name, id, etc.
- **`metadata.xml`**: author, version, etc.
- **`history.xml`**: processes undergone
- **`signature.xml`**: signature of the element and its data
- **`children.xml`**: pointers to child element files.

For the base element:

- **`certificates.xml`** (base element): certificates to use for validating signatures
- **`policy.xml`**: processing rules, for example, for importing into EverSuite with the **Compliance** solution, what to do in case of success or failure.

Maximum structure for an element.xml file (or an elements.xml file)

The tags specific to this base file are id and version.

```
<element>
  <id>1193756481976</id>
  <!-- ------------------- -->
  <version>
              <major>…</major>
         <minor>…</minor>
               <patch>…</patch>
  </version>
  <!-- ------------------- -->
  <attributes>
    <attribute>
    <attr_name>...</attr_name>
    <values>
      <value>...</value>
     <value>...</value>
    </values>
  </attribute>
   ...
  </attributes>
  <!—for a basic table field ---->
  <metadata>
   <data>
    <meta_name>...</meta_name>
    <values>
     <value>...</value>
     <value>...</value>
    </values>
  </data>
   ...
  </metadata>

  <!—for a field linked to a table ---->
  <metadata>
   <data>
    <meta_name>...</meta_name>
    <references>
     <reference>...</reference>
     <reference>...</reference>
    </references>
  </data>
   ...
  </metadata>
```

For a record import, a <data> tag contains:

- either one or more **`<value>`** tag(s) for the column values,

- or one or more **`<reference>`** tag(s) to give the ID of another *`record`* element already known.

```
<!-- ------------------- -->
<history>
  <events>
     <event>
        <date>2007-05-29 15:02:02</date>
        <attributes>
            <attribute>
                <hevt_name>...</hevt_name>
                <value>...</value>
         </attribute>
   ...
       </attributes>
     </event>
   </events>
</history>
<!-- ------------------- -->
<policy>
  <rule>
     <rule>
         <rulename>...</rulename>
         <classname>...<classname>
         <classparams>
           ... parameters specific to the classname ...
          </classparams>
       </rule>
      ...
     </rules>
</policy>
<!-- ------------------- -->
<signatures>
   <signature>
     <date>2007-05-29 15:02:03</date>
     <signer>...</signer>
     <algorithm>...</algorithm>
     <data>...</data>
     <list>
        <path>...</path>
        <path>...</path>
      </list>
   </signature>
 ...
</signatures>
<!-- ------------------- -->
<certificates>
   <certificate>
     <alias>...</alias>
```

Tag specific to the base file: element.xml (root)

Tag specific to the base file: element.xml (root)

```
        <values>
            <certpath>...</certpath>
    </values>
    ...
      </certificate>
   ...
  </certificates>
  <!-- ------------------- -->
  <children>
      <child>
        <id>4489754</id>
        <path>...</path>
      </child>
   ...
  </children>
</element>
```

Tree

Data organization may be represented by a directory tree. The example of a tree shown below is the EverSuite default tree. It is for the pivot format adapted for an import with the **Compliance** solution. But any user may create his own pivot format.

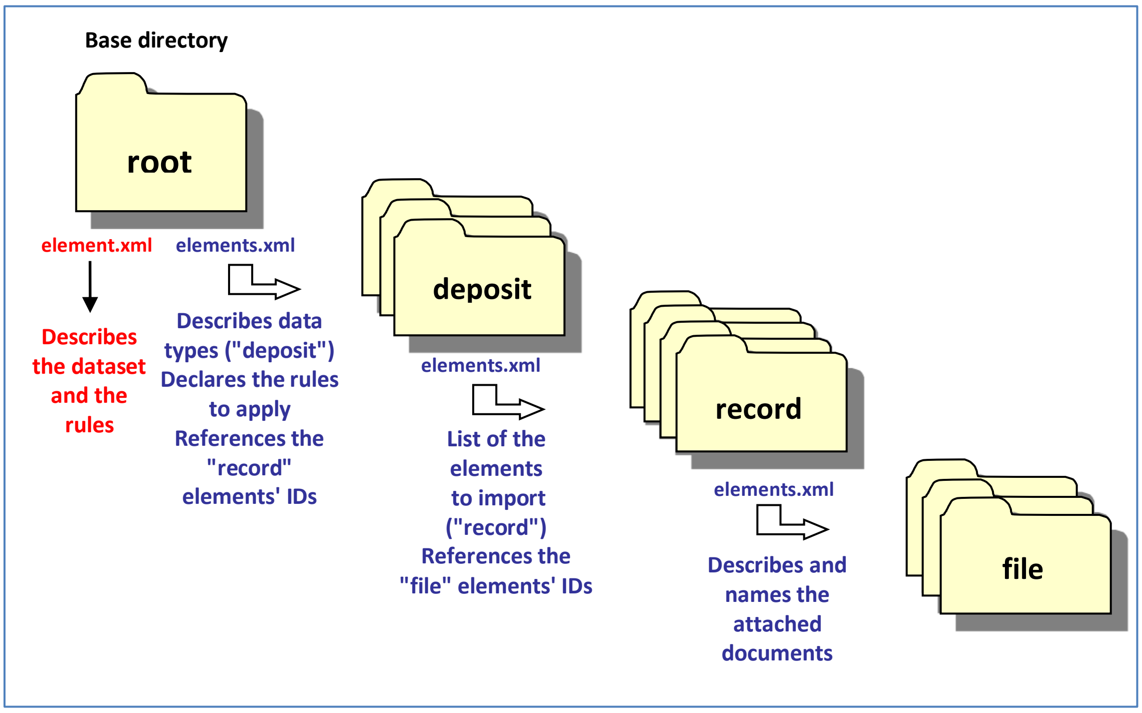

Base directory
root
element.xml
Describes the dataset and the rules
elements.xml
Describes data types ("deposit") Declares the rules to apply References the "record" elements' IDs
deposit
elements.xml
List of the elements to import ("record") References the "file" elements' IDs
record
elements.xml
Describes and names the attached documents
file

Figure 38: pivot format file tree

Root

At the **root**, the folder contains:

- the **element.xml** file which describes the dataset and the rules used, and references the corresponding data folders.

- the **elements.xml** file which describes the **deposit** level, i.e. the different datatypes (imported table), names the rules to apply to each type and references the IDs of the corresponding **record** elements.
- the deposit folders for each datatype.

Deposit folder

Each deposit folder contains:

- the **elements.xml** file which describes the **record** level, i.e. metadata for each record, and references the IDs of the corresponding **file** elements.
- metadata folders.

Data folder

Each data folder contains:

- the **elements.xml** file which describes the **file** level, i.e. the documents themselves and their names.
- attached files.

Example

The dataset example chosen for this demonstration correspond to the contents of two EverSuite system tables: CSUSER (users) and CSROLE (roles); and the table of links between these two tables, CSUSERROLES.

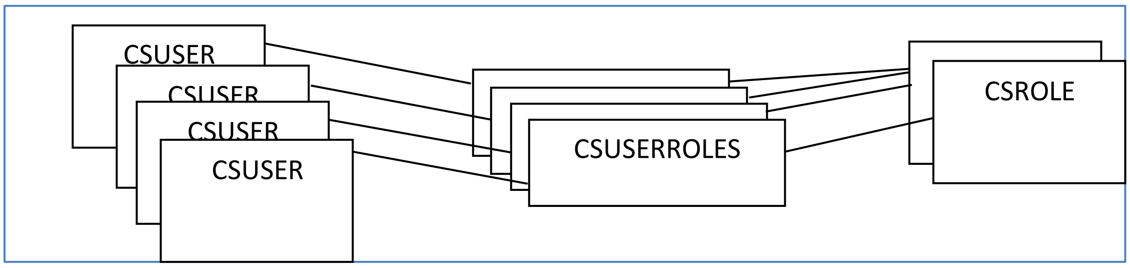


Figure 39: Dataset for the CSUSER and CSROLE tables

Base directory

The base directory contains the **root element** of the tree, along with the data needed to describe global content of encapsulated data. The root element is represented by the **element.xml** file, the only file with this name in the whole pivot format.

This file declares the format version, attributes, policy, etc. which apply to the data.

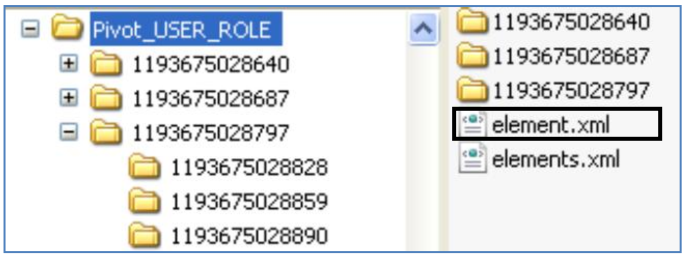


Figure 40: element.xml file in the base directory

Tag examples

The examples of tags described below may be present in the **element.xml** file or **elements.xml** files, depending on the case.

ID tag

Any element has a unique identifier, **ID**, which may:

- be generated automatically by ES-Capture (using today's date for example)
- be a value (character string)
- be a URI followed by the key.

```
 <elements>
   <element>
      <id>1193756484054</id>
         ..........
   </element>
</elements>
```

Version tag

The version of the format can be found only in the base element file, **element.xml**. The version of the pivot format is currently *1.00*; it determines if data can be reread.

```
<element>
  <id>1193675028593</id>
  <Version>
    <Major>1</Major>
    <Minor>0</Minor>
    <Patch>0</Patch>
  </Version>
```

Attribute tag

Attributes allow you to define the element type, give the ID of the data element it comes from (for example the ES-Capture input Dataset), etc. and the name of any rule to apply.
An attribute always contains two parameters: **attr_name** and **values**.

The **<attr_name>** attribute in the pivot format can take values:

- **elt_name**: element name
- **type**: element type (*root*, *file*, *record*, *deposit*, etc.)
- **fromid**: old ID for the element in the previous Dataset
- **parentid**: ID of the parent element

- **`datatype`**: data type
- **`datapath`**: absolute path to the data
- **`dataoffset`**: range - beginning of the data in the content
- **`datalength`**: range - data length
- **`rangeunit`**: unit for dataoffset and datalength (byte or page)
- **`rulename`**: name of a rule defined in the base element Policy.

Example for a deposit element

```
<attributes>
  <attribute>
   <attr_name>type</attr_name>
   <values>
    <value>deposit</value>
   </values>
  </attribute>
  <attribute>
   <attr_name>elt_name</attr_name>
   <values>
    <value>ESDMS_DEMO</value>
   </values>
 </attribute>
 <attribute>
   <attr_name>parentid</attr_name>
   <values>
    <value>1193756481976</value>
   </values>
  </attribute>
  <attribute>
   <attr_name>fromid</attr_name>
   <values>
    <value>records</value>
   </values>
  </attribute>
  <attribute>
   <attr_name>rulename</attr_name>
   <values>
    <value>ESDMS_DEMO_rule</value>
   </values>
  </attribute>
</attributes>
```

Rule to apply

Example for a file element

To obtain the path of a file associated with a *`file`* element, you start with the value of the **`datapath`** attribute.

- If it is empty or missing, ES-Capture takes the relative path of the current element, constructs the relative path of the sub-directories to arrive at this element, and adds its ID.

- If the value starts with "./" (period slash), ES-Capture takes the relative path given and removes the "period" then adds the base directory to the relative path of the current folder constructed as above.
- If the value is absolute, it is read directly.

```
<attribute>
        <attr_name>type</attr_name>
         <values>
           <value>file</value>
       </values>
</attribute>
<attribute>
   <attr_name>elt_name</attr_name>
   <values>
       <value>JUnitTest</value>
     </values>
</attribute>
<attribute>
```

File path

```
   <attr_name>datapath</attr_name>
    <values>
        <value>D:\src\es\modules\es_dev\WEB-INF\web.xml</value>
      </values>
</attribute>
<attribute>
   <attr_name>datatype</attr_name>
   <values>
       <value>xml</value>
      </values>
</attribute>
```

certificates tag

Certificates are stored either in a subdirectory or exported directly in the XML file. They are listed in the base element which details all the certificates for validating all the signatures in the format.

Each signer's alias must therefore be unique. It is however possible to store several certificates per signer. Each <certificates> tag is either in the base element file, or in a `certificates.xml`file.

The `<certpath>` attribute contains the certification path, that is, the signer's certificate followed by the certificates which were used to generate it and enable it to be validated.

```
<certificates>
 <certificate>
   <alias>webapp</alias>
  <values>
<certPath>MIIFOjCCApwwgglFoAMCAQICCCH1UmoQ3gzfMA0GCSqGSIb3DQEBBQUAMF0xFjAUBgNVB
AMTDVRydXN0ZWRTaWduZXIxEDAOBgNVBAYTB0NvdW50cnkxFTATBgNVBAoTDE9yZ2FuaXphdGlvbjE
aMBgG
A1UECxMRT3JnYW5pemF0aW9uIFVuaXQwIBcNMDUxMTE4MTUwNDUzWhgPMzAwNTAzMjExNTE0N
TNa
MF0xFjAUBgNVBAMTDVRydXN0ZWRTaWduZXIxEDAOBgNVBAYTB0NvdW50cnkxFTATBgNVBAoTDE9y
```

```
Z2FuaXphdGlvbjEaMBgGA1UECxMRT3JnYW5pemF0aW9uIFVuaXQwgZ8wDQYJKoZIhvcNAQEBBQAD
gY0AMIGJAoGBAMYczHxGSmZn1hdiF5KYSqVBri+W7EaTUI2N8L6bb+MfaCLGPiIr1f617fotLL3q
RmPOOOfLV3yJwETuJ9leGvqAbKyBYsuf9JN+GtHbEHzGAPI0PhSZaETKaeL5s5O8HKBTvNVooZlw
1kDsv53AnMRSP0VHLQ8aDYynGGEBvZhZAgMBAAGjYzBhMB0GA1UdDgQWBBSPFXGWBH3Grdkngmu
E
Se9eXPDkEDAfBgNVHSMEGDAWgBSPFXGWBH3GrdkngmuESe9eXPDkEDAMBgNVHRMEBTADAQH/M
BEG
CWCGSAGG+EIBAQQEAwIA9zANBgkqhkiG9w0BAQUFAAOBgQAHB8RwvUxnF1KCHSaw0DAzt8rofEYx
tu4SgSgToFpFKCTCBPT0E+NsHnVx8XZQS38OiDhbk1b0sCQFenz2BLwiui+6vYnTRwqf5xJx9cNn
5Vq5LS1yQU+PmMHCN5ob35kkQYHr9TUgf1xOsYYXr3nj9h1lW5PPuxBi/zkgu5/tODCCApYwggH/
oAMCAQICCAjEETCzynwiMA0GCSqGSIb3DQEBBQUAMF0xFjAUBgNVBAMTDVRydXN0ZWRTaWduZXIx
EDAOBgNVBAYTB0NvdW50cnkxFTATBgNVBAoTDE9yZ2FuaXphdGlvbjEaMBgGA1UECxMRT3JnYW5p
emF0aW9uIFVuaXQwIBcNMDUxMTE4MTUwNDU0WhgPMzAwNTAzMjExNTE0NTRaMFoxJjAkBgNVBA
sT HUV2ZXJTdWl0ZSBQcml2YXRlIENlcnRpZmljYXRlMRIwEAYDVQQKEwlFdmVyLVRlYW0xCzAJBgNV
BAYTAkZSMQ8wDQYDVQQDEwZ3ZWJhcHAwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOXyozr
z sJY6MwfsP53s7KPzXMq5cIuAw9b4pPgad2Q+bkFuIk3Yi799IzgKLsRwec034wJKPiXsmz5tzd9Q
rF69/4yPozm6lzv/Sw7vk68xRCsUwU7nhXD2AE/317FPOVvFSkLCSSQv2QNsGuS2OXbc9BCm2oXj
RuQxxheEXzpZAgMBAAGjYDBeMB0GA1UdDgQWBBQtp2vQd39H9J+McGKc3yCkHSCAlzAfBgNVHSME
GDAWgBSPFXGWBH3GrdkngmuESe9eXPDkEDAJBgNVHRMEAjAAMBEGCWCGSAGG+EIBAQQEAwIE8D
AN BgkqhkiG9w0BAQUFAAOBgQA84eh3e3a0oOQtElif5+pbYbxv3HHgInwm7jim58V4OaTgllwG6dqV
EWIzId5IxEz4rhe7jFbHBWaMo87u+jR8ZAHq5mrOKkgd4Wkfq/f2RfxhJds274MCkYfjRgKlo3Cx
MyMTQdI/9cCTaoQU0RDX7ox/h81LJGUFLy6xnhbssg==
  </certpath>
   </values>
 </certificate>
</certificates>
```

Information
The <certpath> tag contains a series of X509 certificates which must be able to pass the validity test in the "PKIX" algorithm. Their order must therefore be preserved and they are encoded in "PkiPath" format which is an ASN.1 sequence encoded using DER. Then the resulting binary is encoded in Base64 and placed in the tag.

Example of generation in Java:

```
Certificate[] result_certs = _KeyStore.getCertificateChain( alias );
List certlist = Arrays.asList( result_certs );
CertificateFactory certfactory = CertificateFactory.getInstance( "X.509" );
CertPath certpath = certfactory.generateCertPath( certlist );
byte[] certpathencoded = certpath.getEncoded( "PkiPath" );
result = new Base64String( certpathencoded ).toString();
```

To know more about this topic, refer to the online Oracle documentation, JavaTM PKI Programmer's Guide

signatures tag

A signature must only be generated using an algorithm known to Java 1.4, so that it can be validated.

Information
The binary resulting from the signature is encoded in Base64 and placed in the tag.

Example of generation in Java:

```
byte[] signature = ….sign( content_path ) ;
```

result = new Base64String(certpathencoded).toString();

To know more about this topic, refer to the online Oracle documentation, Java Cryptography Architecture (JCA) Reference Guide<

Policy tag

1. XML location: only the `root` element in the pivot format is queried by **Compliance** for a data processing *Policy*. The **`Policy`** information is a rule to apply when importing data. It may be either in the same XML file as the base element, or in a policy.xml file next to it.

2. **`rulename`** attribute: in a pivot's elements, each `deposit` element has a **`rulename`** attribute whose value is the name of a rule described in the **`root`** element.xml file (**`rule`** in **`policy`**). This attribute determines the destination of all `record` elements belonging to this `deposit`.

3. XML structure of Policy: the **`<policy>`** tag may contain several **`<rule>`** tags. Each **`<rule>`** tag must have:

- a rule name,
- the name or label of a class to instantiate to apply the rule,
- parameters specific to this class.

```
<policy>
 <rules>
  <rule>
   <rulename>CSUSER_rule</rulename>
   <classname>es.cls.core.pivot.PivotImportRuleImpl</classname>
   <classparams>
            ............
   </classparams>
  </rule>
 </rules>
</policy>
```

4. Specific rule supplied with EverSuite: a specific importing rule supplied with EverSuite may be used to avoid importing duplicates and to specify the destination of the data. The class is called `es.cls.core.pivot.PivotImportRuleImpl`.

The parameters to insert into **`<classparams>`** for the example above are the following:

```
.....
<classparams>
   <dbname>dPortal</dbname>
   <tablename>CSUSER</tablename>
   <profile />
   <double>no</double>
   <unicityfieldcheck>USERCODE</unicityfieldcheck>
   <actionondouble>ignore</actionondouble>
</classparams>
.....
```

`dbname`: database name

`tablename`: import table name

`profile`: EverSuite profile associated with the data

`double`: uniqueness test: *`yes`* (the import runs no checks)| *`no`*

`unicityfieldcheck`: field(s) tested for uniqueness
syntax: field1,field2, etc.

`actionondouble`: action to take when the uniqueness test fails:

- *`error`*: stopsprocessing and generates an error
- *`ignore`*: does not import the data
- *`update`*: updates the record

children tag

- The **`<children>`** tag in a **`element.xml`** file defines the "child" elements in the tree. For an import, the base element tag indicates the three `deposit` folders, containing data of different types (corresponding to the three import rules declared in the `<policy>` tag). These folders are for the **CSUSER**, **CSROLE** and **CSUSERROLES** tables.

```
<element>
 <id>1193675028593</id>
 .......
<attributes>
 <attribute>
  <attr_name>type</attr_name>
   <values>
     <value>root</value>
  </values>
 </attribute>
 ..........
<children>
 <child>
  <id>1193675028640</id>
  <path />
 </child>
 <child>
  <id>1193675028687</id>
  <path />
  </child>
  <child>
  <id>1193675028797</id>
  <path />
  </child>
</children>
```

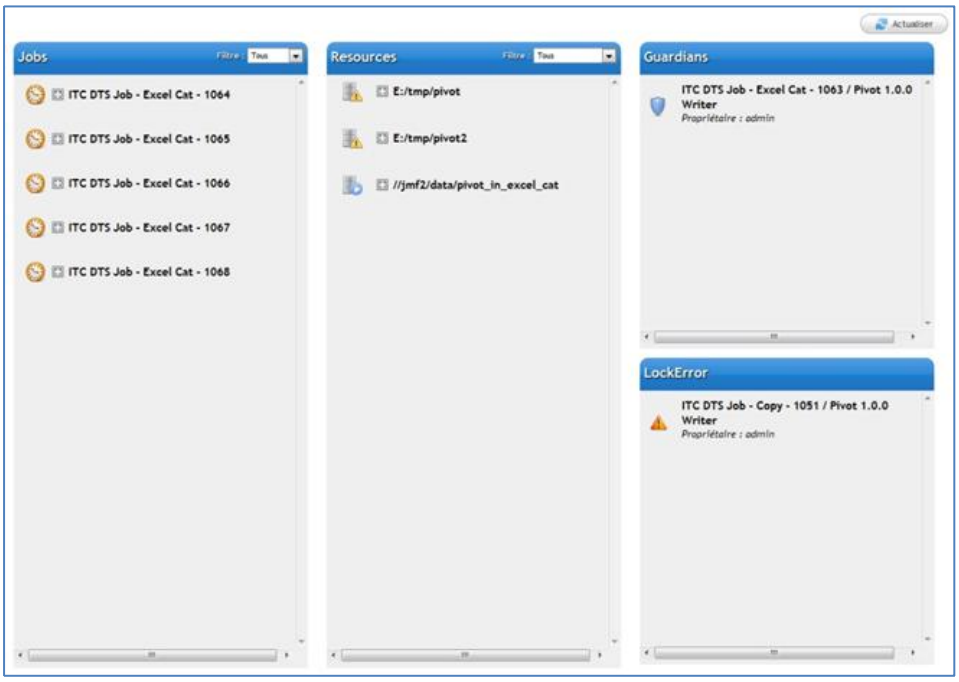
- The **`<children>`** tag in a `deposit` **`elements.xml`** file defines the "child" elements in each dataset, that is, the `record` folders of the elements themselves to import.

```
<element>
 <id>1193675028687</id>
 <attributes>
  <attribute>
  <attr_name>type</attr_name>
  <values>
    <value>deposit</value>
```

```
      </values>
      </attribute>
      <attribute>
        <attr_name>elt_name</attr_name>
        <values>
          <value>CSUSER</value>
        </values>
    </attribute>
        ........
</attributes>
<children>
  <child>
    <id>1193675028718</id>
    <path />
  </child>
  <child>
    <id>1193675028750</id>
    <path />
  </child>
<child>
  <id>1193675028781</id>
  <path />
  </child>
 </children>
</element>
```

2.3 Lock console

This console allows you to see links between different jobs, resources and guardians. The screen is divided into several columns.



ITC DTS Job - Excel Cat - 1064
ITC DTS Job - Excel Cat - 1065
ITC DTS Job - Excel Cat - 1066
ITC DTS Job - Excel Cat - 1067
ITC DTS Job - Excel Cat - 1068

E:/tmp/pivot
E:/tmp/pivot2
//jmf2/data/pivot_in_excel_cat

ITC DTS Job - Excel Cat - 1063 / Pivot 1.0.0
Writer
Propriétaire : admin

ITC DTS Job - Copy - 1051 / Pivot 1.0.0
Writer
Propriétaire : admin

Figure 41: Lock console

2.3.1 Jobs

The **Jobs** column lists locks by job.

An icon gives the status of the jobs (*Pending*/*Executing*), knowing that a job is executing if all its locks are active.

A click on the icon expands the job to obtain the list of its locks (with the status of each lock).

2.3.2 Resources

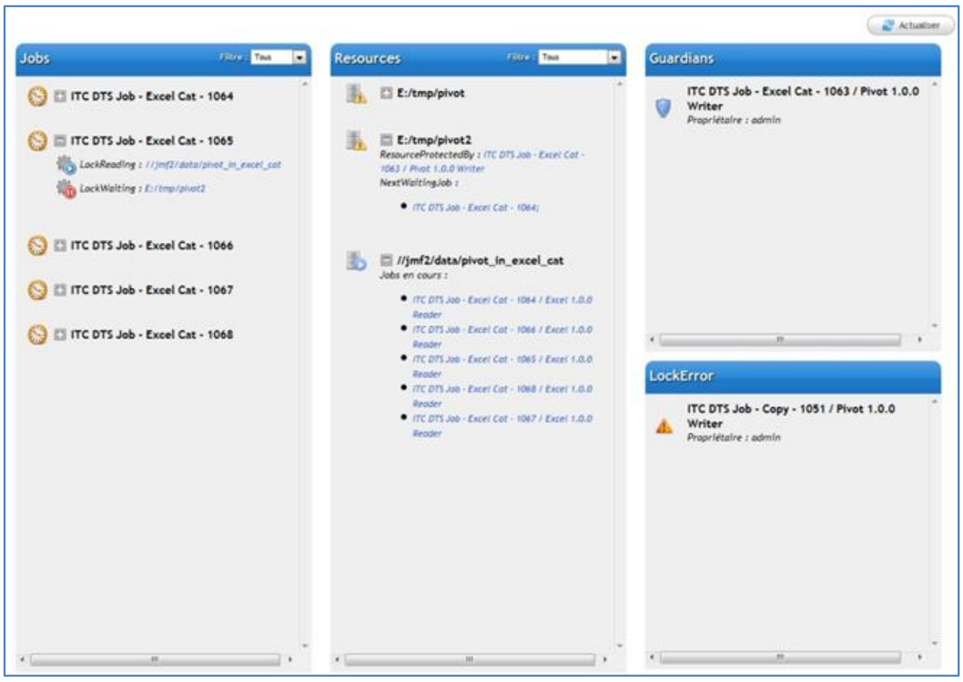
An icon gives the status of the resources (*Protected*/*Writing*/*Reading*).

A click on the icon expands the resource and shows its active lock and, if necessary, the list of pending jobs.

2.3.3 Guardians

This column displays the guardians and allows you to delete them if necessary.

If there are jobs with errors, they are displayed in a second frame, below the guardians (as a general rule, the column only shows the guardians).



ITC DTS Job - Excel Cat - 1064
ITC DTS Job - Excel Cat - 1065
LockReading : //jmf2/data/pivot_in_excel_cat
LockWaiting : E:/tmp/pivot2
ITC DTS Job - Excel Cat - 1066
ITC DTS Job - Excel Cat - 1067
ITC DTS Job - Excel Cat - 1068

E:/tmp/pivot
E:/tmp/pivot2
ResourceProtectedBy : ITC DTS Job - Excel Cat - 1063 / Pivot 1.0.0 Writer
NextWaitingJob :
- ITC DTS Job - Excel Cat - 1064;

//jmf2/data/pivot_in_excel_cat
Jobs en cours :
- ITC DTS Job - Excel Cat - 1064 / Excel 1.0.0 Reader
- ITC DTS Job - Excel Cat - 1066 / Excel 1.0.0 Reader
- ITC DTS Job - Excel Cat - 1065 / Excel 1.0.0 Reader
- ITC DTS Job - Excel Cat - 1068 / Excel 1.0.0 Reader
- ITC DTS Job - Excel Cat - 1067 / Excel 1.0.0 Reader

ITC DTS Job - Excel Cat - 1063 / Pivot 1.0.0 Writer
Propriétaire : admin

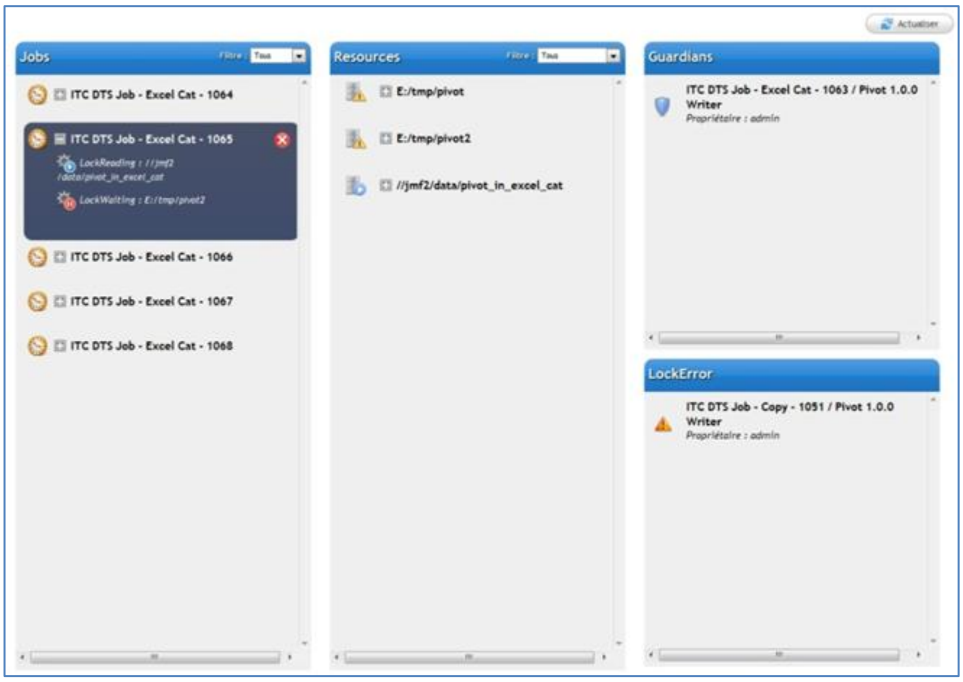
ITC DTS Job - Copy - 1051 / Pivot 1.0.0 Writer
Propriétaire : admin

Figure 42: Display for a job with errors

Once expanded, locks in the first two columns are clickable.

A click on a job's lock (column on the left) highlights the resource associated to this same lock in the middle column. A click on a resource's lock (middle column) highlights the lock in the column on the left (if it is a lock for a job which is active or pending) or in the column on the right (if it is a guardian).

If the mouse hovers over a job or a guardian, an icon is displayed (red cross) to delete locks for the job (column on the left) or for the guardian or a job with errors (column on the right).



ITC DTS Job - Excel Cat - 1064
ITC DTS Job - Excel Cat - 1065
LockReading : //jmf2/data/pivot_in_excel_cat
LockWaiting : E:/tmp/pivot2
ITC DTS Job - Excel Cat - 1066
ITC DTS Job - Excel Cat - 1067
ITC DTS Job - Excel Cat - 1068

E:/tmp/pivot
E:/tmp/pivot2
//jmf2/data/pivot_in_excel_cat

ITC DTS Job - Excel Cat - 1063 / Pivot 1.0.0 Writer
Propriétaire : admin

ITC DTS Job - Copy - 1051 / Pivot 1.0.0 Writer
Propriétaire : admin

Figure 43: Highlighting the job's lock

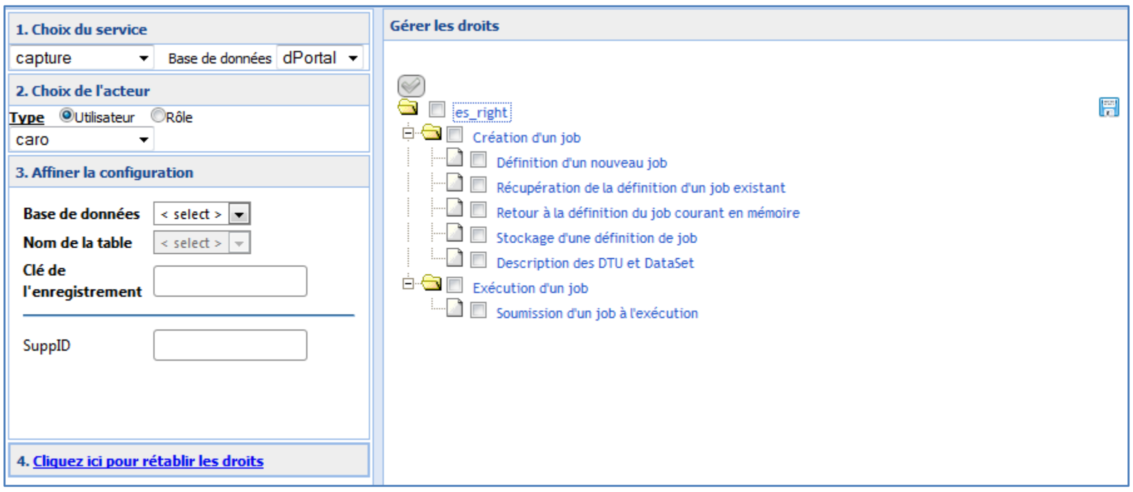
3 SECURITY

The functions available in the ES-Capture service depend on users' (or roles') **rights**.

To set up roles and/or users with these rights, go to the Administration > Security > Service security menu, service: capture.

The window displayed enables access to roles or to users in order to configure the interface.

In the Choose actor section, Role is selected by default and the roles appear in a drop-down list. To select a user, you just check `User` and select the code for the user (here `caro`), so that the window looks like this:



1. Choix du service
capture | Base de données dPortal

2. Choix de l'acteur
Type [✓ Utilisateur] Rôle
caro

3. Affiner la configuration
Base de données < select >
Nom de la table < select >
Clé de l'enregistrement
SuppID

4. Cliquez ici pour rétablir les droits

Gérer les droits
- es_right
  - Création d'un job
    - Définition d'un nouveau job
    - Récupération de la définition d'un job existant
    - Retour à la définition du job courant en mémoire
    - Stockage d'une définition de job
    - Description des DTU et DataSet
  - Exécution d'un job
    - Soumission d'un job à l'exécution

Figure 44: Security and rights management

A click on the ✓ icon selects or deselects all the rights.

A click on [4. Click to restore rights] restores the rights last saved.

A click on 💾 [Save] saves the rights declared.

4 CONFIGURATION

4.1 PARAMETER LIST FOR THE ES-DTS SERVICE

Parameters specific to ES-DTS are accessible in the Administration > Parameters > Service menu, service dts.

This screen contains all the parameters which the administrator can modify for his/her application.

Reference
For a detailed explanation of all the ES-DTS parameters, please refer to<t0/> the<t1/>EverSuite CORE Parameters<t2/> guide.

Parameter	Description	Default value
DTSMergeXMLElement	Export all parts of an object in a single file, element.xml true/false	true
DTSMergeXMLList	Export elements at the same level in a single elements.xml file true/false	true
DTUMap	Mapping of operations between input data and output data	
ErrorMailer	Parameters for emails sent when there is an error in a Capture job (From / To / Subject / Form used to construct the message body)	
JobThreadMaxPriority	Maximum priority that can be given to a job thread Between 1 (minimum) and 10 (maximum)	4
JobUserCode	Code of the user who must execute the job, when this is required	
MaxJobsLaunchAtOnce	Maximum number of jobs to be launched by a single call to ScheduleJobs, in order to be able to reserve enough resources	5
MaxJobsRunning	Maximum number of jobs running at the same time	5

Figure 45: ES-DTS parameters

4.2 USING SCHEDULEMANAGER

When EverSuite starts up, the **ScheldeleManager** servlet executes a certain number of methods based on the frequency set up for them, and on the date they were first executed.

The **JobScheduler** class may be attached to EverSuite's **ScheduleManager** which executes the **JobManager.scheduleJobs()** method periodically. This class decides the order in which the jobs must be run and how many jobs can execute at the same time. It may be called by another class, for example when a job is submitted and placed in the queue, or when a job completes.

4.3 USING ES-BIS

The ES-BIS **service** (Activity statistics) stores ES-Capture process events. It is informed when jobs are created, submitted, or executed, about progress through the steps, and whether they run successfully or not.

ES-BIS may be queried on the progress of all processing cycles, current or complete, successful or with errors.

You should check that the ES-BIS service is set up correctly:

Parameter	Description	Default value
bamDBName	Name of the ES-BIS service database	*dPortal*
bamLogClass	Name of the default class for managing logs	*es.cls.bam.traces.bamLogTraDebug*
bamPath	ES-BIS path	*/apps/bam*
bamSpecificActions	ES-BIS specific actions	*<ACTIONS LABEL="ExtraLog"> </ACTIONS>*

Figure 46: ES-BIS parameters

5 PREDEFINED JOBS

The ES-Capture > Predefined Jobs menu makes it easy for the administrator to implement import solutions:

- [<t0/Standard import]: this is for data to import described in a simple schema (pivot light).
- [Automatic deployment]: two datasets come as standard for importing request (Case Management Solution) or emails (ES-MMS service).

5.1 STANDARD IMPORT

The ES-Capture > Predefined Jobs > Standard import menu gives access to the following form:

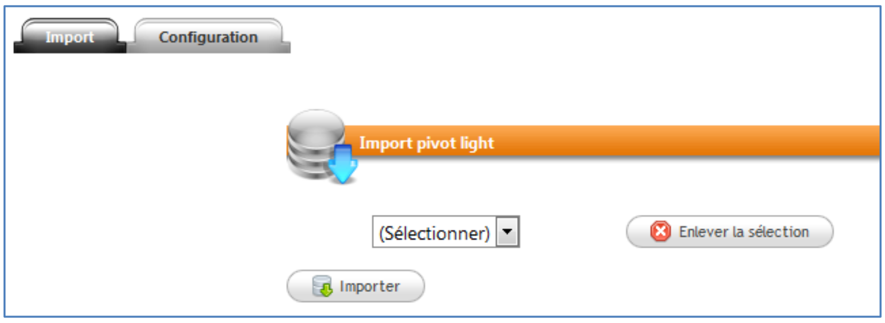


Figure 47: pivot light import format

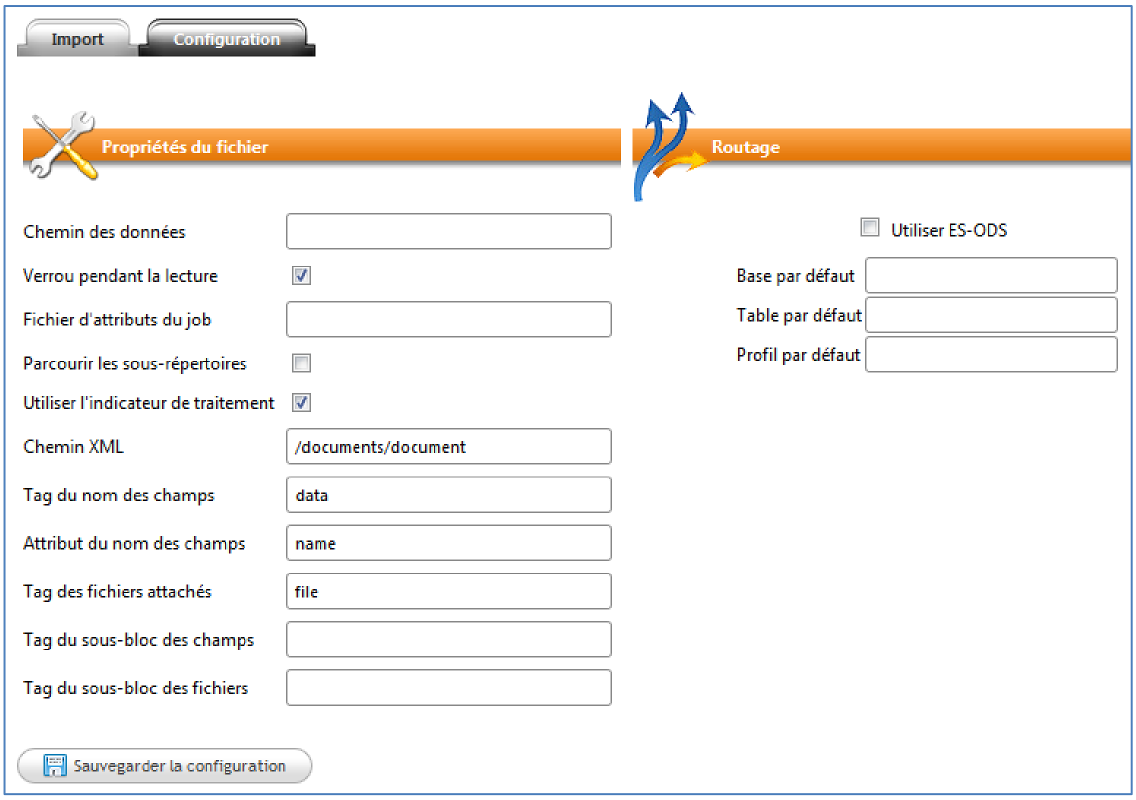
Import data must be in a simple format such as the pivot light format supplied by EVER TEAM. The data are described in the [Configuration] tab and imported in the [Import]tab.

5.1.1 Declaring a configuration

The [Configuration] tab provides a simple environment for defining the data structure and location and declaring the target for the import (routing for mapping).

The data for the imported document metadata are in a single file whose structure is relatively simple.

The original documents (text, images or pdf) are stored in the same directory as the import file or in sub-directories.



Chemin des données	
Verrou pendant la lecture	✓
Fichier d'attributs du job	
Parcourir les sous-répertoires	☐
Utiliser l'indicateur de traitement	✓
Chemin XML	/documents/document
Tag du nom des champs	data
Attribut du nom des champs	name
Tag des fichiers attachés	file
Tag du sous-bloc des champs	
Tag du sous-bloc des fichiers	

Utiliser ES-ODS	☐
Base par défaut	
Table par défaut	
Profil par défaut	

Figure 48: Configuring a capture

File properties

The first set of information in the **File properties** column is about the location and structure of the data file. Tags and tag attributes are given.

If the tags containing field names are left empty, the names of the tags containing field values correspond to the target table field names.

In the example above, the data must be in this form:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<documents>
 <document>
     <data name="EXTRAITDE">Enjeux Les Echos</data>
     <data name="AUTEUR">BACHET Emma</data>
     <data name="TITRE">Palmyre, Venise des sables</data>
     <data name="NUM">Special History Issue</data>
     <data name="MOTSCLES">History, Civilisation</data>
     <file>img-Palmyre.pdf</file>
 </document>
 <document>
     <data name="EXTRAITDE">Solutions et Logiciels</data>
     <data name="AUTEUR">VOROIT Claire</data>
     <data name="TITRE">Office 2010, au-delà de la bureautique</data>
     <data name="NUM">Microsoft Office 2010 edition</data>
     <data name="MOTSCLES">office, word processing</data>
     <file>img-Office-2010.pdf</file>
     <file>img-Presse-0001.pdf</file>
 </document>
```

Tag for information about a record

Tag for different record fields

Attribute giving the name of the target field

Tag giving the name of the document in the same directory

```
</documents>
```

Data path: path and name of the data file.
Example: D:/data/file.xml

Lock during read: lock data while the file is being used by ES-Capture.

Job attributes file: xml file with properties used in the job.

Parse sub-directories: check if there are files to process in the subdirectories.

Use processing mark: add a start file, which will be deleted afterwards if processing is successful.

XML path: series of tags and name of the tag to get metadata describing a document.
Example: /documents/document for <documents><document>… </document></documents>

Field name tag: name of the tag containing the values to import (metadata)

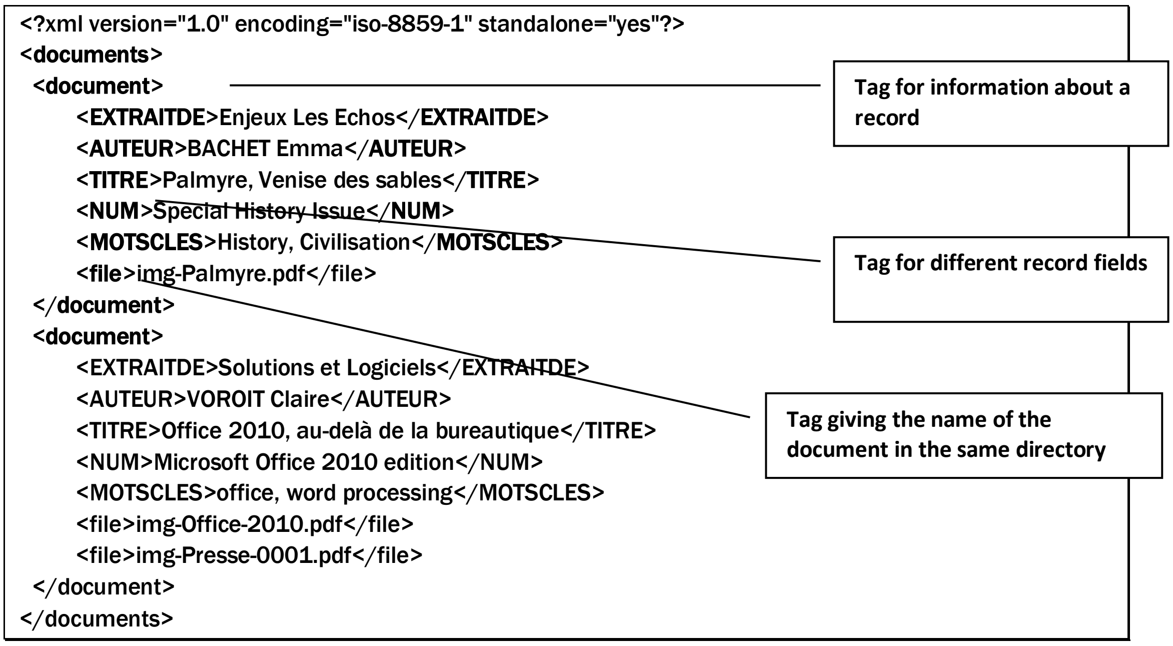
Field name atttribute: name of the attribute containing field names.

Attached files tag: name of the tag which contains document names.

Field sub-block tag: *(if there are sub-blocks in the file structure).*

File sub-block tag: *(if there are sub-blocks in the file structure).*

If only the data path and the XML path are supplied, that means it is a file of the same type as the one shown below, where the field names are given directly in the tag:


Tag for information about a record
Tag for different record fields
Tag giving the name of the document in the same directory

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<documents>
 <document>
     <EXTRAITDE>Enjeux Les Echos</EXTRAITDE>
     <AUTEUR>BACHET Emma</AUTEUR>
     <TITRE>Palmyre, Venise des sables</TITRE>
     <NUM>Special History Issue</NUM>
     <MOTSCLES>History, Civilisation</MOTSCLES>
     <file>img-Palmyre.pdf</file>
 </document>
 <document>
     <EXTRAITDE>Solutions et Logiciels</EXTRAITDE>
     <AUTEUR>VOROIT Claire</AUTEUR>
     <TITRE>Office 2010, au-delà de la bureautique</TITRE>
     <NUM>Microsoft Office 2010 edition</NUM>
     <MOTSCLES>office, word processing</MOTSCLES>
     <file>img-Office-2010.pdf</file>
     <file>img-Presse-0001.pdf</file>
 </document>
</documents>
```

Other possible formats

Data may be in more complex files, provided that it is possible to enter information in the block or sub-block for metadata and the path where the original documents can be found.

Example:
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>

```
<root>
  <this>
    <that>aaaaa</that>
  </this>
  <list>
   <record>
    <field colname="type_document">Legal clauses</field>
    <field colname="archive_box">1</field>
    <field colname="member_num">01234567</field>
    <this>
     <that>bbbb</that>
     <this>
      <that>cccc</that>
     </this>
    </this>
    <field colname="contract">CCFPrev01</field>
    <field colname="name">MARTIN</field>
    <field colname="first_name">SYLVIE</field>
    <field colname="date_received">02/12/2012</field>
    <image>0001.tif</image>
   </record>
   <this>
    <that>dddd</that>
   </this>
   <record>
    <field colname="type_document">legal clauses</field>
    <field colname="archive_box">1</field>
    <field colname="member_num">00445121</field>
    <field colname="contract">CCFPrev01</field>
    <field colname="name">VILUS</field>
    <field colname="first_name">RAYMONDE</field>
    <field colname="date_received">04/09/2012</field>
    <image>0002.tif</image>
    <image>0003.tif</image>
   </record>
  </list>
</root>
```

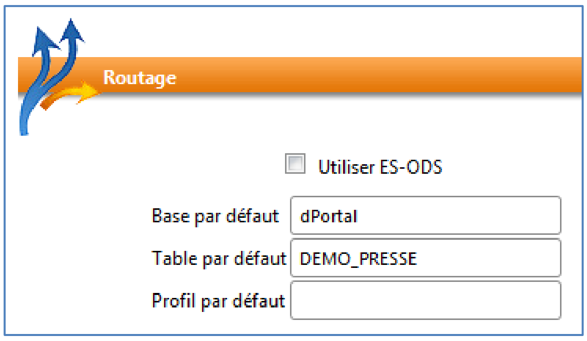
where the information supplied is:

XML path	*<root><list><record>*
Field name tag	*field*
Field name attribute	*colname*
Attached file tag	*image*

Routing

The target table for the import can be given directly in the `Routing` column or by using the routing tools in the EverSuite **ES-ODS** service.

Direct input



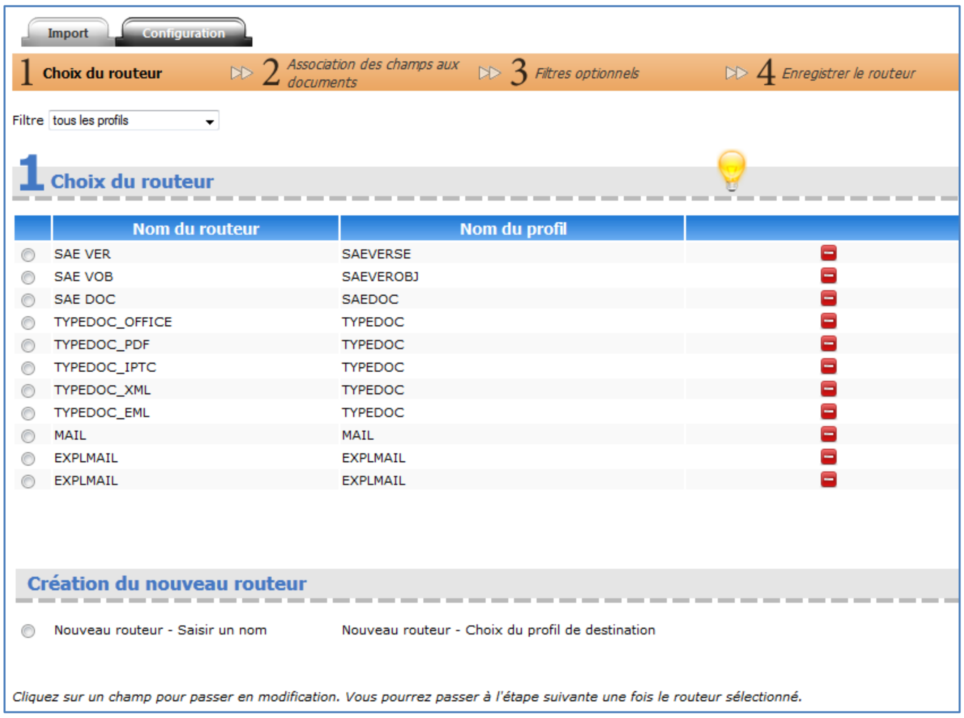
- [ ] Utiliser ES-ODS

Base par défaut	dPortal
Table par défaut	DEMO_PRESSE
Profil par défaut	

Figure 49: Direct input

Via the ES-ODS service

If you check the **Use ES-ODS** option, you get access to the ES-ODS service to define routing or select an existing router.



Filtre tous les profils

1 Choix du routeur

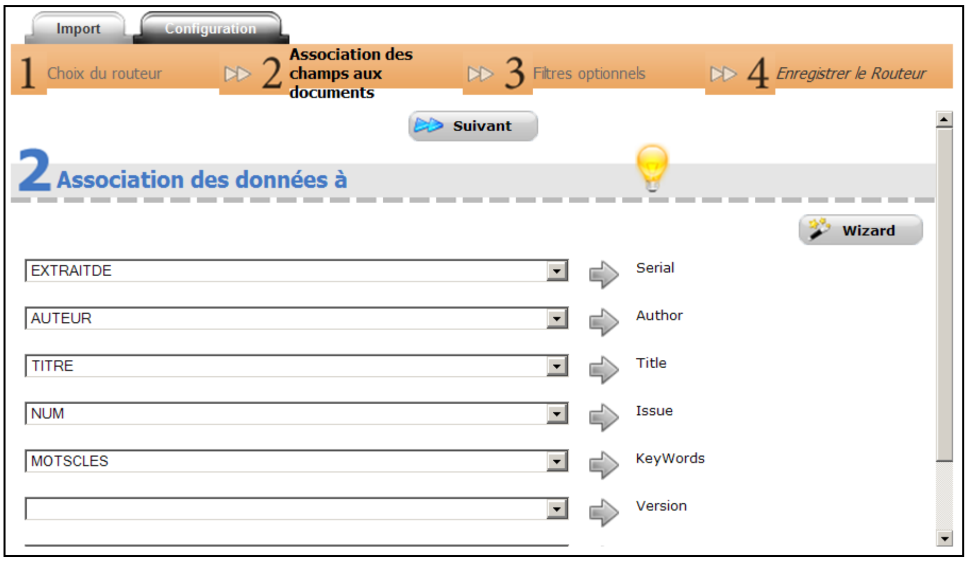
Nom du routeur	Nom du profil
SAE VER	SAEVERSE
SAE VOB	SAEVEROBJ
SAE DOC	SAEDOC
TYPEDOC_OFFICE	TYPEDOC
TYPEDOC_PDF	TYPEDOC
TYPEDOC_IPTC	TYPEDOC
TYPEDOC_XML	TYPEDOC
TYPEDOC_EML	TYPEDOC
MAIL	MAIL
EXPLMAIL	EXPLMAIL
EXPLMAIL	EXPLMAIL

Création du nouveau routeur

Nouveau routeur - Saisir un nom   Nouveau routeur - Choix du profil de destination

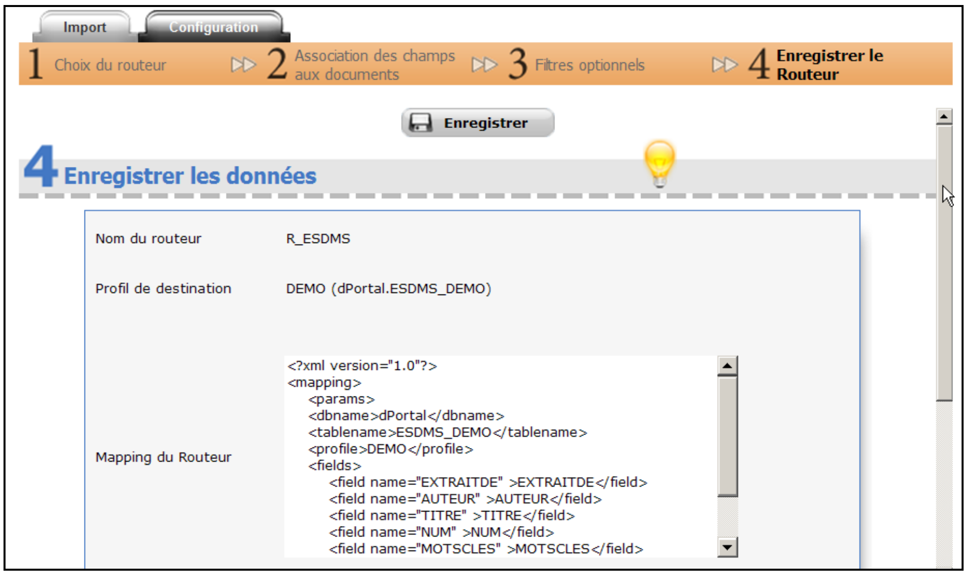
Cliquez sur un champ pour passer en modification. Vous pourrez passer à l'étape suivante une fois le routeur sélectionné.

Figure 50: Input via ES-ODS / Choosing a router



EXTRAITDE	Serial
AUTEUR	Author
TITRE	Title
NUM	Issue
MOTSCLES	KeyWords
	Version

Figure 51: Input via ES-ODS / Mapping data to fields



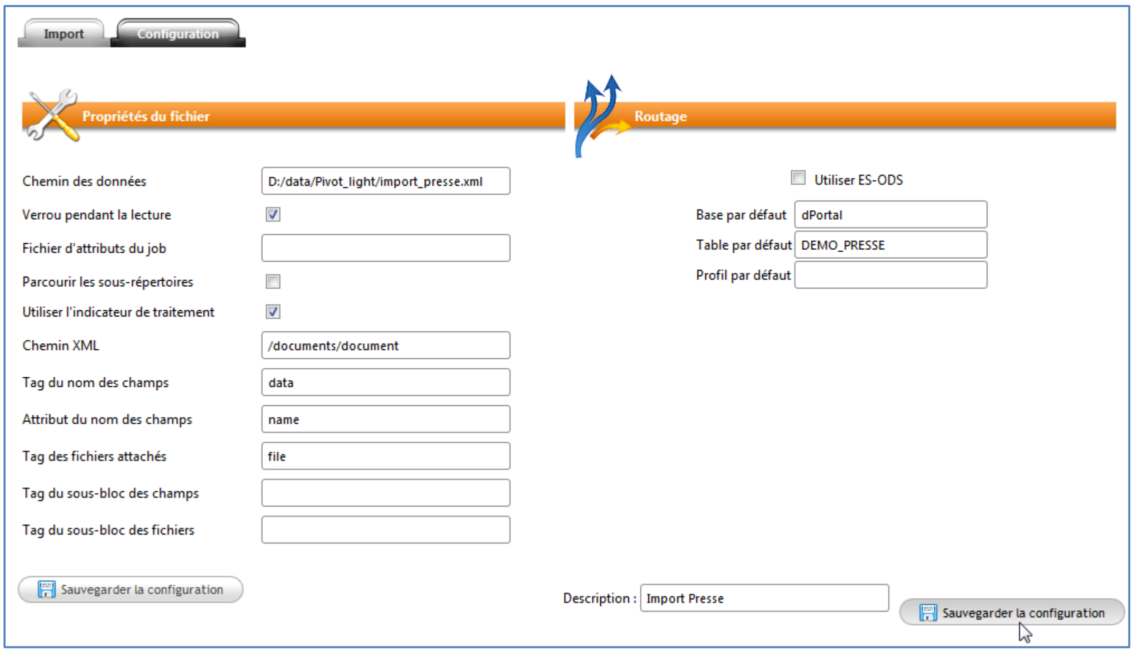
Nom du routeur	R_ESDMS
Profil de destination	DEMO (dPortal.ESDMS_DEMO)
Mapping du Routeur	

```
<?xml version="1.0"?>
<mapping>
   <params>
   <dbname>dPortal</dbname>
   <tablename>ESDMS_DEMO</tablename>
   <profile>DEMO</profile>
   <fields>
      <field name="EXTRAITDE" >EXTRAITDE</field>
      <field name="AUTEUR" >AUTEUR</field>
      <field name="TITRE" >TITRE</field>
      <field name="NUM" >NUM</field>
      <field name="MOTSCLES" >MOTSCLES</field>
```

Figure 52: Input via ES-ODS / Save router

5.1.2 Saving a setup

A click on the [Save configuration] button opens a Description box where you can enter the name of the import then click on the [Save configuration] button on the right.



Chemin des données	D:/data/Pivot_light/import_presse.xml
Verrou pendant la lecture	✓
Fichier d'attributs du job	
Parcourir les sous-répertoires	☐
Utiliser l'indicateur de traitement	✓
Chemin XML	/documents/document
Tag du nom des champs	data
Attribut du nom des champs	name
Tag des fichiers attachés	file
Tag du sous-bloc des champs	
Tag du sous-bloc des fichiers	

Utiliser ES-ODS	☐
Base par défaut	dPortal
Table par défaut	DEMO_PRESSE
Profil par défaut	
Description :	Import Presse

Figure 53: Saving a setup

5.1.3 Importing data

Once the configurations have been described, click on the [Import] tag to import data straight into the target table in your EverSuite application. Select the configuration to use:

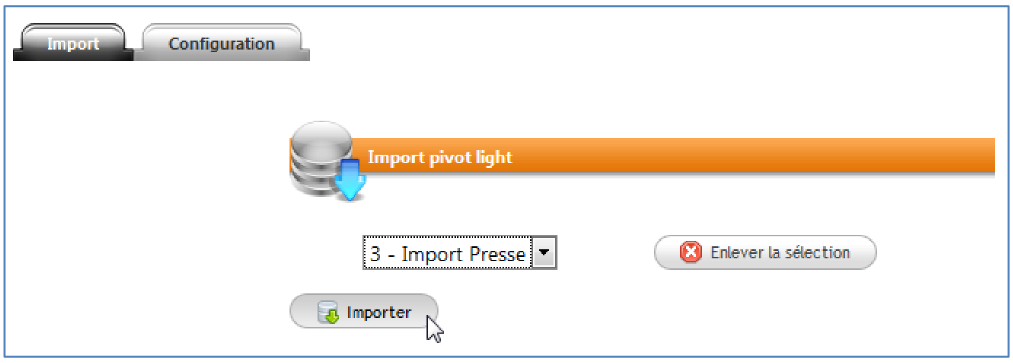


Figure 54: Importing data in pivot light format

The import is confirmed with the following message:

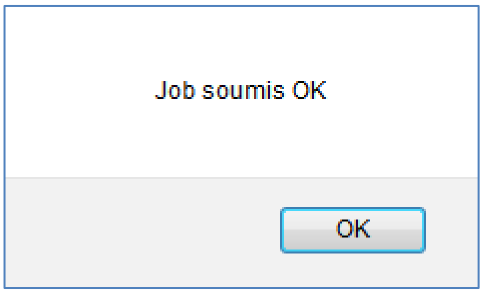
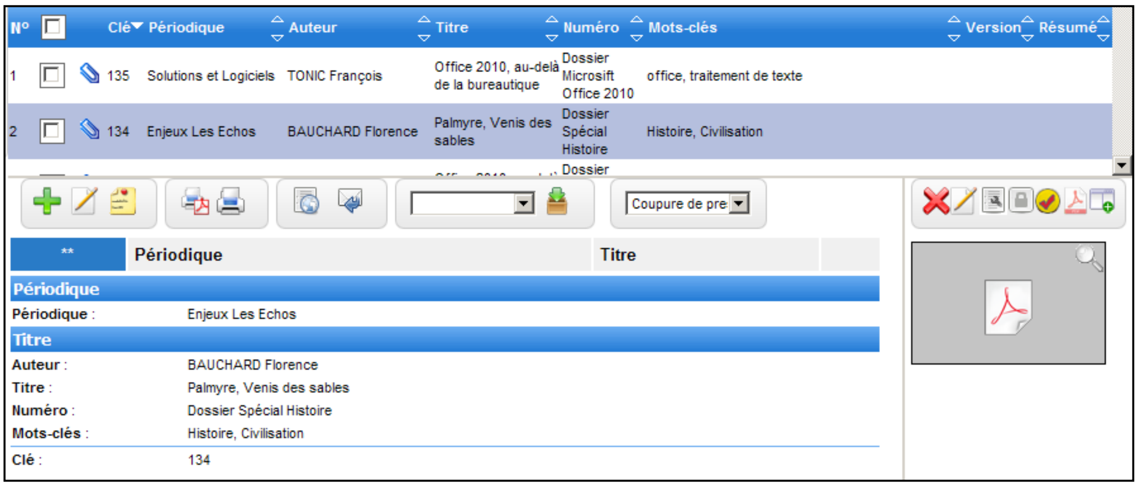

Job soumis OK

Figure 55: Message confirming an import

By querying the import table (e.g. **DEMO_PRESSE**) we can find the imported records, with their attached documents:



Nº		Clé	Périodique	Auteur	Titre	Numéro	Mots-clés	Version	Résumé
1	☐	135	Solutions et Logiciels	TONIC François	Office 2010, au-delà de la bureautique	Dossier Microsift Office 2010	office, traitement de texte		
2	☐	134	Enjeux Les Echos	BAUCHARD Florence	Palmyre, Venis des sables	Dossier Spécial Histoire	Histoire, Civilisation		

Coupure de pre

**	Périodique	Titre

Périodique

Périodique :	Enjeux Les Echos

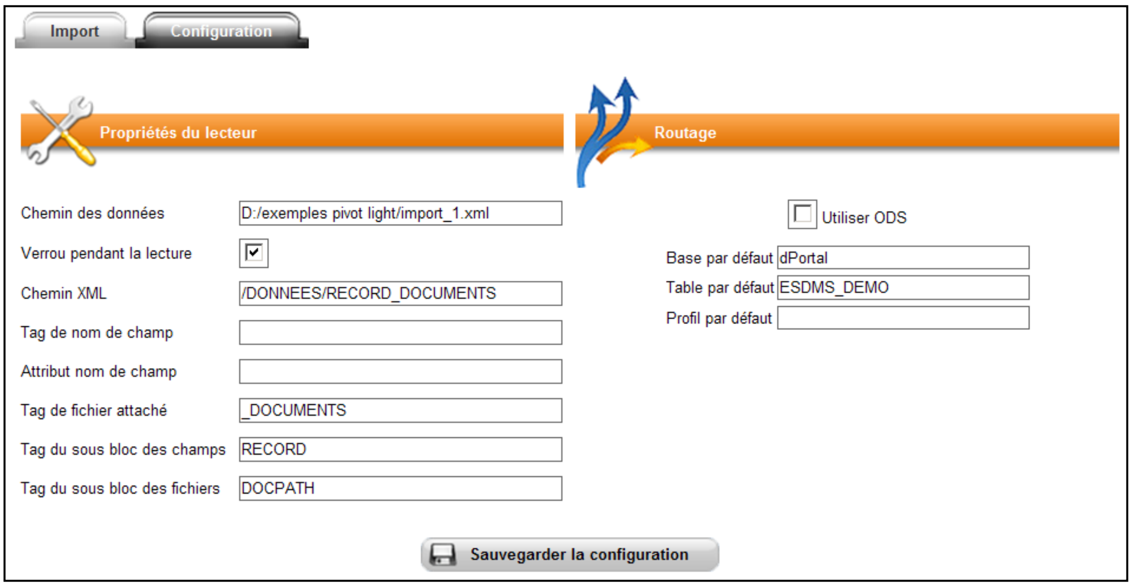
Titre

Auteur :	BAUCHARD Florence
Titre :	Palmyre, Venis des sables
Numéro :	Dossier Spécial Histoire
Mots-clés :	Histoire, Civilisation
Clé :	134

Figure 56: List of imported records

Caution
When the import is successful, the data file is deleted from the directory.
Before importing, you are therefore advised to back up your file.

5.1.4 Example



Chemin des données	D:/exemples pivot light/import_1.xml
Verrou pendant la lecture	☑
Chemin XML	/DONNEES/RECORD_DOCUMENTS
Tag de nom de champ	
Attribut nom de champ	
Tag de fichier attaché	_DOCUMENTS
Tag du sous bloc des champs	RECORD
Tag du sous bloc des fichiers	DOCPATH

Utiliser ODS	☐
Base par défaut	dPortal
Table par défaut	ESDMS_DEMO
Profil par défaut	

Figure 57: Setup declaration

This setup is based on a file like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<DATA>
  <DB_Name>dPortal</DB_Name>
  <Table_Name>ESDMS_DEMO</Table_Name>
  <RECORD_DOCUMENTS>
      <RECORD>
        <EXTRAITDE><![CDATA[Le Monde]]></EXTRAITDE>
        <AUTEUR><![CDATA[MATINUS Marc]]></AUTEUR>
        <TITRE><![CDATA[Analyse des nouvelles fonctions]]></TITRE>
      </RECORD>
      <_DOCUMENTS>
        <DOCPATH>140/img-01212.pdf</DOCPATH>
      </_DOCUMENTS>
  </RECORD_DOCUMENTS>

  <RECORD_DOCUMENTS>
      <RECORD>
        <EXTRAITDE><![CDATA[Le Monde]]></EXTRAITDE>
        <AUTEUR><![CDATA[GOUTALAN Cécile]]></AUTEUR>
        <TITRE><![CDATA[Le cinéma aujourd'hui]]></TITRE>
      </RECORD>
      <_DOCUMENTS>
              <DOCPATH>141/img-01247-0002.pdf</DOCPATH>
              <DOCPATH>141/img-01245-0001.pdf</DOCPATH>
      </_DOCUMENTS>
  </RECORD_DOCUMENTS>
</DATA>
```

5.2 AUTOMATIC DEPLOYMENT

Two datasets come as standard:

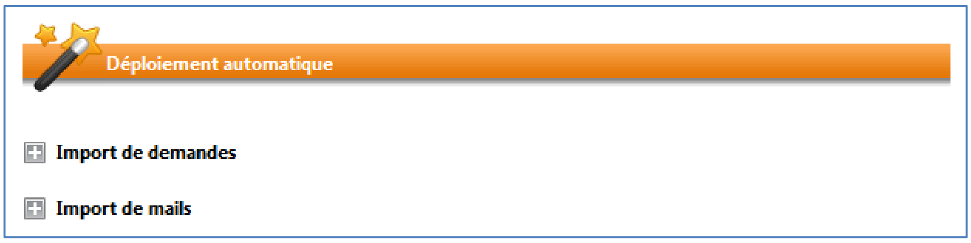
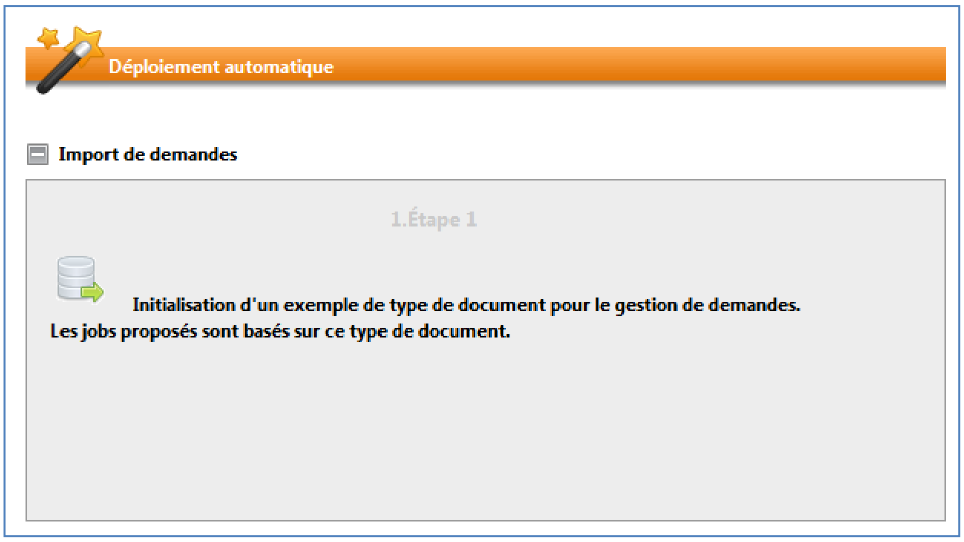


Figure 58: Automatic deployment

5.2.1 Request import

The first standard import configuration is for data processed in the Case Management solution. Jobs are based on this document type.


Déploiement automatique

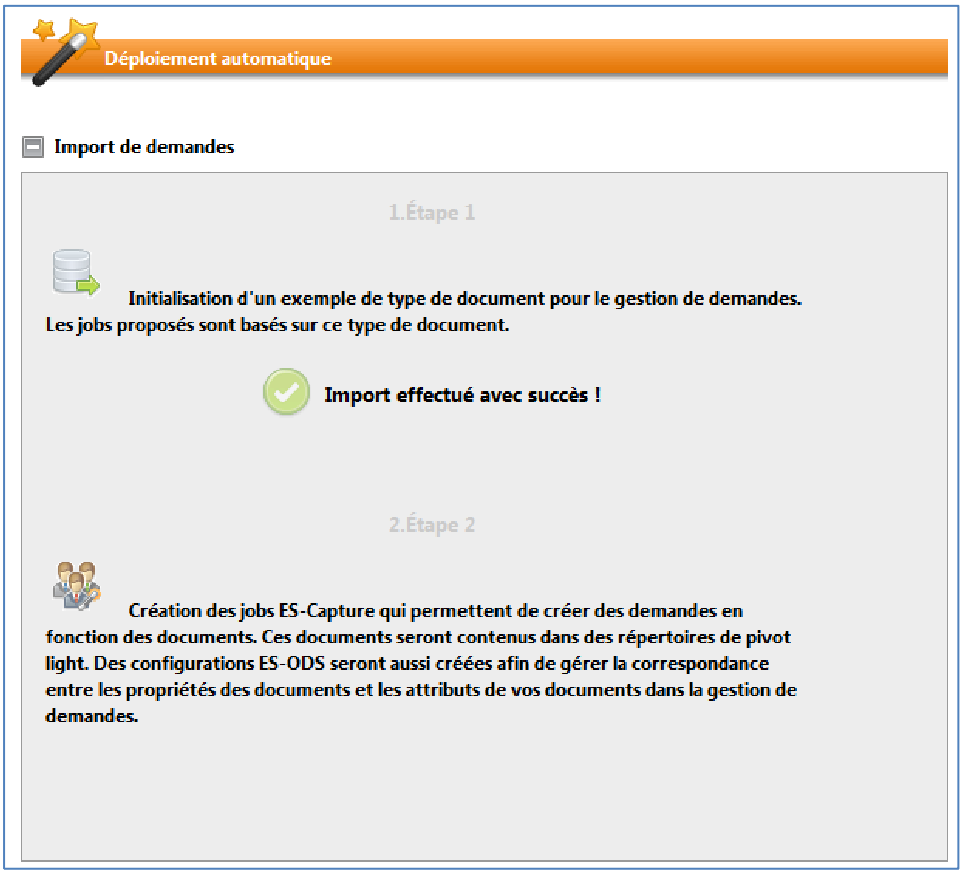
Import de demandes

1.Étape 1

Initialisation d'un exemple de type de document pour le gestion de demandes. Les jobs proposés sont basés sur ce type de document.

Figure 59: Request import / Step 1

Click on the button to start importing requests:


Déploiement automatique

Import de demandes

1.Étape 1

Initialisation d'un exemple de type de document pour le gestion de demandes. Les jobs proposés sont basés sur ce type de document.

Import effectué avec succès !

2.Étape 2

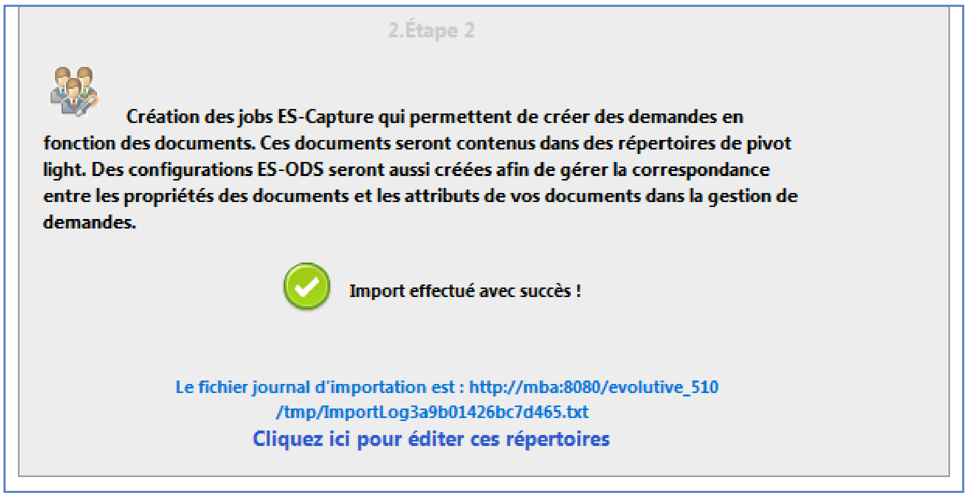
Création des jobs ES-Capture qui permettent de créer des demandes en fonction des documents. Ces documents seront contenus dans des répertoires de pivot light. Des configurations ES-ODS seront aussi créées afin de gérer la correspondance entre les propriétés des documents et les attributs de vos documents dans la gestion de demandes.

Figure 60: Request import / Step 2

Click on the button to generate the corresponding ES-Capture jobs:

Deploying these data creates ES-Capture jobs for creating requests based on documents. These documents will be in the directories of data in pivot light format.

ES-ODS configurations will also be created to handle mappings between document properties and the attributes of your documents in case management.


2.Étape 2

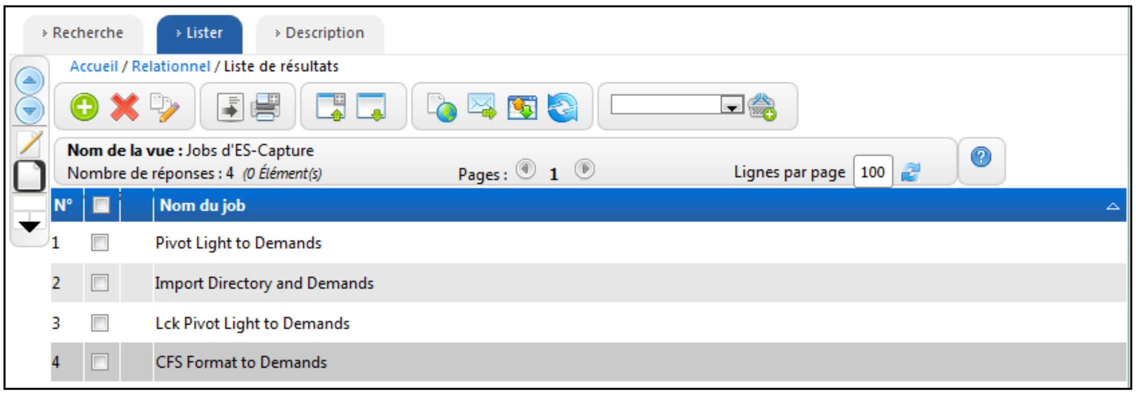
Création des jobs ES-Capture qui permettent de créer des demandes en fonction des documents. Ces documents seront contenus dans des répertoires de pivot light. Des configurations ES-ODS seront aussi créées afin de gérer la correspondance entre les propriétés des documents et les attributs de vos documents dans la gestion de demandes.

Import effectué avec succès !

Le fichier journal d'importation est : http://mba:8080/evolutive_510/tmp/ImportLog3a9b01426bc7d465.txt

Cliquez ici pour éditer ces répertoires

Figure 61: Request import / Access the import log file

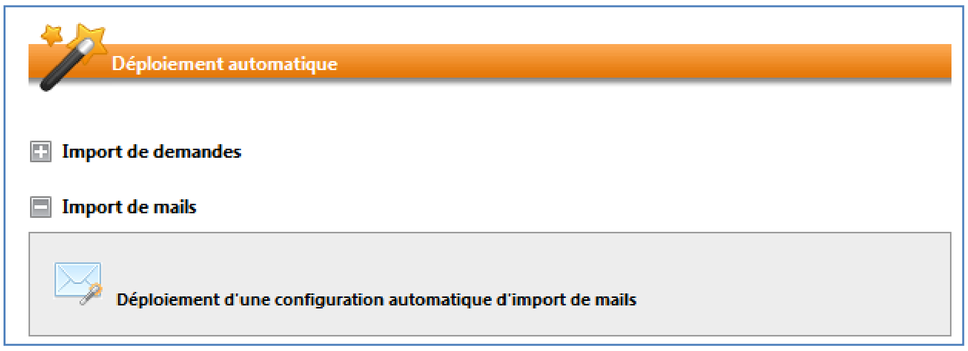


Nom de la vue : Jobs d'ES-Capture
Nombre de réponses : 4 (0 Élément(s))
Pages : 1
Lignes par page 100

N°	☐	Nom du job
1	☐	Pivot Light to Demands
2	☐	Import Directory and Demands
3	☐	Lck Pivot Light to Demands
4	☐	CFS Format to Demands

Figure 62: Imported ES-Capture jobs

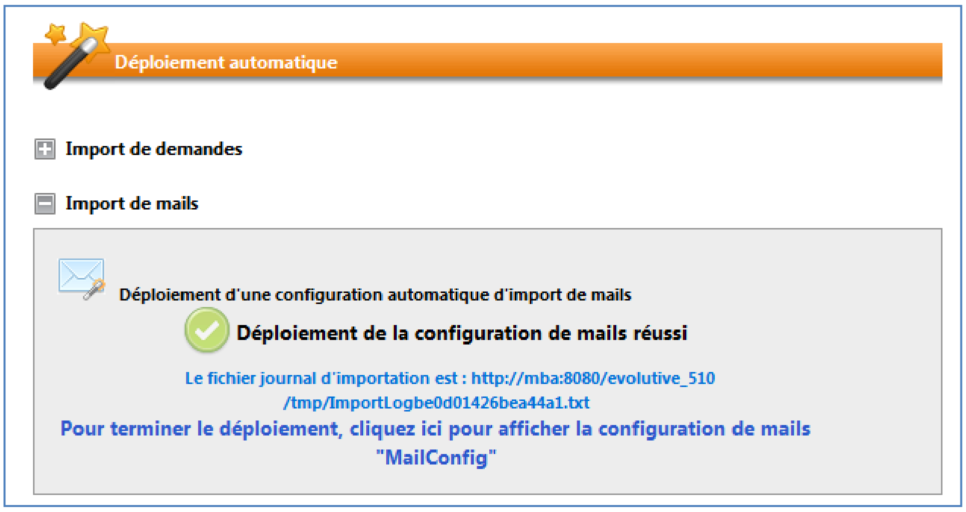
5.2.2 Email import



Déploiement d'une configuration automatique d'import de mails

Figure 63: Email import

Click on the button to start importing emails:


Déploiement automatique

Import de demandes

Import de mails

Déploiement d'une configuration automatique d'import de mails

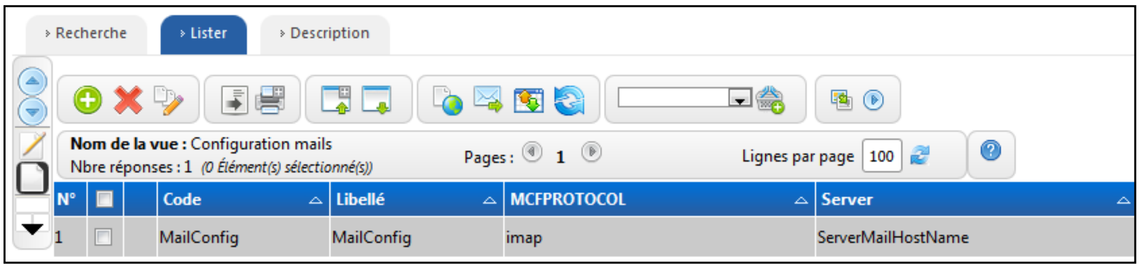
Déploiement de la configuration de mails réussi

Le fichier journal d'importation est : http://mba:8080/evolutive_510/tmp/ImportLogbe0d01426bea44a1.txt

Pour terminer le déploiement, cliquez ici pour afficher la configuration de mails "MailConfig"

Figure 64: Finishing deployment of the email import

Deployment adds the `MailConfig` configuration, which you should adapt to your application.



Nom de la vue : Configuration mails
Nbre réponses : 1 (0 Élément(s) sélectionné(s))
Pages : 1
Lignes par page 100

N°		Code	Libellé	MCFPROTOCOL	Server
1	☐	MailConfig	MailConfig	imap	ServerMailHostName

Figure 65: MailConfig email configuration deployed

6 Appendices

6.1 Processing components

6.1.1 Jobs

A job definition can be represented in XML and saved by the `JobBuilder`. It is handled in Java by the Job class. It can be reloaded, modified and reused by the `JobBuilder`.

```
<?xml version="1.0"?>
<Job>
        <Name></Name>
        <StartPriority></StartPriority>
        <ThreadPriority></ThreadPriority>
        <Step>
                <Name></Name>
                <StopOnError></ StopOnError >
                <DeleteInputOnSuccess></DeleteInputOnSuccess>
                <DeleteOutputOnError></ DeleteOutputOnError >
                <Input>
                        <ClassName></ClassName>
                        <URI></URI>
                </Input>
                <DTU>
                        <ClassName></ClassName>
                </DTU>
                <Output>
                        <ClassName></ClassName>
                        <URI></URI>
                </Output>
        </Step>
        <Step>
                <Name></Name>
                <StopOnError></ StopOnError >
                <DeleteInputOnSuccess></DeleteInputOnSuccess>
                <DeleteOutputOnError></ DeleteOutputOnError >
                <!– Input is previous output –>
                <DTU>
                        <ClassName></ClassName>
                </DTU>
                <Output>
                        <ClassName></ClassName>
                        <URI></URI>
                </Output>
        </Step>
        <Step>
                <Name></Name>
                <StopOnError></ StopOnError >
                <DeleteInputOnSuccess></DeleteInputOnSuccess>
                <DeleteOutputOnError></ DeleteOutputOnError >
                <!– Input is previous output –>
                <DTU>
                        <ClassName></ClassName>
```

```
            </DTU>
            <Output>
                <ClassName></ClassName>
                <URI></URI>
            </Output>
      </Step>
</Job>
```

Figure 66: XML representation of a job

Name tag

Name of the Job, used for display purposes and/or as a file name.

StartPriority tag

Starting priority.

This is represented by a signed number (default `0`). A positive number (`+1` or higher) allows a job to be started before ones with no priority, and a negative (`-1` or less) will make it start after them.

The starting order is not of great importance, as it influences only the order in which threads are created.

However using a **`MaxJobsRunning`** parameter to limit the number of active jobs allows you to delay execution of less urgent processes, which helps the more urgent ones by leaving them more machine resources.

ThreadPriority tag

Manages the priority of the job thread. Possible values are the ones accepted by the Thread Java class, from 1 to 10, 5 being the normal priority:

- 1 = Thread.MIN_PRIORITY
- 5 = Thread.NORM_PRIORITY
- 10 = Thread.MAX_PRIORITY

All the job threads are created in a ThreadGroup whose maximum value can be specified in the **`JobThreadMaxPriority`** parameter in ES-DTS.

The priority given to a job cannot exceed the maximum allocated to its group: **`ThreadPriority`** is therefore reduced by **`JobThreadMaxPriority`**.

Sheduler

The **`scheduleJobs()`** method is called periodically by the schedulemanager or when a job is submitted or completes. It reads the list of pending jobs: it examines start priorities (StartPriority tag) and the maximum number of jobs which can be active at the same time (**`MaxJobsRunning`** parameter), and decides which, and how many, jobs must be started.

It creates a new JobThread for each job to be started.

Job completion and cleanup

When the steps are finished, the JobThread class:

- cleans up the Datasets
- alerts the JobManager
- stops the thread.

The **`delete()`** Dataset method is called depending on the **`DeleteInputOnSuccess`** and **`DeleteOutputOnError`** attributes.

Once the job is finished, if there has been a fatal error (StopOnError=true) and depending on the step, you must, after correcting the error:

- either restart the job which will resume the current step, at the place the error was found,
- or pass a job with a "resume" step which will put the input Dataset back to its initial state. The history may be used by the DTU which resumes.

Caution
Giving too high a priority to a thread which lasts a long time could penalize EverSuite application threads.

6.1.2 Step

A step is executed by an instance of the **`JobStep`** class, the **`JobThread`** class executes each **`JobStep`** one after the other using the **`StepsManager`** servlet which:

- creates instances of the Dataset and DTU classes defined for this step
- calls the interface execution method DTU.execute()
- when processing is complete, saves the history in the output Dataset, including any error.
- closes the Datasets. input data can be deleted if processing was successful, depending on the **`DeleteInputOnSuccess`** attribute.

If there is an error, if it is impossible to skip the step, the job is interrupted and the output directory is cleared, depending on the **`DeleteOutputOnError`** attribute.

Only the first step has an Input tag. At every other **`JobStep`** the **`StepManager`** pass the Dataset Output by the previous **`JobStep`** as Input to the next.

The **`JobStep.rollback()`** method is executed if there is an error.

Name tag

Name used for display, in the history, etc.

StopOnError tag

If this is set to `true`, the step cannot be skipped: if there is an error the job stops.

DeleteInputOnSuccess tag

If this is set to `true` and the step completed successfully, the **`delete()`** method for the input Dataset is called.

It has no effect if the same Dataset is specified as Input and Output.

DeleteOutputOnError tag

If this is set to `true` and the step has an error, the **`delete()`** method for the output Dataset is called.

It has no effect if the same Dataset is specified as Input and Output.

Input tag

Defines the type and location of input data: the Dataset to use, the source URI (absolute path) for reading them.

The URI can be any character string as long as a class implementing ESURI is present in the es.cls.uri.driver package to handle it.

DTU tag

Contains the name of the Java class for processing the Dataset.

Output tag

Defines the type and location of data to be created by the process: the Dataset to use, the source URI (absolute path) for reading them.

The URI can be any character string as long as a class implementing ESURI is present in the es.cls.uri.driver package to handle it.

6.1.3 Dataset

Each Dataset class implements the Dataset interface and inherits the DatasetImpl (abstract) which groups some operations common to all data types.

For example, a Dataset depends on a version corresponding to the version of the format or of the data arrangement. A different sub-class can be loaded to handle a different version.

All the Dataset classes are in a subpackage, which means that any external client classes can be detected there dynamically. They can handle specific formats unknown to EverSuite.

A Dataset can be read using the DatasetReader interface, and created using the DatasetWriter interface. Instances are obtained by Dataset.getReader() and Dataset.getWriter().

Depending on whether the Dataset class is implementing a Reader and/or a Writer, it can be used as input and/or output.

Their respective methods, getElement()/getChildren() and addElement(), can obtain the input tree, and build the output tree. They communicate in a DTSElement interface.

The DTSElement structure obtained by getElement() represents a piece of input data or a tree node. The one passed to addElement() represents either a new piece of data created by the DTU, or the input data which was able to be transformed.

DatasetReader

getDataset()

getFormatVersion()

isDataReady()

validate()

getRootElement()

getChildren()

DatasetWriter

getDataset()

getFormatVersion()

addElement()

getElement()

copyData()

export()

6.1.4 DTU

The simplest DTU just copies a Dataset format into another one: its only job is to loop through all the elements in the input Dataset and add them to the output Dataset.

It must begin by using the input DatasetReader methods, isDataReady() then validate().

The same Dataset may be specified as input and output, if you wish to enrich or transform data, without copying them. In that case, a "Rollback" is impossible on this step, and is pushed back to the previous step.

The DTU obtains Dataset data by reading a tree recursively. The tree is not passed all at once to the DTU, but element by element, and each one can have a list of children: getRootElement() then getChildren().

While reading the tree, it must call its isCancelled() method and interrupt processing if the response is ‘true’, as its cancel() method may be called from another thread.

Interface

getName()

getArgs()

execute()

cancel()

isCancelled()

6.1.5 DTSElement

In the Dataset, whatever its internal storage structure, data can be represented in a tree. Each element in the tree may be made up of:

- a history
- metadata and references
- attributes
- certificates
- signatures
- import rules

Example of an XML file:

```
<element>
   <!-- ------------------- -->
   <attributes>
    <attribute>
     <name>...</name>
     <values>
       <value>...</value>
       <value>...</value>
     </values>
    </attribute>
    ...
   </attributes>
   <!-- ------------------- -->
   <metadata>
    <data>
     <name>...</name>
     <values>
       <value>...</value>
       <value>...</value>
     </values>
    </data>
    ...
   </metadata>
   <!-- ------------------- -->
   <history>
    <events>
     <event>
      <date>2007-05-29 15:02:02</date>
      <attributes>
       <attribute>
         <name>...</name>
         <value>...</value>
       </attribute>
       ...
      </attributes>
     </event>
    </events>
   </history>
   <!-- ------------------- -->
   <policy>
    <rule>
     <name>...</name>
     <values>
       <value>...</value>
       <value>...</value>
     </values>
    </rule>
    ...
   </policy>
   <!-- ------------------- -->
   <signatures>
    <signature>
     <date>2007-05-29 15:02:03</date>
     <signer>...</signer>
     <algorithm>...</algorithm>
     <data>...</data>
     <list>
       <path>...</path>
       <path>...</path>
     </list>
    </signature>
    ...
   </signatures>
   <!-- ------------------- -->
   <certificates>
    <certificate>
     <alias>…</alias>
     <values>
       <certpath>...</certpath>
     </values>
     ...
    </certificate>
    ...
   </certificates>
   <!-- ------------------- -->
   <children>
    <child>
```

```
        <id>4489754</id>
        <path>...</path>
      </child>
      ...
    </children>
</element>
```

History

The history is restricted to the element itself: child element histories are not included.

The history of a base element contains the history of all the processes applied to the Dataset whatever format transformations it may have undergone: each DTU at each step (and in each job) adds its information to it.

each event in an element's history must contain:

- its date
- its type (start of processing, end, etc.)
- information about its type (e.g.: result code or message for end of processing)

Metadata

This is a list of information on the data represented by the element. The list is just for the element, not its children.

It may contain indications coming from two different sources:

- data extracts: for example properties of an MS Office document
- supplied by the creator of the Dataset: for example by an XML file.

The Dataset has to retrieve and group together in a single capsule all an element's metadata.

For a RECORD element, the metadata represent the columns in the target table, and references to other records may be associated with them.

Attributes

These characterize the element itself: type (folder, file, mailbox, email, page, etc.), name, format of associated data, location, etc.

Unique identifier

It may be unique only within a dataset. It is created by the Dataset and its format is decided by the Dataset. This may be a number, a relative file path, etc.

It is passed by the DTU to the output Dataset when the element is added. The output Dataset stores the identifier in its **`fromid`** attribute. It will create its own identifier at the time which suits it, for example only when it receives a request for the element. It can also give the same identifier as **`fromid`** if it can be sure of its uniqueness.

Source identifier

This contains the unique identifier which was used to create this element, for example the unique identifier of the input Dataset, passed by addElement().

Object name

For example, a file name or a page number.

Element type

Indicates if it is a file, a record, an email, or a folder, etc.

Version

The version is in the root element.xml file:

```
<version>
        <major>…</major>
        <minor>…</minor>
        <patch>…</patch>
</version>
```

Certificates

Certificates are either stored in a sub-directory or exported directly in the XML. They are listed in the root element which details all the certificates used for signing or encrypting elements. Each signer's alias must therefore be unique. It is however possible to store several certificates per signer.

```
<certificates>
 <certificate>
  <alias>…</alias>
  <values>
    <certpath>...</certpath>
  </values>
  ...
 </certificate>
 ...
</certificates>
```

6.2 COMMANDS AND PARAMETERS

The ES-Capture service is based on a business-oriented API, which uses internal APIs, such as `JobBuilder` and `JobManager`.

6.2.1 Create a job

Basic concepts

An instance of the **JobBuilder** class is created by the Capture API (**Action**=*Build*) and kept in memory for the current user session. This instance keeps the definition of the job being built (instance of DefJob) and, thanks to its API, can:

- create a new job step by step
- resume a job created earlier
- save a job to memory
- remove a job from memory
- submit a job to the **JobManager** for execution.

A created job is just a basic XML definition, which does not need to be complete. A job definition is not validated until it is submitted.

The **Capture** servlet governs all commands for building a job, with the **Action**=*Build* and **Cmd** parameters:

```
../Capture?Action=Build&Cmd=...
```

Build commands create and enrich a job definition instance in memory (**DefJob**). They return their result as XML intended for web clients, to refresh their display.

The right to use commands is defined in **Building jobs** security parameters in the Administration > Security > Service security menu, service: capture.

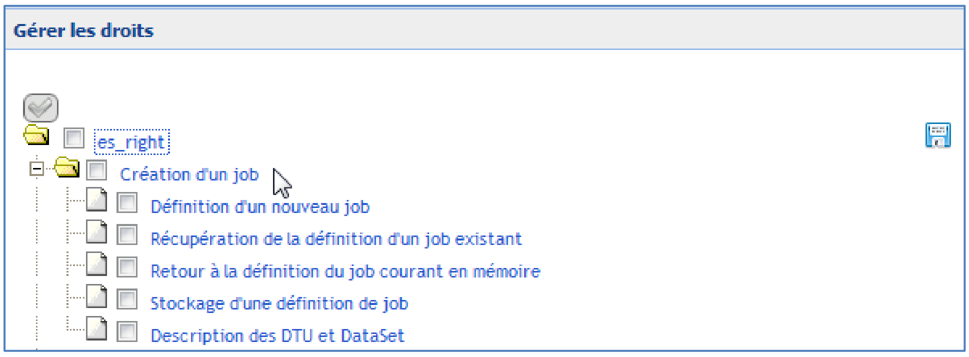

Gérer les droits
es_right
Création d'un job
Définition d'un nouveau job
Récupération de la définition d'un job existant
Retour à la définition du job courant en mémoire
Stockage d'une définition de job
Description des DTU et DataSet

Figure 67: Rights to use job build commands

List of commands for the Cmd parameter

GetDTUs – Describe DTUs and Datasets

GetCurrentJob – Return to the definition of the current job in memory

GetNewJob – Define a new job
Confirmation: *true*/*false* depending on user confirmation

GetJob – Retrieve definition of an existing job

Name: name of the stored job
Confirmation: `true`/`false` depending on user confirmation

PutJob – Store a job definition
Confirmation: `true`/`false` depending on user confirmation

ChangeDatasetType – Change the format of data to read/write
StepIndex: identifier (index) of the step to modify
InOut: `in`/`out` whether the Dataset is the source or the target
Name: name of the new Dataset type
ClassName: name of the new class to use

ChangeDataset – Change an attribute of the Dataset
StepIndex: identifier (index) of the step to modify
InOut: `in`/`out` whether the Dataset is the source or the target
BaseURI: URI of the database directory

ChangeDTUType – Change the type of processing to apply
StepIndex: identifier (index) of the step to modify
Name: name of the new type of processing
ClassName: name of the new DTU class to use

ChangeDTU - Change process attributes
StepIndex: identifier (index) of the step to modify
DefTableName: name of the default table where elements are imported

ChangeJob - Change the attributes of the job being edited
Name: new job name
StartPriority: priority for submission before execution (integer)
ThreadPriority: priority of the Java execution thread (0-5-10)

ChangeStep - Change attributes of a step
StepIndex: identifier (index) of the step to modify
Name: new step name
StopOnError: `true`/`false` fatal error
DeleteInputOnSuccess: `true`/`false` delete data if ok
DeleteOutputOnError: `true`/`false` delete result if error

InsertStep – Insert a step between two components
StepIndex: identifier (index) of the step where a new step is to be inserted
Location: the components of this step between which to insert the new step

DeleteStep – Delete two components starting at a position
StepIndex: identifier (index) of the step to delete
Location: the component of this step from which to start deleting

6.2.2 Execute a job

Basic concepts

A **JobManager** class, created by the Capture API (**Action**=*Run*), presents an API which can:

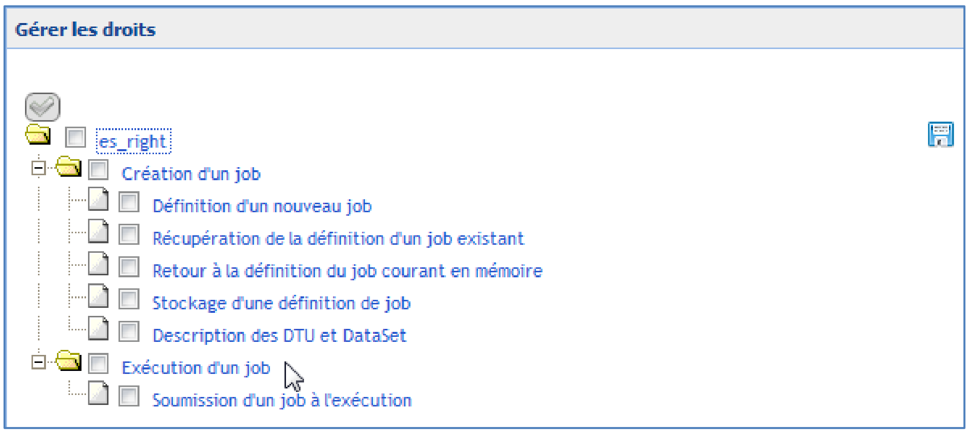
- receive the jobs to execute submitted by a requestor (e.g.: the **APIDispatcher** class) and keep the list in memory: **submit()**
- start jobs according to a configurable policy, with priority management: **scheduleJobs()**
- monitor their execution
- interrupt their execution.

The instance of **JobManager** is unique and obtained by **getInstance()**.

The **Capture** servlet governs all commands for executing a job, with the **Action**=***run*** and ***Cmd*** parameters:

```
../Capture?Action=Run&Cmd=...
```

The right to use commands is defined in **Running jobs security parameters in the** Administration > Security > Service security menu, service: capture.


Gérer les droits

- es_right
  - Création d'un job
    - Définition d'un nouveau job
    - Récupération de la définition d'un job existant
    - Retour à la définition du job courant en mémoire
    - Stockage d'une définition de job
    - Description des DTU et DataSet
  - Exécution d'un job
    - Soumission d'un job à l'exécution

Figure 68: Rights to use job execution commands

List of commands for the Cmd parameter

Submit – Submit a job for execution
Name: name of a stored job, or nothing for the job being edited

ScheduleJobs – Starts jobs in the queue depending on setup

ListWaitQueue – List jobs in the queue

ListActiveQueue - List jobs being executed

ListEndQueue – List completed jobs, by period

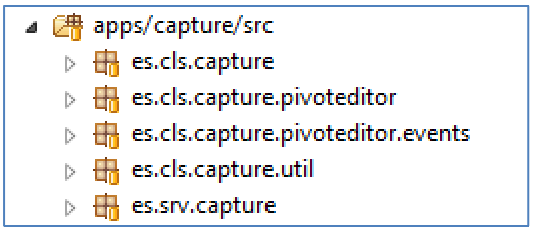
PastDay: 0=today, 1=yesterday, 2=day before yesterday, …, 7=all the rest

6.3 JAVA & HTML SOURCES

6.3.1 ES-Capture

Package

The ES-Capture package is in the `apps/capture/src/` folder:



```
apps/capture/src
    es.cls.capture
    es.cls.capture.pivoteditor
    es.cls.capture.pivoteditor.events
    es.cls.capture.util
    es.srv.capture
```

Figure 69: ES-Capture package

Forms

The ES-Capture forms are in the `apps/capture/templates/`folder.

Images

ES-Capture images are in the `apps/capture/images/`folder.

Scripts

ES-Capture scripts are in the `apps/capture/scripts/`folder.

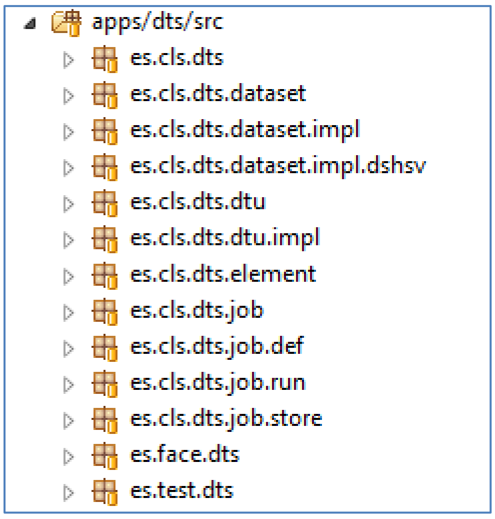
CSS

ES-Capture css are in the `apps/capture/css/`css.

6.3.2 ES-DTS

Package

The ES-DTS package is in the `apps/dts/src/` folder:



apps/dts/src
- es.cls.dts
- es.cls.dts.dataset
- es.cls.dts.dataset.impl
- es.cls.dts.dataset.impl.dshsv
- es.cls.dts.dtu
- es.cls.dts.dtu.impl
- es.cls.dts.element
- es.cls.dts.job
- es.cls.dts.job.def
- es.cls.dts.job.run
- es.cls.dts.job.store
- es.face.dts
- es.test.dts

Figure 70: ES-DTS package

TABLE OF ILLUSTRATIONS