DELIVERABLE

**Project Acronym:** FLAVIUS

**Grant Agreement number:** ICT-PSP-250528

**Project Title:** Foreign LAnguage Versions of Internet and User generated Sites

# D2.4 Platform able to translate websites automatically

**Revision: 2.2**

**Authors:**

Joël Benchitrit (Softissimo)

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

Revision History

| Revision | Date | Author | Organization | Description |
|---|---|---|---|---|
| 0.1 | March, 14th | Christophe Brun-Franc | Softissimo | Draft version |
| 1.0 | March, 31st | Christophe Brun-Franc | Softissimo | Version sent to EC |
| 2.0 | December, 7th | Joel Benchitrit | Softissimo | New version of D2.4 |
| 2.1 | December 26th | Elsa Monségur | Softissimo | Revised version |
| 2.2 | December 26th | Joel Benchitrit | Softissimo | Final version sent to EC |

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

## Table of contents

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

## 1. Document References

Deliverable D2.2 Website specification document v2

Deliverable D2.5 Interface to create personal dictionaries

Deliverable D4.4 Translation Memory Module able to provide translation

## 2. Introduction

The objective of this deliverable is to describe the technical implementation aspects of the first version of the FLAVIUS platform delivered at M18. This deliverable is mainly based on the Website functional specifications from D2.2. For a general overview of the architecture, please refer to D2.2.

FLAVIUS, Foreign LAnguage Versions of Internet and User-generated Sites, aims at bridging the language gap between content publishers and users by providing an online platform accessible to websites owners that will enable them to generate multilingual versions of their site, quickly, easily and efficiently in as many languages as they want.

FLAVIUS is a European project under the Competitiveness and Innovation framework Programme (call identifier: CIP-ICT-PSP-2009-3)

# 3. Translation workflow overview

The workflow is implemented as described in D2.2:

- without the post-edition module (it will be delivered in D2.6)
- with the spell-checking option only in the “File” scenario (for the “URL” scenario it will be delivered in D2.9)

# 4. User Account and authentication

## 4.1 Overview

The authentication is implemented as described in D2.2:

- Without the reviewer management (it will be delivered in D2.6 with post-edition)

## 4.2 User account and authentication

### 4.2.1 Roles

The security of the FLAVIUS platform is based on the role-based security and user authentication model provided by .Net Framework.

The role-based security model is a means of implementing an authorization mechanism that eases the work of the Administrator and allows handling the notion of group of users.

The users could belong to different roles:

- Guest (Anonymous)
- Basic (Authenticated user with basic rights)
- Premium (Authenticated user with advanced rights)
- Administrator (Admin role)

A “Guest” user can go to the Flavius home page, create an account and browse published websites.

A “Basic” user can create a translation job with some limitations in terms of number and size of files to be translated. He/she also can view his/her completed translation jobs and update his/her user account.

A “Premium” user is a “Basic” user that can create a translation job with fewer limitations.

## 4.2.2 Properties management

A set of properties can be defined by an administrator in Flavius:

| Property name | Description | Default Value |
|---|---|---|
| RET_pathRepository | Path for processing translation and storing data | C:\Flavius\Data\Job |
| PUB_pathPublishing | Path for publishing websites | C:\Flavius\Publishing |
| RET_SizeMaxUrl | Max size allowed for retrieving websites data | 500Ko |
| RET_FileCountMaxUrl | Max file count allowed for retrieving websites data | 2500 |
| RET_LevelCrawlUrl | Max level allowed for crawling websites | 1 |
| TRS_DefaultWS | Default Web service for non granted users | * |
| TRS_DefaultIdEngine | Default Engine Id for non granted users | * |
| TRS_pathTranslation | Path for processing translation | C:\Flavius\Data\Processing\Translation |
| GEN_FlagCountDisplay | Maximum flag count to display in list | 3 |
| GEN_PathParameter | Parameters Path | C:\Flavius\Data\Config |
| EXT_PathParameterXmlTemplate | Path to locate the xml template file for Data extraction | C:\Flavius\Data\Config\XMLTemplate |
| RET_MaxSizeZip | Max size allowed for retrieving zip files | 500ko |
| RET_MaxSizeXml | Max size allowed for retrieving xml files | 30000ko |
| PUB_urlPublishing | URL for publishing | http://flaviuspub.reverso.net/ |
| GEN_EmailFrom | Sender of email | flavius@reverso.net |
| TM_idServer | Translation Memory server | 5ae747ff-c3cc-4e6f-81ad-3aaa3ef32084 |
| TM_usrLogin | Translation Memory login | default |
| TM_usrPassword | Translation Memory password | |
| TM_LevelSimiliraty | Translation Memory similarity | 90 |
| RET_Cumulative_MaxSizeUrl | Cumulative quota for URL scenario | 30000ko |
| RET_pathTMXRepository | Path for imported translation memory | C:\Flavius\Data\TM |
| EXT_PathParameterTmxDtd | Path to locate the TMX Dtd file for TMX validation | C:\Flavius\Data\Config\TM |
| EXT_PathParameterTmxImportTemplate | Path to locate the TMX import template for TMX import | C:\Flavius\Data\Config\Across |
| TRS_pathRevision | Path for processing spell check | C:\Flavius\Data\Processing\Revision |
| TRS_pathWget | Path to Wget application | C:\Program Files\GnuWin32\bin |
| EXT_PathParameterConfigFlavius | Path to FLAVIUS SMT configuration file | C:\Flavius\Data\Config\MT\ConfigFlavius.xml |
| TM_DBConnectionString | Connection string to DB TM Server | ******* |

| | | |
|---|---|---|
| TM_DBMain | Main DB name | across |
| TM_DBTank | Optional DB name | ctank_across |
| RET_pathEngineDictionaryRepository | Path for imported dictionary | C:\Flavius\Data\EngineDictionary |
| TM_MaxEntriesPerDirection | TM max entries per direction | 400 |
| TM_MaxSizeEntries | TM max size entries | 1000 |
| ED_MaxEntriesPerDirection | Dictionary max entries per direction | 400 |
| ED_MaxSizeEntries | Dictionary max size entry | 120 |
| ED_MaxSizeFile | Dictionary max size file | 50000ko |
| TM_MaxSizeFile | TMX max size file | 500ko |
| RET_Cumulative_MaxSizeXML | Cumulative quota for XML scenario | 50000ko |
| RET_MaxSizePerFileXML | Max size allowed per XML file | 30000ko |
| PUB_mailReportAbuse | Email for reporting an abuse | flavius@reverso.net |
| SPC_ConfidenceLevel | Confidence level for spell checking | 90 |
| SPC_idServer | Spelling checker server | http://api.daedalus.es/flavius |
| SPC_usrLogin | Spelling checker login | ******** |
| RET_SegmentBlockCharCount | Max char to send in one block for the translation | 2000 |
| TRS_ShowToolTip | Show source text in tooltip | 1 |
| PUB_pathPublishingArchive | Path for archiving published website | C:\Flavius\Data\PublishingArchive |
| GEN_availableCultures | Flavius translation available | fr:français;en:english |
| TRS_SourceCodeToolTip | Path to Reverso Tooltip source code | C:\Flavius\Data\Config\MT\tooltip.txt |
| TRS_SourceCodeGoogle | Path to Google Analytics source code | C:\Flavius\Data\Config\MT\google.txt |
| TRS_SourceCodeToolBar | Path to Toolbar source code | C:\Flavius\data\Config\MT\toolbar.txt |

These properties can be used at three levels:

- Flavius level (this is the default properties)
- Role level (the default properties are overridden by the role ones)
- User level (The role properties are overridden by the user ones)

These properties could be used for example to put some quota on the size of crawled data (RET_SizeMaxUrl). This limitation could be set for the application as a default value, for a specific role (ex: Premium), or for a specific user.

As a matter of fact, users inherit the properties and rights of their role. For instance, the limitation of the platform use is defined for each role.

The “Administrator” users are allowed to change the value of these properties at each level.

Here is the description of the retrieval process:

- When a global property is required:
    - Get the property value at Flavius level
- When a role property is required:
    - Get the property value of the role
    - If the role property does not exist get the property at Flavius level
- When a user property is required:
    - Get the property value for this user
    - If the user property does not exist, get the property at Role level
    - If the role property does not exist, get the property at Flavius level

# 5. Job configuration

## 5.1 Overview

The job configuration for “URL” and “File” scenario is implemented as described in D2.2:

- Without the PHP format for “File scenario” (it will be delivered in D2.9)
- Without the XLIFF format for “File scenario” (it will be delivered in D2.9)

## 5.2 Website Crawling (“URL” scenario)

After a “URL” job is created, the website is crawled from the URL entry point provided by the user. The website crawling is done by WGET tool: (http://gnuwin32.sourceforge.net/packages/wget.htm). This crawling tool is able to manage most of websites. Some remaining issues still exist on some websites (see “Crawling issues” section).

According to the user access rights, the crawling is restricted:

- By the depth level
- By the total website size to be crawled

The WGET tool does not accept a file count quota. It is therefore not possible to set easily a file count limitation.

These constraints can be defined through Flavius properties at any level (Global, Role, and User).

We also set a list of parameters for WGET:

- Convert Links = true: The links of web pages are converted so to point to local crawled pages instead of to original ones.

- Reject list = gif, jpg, jpeg, swf, png: This list indicates the file extensions that are not to be crawled (images, Flash, etc.). The crawled webpages containing references to these files are pointing to the original files.
- UserAgent: The Firefox user agent is defined to allow WGET to work with the greatest number of websites.
- Quota: The maximum file size to crawl (by default, 3000ko)
- RecLevel: The depth level to crawl (by default, 1, which means first page with its external links)

The global crawling process can be summarized in the following steps:

- First, the user configures and creates a “URL” job by clicking on the “create a job” button.
- A new WGET task is launched in the Post Processing Windows Service (See “Processing of source texts” section). The crawled files are saved on the job directory.
- Once the crawling is completed, the log file generated by WGET (wget.log) is parsed, and page information is extracted and saved in database:
    - Original crawled URL
    - File path of the crawled page
    - Size of the webpage in bytes
    - Extension File Type extracted from the Content Type of the original HTTP header page (Html, CSS, JavaScript)

### 5.2.1 Crawling issues

#### *JavaScript links*

The main remaining issues of the crawling step are related to JavaScript. Sometimes, links inside HTML pages are dynamically generated by JavaScript, and cannot be found by WGET tool. These links cannot be retrieved and it generates some dead links.

A first approach to manage them is to detect some JavaScript patterns with regular expressions, inside some specific HTML tags and attributes (ex: onclick):

- div onclick="document.location.href='http://foo.com/'">
- <tr onclick="myfunction('index.html')"><a href="#" onclick="myfunction()">new page</a>
- <a href="javascript:void(0)" onclick="window.open ('welcome.html')">open new window</a>

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

Then, we could try to rebuild these links and crawl the associated pages. After some tests on a set of 20 websites, it appears that it will cover most of the problems encountered, but it will not address all the cases (highly dynamic AJAX sites, etc.).

### *Page redirection*

Sometimes, websites make some page redirection, using JavaScript, or using HTTP headers. During the crawling step, we want to access to the final page, not to the intermediate page. For the JavaScript redirection, it could be managed by the solution above (See “JavaScript links”). For HTTP header redirection, it is managed by the crawling tool by detecting Redirection HTTP header (HTTP 301 code, etc.).

### *Ajax calls*

Some websites (like http://www.lemonde.fr/) get their content from AJAX calls (in asynchronous calls in JavaScript).

During the crawling, we do not execute the JavaScript code, so this dynamic content is never retrieved and translated. Moreover, if the URL called by the AJAX scripts is a relative one, it will cause a JavaScript error (The URL that deliver dynamic content called by AJAX is not present on the Flavius Publishing Server). If the URL is an absolute URL, the content will be retrieved, but not translated (it will come from the original website).

One solution could be to interpret the JavaScript on crawled page (with a JavaScript Interpreter) and to intercept the Ajax calls. As it is a whole research project, it is out of the Flavius scope and we will not address it for now.

### *Global dead links management*

Another solution could be to manage failed requests (JavaScript, AJAX, Webpages, etc.) once the website is published. When a user is navigating, if we get a 404 Not Found response from the Publishing Flavius Server, we could redirect the request to the original server, get the file, translate it if needed in real time (for html files), and publish it.

Advantages:

- We will not have some “dead links” anymore (JavaScript, HTML pages, etc.)

Disadvantages:

- We need to manage quotas more deeply with these new files.

- We need to translate the new file in “real time” (The Flavius Queue System cannot be used)
- We need to re-integrate the new file in the workflow (for post edition, etc.)
- We need to ensure that the “404 code” is a “real not crawled web page”, and not a random URL typed by the user (We could check it on the original server)

## 5.3 XML Processing (“File” scenario)

XML is a universal data format, used by the largest part of developers to process and exchange their data. Most of the content management systems (CMS like WordPress, Drupal, SharePoint, etc.) have a way (native or with external plugins) to export their data in XML.

In addition, some web technologies use XML format to localize their content (ex: ASP.NET), and to manage their resources. XML also became a standard to store documents (see DOCX and ODT format). XML processing in Flavius will permit to meet a large part of user needs.

### 5.3.1 Use of XPATH technology

Flavius Platform allows users to translate any XML files.

To separate useful from useless data, the system uses a dedicated template to define which XML nodes need to be translated. The filter is based on the XPATH standard (http://www.w3.org/TR/xpath/). The data contained in a node matched by the XPATH will be extracted to be translated. The data contained in the other nodes will be ignored.

Sample of a XPATH template configuration file used in Flavius

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<!--
        transType values:
                0 - do not translate
                1 - replace and if not found, do not translate it
                2 - translate
                3 - replace and if not found, translate it
        defaultBehavior - unrefferencedTags values:
                true - all unrefferenced tags will be translated by default
                false - all unrefferenced tags will NOT be translated by default
-->

<configuration>
        <defaultBehavior>
                <unrefferencedTags translate='false'/>
        </defaultBehavior>
        <config_item>
                <xpath value="/rss/channel/title"/>
                <transType value="2"/>
        </config_item>
</configuration>
```

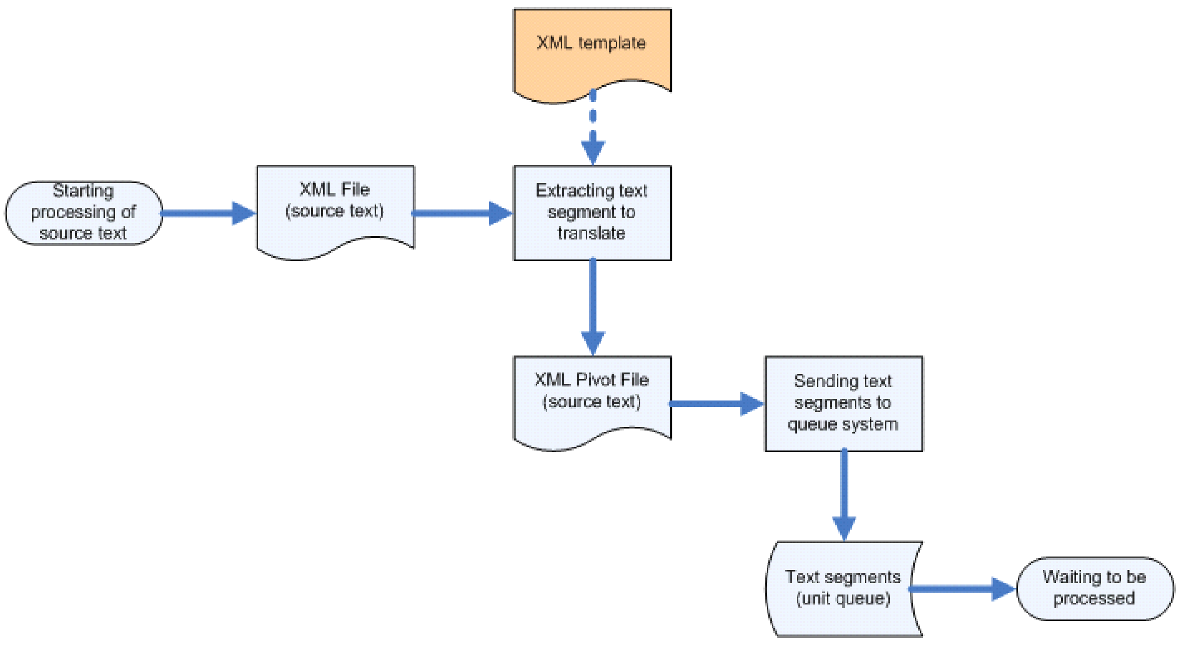
In this example, we do not want to translate any tags (default behavior), except the ones that match the XPATH "/rss/channel/title".

In the XML file below, the following section (in red) will be translated:

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rss>
        <channel>
                <title>Title of my article</title>
                <author>John Doe</author>
        </channel>
</rss>
```

### 5.3.2 Flavius XML Parser

At first, we check if the XML provided by the Flavius user is syntactically correct. Then, the XML Parser extracts the translatable content from the XML file (with the XPATH technology), divides it in segments, and stores it in a sequential manner. These segments are processed by the different engines (Speller, Translation Memory, and Machine Translation) and the result is stored in the same order on the disk. Then, the original XML file is parsed a second time to be able to replace the initial segments with the new ones, and to rebuild the final XML file.


XML template
Starting processing of source text
XML File (source text)
Extracting text segment to translate
XML Pivot File (source text)
Sending text segments to queue system
Text segments (unit queue)
Waiting to be processed

**<u>Information</u>**

The XML can contain HTML segments (from CMS for example) with encoded tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <node1>&lt;div&gt; Text to translate &amp;  &lt;/div&gt; </node1>
</root>
```

### 5.3.3 Existing template configuration files

Currently, the templates cannot be uploaded by the user, so he has to select an existing template. For now, the following templates have been implemented for Flavius:

- **«All Data» Template**

All content inside XML tags is considered valid for the translation process. This template is used when the user provides an XML file with a proprietary XML format containing only data ready for translation.

- **RESX template**

This template allows translating a .NET resource file. In ASP.NET development, a developer who wants to localize easily his website should separate his work in two parts:

- The source code containing the website behavior,
- The resources files containing all the localizable information like labels or messages (RESX file).

In this case, the user only needs to translate the RESX files to generate a new translated website.

Using XPATH template configuration files, adding the support of a new XML format is very easy: it could be done just with configuration.

<u>Example of a RESX .Net file</u>

```
<?xml version="1.0" encoding="utf-8"?>
<root>
    <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="btn_createnewuser_text" xml:space="preserve">
    <value>Create user</value>
  </data>
  <data name="btn_login_text" xml:space="preserve">
    <value>login</value>
  </data>
  <data name="lbl_createnewuser_details_country_text" xml:space="preserve">
    <value>Country</value>
  </data>
</root>
```

XPath Template used for Resx:

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<configuration type="WordPress">
        <defaultBehavior>
                <unrefferencedTags translate='false'/>
        </defaultBehavior>
        <config_item>
                <xpath value="/root/data/value"/>
                <transType value="2"/>
        </config_item>
</configuration>
```

- **WordPress template**

  This template allows translating a WordPress export XML file. The translated XML could be imported in a new WordPress instance.

<u>XPath Template used for WordPress:</u>

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<configuration type="WordPress">
        <defaultBehavior>
                <unrefferencedTags translate='false'/>
        </defaultBehavior>
        <config_item><xpath value="/rss/channel/title"/></config_item>
        <config_item>   <xpath value="/rss/channel/wp:category/wp:cat_name"/></config_item>
        <config_item>   <xpath value="/rss/channel/wp:tag/wp:tag_name"/></config_item>
        <config_item>   <xpath value="/rss/channel/description"/></config_item>
        <config_item>   <xpath value="/rss/channel/item/title"/></config_item>
        <config_item>   <xpath value="/rss/channel/item/category"/></config_item>
        <config_item>   <xpath value="/rss/channel/item/wp:post_name"/> </config_item>
        <config_item><xpath value="/rss/channel/item/wp:comment/wp:comment_content"/>
        </config_item>
        <config_item><xpath value="/rss/channel/item/content:encoded"/></config_item>
</configuration>
```

## 5.4 Processing of source texts

### 5.4.1 General Overview

The processing of each specific operation (translation, text correction, etc.) is handled by a dedicated Windows Service. For a more detailed description, please refer to D2.2 / section “Flavius Back Office”.

Currently, three Windows Services are deployed in Flavius platform:

- A Windows Service which hosts the Pre and Post Processing steps : website crawling, extracting and parsing XML files
- A Windows Service which sends extracted segments to the Spell Checking API (See Flavius Queue System below)
- A Windows Service which sends extracted segments to the translation API (See Flavius Queue System below)

### 5.4.2 Flavius Queue System

The Queue system mainly works with the notion of “job”. A job represents the process of translating (or spell checking, etc.) one or several files in a sequential manner and is formed internally of several important steps:

1- Iterate the folder to be translated and push all the file names and paths that meet the selection criteria (chosen in the interface) to a dedicated queue (described later)

2- De-queue one file at a time (sequential file processing) and process it:

  - According to the type of file (e.g. .xml), send it for pre-processing. For xml files, the pre-processing is actually the process of breaking down the respective xml file into several smaller xml part called “units”. For the rest of the file types (e.g. HTML), a unit is considered to be the entire file.
  - Push each unit into a dedicated queue (described later) until all units that form the main file, are there

- De-queue several units at a time (concurrent unit processing) and send them for processing (translation, spell checking, etc.) – the number of units processed at a time is configurable (described later in configuration files); the processing of each unit is done in a separate thread (see thread management) and is formed by a call to the corresponding WebService (Daedalus API, LW API, etc.).
- Push the results from the previous step into a dedicated queue (described later)

3- Monitor the result queue and detect when all the units forming the file being processed are there
   - Build the processed file; since the unit processing is done in a concurrent manner, the units need to be ordered correctly (for non XML files, no ordering is necessary because they only have 1 unit)
   - Save it with a different name

4- Continue from Step 2 until all files have been processed

5- Build a status report for the processed job

Flavius Queue System uses an internal queuing system for managing all aspects related with a job. The system is integrated based on Microsoft Message Queuing version 3.0 (higher versions can be used).

The system uses three types of queues based on the action they serve:

- Request queues
- Response queue
- Context queue

There are two request queues:

- **File queue**
  - used for storing all files to be processed in a job
  - only allowsfiles from a single job at a time (Two jobs cannot run at the same time)
  - it stores only the file name and path of the file to be processed and not the entire content of the file
- **Unit queue**
  - used for storing all the units that form a file – for HTML files, a unit is equivalent to the entire file; for XML files, a unit is a small part of the file

- it stores information related to the unit to be processed (unit id, job id, file name, etc.)

Both request queues work in a sequential manner, meaning that first the **File queue** is filled in with the job files, then the first file is extracted and converted into units, units which are pushed into the **Unit queue**.

The results of the process are stored in a single **Result queue**. The **Result queue** becomes active as soon as the first unit from the Unit queue has been processed, and continues to be active until all the units have been processed. While the request queues are populated in a sequential ordered manner, the **Result queue** is exactly the opposite, meaning that results are pushed concurrently (because units are processed concurrently) and thus the order is not maintained.

Besides the request and response queues, there is also a Context queue. The context queue is used for storing context related information like the current job in progress, file in progress, total number of units of the file in progress and number of processed units from the file in progress. All this information is used for allowing job restoration in case of failure: if the application crashes from different reasons (internal crash or external causes like power failure etc.), the first time you start the application, the context queue will be used to restore the job in progress and continue from where it crashed.

All the queues are private transactional queues and can only be accessed from local computer (computer on which the Windows Service has been deployed). All the operations related to queues are done in a transactional manner, meaning that no information is lost if the system encounters a failure.

Neither of the queues uses the internal prioritization of messages, meaning that all messages share the same priority within the queue.

## Thread Management

The translation of each unit is done on a separate thread. This allows for concurrent processing of units, and thus improves the overall time needed to finish a job.

Thread management is done using an internal pooling system (Dispatcher) based on manual reset events class from .NET Framework (Alert one or several waiting threads that an event occurred). The feeding of translation threads is done using a balancer that automatically cycles all the available threads avoiding the overload of a single thread (similar to Round Robin mechanism).

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

The reason for which a custom pooling mechanism was used instead of the internal .NET thread pool was to allow a more in depth control over the entire pool of threads; some of the advantages gained by building the custom pool are:

- Allow stopping any thread at any time during the process (the thread is not aborted – because it would cause a ThreadAbortedException which would disrupt the entire process – but it is allowed to finish its work and then it is blocked from receiving new work)
- Allow freeing resources at any time without relying on the .NET suppression mechanism – useful when the user aborts the job

Once a unit has been processed, it is placed in the **Result queue**. In case the status of the unit is failed (failed because of internal processing causes from Spell Checking API, or SMT API), the unit is sent back for reprocessing – it is pushed at the end of the **Unit queue** and will be reprocessed at the end. The total number of times that a unit can be reprocessed is configurable in the configuration file.

Besides the threads used for units processing, the application uses several new threads:

- File manager thread
  - Used to push/de-queue files for current job into the File queue
  - Used to preprocess the file in progress (break down into units)
  - Used to send the units into the **Unit queue**
  - Used to update the progress display in GUI (in the Flavius Website Database)
- Unit manager thread
  - Used to de-queue units and send them for processing
  - This is the thread where the Dispatcher resides (custom thread pool)
    - Within each translation thread (managed by the Dispatcher) the processed unit is pushed into the **Result Queue**
- Result manager thread
  - Used to monitor the results queue and de-queue all the units once the file in progress is completely processed
  - Used to rebuild the processed file from the processed units
- GUI progress display thread
  - Used to display job progress information/status

After a file has been processed, it is automatically rebuilt by the Result manager thread. All units will be assembled back in the same order as in the original file, and this includes also the failed units; the failed units will be placed back into the output file without being altered (same content from original file).

At the end of a file processing, all failed units (if any) will be reprocessed if the reprocessing counter is higher than 1 in configuration file.

| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
|---|---|---|
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

The units that failed completely will be placed also in a separate file that can be used for post analysis and eventually re-processing.

## 5.5 Encoding management

In this version, the Flavius platform is implemented to be compliant only with UTF-8 encoding files. LW and Daedalus APIs can only manage UTF-8 for now.

For the “File” scenario, users are warned if their file sent is not an UTF-8 encoding file (or is not detected as an UTF-8 encoding file), but in any case, the job is processed.
To detect the file encoding, we check the following two points:

- The XML header (for xml files) declares that the file is encoded in UTF-8:
  <?xml version="1.0" encoding="UTF-8"?>
- The UTF-8 BOM exists at the beginning of the file

Otherwise, the file is not detected as an UTF-8 encoding file.

For “URL” scenario, every encoding is accepted in Flavius, but crawled files are sent to the LW API and Daedalus API as UTF-8.

## 5.6 SQL database and file systems

The data are stored both in a SQL database and in the File System.

The SQL database is used to stored configuration data and workflow steps: job description, job workflow, statistics, user account, etc. The Flavius database is shared by the Flavius Website and the Windows Services.

The File System is used to store source files (crawled website, xml, etc.), intermediate results (spell checked files, etc.) and final result (translated files, archives, etc.)

Each process creates its own subfolder (in the corresponding job folder) to store files:

- Source : contains the source files
- Revised : contains the spell-checked files
- Translated : contains the translated files

Finally, the ZIP archive that contains all these files is created and saved in the job folder.

# 6. Grammar and spell-checking

## 6.1 Overview

The Spell Checking is implemented as described in D2.2:

- Only for “file scenario” (the spell-checking option for URL scenario will be delivered in D2.9)
- Some minor interface design

## 6.2 Use of Daedalus API for Flavius (“File” scenario)

The Flavius Spell checking Process is done through the Daedalus API. If spell checking is enabled in the “Job Creation” panel, the process is launched using the Flavius Queue System (See Processing of Source text section). The spell-checking option is available on the interface only for the following source languages:

- French
- English
- Spanish
- Italian

The Daedalus API (See Deliverable D3.0 Specification document for text correction adaptation) is called on each segment, with the following parameters:

- Input format = txt: HTML or plain text segments
- Confidence score = 90: Minimal confidence score for auto replacement
- Output format = auto: The segment is auto-corrected by the API, according to the confidence score and the result (auto-corrected segment + error detail) is sent in a XML stream.

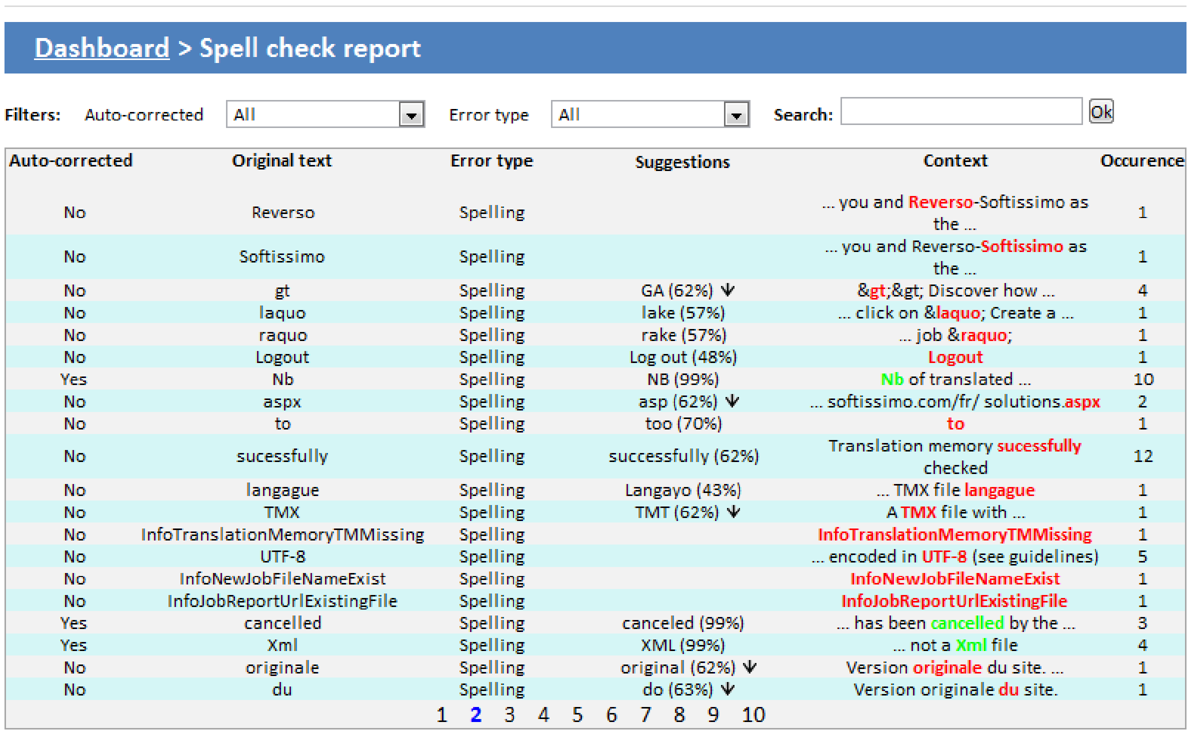
During spell-checking of a XML file, we serialize (in XML for now) each XML API result (one result by segment) in a unique file with the same name and a “.speller” extension.

Here is an example of a serialized “speller” file:

```
<SpellCheck xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/Softissimo.Module.SpellCheck.Daedalus">
  <SpellCheckResults>
    <SpellCheckResult>
<SpellCheckResult>
      <AutoCorrectedSegment>&lt;p f="bl"&gt;Webpages list&lt;/p&gt;</AutoCorrectedSegment>
      <OriginalSegment>&lt;p f="bl"&gt;Webpages list&lt;/p&gt;</OriginalSegment>
      <SpellCheckUnits>
        <SpellCheckUnit>
          <Changed>false</Changed>
          <EndOffset>18</EndOffset>
          <ErrorMessage>Possible spelling error.</ErrorMessage>
          <ErrorType>Spelling</ErrorType>
          <OriginalText>Webpages</OriginalText>
          <SpellCheckSuggestions>
            <SpellCheckSuggestion>
              <Confidence>48</Confidence>
              <SuggestedText>Web pages</SuggestedText>
            </SpellCheckSuggestion>
          </SpellCheckSuggestions>
          <StartOffset>10</StartOffset>
        </SpellCheckUnit>
      </SpellCheckUnits>
      <Status>OK</Status>
    </SpellCheckResult>
  </SpellCheckResults>
</SpellCheck>
```

Then, we replace original segments by auto-corrected segments if needed (depending on the confidence score) and continue the process with the translation step.

Once the spell-checking process is finished, the “speller” file can be read and displayed on the web interface inside a paginated Data Grid. In the “context” column, we display the original words in red (using the offset position given by the API), and three words (before and after) around the error. We also group errors and compute the occurrence count.



Dashboard > Spell check report

Filters: Auto-corrected All | Error type All | Search: | Ok

| Auto-corrected | Original text | Error type | Suggestions | Context | Occurence |
|---|---|---|---|---|---|
| No | Reverso | Spelling | | ... you and Reverso-Softissimo as the ... | 1 |
| No | Softissimo | Spelling | | ... you and Reverso-Softissimo as the ... | 1 |
| No | gt | Spelling | GA (62%) ↓ | &gt;&gt; Discover how ... | 4 |
| No | laquo | Spelling | lake (57%) | ... click on &laquo; Create a ... | 1 |
| No | raquo | Spelling | rake (57%) | ... job &raquo; | 1 |
| No | Logout | Spelling | Log out (48%) | Logout | 1 |
| Yes | Nb | Spelling | NB (99%) | Nb of translated ... | 10 |
| No | aspx | Spelling | asp (62%) ↓ | ... softissimo.com/fr/ solutions.aspx | 2 |
| No | to | Spelling | too (70%) | to | 1 |
| No | sucessfully | Spelling | successfully (62%) | Translation memory sucessfully checked | 12 |
| No | langague | Spelling | Langayo (43%) | ... TMX file langague | 1 |
| No | TMX | Spelling | TMT (62%) ↓ | A TMX file with ... | 1 |
| No | InfoTranslationMemoryTMMissing | Spelling | | InfoTranslationMemoryTMMissing | 1 |
| No | UTF-8 | Spelling | | ... encoded in UTF-8 (see guidelines) | 5 |
| No | InfoNewJobFileNameExist | Spelling | | InfoNewJobFileNameExist | 1 |
| No | InfoJobReportUrlExistingFile | Spelling | | InfoJobReportUrlExistingFile | 1 |
| Yes | cancelled | Spelling | canceled (99%) | ... has been cancelled by the ... | 3 |
| Yes | Xml | Spelling | XML (99%) | ... not a Xml file | 4 |
| No | originale | Spelling | original (62%) ↓ | Version originale du site. ... | 1 |
| No | du | Spelling | do (63%) ↓ | Version originale du site. | 1 |

1 2 3 4 5 6 7 8 9 10

Spell Checking report for RESX (XML file) translation

# 7. Translation memory

## 7.1 Overview

The translation memory implementation is described in deliverable D4.4. “Translation Memory Module able to provide translation”.

# 8. Personal Dictionary

## 8.1 Overview

The personal dictionary is implemented as described in D2.2.

The personal dictionary implementation is described in deliverable D2.5 “Interface to create personal dictionaries.” (without online entry editing, described below).

## 8.2 Online entry editing

In addition to the CSV import mode, we added an online entry editing feature for dictionaries (See D2.2).

We can now manually add / edit / delete dictionary entries. The entries (in CSV import mode or editing mode) are saved in Flavius Database. Thus, if the user leaves the editing page without saving, he will not loose his modifications. The edited dictionary is sent to the LW API globally, when the user clicks on the “submit” button: The former dictionary is deleted from the Language Weaver API side and a new one is created including all edited entries (currently, we only allow one dictionary per direction). The creation and deletion steps use the same Language Weaver API calls than the former CSV import (See D2.5).

# 9. Automatic translation

## 9.1 Overview

The automatic translation module is implemented as described in D2.2.

## 9.2 Use of LW SMT Engine

The translation in Flavius is done through the LW API (See deliverable D4.5 V2).

All information about web service addresses, engines used and directions are read and stored into the Flavius platform cache. Moreover, the system can combine different engines from different providers. For now, we configured it with Language Weaver engines, but we could support some additional engines (ex: French <=> Russian from Softissimo, etc.).

When the language direction does not exist in LW API, we can configure some “pivot” language and use it in Flavius as it would be a real direction. For example, for French to Romanian, we use English as a pivot language: we first translate French to English, and then English to Romanian.

### 9.2.1 Configuration file

A configuration file allows the customization of the platform.

<u>Web service section</u>

This section enumerates the web services which can be reached from the Flavius platform.

```
<webservices>
  <webservice name="LW">
    <URL value="http://193.39.117.4/riws_dev/translation.asmx" />
    <User value="flavius" />
    <Password value="********"/>
    <Engine value="*" />
    <TimeOut value="60000" />
    <Block value="40" />
    </webservice>
</webservices>
```

- Name: Web service name. Mandatory to identify the Web Service.
- URL: Web Service URL
- User/Password: Access rights to the Web Service
- Engine: Id engine list to publish
    - 1,5 engines 1 and 5 are allowed
    - 6 engine 6 is allowed
    - * all engines are allowed
- Timeout: time out in millisecond.
- Block: Segment count to send in one block to the translation Web Service.

## Direction section

This section shows the directions available for the user:

```
  [0] [Virtual Engine]
  [1] [Reverso 5]
  [2] [LEC and partners]
  [3] [Softissimo ERRE]
  [4] [LanguageWeaver]
  [5] [LanguageWeaver(Saas)]
  [6] [Prompt9]

  <directions>
    <direction source="fr" target="en" pivot="  " ws1="LW" eg1="5"/>
    <direction source="fr" target="es" pivot="  " ws1=" LW" eg1="5"/>
    <direction source="fr" target="de" pivot="  " ws1=" LW" eg1="5"/>
    <direction source="fr" target="it" pivot="  " ws1=" LW" eg1="5"/>
    <direction source="fr" target="ro" pivot="en" ws1=" LW" eg1="5" ws2="LW" eg2="5"/>
    <direction source="fr" target="ru" pivot="en" ws1="LW" eg1="1" ws2="LW" eg2="3"/>
…
```

- Source: source language of the direction
- Target: target language of the direction
- Pivot: if the direction needs a pivot language
- Ws1/eg1: web service name / id engine which provide direction
- Ws2/Eg2: web service name / id engine which provide direction for the second direction if pivot is needed.

# 10. Post-edition

## 10.1 Overview

The functional specifications of the Post-Edition module are described in the deliverable D2.2.

The implementation part for “URL” and “File” scenario will be described in the deliverable D2.6.

# 11. Data retrieval

## 11.1 Overview

Data retrieval has been implemented as described in D2.2:

- The download of the translated versions (for URL scenario) is disabled for now, only publishing is available
- The feedback suggestion module is not yet implemented and will be delivered in D2.8.

## 11.2 Website publishing (“URL” scenario)

Website publishing follows several steps:

### 11.2.1 File Copy

First, the translated files are copied from the translation folder repository to the publishing server in a specific folder. The hierarchy structure of the Website is kept.

Assuming that the translated pages are located in the translation folder jobs \[idjob]\translated\[lng], they are copied in the publishing folder to [domainName]\[idjob]\[lng].

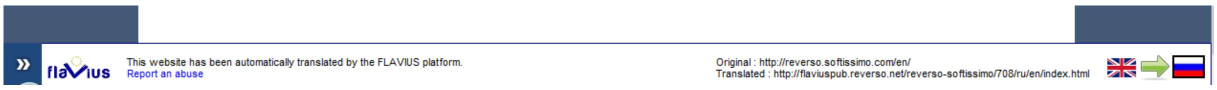
### 11.2.2 Flavius intrusive bottom bar generation

The published translated pages are not included in HTML Frame object anymore. A JavaScript bottom bar is used to navigate among the different language versions of the Website. This bottom bar is generated using a JavaScript file (bottomBar.js) and a CSS file (bottomBar.css) shared by all published website. Thus, we can change the bottom bar design without having to republish former published websites. The variable content of the bottom bar for each page is displayed by setting parameters to the JavaScript function (See below).

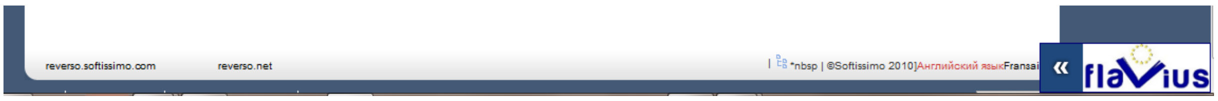
The JavaScript code is inserted on each page, just before the “</body>” tag:

```
<script type="text/javascript" src="/bottomBar.js"></script>
<script type="text/javascript">
        $(document).ready(function() {
                InitBottomBar('en',
                'fr',
                'http://reverso.softissimo.com/en/company',
                'http://flaviuspub.reverso.net/reverso-softissimo/753/fr/en/company.html',
                new Array('http://flaviuspub.reverso.net/reverso-
softissimo/753/fr/en/company.html'),
                new Array('fr'));
        });
</script>
```

This bottom bar can be collapsed or expanded:


This website has been automatically translated by the FLAVIUS platform.
Report an abuse
Original : http://reverso.softissimo.com/en/
Translated : http://flaviuspub.reverso.net/reverso-softissimo/708/ru/en/index.html

The bottom bar is expanded


reverso.softissimo.com reverso.net | *nbsp | ©Softissimo 2010]Английский языкFransai

The bottom bar is collapsed.

**Note:**

To avoid CSS / JavaScript conflicts with the existing content of the published Website, main styles attributes of the bottom bar are overloaded (highest priority of cascading style sheets).

With this new bottom bar solution, some new issues were raised. If the user navigates and clicks on external links (for example, absolute links pointing on the original website), the user will leave the publishing Website and will not be alerted. One solution could be to add some JQuery code to listen to the unload event of the page and to display a warning popup.

Handling error

Some published pages could have dead links (page not crawled because of quota limitation, etc.). For now, we use the internal IIS mechanism to replace the 404 error page with a specific HTML page, indicating that the page called is not translated.

### 11.2.3 Website hosting and SEO

One of the main goals of Flavius is to index the published websites to search engines (Google, Bing, etc.). With the bottom bar "solution", published Flavius Websites are now well indexed (no more IFrame issues).

We still need to make some analysis on the impact on SEO, by testing the ranking on a set of testing published Websites. The indexing also depends significantly on the original website structure and on the backlinks pointing to the Flavius published website.

### 11.2.4 Translation of meta-tag description and keyword

Description and keyword meta-tags are extracted from each page, aggregated and inserted into a temporary HTML file. This file is translated in the different target languages.

Then, during the publishing step, these meta-tags are inserted in the corresponding translated page.

### 11.2.5 Google Analytics

We added a global tracking system to monitor the traffic generated by published Flavius websites.

For now, the traffic is logged through Google Analytics using a global Flavius Account. Google Analytics was chosen because of its statistic visualisation module and its simplicity.

This JavaScript code (see below) is inserted during the Website publishing step, on each page:

```
<script type="text/javascript">
        var tryNb = 5;
        function checkIfAnalyticsLoaded() {
                if (window._gaq && window._gaq.push) {asyncPageTracking();}
                else if (window._gat && window._gat._getTracker) {asyncPageTracking();}
                else if (window.urchinTracker) {
                        asyncPageTracking();
                        insertAnalyticsCode();
                }
                else {
                        if (tryNb > 0) {
                                tryNb--;
                                setTimeout('checkIfAnalyticsLoaded()', 100);
                        }
                        else {
                                asyncPageTracking();
                                insertAnalyticsCode();
                        }
                }
        }

        function asyncPageTracking() {
                var _gaq = _gaq || [];
                _gaq.push(['flaviusTracker._setAccount', 'UA-25814413-1']);
                _gaq.push(['flaviusTracker._trackPageview']);
        }

        function insertAnalyticsCode() {
                var _gaq = document.createElement('script'); _gaq.type = 'text/javascript'; _gaq.async = true;
                _gaq.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-
analytics.com/ga.js';
                var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(_gaq, s);
        }
        checkIfAnalyticsLoaded();
</script>
```

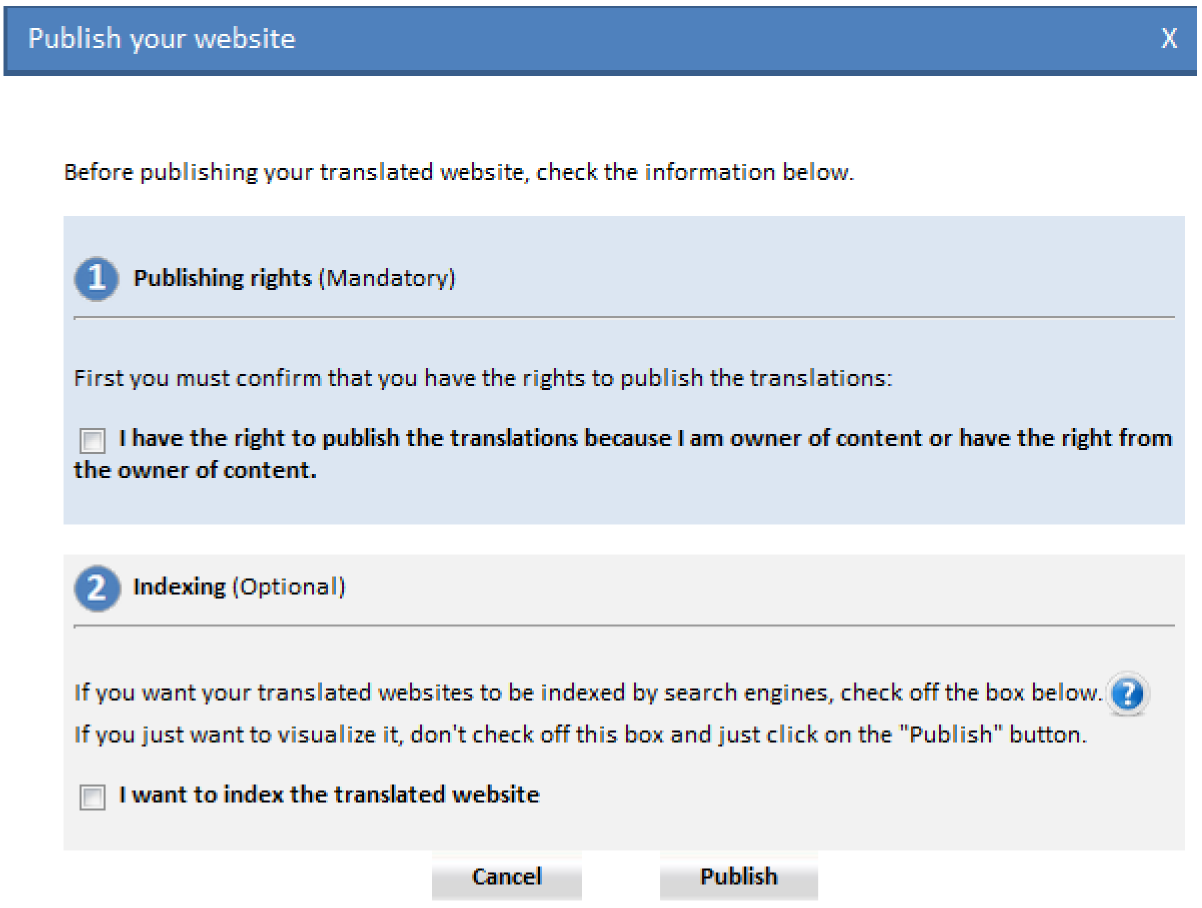
This JavaScript code is compliant with the new version of the Google Analytics API.

If the webmaster already inserted some Google Analytics JavaScript code in his original pages, we try to detect his Google Analytics version, and adapt our call to the Google API.

## 11.2.6 Indexing Websites and Terms of use

*Indexing*

Webmasters can now decide if their translated websites should be indexed or not by search engines.



Publish your website X

Before publishing your translated website, check the information below.

**1 Publishing rights** (Mandatory)

First you must confirm that you have the rights to publish the translations:

☐ **I have the right to publish the translations because I am owner of content or have the right from the owner of content.**

**2 Indexing** (Optional)

If you want your translated websites to be indexed by search engines, check off the box below.

If you just want to visualize it, don't check off this box and just click on the "Publish" button.

☐ **I want to index the translated website**

Publishing confirmation popup

The indexing choice is stored in the database and used to generate dynamically:

- The “robots.txt” file for the Flavius publishing Web Server: this file is used by search engines to know which part of a website should be indexed.
- The list of published Websites available on the Flavius Homepage (and on the "All Website" page).

Here is a sample of a generated robots.txt file preventing search engines to index the Website published in the “/reverso/648” folder:

```
User-agent: *
Disallow: /reverso/648/
```

### *Terms and conditions*

We added a “term of use” validation when a user creates an account in Flavius. Also, during the publishing step, he must validate a second time that he is the real website owner (See publishing confirmation popup above).

We also added a “report an abuse” link on all published Website allowing visitors to alert us if a Website has been published without authorization. In the admin panel of Flavius, we can now remove a Website from publishing.

A validation system based on the one used by Google could be added but some users risk to give up using Flavius. For example, adding a meta tag in the homepage header is not always easy, and sometimes this is not possible at all: (ex: Overblog). Moreover, allowing the user to add a marker anywhere in the page (like in a published article for example, to overcome the Overblog case) is dangerous because visitors could also add it in a comment and they will still be able to publish the website in Flavius.

However, if we have lots of “abuse notifications” after reaching a certain traffic, we will analyse them and if required, we will add such strong validation system (special file checking on server, or tag added to the homepage header).

## 11.2.7 Publishing issues

The publishing of a generated website is a complex step, which requires some fine-tuning in order to resolve the remaining issues.

### *Dead links*

Some websites (e.g. http://reverso.softissimo.com/) use JavaScript to build dynamically HTML links inside theirs pages. Thus, these pages are not crawled by the Flavius crawling module, and the link is not rewritten. When the user navigates on the published page, the link is built with the relative original URL (ex: /products) and the page cannot be retrieved.

One solution is to use a URL rewriting module to rewrite all the published pages of the Flavius Publishing Server, with the same relative URL than the original one.

In our case, navigating on “http://flaviuspub.reverso.net/reverso-softissimo/300/fr/products/”, should display the associated crawled page (for example “product.html”).

Advantages:

- We avoid some “dead links” problems on some websites

Disadvantages:

- The rewritten URL cannot be localized in the target language (bad point for SEO)

### *Forms*

Lots of websites have forms to gather some information from the users (ex: Contact forms, etc.). The published website cannot manage the forms. One solution could be to intercept with JavaScript all the form requests (links from “<input>” HTML tag for example) and to redirect the POST content to the form page of the original Website. Another solution could be to rewrite all the form links during the publishing step to point to the original Form page.

Both solutions could be explored but we will get the original page response, without translation. A last solution to explore could be to intercept all requests containing POST data, redirect the request to the original page, get the response result and dynamically translate it (See Crawling issue section).

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

## 11.3 ZIP Archive Download (“File scenario”)

Once the files are processed for the File scenario, a ZIP archive is generated and ready to be downloaded. The ZIP archive is created using the .NET Open Source Library SharpZipLib: http://www.icsharpcode.net/opensource/sharpziplib/.

The archive directories are described in the “12.3.5 Retrieve the job result” section.

# 12. Flavius API

## 12.1 Overview

Concerning the API, we defined a first version after some discussion with the power users of the consortium (Overblog, TVTrip, and Qype). The API covers a small part of future Flavius users, and we focused for now on the power user's needs. It will be implemented in the next phases (See deliverable D2.3: Implementation plan) and delivered in D2.9.

The Flavius API will be a REST based Application Programming Interface (API) accessible over the Internet and using existing technologies and protocols such as HTTP, SSL, and XML.

### *REST*

REST (Representational State Transfer) is an architectural style in which clients can make requests to the server, which will send responses to those requests. Requests and responses are built around the transfer of representations of resources.

REST uses HTTP methods for accessing and manipulating the state of the resources that are identified by a URI, as implemented on the World Wide Web.

The following methods are typically used to access and manipulate resources:

• HTTP POST + URI = Create a new resource
• HTTP GET + URI = Retrieve a representation of a resource
• HTTP DELETE + URI = Delete a resource

The Flavius API will be asynchronous. To meet the requirements defined by power users, it will provide the ability to:

- Create a job
- Cancel a job
- Delete a job
- Get a job status
- Retrieve a job result

## 12.2 Security Management

The Flavius API will be secured by a custom security mechanism. Every request will be authenticated. This is accomplished by attaching a custom header. Flavius API will use HTTP as the communication protocol and all data will be encrypted with SSL (HTTPS).

Softissimo will provide to each power user a username and a password that will be used to authenticate the access rights to the Flavius API.

The password will never be sent in any HTTP request. Every HTTP request to Flavius API should contain the following HTTP headers:

- Created: Contains the current date and time (ex: Flavius_date: “”11/25/2011 19:04:40”)
- Username: Contains the username
- Signature: Contains the signature string used to authenticate the request.(ex: Signature:”afsfsqfzf=fzsfs354”)

<u>Created:</u>

This date is used to mark the exact date and time of the call.

<u>Username:</u>

The username will be the client login.

<u>Signature:</u>

The signature is generated by concatenating the first two fields (Username, Created) and encrypting them using a secured hash algorithm.

The algorithm used should be HMAC-SHA1 which is a type of message authentication code (MAC) calculated using a specific algorithm involving a cryptographic hash function (SHA-1) in combination with a secret key. The HMAC process mixes a secret key with the message data, hashes the result with the hash function, mixes this hash value with the secret key again and then applies the hash function a second time. The secret key will be the client password provided by Softissimo.

### *Server Side Authentication Checking*

If the header is not present in the header section, the call will be rejected. Also if either the Username or Created field is empty (no set), the call will also fail. If the above conditions are met successfully, the authentication continues by generating the signature locally (at Flavius server) and checking it with the one provided in the custom header. The key used to generate the signature is in fact the client's password. If the signatures match, a further checking is made: the creation date of the custom header is checked against the client's last call. If the creation date is higher than the last call date, authentication succeeds (if not call is rejected with a specific error message).

## 12.3 API Definition

The global access will be:
https://flavius.reverso.net/api/v1/<resource>/

Every API function will return an HTTP response code as well as an XML document. The XML document will provide the response data for the API function call if the HTTP response code is an HTTP 200 OK. The XML document will provide the error data for the API function call if the HTTP response code is an HTTP 40x.

Error XML format

In case of error, the XML document will be sent in the following format:

```
<Error xmlns="http://www.flavius.reverso.net/flaviusAPI/">
   <ErrorException> INTERNAL_ERROR_EXCEPTION</ ErrorException >
   <ErrorMessage>Error message</ErrorMessage>
    <ErrorSource> Error source</ErrorSource>
 </Error>
```

### 12.3.1 Create a new job

**URL Resource**
https://flavius.reverso.net/api/v1/job/

**Description**
HTTP POST: This submits a Job creation request to Flavius. Each job created will be launched in an asynchronous way (See getting job progress). The parameters must be supplied via HTTP POST.

**Mandatory Parameters description**
*inputFormat*: {xml; xmlZip; url; php; xliff; resx; xmlWordpress}. This parameter contains the input format to process in Flavius:

- *xml*: An xml file
- *xmlZIP*: An archive in ZIP format that contains xml files at the root directory
- *url*: the URL of the website home page to crawl and to process.
- *xliff*: A XLIFF file to process
- *xmlWordPress*: an XML file in the WordPress export format.
- *resx*: A .NET Resource file

*inputContent*: The file content to upload in binary format:

- In case of inputFormat = url, it should contain the Homepage URL to process (ex: http://reverso.softissimo.com).
- In case of inputFormat = xml, it should contain a well formed XML file with an *.xml extension (UTF-8 with BOM).
- In case of inputFormat = xmlZIP, it should contain a valid ZIP archive with only xml files at the root directory.
- In case of inputFormat = xliff, it should contain a valid XLIFF file according to the DTD: http://www.oasis-open.org/committees/xliff/documents/xliff.dtd (V1.2)
- In case of inputFormat = xmlWordPress: it should contain a valid export xml file from the WordPress default export module (V3)
- In case of inputFormat = resx, it should contain a valid RESX file generated by .NET applications.

*srcLanguage*: {en;fr:es;de;it;pl;ro;sv} This parameter specifies the source language using the two-letter convention (ex: “fr” for French, “en” for English, etc.).

*tgtLanguages*: {en;fr:es;de;it;pl;ro;sv} This parameter specifies the target languages using the two-letter convention (ex: “fr” for French, “en” for English, etc.). To use multiple target languages, they should be separated by semi-colons, without spaces (ex: fr;en).

**Optional Parameters description**

*xmlTemplate*: {AllData} This parameter contains the name of the XML template to apply. By default, the “AllData” template is used for xml input format.

*inputEncoding*: {UTF-8} This parameter allows to define the input encoding format used by your input content. For now, Flavius API will support only UTF-8 as the encoding for the input content. By default, the UTF-8 encoding is used.

*enableSpellCheck*: {false;true} This parameter enables / disables the spell check of the input content. The spell-checking module will correct errors detected automatically.
The enableSpellCheck parameter will be ignored for url input format (always disabled).
By default, this parameter is set to true.

*enableDictionary*: {false;true} This parameter enables / disables the use of dictionary created and enabled in the Flavius Website interface.
By default, this parameter is set to false.

*enableTM* {false;true} This parameter enables / disables the use of TMX created and enabled in the Flavius Website interface for the requested language direction.
By default, this parameter is set to true. If no TMX exists for the requested direction, this parameter will be ignored.

*crawlDepth* {1-5}.This parameter specifies the crawl depth for web crawling (inputFormat = url). For example, a crawl depth of 1 will “crawl” the first page and its external links. If the input format is not set to url, this parameter will be ignored.
By default this parameter is set to 1.

*callbackUrl* {ValidURL}: When the job is finished, if this parameter is set with a valid URL, this URL will be called by FLAVIUS using HTTP protocol. The use of callbackUrl is recommended

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

to avoid Flavius system overload. If the callback failed (Timeout, etc.), the job status can still be retrieved with the "Get Job status" call.

**Output Response**

Successful HTTP requests will return an HTTP 200 response code and an XML document:

```
<FlaviusResponse xmlns="http://www.flavius.reverso.net/flaviusAPI/">
        <jobid>33</jobid>
</ FlaviusResponse>
```

This is a non-blocking method. The job id will be required to view the progress result and to download it.

**Error Response**

Failed HTTP requests will result in an HTTP 400 response code as well as an XML document containing an error message (See error XML format)

| Error Exception | Error Message | Error source |
|---|---|---|
| SECURITY_HEADER_NOTFOUND | no security header has been provided for the call | AuthenticateRequest |
| SECURITY_HEADER_INVALID | when the provided security header is invalid | AuthenticateRequest |
| SECURITY_HEADER_INVALID_DATE | the provided security header contains an invalid date | AuthenticateRequest |
| LANGUAGE_PAIR_NOTFOUND | the provided direction is not available | JobCreationRequest |
| XML_TEMPLATE_NOT_AVAILABLE | The xml template provided is not available | JobCreationRequest |
| INTERNAL_ERROR_EXCEPTION | An internal error exception occurred | JobCreationRequest |
| CRAWLDEPTH_OVER_QUOTA | The crawl depth is over quota for this user | JobCreationRequest |
| NO_TM_ENABLED_OR_DEFINED | No tmx enabled or defined for this direction | JobCreationRequest |
| NO_DICTIONARY_ENABLED_OR_DEFINED | No dictionary enabled or defined for this direction | JobCreationRequest |
| INPUT_ENCODING_NOT_FOUND | The encoding specified is not found | JobCreationRequest |
| INPUT_CONTENT_NOT_FOUND | The input content is not found | JobCreationRequest |
| INVALID_INPUT_FORMAT | The input format is invalid | JobCreationRequest |

### 12.3.2 Cancel a job

**URL Resource**
https://flavius.reverso.net/api/v1/canceljob/{JobId}

**Description**
HTTP POST: it submits a job cancellation request to Flavius. If the job is in progress, the job is cancelled. Each job cancellation will be launched in an asynchronous way. The {JobId} should be to the job id to cancel (id generated by “create a job” request).

**Output Response**
Successful HTTP requests will return an HTTP 200 response code and an XML document:

```
<FlaviusResponse xmlns="http://www.flavius.reverso.net/flaviusAPI/">
        <ResponseStatus>OK</ResponseStatus>
</ FlaviusResponse>
```

This is a non-blocking method. The job id will be required to view the progress result.

**Error Response**
Failed HTTP requests will result in an HTTP 400 response code as well as an XML document containing an error message (See error XML format).

| Error Exception | Error Message | Error source |
|---|---|---|
| SECURITY_HEADER_NOTFOUND | no security header has been provided for the call | AuthenticateRequest |
| SECURITY_HEADER_INVALID | when the provided security header is invalid | AuthenticateRequest |
| SECURITY_HEADER_INVALID_DATE | the provided security header contains an invalid date | AuthenticateRequest |
| JOBID_NOT_FOUND | the provided job id is not found | JobCancelRequest |
| JOB_STATUS_INVALID | The is not IN_PROGRESS or WAITING_FOR status | JobCancelRequest |
| INTERNAL_ERROR_EXCEPTION | An internal error exception occurred | JobCancelRequest |

### 12.3.3 Delete a job

**URL Resource**
https://flavius.reverso.net/api/v1/job/{JobId}

**Description**
HTTP DELETE: This submits a Job delete request to Flavius. If the job is in progress, the job cannot be deleted. Each job deletion will be launched in a synchronous way. The {JobId} should be to the job id to delete (id generated by “create a job” request).

**Output Response**
Successful HTTP requests will return an HTTP 200 response code and an XML document:

```
<FlaviusResponse xmlns="http://www.flavius.reverso.net/flaviusAPI/">
        <ResponseStatus>OK</ResponseStatus>
</ FlaviusResponse >
```

**Error Response**
Failed HTTP requests will result in an HTTP 400 response code as well as an XML document containing an error message (See error XML format).

| Error Exception | Error Message | Error source |
|---|---|---|
| SECURITY_HEADER_NOTFOUND | no security header has been provided for the call | AuthenticateRequest |
| SECURITY_HEADER_INVALID | when the provided security header is invalid | AuthenticateRequest |
| SECURITY_HEADER_INVALID_DATE | the provided security header contains an invalid date | AuthenticateRequest |
| JOBID_NOT_FOUND | the provided job id is not found | JobDeleteRequest |
| JOB_STATUS_INVALID | The job is not in ENDED status | JobDeleteRequest |
| INTERNAL_ERROR_EXCEPTION | An internal error exception occurred | JobDeleteRequest |

| | | |
|---|---|---|
| Project co-funded by the European Commission within the ICT Policy Support Programme | | |
| Dissemination Level | | |
| C | Confidential, only for members of the consortium and the Commission Services | |

## 12.3.4 Get job Status

**URL Resource**
https://flavius.reverso.net/api/v1/job/{JobId}

**Description**
HTTP GET: it submits a Job status request to Flavius. Each job status request will be launched in a synchronous way. The {JobId} should be to the job id of the job to consult (id generated by “create a job” request).

**Output Response**
Successful HTTP requests will return an HTTP 200 response code and an XML document:

```
<FlaviusResponse xmlns="http://www.flavius.reverso.net/flaviusAPI/">
        <JobStatus>{JobStatusCode}</JobStatus>
        <WorkflowStep>{WorkflowStepCode}</WorkflowStep>
        <StepProgressStatusPercentage>
                {StepProgressStatusPercentage}
        </StepProgressStatusPercentage>
</ FlaviusResponse>
```

{JobStatusCode} describes the progress status of the job and can have the following values:

- UNKNOWN: Not used for now
- WAITING_FOR: The job is waiting for a user action.
- CANCELED: The job is cancelled.
- IN_PROGRESS: The job is in progress.
- ABORTED: The Job is aborted.
- ENDED: The job successfully ended. It can be retrieve with the job retrieve request.

{WorkflowStepCode} indicates the current workflow step of the job and can have the following values:

- UNKNOWN: Not used for now
- EXTRACTING: “extracting” step (Crawling, Archive unzipping)

- MANUAL_EXTRACTING_VALIDATION: The job is waiting for a user action in “extracting step.
- REVISING: Spell checking step.
- MEMORY_TRANSLATING: Translation memory matching step.
- AUTO_TRANSLATING: Machine translation step.
- MANUAL_EDITING: Post editing step (Manual post edition from the user)
- ARCHIVING: Archiving step (Zip archive creation, publishing)

{StepProgressStatusPercentage} indicates the progress status of the current step in percentage. This tag is filled only for AUTO_TRANSLATING and REVISING step.

**Error Response**
Failed HTTP requests will result in an HTTP 400 response code as well as an XML document containing an error message (See error XML format).

| Error Exception | Error Message | Error source |
|---|---|---|
| SECURITY_HEADER_NOTFOUND | no security header has been provided for the call | AuthenticateRequest |
| SECURITY_HEADER_INVALID | when the provided security header is invalid | AuthenticateRequest |
| SECURITY_HEADER_INVALID_DATE | the provided security header contains an invalid date | AuthenticateRequest |
| JOBID_NOT_FOUND | the provided job id is not found | JobStatusRequest |
| INTERNAL_ERROR_EXCEPTION | An internal error exception occurred | JobStatusRequest |

## 12.3.5 Retrieve the job result

**URL Resource**
https://flavius.reverso.net/api/v1/job/{JobId}/files

**Description**
HTTP GET: This submits a Job retrieval request to Flavius. Each job retrieval request will be launched in a synchronous way. The {JobId} should be the job id of the job to retrieve (id generated by "create a job" request). This request should be called only if the job status is in "ENDED" status ("job status" request).

**Optional Parameters description**
*includeOriginalFiles*: {false;true} This parameter disables / enables the sending of Original files in the archive result (OriginalFiles directory, see below)
By default, this parameter is set to false.

**Output Response**
Successful HTTP requests will return an HTTP 200 response code and an XML document:

```
<FlaviusResponse xmlns="http://www.flavius.reverso.net/flaviusAPI/">
        <Filename>{ArchiveFileName}</Filename>
        <OutputContent>{BinaryStream}</OutputContent>
</ FlaviusResponse >
```

{Filename} contains the file name of the ZIP archive.

{OutputContent} contains the binary stream of a ZIP archive containing the result files.

For File scenario, it contains all translated versions of the requested files, with the following tree structure:

/OriginalFiles/

/FlaviusSpellCheckedFiles/

/FlaviusTranslatedFiles/

For URL scenario, it contains all translated versions of the requested website, with the following tree structure:

/OriginalFiles/

/OrginalFiles/{Original Website Domain}

/OriginalFile/{Original Crawled Website Domain}/{Original Crawled Files}

/FlaviusTranslatedFiles/

/FlaviusTranslatedFiles/{language code requested}

/FlaviusTranslatedFiles/{language code requested}/{Original Website Domain}

/FlaviusTranslatedFiles/{language code requested}/{Original Website Domain}/{Translated Files}

**Error Response**
Failed HTTP requests will result in an HTTP 400 response code as well as an XML document containing an error message (See error XML format).

| Error Exception | Error Message | Error source |
|---|---|---|
| SECURITY_HEADER_NOTFOUND | no security header has been provided for the call | AuthenticateRequest |
| SECURITY_HEADER_INVALID | when the provided security header is invalid | AuthenticateRequest |
| SECURITY_HEADER_INVALID_DATE | the provided security header contains an invalid date | AuthenticateRequest |
| JOBID_NOT_FOUND | the provided job id is not found | JobRetrieveRequest |
| JOB_STATUS_INVALID | The job to retrieve is not in ENDED status | JobRetrieveRequest |
| INTERNAL_ERROR_EXCEPTION | An internal error exception occurred | JobRetrieveRequest |