Deliverable D 3.5.1

Automatic Test Set

Author(s): Juli Bakagianni, Sokratis Sofianopoulos, Stelios Piperidis (ILSP), Arle Lommel, Aljoscha Burchardt (DFKI)

Dissemination Level:	Public
Date:	11.01.2014 (revised 10.12.2014)

D 3.5.1: Automatic Test Set

Grant agreement no.	296347
Project acronym	QTLaunchPad
Project full title	Preparation and Launch of a Large-scale Action for Quality Translation Technology
Funding scheme	Coordination and Support Action
Coordinator	Prof. Hans Uszkoreit (DFKI)
Start date, duration	1 July 2012, 24 months
Distribution	Restricted
Contractual date of delivery	December 2013
Actual date of delivery	January 2014 (revised December 2014)
Deliverable number	D3.5.1
Deliverable title	Automatic Test Set
Type	Other – Report
Status and version	Final v1.0
Number of pages	42
Contributing partners	ILSP, DFKI
WP leader	DFKI
Task leader	ILSP
Authors	Juli Bakagianni, Sokratis Sofianopoulos, Stelios Piperidis, Arle Lommel, Aljoscha Burchardt
EC project officer	Kimmo Rossi
The partners in QTLaunchPad are:	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Dublin City University (DCU), Ireland
	Institute for Language and Speech Processing, R.C. “Athena” (ILSP/ATHENA RC), Greece
	The University of Sheffield (USFD), United Kingdom

For copies of reports, updates on project activities and other QTLaunchPad-related information, contact:
DFKI GmbH
QTLaunchPad
Dr. Aljoscha Burchardt aljoscha.burchardt@dfki.de
Alt-Moabit 91c Phone: +49 (30) 23895-1838
10559 Berlin, Germany Fax: +49 (30) 23895-1810
Copies of reports and other material can also be accessed via http://www.qt21.eu/launchpad

Contents

Update

This deliverable has been updated to clarify a few points and to correct some minor formatting issues.

1 Introduction

This report presents the different types of processes defined and implemented for testing the layers of the MT-infrastructure prototype developed in QTLaunchPad. As reported in Deliverables 3.4.1 and 7.1.3, two layers are currently under development: one layer that caters for the data infrastructure and data (pre)processing workflows, and another one that supports MT quality evaluation workflows.

The data infrastructure and data (pre)processing workflows are supported by the META-SHARE infrastructure and its extensions for the purposes of QTLaunchpad. To this end, we have started designing and implementing at a prototype level the extension of the current META-SHARE functionalities by providing an additional language processing mechanism for processing language datasets (essentially language corpora) with appropriate natural language tools. Language processing tools are provided as SOAP web services. According to the current design, when the user selects to process a dataset, a list of all available annotation services for each relevant annotation level (e.g. tokenization & sentence splitting, pos tagging, lemmatization, alignment) are provided for the given language, and resource type. As soon as the user selects a tool, the server invokes a service that dispatches the corpus to the specific web service for processing. The system informs the user about the requested job via the messaging service of the platform. When the processing has been completed, the new (annotated) dataset is automatically stored and indexed in the repository, and the user is appropriately informed. If the user, for any reason, requests to process a dataset with a specific tool, and this dataset has already been processed by the specific tool, then the system will just forward the user to the processed dataset that has been created and stored in the repository. Section 2 reports on the automatic procedures that are being followed for testing this data documentation, storage and data processing layer of the infrastructure prototype under development.

The quality evaluation layer of the infrastructure prototype (Deliverable 3.4.1) takes care of and facilitates building quality evaluation workflows involving human and/or automatic assessment methods. It consists in a data model together with data collections, DB user interfaces and data maintenance tools for the management of MT-related data collections. This application is used to administer existing data sets, e.g. the existing WMT data, and new data, such as the data sets compiled for and by the QTLaunchPad shared task in 2014.The prototype demonstrates an entire workflow that includes at least the following steps: 1) selection of source text to be translated, 2) routing through one or more MT engines, 3) sending to Quest for quality estimation (which can then be passed to translate5), 4) error markup of all or part of the source and/or target text in translate5 based on the results of QE, 5) generation of quality scores in translate5. Section 3 presents the steps that can be followed for a simple user test of the translate5-based translation quality evaluation layer that verifies that the related markup can be added and safely retrieved.

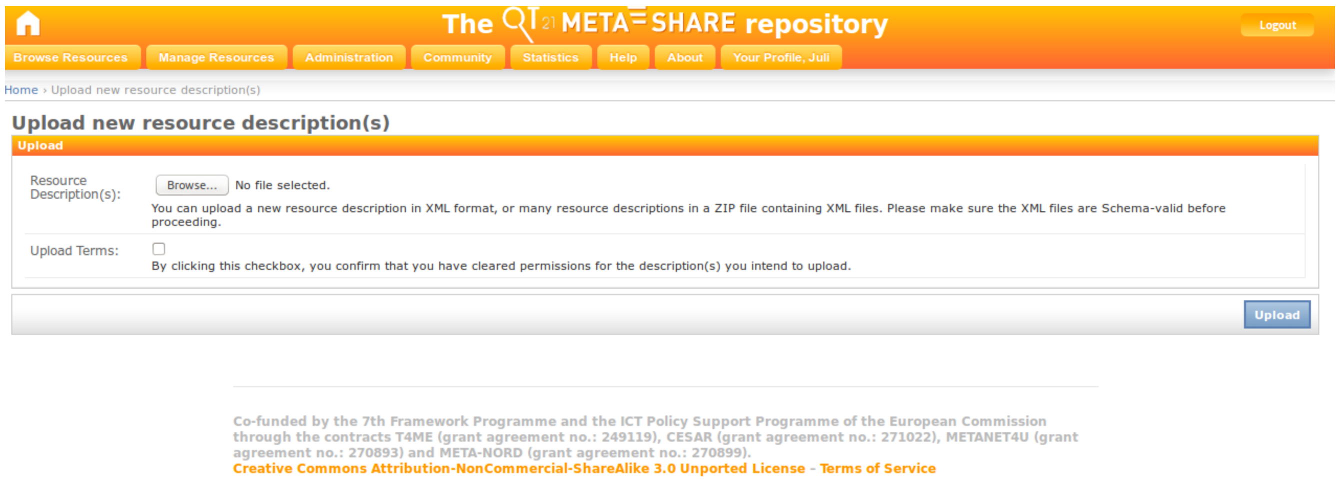
2. Testing the QT21 Resources Repository and its language processing extensions

Tests are created for the entire processing procedure, that is, from importing a resource description and resource actual data to processing a resource by services of various annotation levels and to the integrity of the derived resource. We have run the tests periodically during the implementation in order to guarantee the correctness of the processing mechanism.

Section 2.1 defines the tests that verify the uploading of resource descriptions, while Section 2.2 presents the tests that assert the uploading of resource actual data to the QT21 repository. Section 2.3 defines the tests that validate the stability and correctness of the processing mechanism.

2.1 Upload resource description

In the QT21 repository, providers can upload metadata-based resource descriptions by issuing a command to the command line interface or by using the META-SHARE graphical interface (see Figure 1). Moreover, providers can upload a single archive or a batch of archives by compressing them to a zip archive. A resource description would be successfully imported to the QT21 repository, if it is valid according to the META-SHARE metadata schema.



Upload new resource description(s)

Upload

Resource Description(s): Browse... No file selected.

You can upload a new resource description in XML format, or many resource descriptions in a ZIP file containing XML files. Please make sure the XML files are Schema-valid before proceeding.

Upload Terms: ☐

By clicking this checkbox, you confirm that you have cleared permissions for the description(s) you intend to upload.

Upload

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: 249119), CESAR (grant agreement no.: 271022), METANET4U (grant agreement no.: 270893) and META-NORD (grant agreement no.: 270899).
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License - Terms of Service

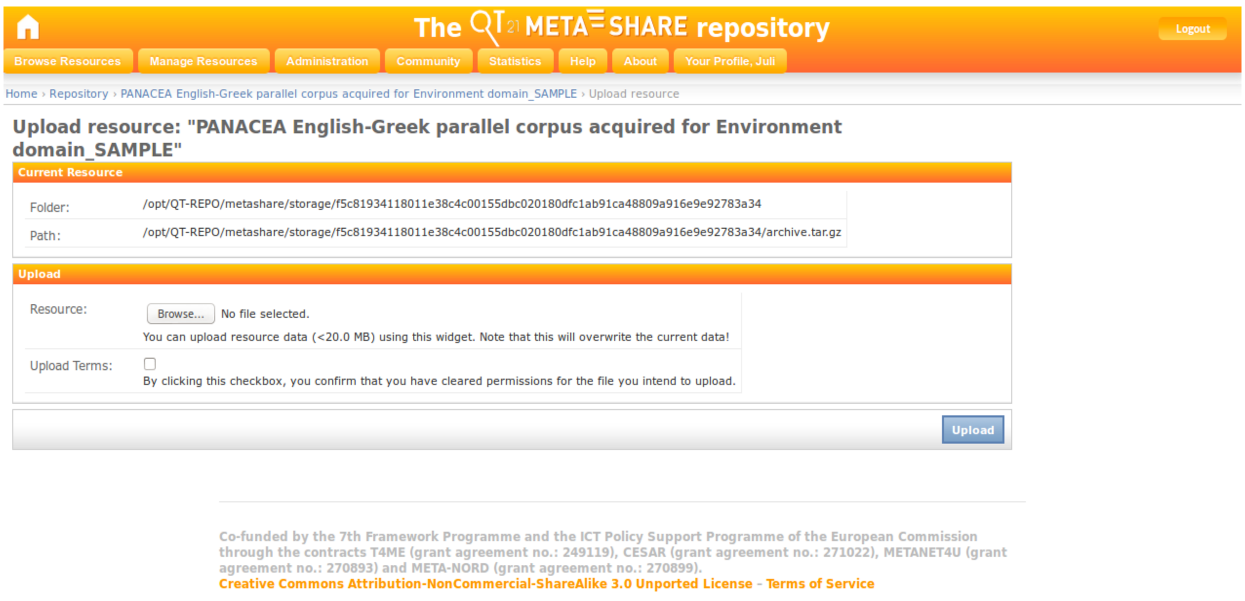
Figure 1

The respective tests to validate the correctness and the stability of uploading xml archives are given in Appendix 1, from line 1 to line 76. To validate that the code works as expected, this collections contains tests that verify the following cases:

- import all xml files in the given directory (line 38 – line 42)
- import valid xml file (line 44 – line 47)
- import invalid xml file (line 59 – line 63)
- import valid xml files that are compressed to zip archive (line 65 – line 69)
- import invalid xml files that are compressed to zip archive (line 71 – line 76)

2.2. Upload data

Providers can upload the actual data of either their or their editor group resources, when the administrator can upload the actual data of any resource. The data must be compressed to an archive that belongs to one of the following file types: zip, tar.gz, gz, tgz, tar or bzip2. Currently, the maximum size of an uploaded file is 10MB. The providers can upload the data from the GUI as shown in Figure 2.


The QT21 META-SHARE repository
Logout
Browse Resources
Manage Resources
Administration
Community
Statistics
Help
About
Your Profile, Juli
Home › Repository › PANACEA English-Greek parallel corpus acquired for Environment domain_SAMPLE › Upload resource

Upload resource: "PANACEA English-Greek parallel corpus acquired for Environment domain_SAMPLE"

Current Resource

Folder:	/opt/QT-REPO/metashare/storage/f5c81934118011e38c4c00155dbc020180dfc1ab91ca48809a916e9e92783a34
Path:	/opt/QT-REPO/metashare/storage/f5c81934118011e38c4c00155dbc020180dfc1ab91ca48809a916e9e92783a34/archive.tar.gz

Upload

Resource:	Browse... No file selected. You can upload resource data (<20.0 MB) using this widget. Note that this will overwrite the current data!
Upload Terms:	☐ By clicking this checkbox, you confirm that you have cleared permissions for the file you intend to upload.

Upload

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: 249119), CESAR (grant agreement no.: 271022), METANET4U (grant agreement no.: 270893) and META-NORD (grant agreement no.: 270899).
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License - Terms of Service

Figure 2

The tests that validate the method of uploading the actual resource data works as expected are shown in Appendix 1, from line 78 to line 267. These tests assert the following cases:

- admin user can upload data to any resource (line 154 – line 178)
- editor user can upload actual resource data to his/her owned resources or to resources that belong to his/her editor group (line 180 – line 224)
- editor user can update actual resource data to his/her owned resources or to

resources that belong to his/her editor group by re-uploading (line 226 – line 255)

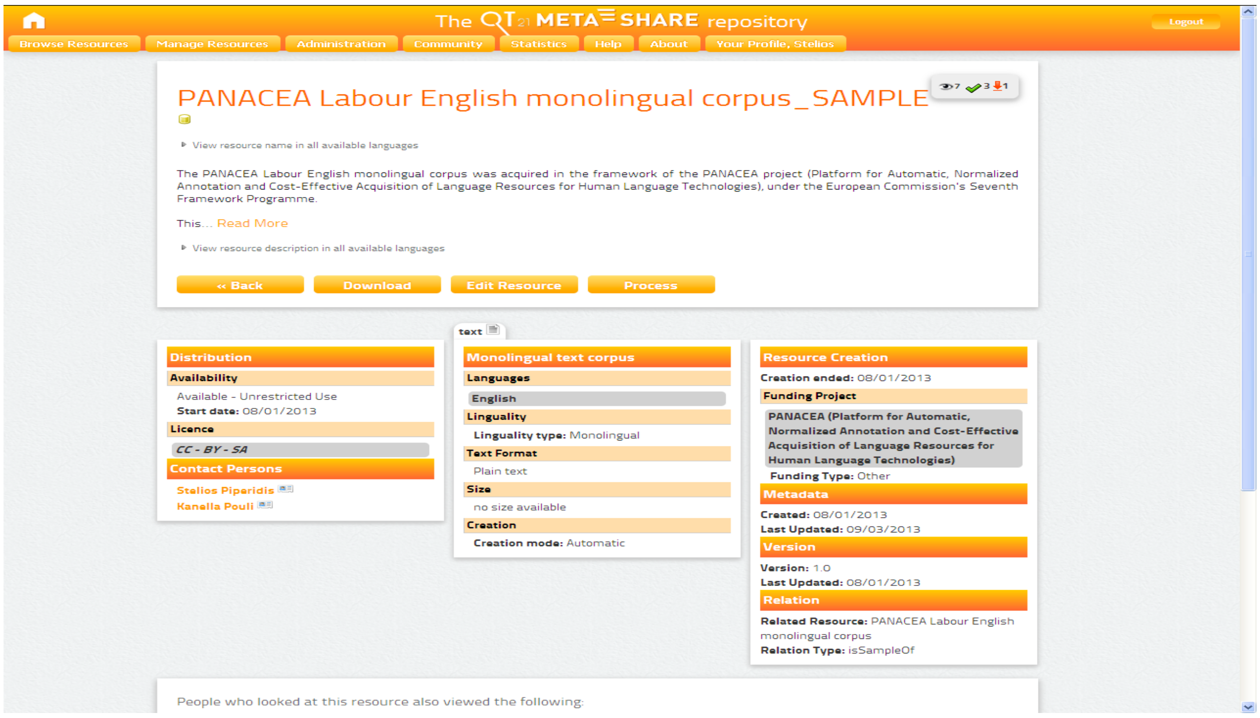
- editor user cannot upload actual resource data to resources that do not belong to his/her editor group (line 257 – line 267)

2.3. Processing

In the QT21 repository, editors can process stored resources with linguistic services, which are also stored in the repository. The services are distinguished in the following annotation levels: tokenization, sentence splitting, part of speech tagging, lemmatization, dependency parsing, chunking and sentence alignment.

Currently, the QT21 repository provides processing and annotation services for Greek and English, by deploying the services developed in the framework of the PANACEA project (http://panacea-lr.eu). The services that are involved in the processing procedure are the ILSP NLP services and the DCU NLP services. The first collection of services is used for processing Greek text corpora complying with the xces data format, whereas the second collection is used for processing English text corpora in the txt data format.

By selecting to process a dataset with a service (see Figure 3), a list of services shows up, from which the user can choose his/her preferred one.



PANACEA Labour English monolingual corpus_SAMPLE

View resource name in all available languages

The PANACEA Labour English monolingual corpus was acquired in the framework of the PANACEA project (Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies), under the European Commission's Seventh Framework Programme.

This... Read More

View resource description in all available languages

Distribution
Availability
Available - Unrestricted Use
Start date: 08/01/2013
Licence
CC - BY - SA
Contact Persons
Stelios Piperidis
Kanella Pouli

Monolingual text corpus
Languages
English
Linguality
Linguality type: Monolingual
Text Format
Plain text
Size
no size available
Creation
Creation mode: Automatic

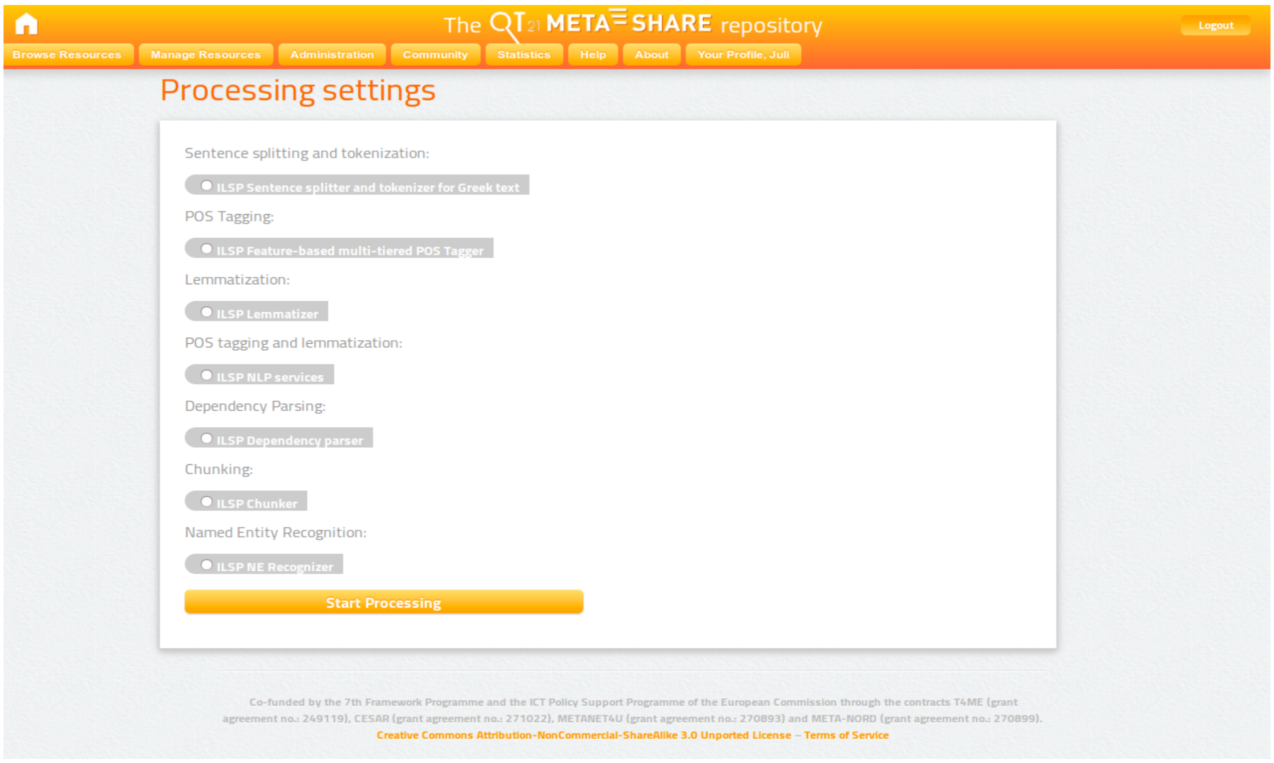
Resource Creation
Creation ended: 08/01/2013
Funding Project
PANACEA (Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies)
Funding Type: Other
Metadata
Created: 08/01/2013
Last Updated: 09/03/2013
Version
Version: 1.0
Last Updated: 08/01/2013
Relation
Related Resource: PANACEA Labour English monolingual corpus
Relation Type: isSampleOf

People who looked at this resource also viewed the following:

Figure 3

The services that are provided for annotation are relevant to the language, the data format and the annotation level of the input resource. The mapping between the specifications of an input resource and the adequate services for processing this input are given below:

- when a resource is a Greek monolingual text corpus having xces data format, then the relevant services are given in Figure 4.



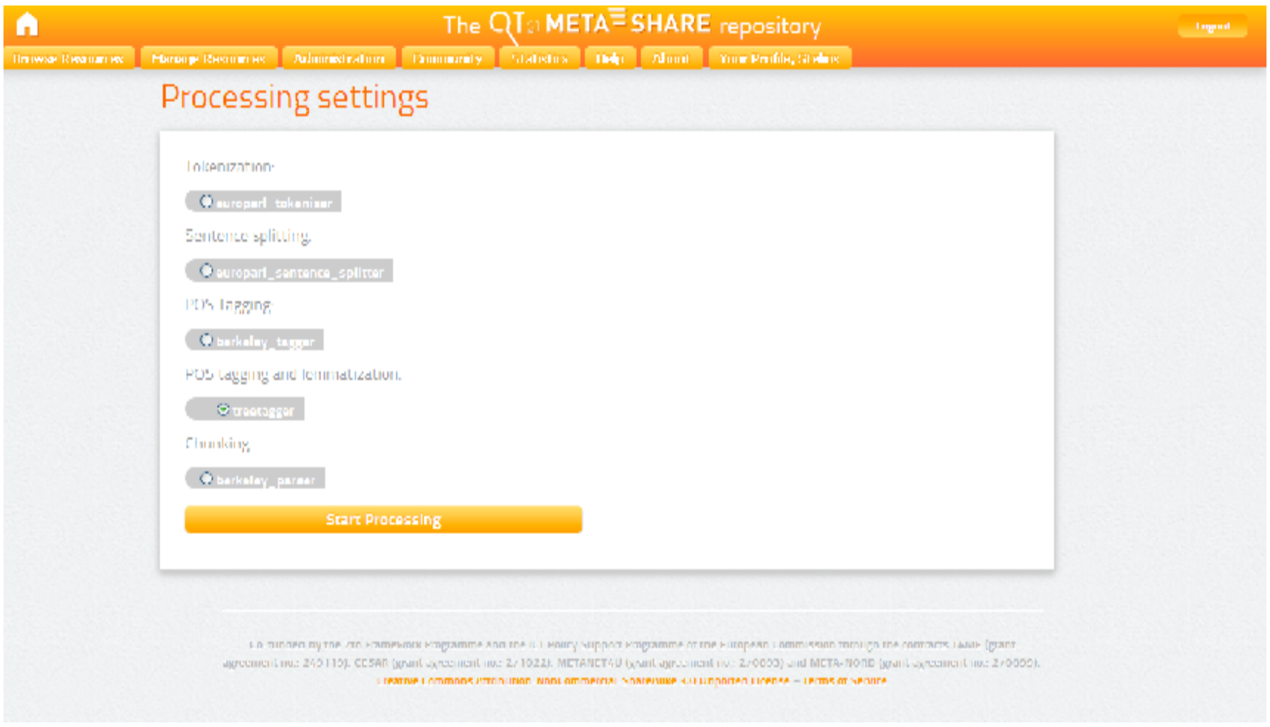
Processing settings

Sentence splitting and tokenization:
ILSP Sentence splitter and tokenizer for Greek text
POS Tagging:
ILSP Feature-based multi-tiered POS Tagger
Lemmatization:
ILSP Lemmatizer
POS tagging and lemmatization:
ILSP NLP services
Dependency Parsing:
ILSP Dependency parser
Chunking:
ILSP Chunker
Named Entity Recognition:
ILSP NE Recognizer

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: 249119), CESAR (grant agreement no.: 271022), METANET4U (grant agreement no.: 270893) and META-NORD (grant agreement no.: 270899).
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License – Terms of Service

Figure 4

- when a resource is an English monolingual text corpus having txt data format, then the relevant services are given in Figure 5, and



Processing settings

Tokenization:

( ) europarl_tokeniser

Sentence splitting:

( ) europarl_sentence_splitter

POS Tagging:

( ) berkeley_tagger

POS tagging and lemmatization:

[✓ treetagger]

Chunking:

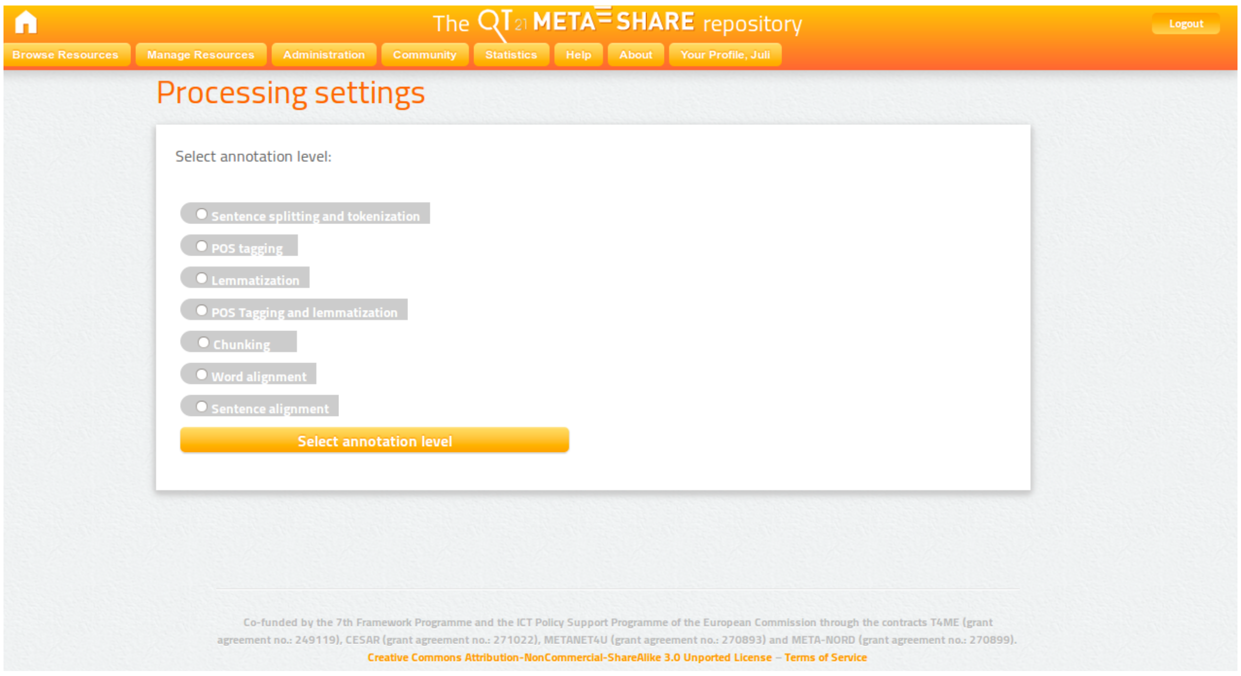
( ) berkeley_parser

Start Processing

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: 249119), CESAR (grant agreement no.: 271022), METANET4U (grant agreement no.: 270893) and META-NORD (grant agreement no.: 270899).
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License – Terms of Service

Figure 5

- when a resource is an English – Greek parallel corpus, then the selection of the preferred services is substantiated in two steps:
    - the selection of the annotation level (see Figure 6) and



Processing settings

Select annotation level:

( ) Sentence splitting and tokenization
( ) POS tagging
( ) Lemmatization
( ) POS Tagging and lemmatization
( ) Chunking
( ) Word alignment
( ) Sentence alignment

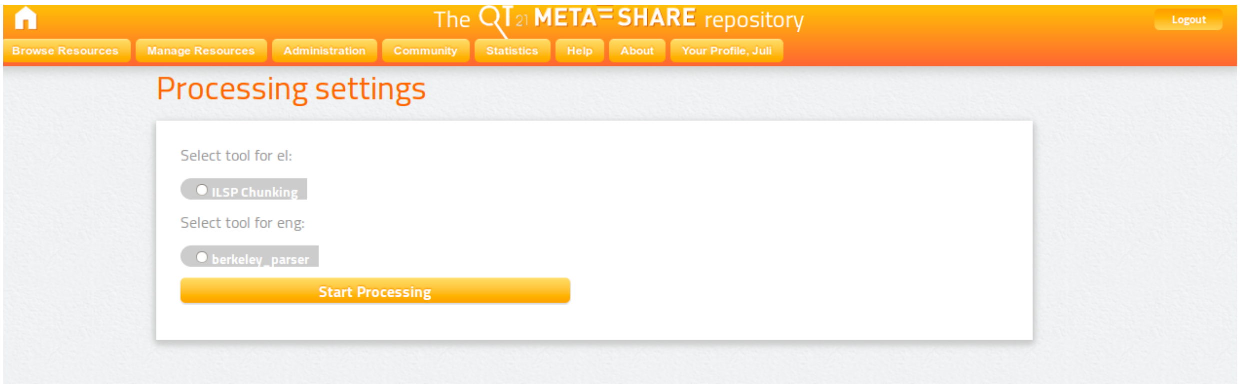
Select annotation level

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: 249119), CESAR (grant agreement no.: 271022), METANET4U (grant agreement no.: 270893) and META-NORD (grant agreement no.: 270899).
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License – Terms of Service

Figure 6

    - the selection of the services that correspond to the selected annotation level (see

Figure 7)



Processing settings

Select tool for el:

ILSP Chunking

Select tool for eng:

berkeley_parser

Figure 7

The tests that verify that the adequate services can process a resource are given in Appendix 1, from line 352 to line 406. Specifically, these tests assert the following cases:

- the editor selects to process a Greek monolingual text corpus, which comprises data in xces format. The services that are shown in Figure 4 must be provided to the user for selection (line 352 – line 369)
- the editor selects to process a English monolingual text corpus, which comprises data in txt format. The services that are presented in Figure 5 must be provided to the user for selection (line 371 – line 387)
- the editor selects to process an English - Greek parallel text corpus. The user selects Chunk annotation and the services that are shown in Figure 7 are provided to the user for selection (line 389 – line 406)

As soon as the user selects a tool, the server will invoke the equivalent service that will dispatch the corpus for processing. In the case of higher-level annotation tools, such as chunkers, processing is implemented using workflows. A workflow consists of a series of services, where the output of a lower annotation level service is the input of a next service of higher level. For example, the service for tagging/lemmatizing a Greek corpus is implemented as a workflow by invoking the processing chain shown in Figure 8.

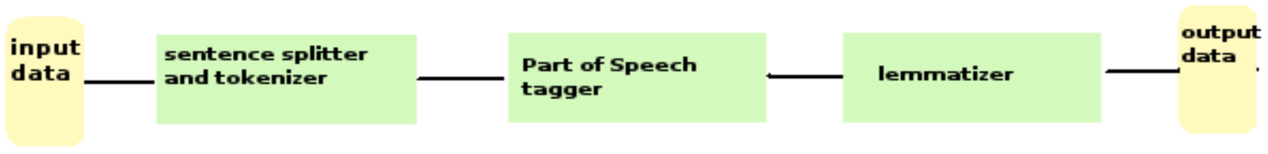
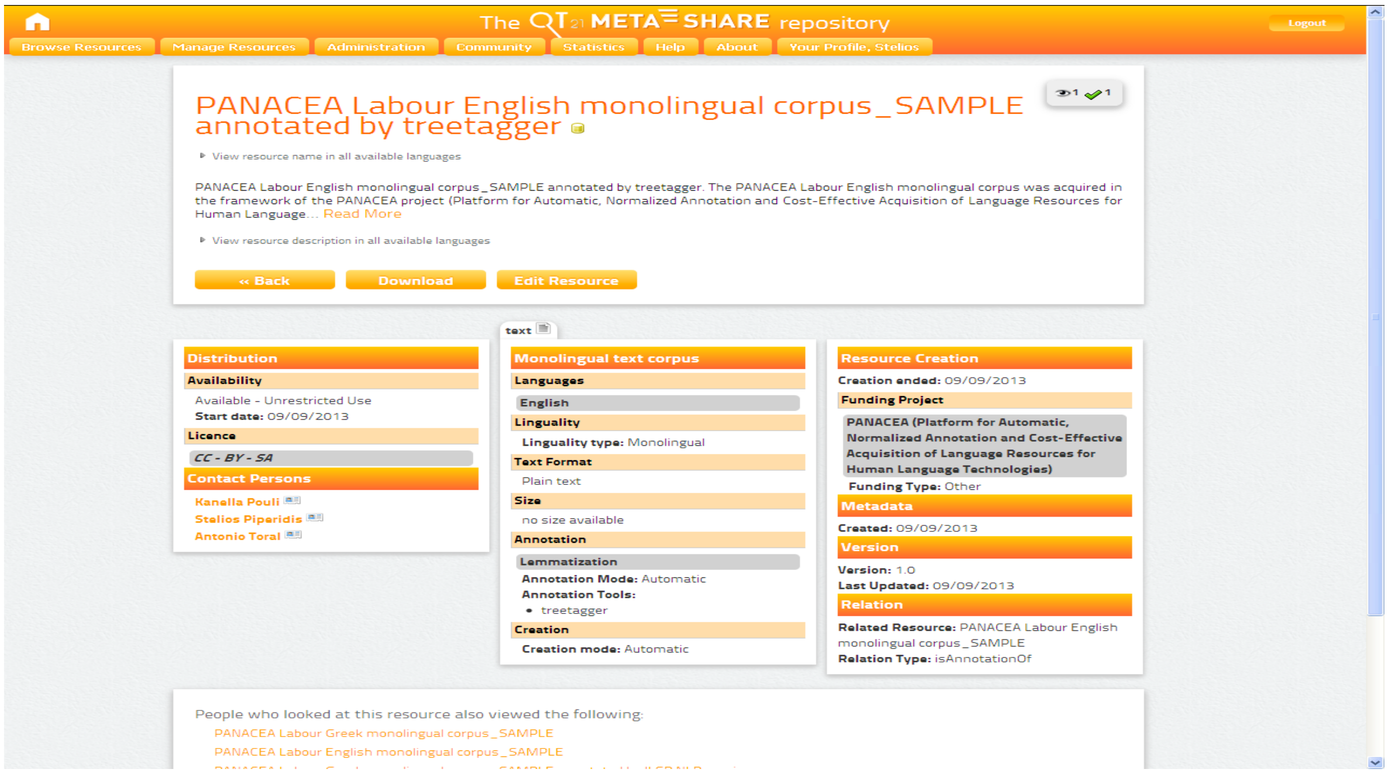


Figure 8

When the processing of the resource has completed successfully, the system will:

- upload the output resource (metadata and data) to the QT21 repository (see Figure 9)



PANACEA Labour English monolingual corpus_SAMPLE annotated by treetagger

View resource name in all available languages

PANACEA Labour English monolingual corpus_SAMPLE annotated by treetagger. The PANACEA Labour English monolingual corpus was acquired in the framework of the PANACEA project (Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language... Read More

View resource description in all available languages

Distribution
Availability
Available - Unrestricted Use
Start date: 09/09/2013
Licence
CC - BY - SA
Contact Persons
Kanella Pouli
Stelios Piperidis
Antonio Toral

Monolingual text corpus
Languages
English
Linguality
Linguality type: Monolingual
Text Format
Plain text
Size
no size available
Annotation
Lemmatization
Annotation Mode: Automatic
Annotation Tools:
- treetagger

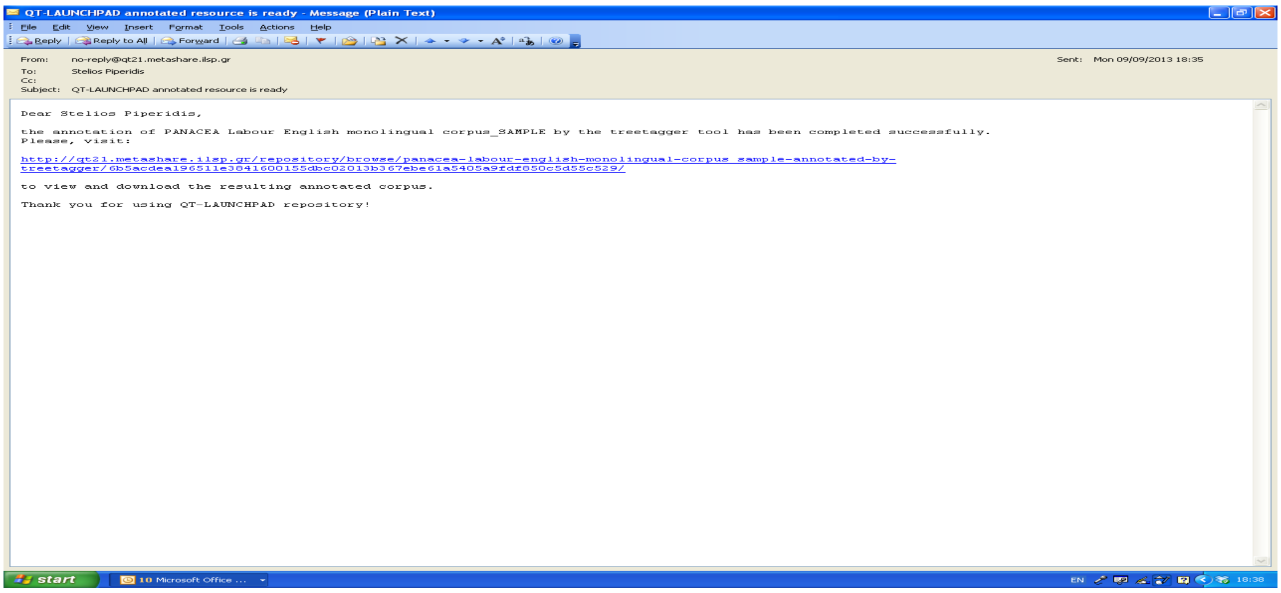
Creation
Creation mode: Automatic

Resource Creation
Creation ended: 09/09/2013
Funding Project
PANACEA (Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies)
Funding Type: Other
Metadata
Created: 09/09/2013
Version
Version: 1.0
Last Updated: 09/09/2013
Relation
Related Resource: PANACEA Labour English monolingual corpus_SAMPLE
Relation Type: isAnnotationOf

People who looked at this resource also viewed the following:
PANACEA Labour Greek monolingual corpus_SAMPLE
PANACEA Labour English monolingual corpus_SAMPLE

Figure 9

- inform the user about the requested job via the messaging service of the platform. The email will contain the link of the output resource on the QT21 web interface (see Figure 10).


QT-LAUNCHPAD annotated resource is ready - Message (Plain Text)
File Edit View Insert Format Tools Actions Help
Reply Reply to All Forward

From: no-reply@qt21.metashare.ilsp.gr
To: Stelios Piperidis
Cc:
Subject: QT-LAUNCHPAD annotated resource is ready
Sent: Mon 09/09/2013 18:35

```
Dear Stelios Piperidis,

the annotation of PANACEA Labour English monolingual corpus_SAMPLE by the treetagger tool has been completed successfully.
Please, visit:

http://qt21.metashare.ilsp.gr/repository/browse/panacea-labour-english-monolingual-corpus_sample-annotated-by-
treetagger/6b5acdea196511e3841600155dbc02013b367ebe61a5405a9fdf850c5d55c529/

to view and download the resulting annotated corpus.

Thank you for using QT-LAUNCHPAD repository!
```

Figure 10

If the processing has failed (for example the service is unavailable for a long time), the user will be informed via email message.

All tests that refer to processing procedures can be found in Appendix 1, from line 408 to line 536 and from line 668 to line 710. These tests verify the cases listed below and they are validated by the following criteria: a) an email has been sent to the user and b) the output resource has been stored to the QT21 repository.

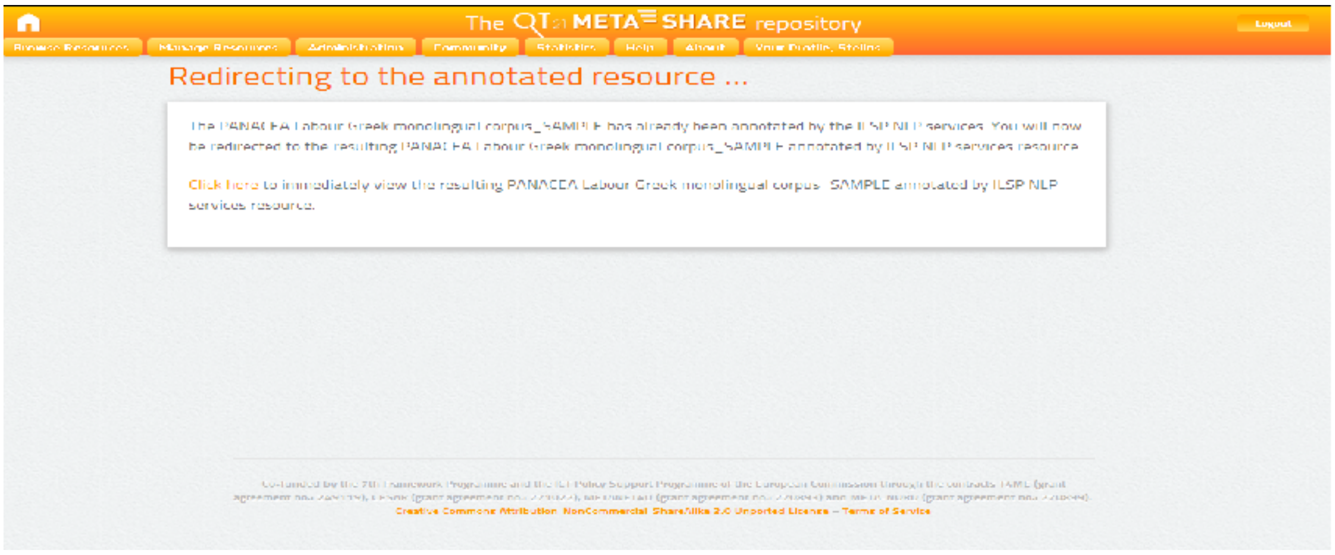
- process Greek monolingual text corpus with sentence splitter and tokenizer (line 408 – line 419)
- process Greek monolingual text corpus with chunker (line 421 – line 432)
- process Greek monolingual text corpus with lemmatizer (line 435 – line 446)
- process English monolingual text corpus with tokenizer (line 448 – line 459)
- process English monolingual text corpus with Part of Speech tagger (line 462 – line 473)
- process English monolingual text corpus with parser (line 476 – line 487)
- process English – Greek parallel text corpus with sentence splitter and tokenizer (line 490 – line 504)
- process English – Greek parallel text corpus with Part of Speech tagger (line 507 – line 521)
- process English – Greek parallel text corpus with sentence aligner (line 524 – line 536)
- Make concurrent calls (namely 1, 3, 5, 10 and 20 calls) to a Greek service

(tagger/lemmatizer) (line 672 – line 683)
- Make concurrent calls (namely 1, 3, 5, 10 and 20 calls) to an English service (treetagger) (line 685 – line 696)
- Make concurrent calls (namely 1, 3, 5, 10 and 20 calls) to a sentence alignment service (Berkeley aligner) (line 698 – line 710)

The annotated resource, which derives from a processing procedure, is stored automatically in the QT21 repository, while the equivalent metadata are also produced. The data of the output resource are stored following the same directory structure as the initial input data. Furthermore, each data filename takes an extension according to the annotation level.

All tests regarding the integrity of the output resource data (all files of the input resource have to be processed and all output files must have the appropriate extension) can be found in Appendix 1, from line 538 to line 632. These tests assert the following cases:
- a sentence-splitter and tokenizer processes a Greek monolingual text corpus and the derived resource data files are as many as the input data files and are named with the appropriate extension ('.tok') (line 538 – line 578)
- Part of Speech taggers-lemmatisers process a Greek - English parallel text corpus and the derived resource data files are as many as the input data files and are named with the appropriate extension ('.taglem') (line 580 – line 632)

If the user requests to process a resource with a specific web service, which has already been provided for another user, then the system will just forward the user to the processed resource that has been created in the repository (see Figure 11).



Redirecting to the annotated resource ...

The PANACEA Labour Greek monolingual corpus_SAMPLE has already been annotated by the ILSP NLP services. You will now be redirected to the resulting PANACEA Labour Greek monolingual corpus_SAMPLE annotated by ILSP NLP services resource.

Click here to immediately view the resulting PANACEA Labour Greek monolingual corpus_SAMPLE annotated by ILSP NLP services resource.

Co-funded by the 7th Framework Programme and the ICT Policy Support Programme of the European Commission through the contracts T4ME (grant agreement no.: [illegible]), CESAR (grant agreement no.: [illegible]), METANET4U (grant agreement no.: [illegible]) and META-NORD (grant agreement no.: [illegible]).
Creative Commons Attribution NonCommercial ShareAlike 3.0 Unported License – Terms of Service

Figure 11

The test that verifies the above procedure is presented in Appendix 1, from line 634 to line 666.

3. User Testing of the translation quality evaluation layer

NOTE (December 2014): The following instructions refer to an older version of translate5 and are not compatible with current versions, although they can be adapted to newer versions. In addition the user credentials are no longer valid. This test set will not be maintained. Note as well that translate5 is a UI-driven tool. At the time this test set was created it was not possible to create an automated test. For more information on translate5 and annotation, please visit http://www.qt21.eu/launchpad/translate5.

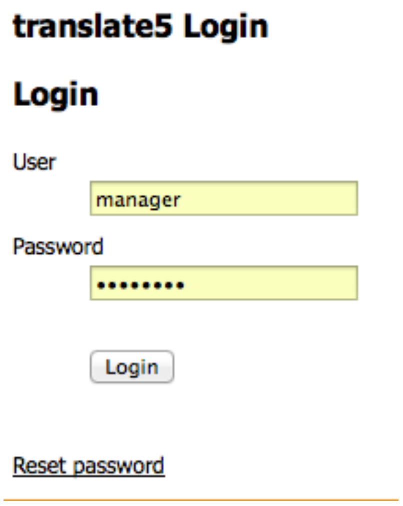
1. Go to http://www.translate5.net

2. Click on Starting the demo version:

3. Login with the following credentials:

User name: manager

Password: asdfasdf


translate5 Login

Login

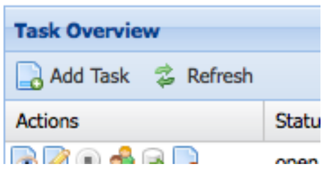
User

manager

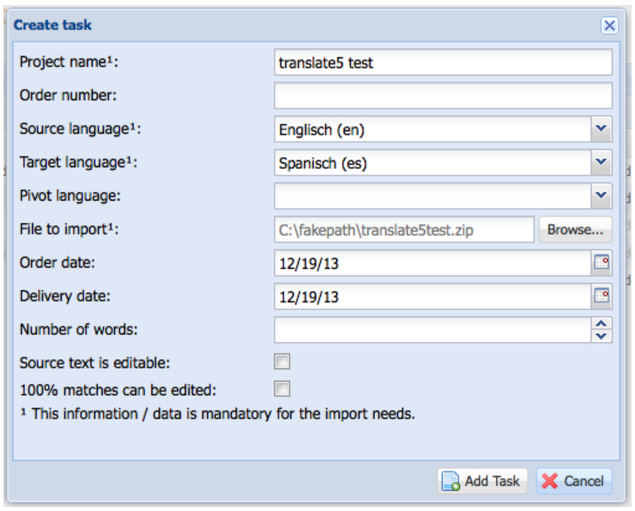
Password

••••••••

5. Click on “Add Task” in the *Task Overview* panel:



6. In the *Create task* window provide a title, specify the source (English) and target (Spanish) languages, and select the file translate5test.zip (available from http://www.qt21.eu/downloads/translate5test.zip) and then click *Add Task*.

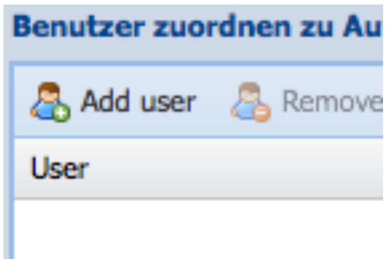


Create task

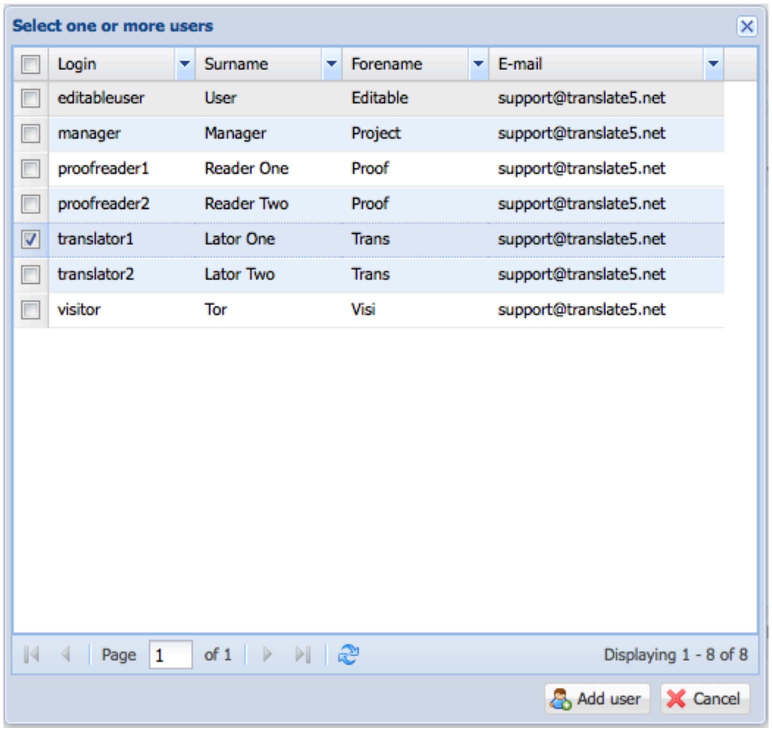
Project name¹:	translate5 test
Order number:	
Source language¹:	Englisch (en)
Target language¹:	Spanisch (es)
Pivot language:	
File to import¹:	C:\fakepath\translate5test.zip
Order date:	12/19/13
Delivery date:	12/19/13
Number of words:	
Source text is editable:	☐
100% matches can be edited:	☐

¹ This information / data is mandatory for the import needs.

7. Click *Add User*

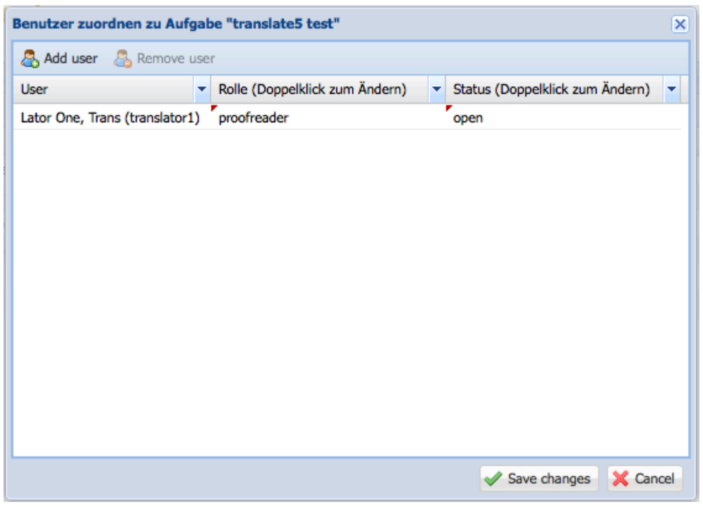


8. Select "translator1" as a user and click *Add User*:


Select one or more users

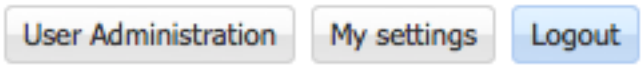
	Login	Surname	Forename	E-mail
☐	editableuser	User	Editable	support@translate5.net
☐	manager	Manager	Project	support@translate5.net
☐	proofreader1	Reader One	Proof	support@translate5.net
☐	proofreader2	Reader Two	Proof	support@translate5.net
✓	translator1	Lator One	Trans	support@translate5.net
☐	translator2	Lator Two	Trans	support@translate5.net
☐	visitor	Tor	Visi	support@translate5.net

9. Click *Save changes*:


Benutzer zuordnen zu Aufgabe "translate5 test"

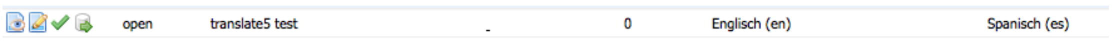
User	Rolle (Doppelklick zum Ändern)	Status (Doppelklick zum Ändern)
Lator One, Trans (translator1)	proofreader	open

10. Click *Logout* in the *Task Overview* window



11. Click *Starting the demo version* once more and repeat the login process using username "translator1" and "asdfasdf" as the password.

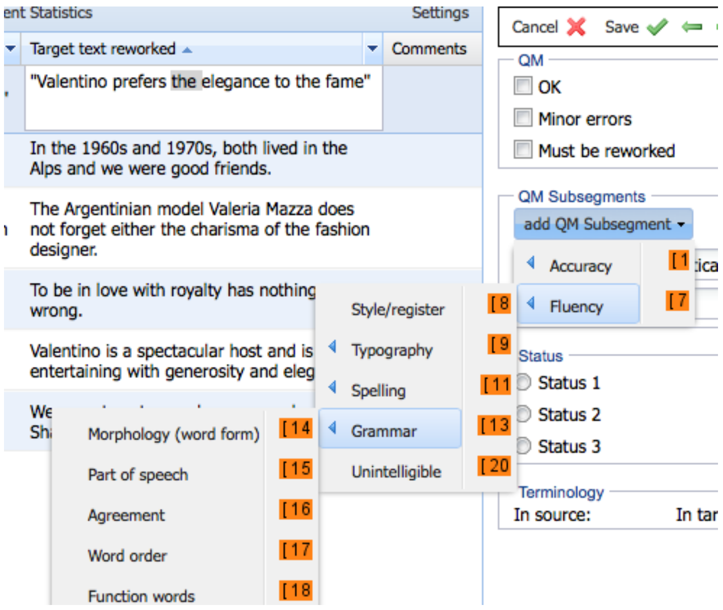
12. When you reach the *Task Overview* window you should see a listing of available projects. One of them should have the title you selected in Step 6. Click on the “Edit task” icon (second from the left) to open the project for editing:



open translate5 test - 0 Englisch (en) Spanisch (es)

13. In the *Segment list and editor* window perform the following edits by double clicking on segments in the *Target text reworked* column, highlighting text, and then selecting issues by clicking on the *Add QM Subsegment* button in the *QM Subsegments* pane and clicking Save after making the noted annotations:

- Segment 1:
  - Mark both instances of “the” with *Fluency:Grammar:Function words.*
- Segment 2:
  - Mark “has nothing” with *Accuracy:Mistranslation*.
- Segment 3:
  - Select the space between *both* and *lived* and then add “of us” to the comment field in the *QM Subsegments* pane and then mark the space with *Accuracy:Omission.*
- Segment 4:
  - Select “is entertaining” and mark it as *Fluency:Grammar:Tense/aspect/mood*
- Segment 5:
  - Select “either the charisma of the fashion designer”, then add “the charisma of the fashion designer either” to the comment field in the *QM Subsegments* pane and then mark the text with *Fluency:Grammar:Word order.*
- Segment 6:
  - Select “Sharon Stone and John Kennedy Jr were” and mark it as *Fluency:Grammar:Word order*.



"Valentino prefers the elegance to the fame"

In the 1960s and 1970s, both lived in the Alps and we were good friends.

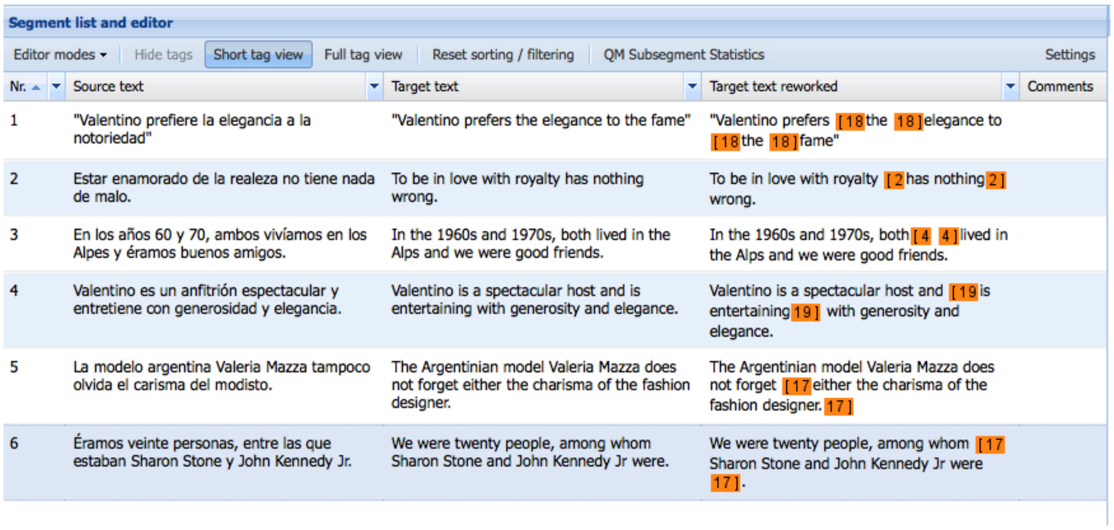
The Argentinian model Valeria Mazza does not forget either the charisma of the fashion designer.

To be in love with royalty has nothing wrong.

Valentino is a spectacular host and is entertaining with generosity and eleg

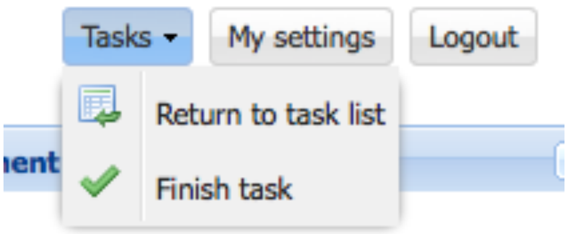
Marking issues in translate5

The results after making all edits should like the following (the sort order of the segments is insignificant):



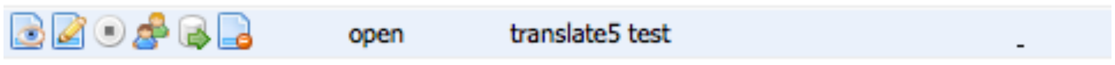
Nr.	Source text	Target text	Target text reworked	Comments
1	"Valentino prefiere la elegancia a la notoriedad"	"Valentino prefers the elegance to the fame"	"Valentino prefers [18 the 18] elegance to [18 the 18] fame"	
2	Estar enamorado de la realeza no tiene nada de malo.	To be in love with royalty has nothing wrong.	To be in love with royalty [2 has nothing 2] wrong.	
3	En los años 60 y 70, ambos vivíamos en los Alpes y éramos buenos amigos.	In the 1960s and 1970s, both lived in the Alps and we were good friends.	In the 1960s and 1970s, both [4 4] lived in the Alps and we were good friends.	
4	Valentino es un anfitrión espectacular y entretiene con generosidad y elegancia.	Valentino is a spectacular host and is entertaining with generosity and elegance.	Valentino is a spectacular host and [19 is entertaining 19] with generosity and elegance.	
5	La modelo argentina Valeria Mazza tampoco olvida el carisma del modisto.	The Argentinian model Valeria Mazza does not forget either the charisma of the fashion designer.	The Argentinian model Valeria Mazza does not forget [17 either the charisma of the fashion designer. 17]	
6	Éramos veinte personas, entre las que estaban Sharon Stone y John Kennedy Jr.	We were twenty people, among whom Sharon Stone and John Kennedy Jr were.	We were twenty people, among whom [17 Sharon Stone and John Kennedy Jr were 17].	

14. Select *Tasks: Finish task*

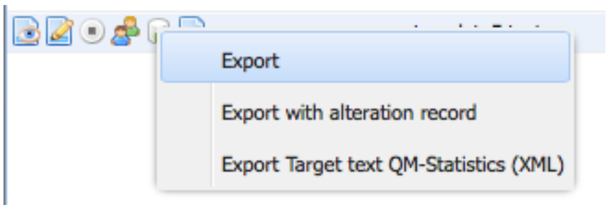


15. Logout from the *Task Overview* page and log back in as "manager".

16. In the Task Overview page, click on the export icon (second from right, depicting a round object with a green arrow) for the project created in steps 6–9:



and select "Export"


Export
Export with alteration record
Export Target text QM-Statistics (XML)

17. After completing Step 16 a zip file will download. After unzipping this file, you should have a directory containing a single CSV file. Opening the CSV file will reveal that it contains XML markup containing the MQM issue tagging. Converting it into tabular format should reveal text equivalent to the following (allowing for whitespace differences and differences in ID values):

mid	quelle	ziel
abces/2012/12/01/239729-1_es_MT2	"Valentino prefiere la elegancia a la notoriedad"	"Valentino prefers <mqm:startIssue type="Function words" severity="critical" note="" agent="Trans Lator One" id="73"/>the <mqm:endIssue id="73"/>elegance to <mqm:startIssue type="Function words" severity="critical" note="" agent="Trans Lator One" id="74"/>the <mqm:endIssue id="74"/>fame"
abces/2012/12/01/239729-10_es_MT1	Estar enamorado de la realeza no tiene nada de malo.	To be in love with royalty <mqm:startIssue type="Terminology" severity="critical" note="" agent="Trans Lator One" id="77"/>has

		nothing<mqm:endIssue id="77"/> wrong.
abces/2012/12/01/239729-12_es_MT1	En los años 60 y 70, ambos vivíamos en los Alpes y éramos buenos amigos.	In the 1960s and 1970s, both<mqm:startIssue type="Omission" severity="critical" note="of us" agent="Trans Lator One" id="75"/><mqm:endIssue id="75"/>lived in the Alps and we were good friends.
abces/2012/12/01/239729-13_es_MT2	Valentino es un anfitrión espectacular y entretiene con generosidad y elegancia.	Valentino is a spectacular host and <mqm:startIssue type="Tense/aspect/mood" severity="critical" note="" agent="Trans Lator One" id="78"/>is entertaining<mqm:endIssue id="78"/> with generosity and elegance.
abces/2012/12/01/239729-18_es_MT2	La modelo argentina Valeria Mazza tampoco olvida el carisma del modisto.	The Argentinian model Valeria Mazza does not forget <mqm:startIssue type="Word order" severity="critical" note="the charisma of the fashion designer either." agent="Trans Lator One" id="76"/>either the charisma of the fashion designer.<mqm:endIssue id="76"/>
abces/2012/12/01/239729-20_es_MT2	Éramos veinte personas, entre las que estaban Sharon Stone y John Kennedy Jr.	We were twenty people, among whom <mqm:startIssue type="Word order" severity="critical" note="" agent="Trans Lator One" id="79"/>Sharon Stone and John Kennedy Jr were<mqm:endIssue id="79"/>.

18. Verify that the MQM tagging in the downloaded file matches the markup highlighted in red above.

Appendix I

```
class ImportTest(TestCase):
    """
    Tests the import procedure for resources
    """

    test_editor_group = None
    super_user = None

    @classmethod
    def setUpClass(cls):
        LOGGER.info("running '{}' tests...".format(cls.__name__))
        test_utils.set_index_active(False)

    @classmethod
    def tearDownClass(cls):
        test_utils.set_index_active(True)
        LOGGER.info("finished '{}' tests".format(cls.__name__))

    def setUp(self):
        """
        Set up the import test
        """
        test_utils.setup_test_storage()

        ImportTest.test_editor_group = EditorGroup.objects.create(
                                                    name='test_editor_group')

        ImportTest.super_user = User.objects.create_superuser('superuser', 'su@example.com', 'secret')

    def tearDown(self):
        """
        Clean up the test
        """
        test_utils.clean_resources_db()
        test_utils.clean_storage()
        test_utils.clean_user_db()

    def _test_import_dir(self, path):
        """Asserts that all XML files in the given directory can be imported."""
        _files = os.listdir(path)
        for _file in _files:
            self._test_import_xml_file("%s%s" % (path, _file))

    def test_import_xml(self):
        """Asserts that a single XML file can be imported."""
        self._test_import_xml_file('{}/repository/fixtures/testfixture.xml'
                                   .format(ROOT_PATH))

    def _test_import_xml_file(self, xml_path):
        """Asserts that the given XML file can be imported."""
        successes, failures = test_utils.import_xml_or_zip(xml_path)
        self.assertEqual(1, len(successes),
            'Could not import file {} -- successes is {}, failures is {}'
                .format(xml_path, successes, failures))
        self.assertEqual(0, len(failures),
            'Could not import file {} -- successes is {}, failures is {}'
                .format(xml_path, successes, failures))

    def test_broken_xml(self):
        _currfile = '{}/repository/fixtures/broken.xml'.format(ROOT_PATH)
        successes, failures = test_utils.import_xml_or_zip(_currfile)
        self.assertEqual(0, len(successes), 'Should not have been able to import file {}'.format(_currfile))
        self.assertEqual(1, len(failures), 'Should not have been able to import file {}'.format(_currfile))

    def test_import_zip(self):
        _currfile = '{}/repository/fixtures/tworesources.zip'.format(ROOT_PATH)
        successes, failures = test_utils.import_xml_or_zip(_currfile)
        self.assertEqual(2, len(successes), 'Could not import file {}'.format(_currfile))
        self.assertEqual(0, len(failures), 'Could not import file {}'.format(_currfile))

    def test_import_broken_zip(self):
        _currfile = '{}/repository/fixtures/onegood_onebroken.zip'.format(ROOT_PATH)
        successes, failures = test_utils.import_xml_or_zip(_currfile)
        self.assertEqual(1, len(successes), 'Could not import file {} -- successes is {}, failures is {}'.format(_currfile, successes, failures))
        self.assertEqual(1, len(failures), 'Could not import file {} -- successes is {}, failures is {}'.format(_currfile, successes, failures))
        self.assertEquals('broken.xml', failures[0][0])

class DataUploadTests(TestCase):
    """
    Tests related to uploading actual resource data for a pure resource
    description.
    """
    # static variables to be initialized in setUpClass():
    test_editor_group = None
    editor_user = None
    editor_login = None
    superuser_login = None
    testfixture = None
    testfixture2 = None
    testfixture3 = None
    testfixture4 = None

    @classmethod
    def setUpClass(cls):
        LOGGER.info("running '{}' tests...".format(cls.__name__))
        test_utils.set_index_active(False)

        DataUploadTests.test_editor_group = \
            EditorGroup.objects.create(name='test_editor_group')

                    # test user accounts
                    DataUploadTests.editor_user = test_utils.create_editor_user(
                        'editoruser', 'editor@example.com', 'secret',
                        (DataUploadTests.test_editor_group,))
                    superuser = User.objects.create_superuser(
                        'superuser', 'su@example.com', 'secret')

                    # login POST dicts
                    DataUploadTests.editor_login = {
                        REDIRECT_FIELD_NAME: ADMINROOT,
                        LOGIN_FORM_KEY: 1,
                        'username': DataUploadTests.editor_user.username,
                        'password': 'secret',
                    }
                    DataUploadTests.superuser_login = {
                        REDIRECT_FIELD_NAME: ADMINROOT,
                        LOGIN_FORM_KEY: 1,
                        'username': superuser.username,
                        'password': 'secret',
                    }

                @classmethod
                def tearDownClass(cls):
                    test_utils.clean_user_db()
                    test_utils.set_index_active(True)
                    LOGGER.info("finished '{}' tests".format(cls.__name__))

                def setUp(self):
                    test_utils.setup_test_storage()
                    # first test resource
                    DataUploadTests.testfixture = \
                        _import_test_resource(DataUploadTests.test_editor_group)
                    # second test resource which is only visible by the superuser
                    DataUploadTests.testfixture2 = \
                        _import_test_resource(None, TESTFIXTURE2_XML)
                    # third test resource which is owned by the editoruser
                    DataUploadTests.testfixture3 = \
                        _import_test_resource(None, TESTFIXTURE3_XML, pub_status=PUBLISHED)
                    DataUploadTests.testfixture3.owners.add(DataUploadTests.editor_user)
                    DataUploadTests.testfixture3.save()
                    # fourth test resource which is owned by the editor user
                    DataUploadTests.testfixture4 = _import_test_resource(
                        DataUploadTests.test_editor_group, TESTFIXTURE4_XML,
                        pub_status=INTERNAL)
                    DataUploadTests.testfixture4.owners.add(DataUploadTests.editor_user)
                    DataUploadTests.testfixture4.save()

                def tearDown(self):
                    # we have to clean up both the resources DB and the storage folder as
                    # uploads change the storage folder
                    test_utils.clean_resources_db()
                    test_utils.clean_storage()

                def test_superuser_can_always_upload_data(self):
                    """
                    Verifies that a superuser may upload actual resource data to any
                    resource.
                    """
                    client = test_utils.get_client_with_user_logged_in(
                        DataUploadTests.superuser_login)
                    for res in (DataUploadTests.testfixture, DataUploadTests.testfixture2,
                                DataUploadTests.testfixture3, DataUploadTests.testfixture4):
                        self.assertFalse(res.storage_object.get_download(),
                            "the test LR must not have a local download before the test")
                        _data_upload_url = "{}repository/resourceinfotype_model/{}/" \
                            "upload-data/".format(ADMINROOT, res.id)
                        response = client.get(_data_upload_url)
                        self.assertEquals(200, response.status_code, "expected that a " \
                                          "superuser may always upload resource data")
                        with open(DATA_UPLOAD_ZIP, 'rb') as _fhandle:
                            response = client.post(_data_upload_url,
                                {'uploadTerms': 'on', 'resource': _fhandle}, follow=True)
                            self.assertContains(response, "was changed successfully.")
                        # refetch the storage object from the data base to make sure we have
                        # the latest version
                        _so = StorageObject.objects.get(pk=res.storage_object.pk)
                        self.assertTrue(_so.get_download(),
                            "uploaded LR data must be available as a local download")

                def test_editor_can_upload_data(self):
                    """
                    Verifies that an editor user may upload actual resource data to owned
                    resources and to resources that are in her editor group.
                    """
                    client = test_utils.get_client_with_user_logged_in(
                        DataUploadTests.editor_login)
                    # make sure data can be uploaded to owned resources:
                    self.assertFalse(DataUploadTests.testfixture3.storage_object \
                            .get_download(),
                        "the test LR must not have a local download before the test")
                    _data_upload_url = "{}repository/resourceinfotype_model/{}/" \
                        "upload-data/".format(ADMINROOT, DataUploadTests.testfixture3.id)
                    response = client.get(_data_upload_url)
                    self.assertEquals(200, response.status_code, "expected that an " \
                                     "editor may upload resource data to owned resources")
                    with open(DATA_UPLOAD_ZIP, 'rb') as _fhandle:
                        response = client.post(_data_upload_url,
                            {'uploadTerms': 'on', 'resource': _fhandle}, follow=True)
                        self.assertContains(response, "was changed successfully.")
                    # refetch the storage object from the data base to make sure we have
                    # the latest version
                    _so = StorageObject.objects.get(
                        pk=DataUploadTests.testfixture3.storage_object.pk)
                    self.assertTrue(_so.get_download(),
                        "expected that uploaded LR data is available as a local download")
                    # make sure data can be uploaded to editable resources:
                    self.assertFalse(
                        DataUploadTests.testfixture.storage_object.get_download(),
                        "the test LR must not have a local download before the test")
                    _data_upload_url = "{}repository/resourceinfotype_model/{}/" \
                        "upload-data/".format(ADMINROOT, DataUploadTests.testfixture.id)
                    response = client.get(_data_upload_url)
                    self.assertEquals(200, response.status_code, "expected that an " \
                        "editor may upload resource data to resources of her editor group")
                    with open(DATA_UPLOAD_ZIP, 'rb') as _fhandle:
                        response = client.post(_data_upload_url,
                            {'uploadTerms': 'on', 'resource': _fhandle}, follow=True)
                        self.assertContains(response, "was changed successfully.")
                    # refetch the storage object from the data base to make sure we have
                    # the latest version
                    _so = StorageObject.objects.get(
                        pk=DataUploadTests.testfixture.storage_object.pk)
                    self.assertTrue(_so.get_download(),
                        "expected that uploaded LR data is available as a local download")

                def test_editor_can_update_own_upload_data(self):
                    """
                    Verifies that an editor user may update actual resource data by
                    re-uploading to owned resources.
                    """
                    client = test_utils.get_client_with_user_logged_in(
                        DataUploadTests.editor_login)
                    # add resource data first which can be updated later:
                    shutil.copy(DATA_UPLOAD_ZIP, u'{}/archive.zip'.format(
                            DataUploadTests.testfixture3.storage_object._storage_folder()))
                    DataUploadTests.testfixture3.storage_object.compute_checksum()
                    DataUploadTests.testfixture3.storage_object.save()
                    # make sure that there is really resource data stored now:
                    _so = StorageObject.objects.get(
                        pk=DataUploadTests.testfixture3.storage_object.pk)
                    _old_checksum = _so.checksum
                    self.assertNotEqual(None, _old_checksum)
                    # vreify that the resource data update works:
                    _data_upload_url = "{}repository/resourceinfotype_model/{}/" \
                        "upload-data/".format(ADMINROOT, DataUploadTests.testfixture3.id)
                    response = client.get(_data_upload_url)
                    self.assertContains(response,
                        DataUploadTests.testfixture3.storage_object.get_download())
                    with open(DATA_UPLOAD_ZIP_2, 'rb') as _fhandle:
                        response = client.post(_data_upload_url,
                            {'uploadTerms': 'on', 'resource': _fhandle}, follow=True)
                        self.assertContains(response, "was changed successfully.")
                    _so = StorageObject.objects.get(
                        pk=DataUploadTests.testfixture3.storage_object.pk)
                    self.assertNotEqual(_so.checksum, _old_checksum)

                def test_editor_cannot_upload_data_to_invisible_resources(self):
                    """
                    Verifies that an editor user must not upload actual resource data to
                    resources that do not belong to her editor group and which are not hers.
                    """
                    client = test_utils.get_client_with_user_logged_in(
                        DataUploadTests.editor_login)
                    response = client.get("{}repository/resourceinfotype_model/{}/" \
                        "upload-data/".format(ADMINROOT, DataUploadTests.testfixture2.id))
                    self.assertIn(response.status_code, (403, 404), "expected that an " \
                        "editor must not upload resource data to invisible resources")

            class ProcessingTests(TestCase):

                @classmethod
                def setUpClass(cls):
                    LOGGER.info("running '{}' tests...".format(cls.__name__))
                    test_utils.set_index_active(False)

                @classmethod
                def tearDownClass(cls):
                    test_utils.set_index_active(True)
                    LOGGER.info("finished '{}' tests".format(cls.__name__))

                def setUp(self):
                    """
                    Creates a new storage object instance for testing.
                    """
                    test_utils.setup_test_storage()

                    self.user = User.objects.create_superuser("superuser", "su@example.com", "secret")
                    self.client = Client()
                    self.client.login(username=self.user.username, password='secret')

                def tearDown(self):
                    """
                    Removes the storage object instance from the database after testing.
                    """
                    test_utils.clean_resources_db()
                    test_utils.clean_user_db()
                    test_utils.clean_storage()

                def add_corpus(self, xmlfile, archive):
                    '''adds corpus (metadata and data)'''
                    res = test_utils.import_xml(
                        '{0}/processing/fixtures/{1}'.format(settings.ROOT_PATH, xmlfile))
                    res.storage_object.publication_status = PUBLISHED
                    res.storage_object.update_storage()

                    data_upload_url = "{}repository/resourceinfotype_model/{}/" \
                                "upload-data/".format(ADMINROOT, res.id)
                    response = self.client.get(data_upload_url)

                    with open('{0}/processing/fixtures/{1}'.format(settings.ROOT_PATH, archive), 'rb') as fhandle:
                                response = self.client.post(data_upload_url,
                                    {'uploadTerms': 'on', 'resource': fhandle}, follow=True)
                    return res

                def add_service(self, xmlfile):
                    '''adds service's metadata '''
                    res = test_utils.import_xml(
                        '{0}/processing/fixtures/{1}'.format(settings.ROOT_PATH, xmlfile))
                    res.storage_object.publication_status = PUBLISHED
                    res.storage_object.update_storage()
                    return res

                def assert_monolingual_corpus_processing(self, _corpusxmlfile, _corpusarchive, _servicefile):
                    _corpus = self.add_corpus(_corpusxmlfile, _corpusarchive)
                    _service = self.add_service(_servicefile)

                    success = ProcessingTask.delay(self.user, _corpus, _service)
                    mail_sent, output_resource_url = success.get()
                    response = self.client.get(output_resource_url)

                    self.assertTrue(success.successful())
                    self.assertEquals(len(mail_sent.outbox), 1)
                    self.assertEquals(mail_sent.outbox[0].subject, 'QT-LAUNCHPAD annotated resource is ready')
                    self.assertEqual(response.status_code, 200)

                def assert_parallel_corpus_processing(self, _corpusxmlfile, _corpusarchive, _firstservicefile, _secondservicefile=None):
                    _corpus = self.add_corpus(_corpusxmlfile, _corpusarchive)
                    _firstservice = self.add_service(_firstservicefile)
                    _secondservice = None
                    if not _secondservicefile is None:
                        _secondservice = self.add_service(_secondservicefile)

                    success = ParallelProcessingTask.delay(self.user, _corpus, _firstservice, _secondservice)
                    mail_sent, output_resource_url = success.get()
                    response = self.client.get(output_resource_url)

                    self.assertTrue(success.successful())
                    self.assertEquals(len(mail_sent.outbox), 1)
                    self.assertEquals(mail_sent.outbox[0].subject, 'QT-LAUNCHPAD annotated resource is ready')
                    self.assertEqual(response.status_code, 200)

                def test_select_right_tools1(self):
                    '''
                    Verifies that the relevant services, regarding the language,
                    the format and the annotation level of the input resource,
                    show up
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus = self.add_corpus('el_corpus.xml', 'ENV_EL.tar.gz')

                    _corpus_id = _corpus.storage_object.identifier

                    response = self.client.post('{0}/repository/processing/{1}/'.format(DJANGO_URL, _corpus_id))

                    ILSP_services = ['ILSP Sentence splitting and tokenization','ILSP FBT Parser', 'ILSP Lemmatization',
                                     'ILSP NLP services', 'ILSP Chunking','ILSP Dependency Parsing',
                                     'ILSP Named Entity Recognition']
                    for service in ILSP_services:
                        self.assertContains(response, service, status_code=200)

                def test_select_right_tools2(self):
                    '''
                    Verifies that the relevant services, regarding the language,
                    the format and the annotation level of the input resource,
                    show up
                    '''
                    #add PANACEA Labour English monolingual corpus_SAMPLE
                    _corpus = self.add_corpus('en_corpus.xml', 'LAB_EN.tar.gz')

                    _corpus_id = _corpus.storage_object.identifier

                    response = self.client.post('{0}/repository/processing/{1}/'.format(DJANGO_URL, _corpus_id))

                    en_services = ['europarl_tokeniser', 'europarl_sentence_splitter', 'berkeley_tagger',
                                       'treetagger', 'berkeley_parser']
                    for service in en_services:
                        self.assertContains(response, service, status_code=200)

                def test_select_right_tools3(self):
                    '''
                    Verifies that the relevant services, regarding the language,
                    the format and the annotation level of the input resource,
                    show up
                    '''
                    #add PANACEA English-Greek parallel corpus acquired
                    #for Labour Legislation domain_SAMPLE
                    _corpus = self.add_corpus('en_el_corpus.xml', 'LAB_EN_EL.tar.gz')

                    _corpus_id = _corpus.storage_object.identifier

                    response = self.client.post('{0}/repository/processing/annotation-levels/{1}/'.format(DJANGO_URL, _corpus_id),
                                        {'annotation_level' : 'chunking'}, follow=True)

                    chunkers = ['ILSP Chunker', 'berkeley_parser']
                    for service in chunkers:
                        self.assertContains(response, service, status_code=200)

                def test_processing_el_corpus_with_ilsp_sst(self):
                    '''
                    Verifies the processing of the PANACEA Environment Greek monolingual corpus_SAMPLE
                    by the ILSP Sentence splitter and tokenizer
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'el_corpus.xml'
                    _corpus_archive = 'ENV_EL.tar.gz'
                    #add ILSP Sentence splitter and tokenizer
                    _ilsp_sst = 'ilsp_sst.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _ilsp_sst)

                def test_processing_el_corpus_with_ilsp_chunker(self):
                    '''
                    Verifies the processing of the PANACEA Environment Greek monolingual corpus_SAMPLE
                    by the ILSP Chunker
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'el_corpus.xml'
                    _corpus_archive = 'ENV_EL.tar.gz'
                    #add ILSP Chunker
                    _ilsp_chunker = 'ilsp_chunker.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _ilsp_chunker)


                def test_processing_el_corpus_with_ilsp_lemmatizer(self):
                    '''
                    Verifies the processing of the PANACEA Environment Greek monolingual corpus_SAMPLE
                    by the ILSP Lemmatizer
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'el_corpus.xml'
                    _corpus_archive = 'ENV_EL.tar.gz'
                    #add ILSP Lemmatizer
                    _ilsp_lemmatizer = 'ilsp_lemmatizer.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _ilsp_lemmatizer)

                def test_processing_en_corpus_with_europarl_tokeniser(self):
                    '''
                    Verifies the processing of the PANACEA Labour English monolingual corpus_SAMPLE
                    by the europarl tokeniser
                    '''
                    #add PANACEA Labour English monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'en_corpus.xml'
                    _corpus_archive = 'LAB_EN.tar.gz'
                    #add europarl tokeniser
                    _europarl_tokeniser = 'europarl_tokeniser.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _europarl_tokeniser)


                def test_processing_en_corpus_with_treetagger(self):
                    '''
                    Verifies the processing of the PANACEA Labour English monolingual corpus_SAMPLE
                    by the treetagger
                    '''
                    #add PANACEA Labour English monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'en_corpus.xml'
                    _corpus_archive = 'LAB_EN.tar.gz'
                    #add treetagger
                    _treetagger = 'treetagger.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _treetagger)


                def test_processing_en_corpus_with_berkeley_parser(self):
                    '''
                    Verifies the processing of the PANACEA Labour English monolingual corpus_SAMPLE
                    by the berkeley parser
                    '''
                    #add PANACEA Labour English monolingual corpus_SAMPLE
                    _corpus_xmlfile = 'en_corpus.xml'
                    _corpus_archive = 'LAB_EN.tar.gz'
                    #add berkeley parser
                    _berkeley_parser = 'berkeley_parser.xml'

                    self.assert_monolingual_corpus_processing(_corpus_xmlfile, _corpus_archive, _berkeley_parser)


                def test_processing_en_el_corpus_with_sentence_splitters(self):
                    '''
                    Verifies the processing of the PANACEA English-Greek parallel corpus acquired
                    for Labour Legislation domain_SAMPLE by sentence splitters
                    '''
                    #add PANACEA English-Greek parallel corpus acquired
                    #for Labour Legislation domain_SAMPLE
                    _corpus_xmlfile = 'en_el_corpus.xml'
                    _corpus_archive = 'LAB_EN_EL.tar.gz'
                    #add ILSP Sentence splitter and tokenizer
                    _ilsp_sst = 'ilsp_sst.xml'
                    #add europarl sentence splitter
                    _europarl_sentence_splitter = 'europarl_sentence_splitter.xml'

                    self.assert_parallel_corpus_processing(_corpus_xmlfile, _corpus_archive, _ilsp_sst, _europarl_sentence_splitter)


                def test_processing_en_el_corpus_with_pos_taggers(self):
                    '''
                    Verifies the processing of the PANACEA English-Greek parallel corpus acquired
                    for Labour Legislation domain_SAMPLE by POS taggers/lemmatizers
                    '''
                    #add PANACEA English-Greek parallel corpus acquired
                    #for Labour Legislation domain_SAMPLE
                    _corpus_xmlfile = 'en_el_corpus.xml'
                    _corpus_archive = 'LAB_EN_EL.tar.gz'
                    #add ILSP NLP services
                    _ilsp_nlp = 'ilsp_nlp.xml'
                    #add treetagger
                    _treetagger = 'treetagger.xml'

                    self.assert_parallel_corpus_processing(_corpus_xmlfile, _corpus_archive, _ilsp_nlp, _treetagger)


                def test_processing_en_el_corpus_with_sentence_aligner(self):
                    '''
                    Verifies the processing of the PANACEA English-Greek parallel corpus acquired
                    for Labour Legislation domain_SAMPLE by sentence aligner
                    '''
                    #add PANACEA English-Greek parallel corpus acquired
                    #for Labour Legislation domain_SAMPLE
                    _corpus_xmlfile = 'en_el_corpus.xml'
                    _corpus_archive = 'LAB_EN_EL.tar.gz'
                    #add hunalign
                    _hunalign = 'hunalign.xml'

                    self.assert_parallel_corpus_processing(_corpus_xmlfile, _corpus_archive, _hunalign)

                def test_output_files1(self):
                    '''
                    Verifies that all the input resource's data has been annotated and that
                    the output resource filenames have the right extension
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus = self.add_corpus('el_corpus.xml', 'ENV_EL.tar.gz')
                    _corpus_data_path = _corpus.storage_object.get_download()
                    #add ILSP sentence splitter and tokenizer
                    _ilsp_sst = self.add_service('ilsp_sst.xml')

                    success = ProcessingTask.delay(self.user, _corpus, _ilsp_sst)
                    self.assertTrue(success.successful())

                    processing = Processing.objects.get(input_resource=_corpus, first_tool=_ilsp_sst)
                    output_resource = processing.output_resource
                    output_data_path = output_resource.storage_object.get_download()
                    input_archive = tarfile.open(_corpus_data_path, "r:gz")
                    input_filenames = []
                    for filename in input_archive.getnames():
                        if len(filename.split('.')) == 1:
                            continue
                        input_filenames.append(os.path.basename(filename))
                    input_archive.close()

                    output_archive = tarfile.open(output_data_path, "r:gz")
                    output_filenames = []
                    _extension  = 'tok'
                    for filename in output_archive.getnames():
                        if len(filename.split('.')) == 1:
                            continue
                        output_filenames.append(os.path.basename(filename))
                    output_archive.close()

                    self.assertEqual(len(input_filenames), len(output_filenames))
                    #verifies that with the deletion of the output file extension
                    #the filenames of the input and output resource are the same
                    output_filenames_noextension = []
                    for filename in output_filenames:
                        output_filenames_noextension.append(filename.replace('.{0}'.format(_extension),'') )
                    self.assertSetEqual(set(input_filenames), set(output_filenames_noextension))

                def test_output_files2(self):
                    '''
                    Verifies that all the input resource's data has been annotated and that
                    the output resource filenames have the right extension
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus = self.add_corpus('en_el_corpus.xml', 'LAB_EN_EL.tar.gz')
                    _corpus_data_path = _corpus.storage_object.get_download()
                    #add ILSP NLP services
                    _ilsp_nlp = self.add_service('ilsp_nlp.xml')
                    #add treetagger
                    _treetagger = self.add_service('treetagger.xml')

                    success = ParallelProcessingTask.delay(self.user, _corpus, _ilsp_nlp, _treetagger)
                    self.assertTrue(success.successful())

                    processing = Processing.objects.get(input_resource=_corpus, first_tool=_ilsp_nlp, second_tool=_treetagger)
                    output_resource = processing.output_resource
                    output_data_path = output_resource.storage_object.get_download()

                    input_archive = tarfile.open(_corpus_data_path)
                    input_filenames = []
                    for filename in input_archive.getnames():
                        if len(filename.split('.')) == 1:
                            continue
                        input_filenames.append(os.path.basename(filename))
                    input_archive.close()

                    output_archive = tarfile.open(output_data_path,"r:gz")
                    output_filenames = []
                    output_en_filenames = []
                    output_el_filenames = []
                    _extension  = 'taglem'
                    for filename in output_archive.getnames():
                        if len(filename.split('.')) == 1:
                            continue
                        elif 'en' in filename:
                            output_en_filenames.append(os.path.basename(filename))
                        elif 'el' in filename:
                            output_el_filenames.append(os.path.basename(filename))
                        else:
                            output_filenames.append(os.path.basename(filename))
                    output_archive.close()

                    self.assertEqual(len(input_filenames), len(output_filenames))
                    self.assertEqual(len(input_filenames), len(output_en_filenames))
                    self.assertEqual(len(input_filenames), len(output_el_filenames))
                    #verifies that with the deletion of the output file extension
                    #the filenames of the input and output resource are the same
                    output_filenames_noextension = []
                    for filename in output_filenames:
                        output_filenames_noextension.append(filename.replace('.{0}'.format(_extension),'') )
                    self.assertSetEqual(set(input_filenames), set(output_filenames_noextension))

                def test_re_processing(self):
                    '''
                    Verifies that when requested processing has already been executed,
                    the user will be informed and redirected to the output resource
                    '''
                    #add PANACEA Environment Greek monolingual corpus_SAMPLE
                    _corpus = self.add_corpus('el_corpus.xml', 'ENV_EL.tar.gz')
                    #add ILSP Feature-based multi-tiered POS Tagger
                    _ilsp_fbt = self.add_service('ilsp_fbt.xml')

                    success = ProcessingTask.delay(self.user, _corpus, _ilsp_fbt)
                    self.assertTrue(success.successful())

                    _corpus_id = _corpus.storage_object.identifier
                    _corpus_name = _corpus.identificationInfo.get_default_resourceName()
                    _ilsp_fbt_name = _ilsp_fbt.identificationInfo.get_default_resourceName()

                    processing = Processing.objects.get(input_resource=_corpus, first_tool=_ilsp_fbt)
                    output_resource = processing.output_resource
                        output_resource_name = output_resource.identificationInfo.get_default_resourceName()
                    output_data_path = output_resource.storage_object.get_download()

                    self.assertEqual(output_resource_name, '{0} annotated by {1}'.format(_corpus_name, _ilsp_fbt_name))
                    self.assertTrue(os.path.exists(output_data_path))


                    response = self.client.post('{0}/repository/processing/{1}/'.format(DJANGO_URL, _corpus_id),
                                  {'tool' : _ilsp_fbt_name}, follow=True)
                    text = 'The {0} has already been annotated by the {1}.'.format(_corpus_name,
                                                              _ilsp_fbt_name)
                    self.assertEqual('repository/processing_message.html',
                      response.templates[0].name)
                    self.assertContains(response, text,status_code=200)

                class processCorpusTest(unittest.TestCase):
                        files = ['1.xml','2.xml','3.xml','5.xml','6.xml','8.xml','9.xml','10.xml','11.xml','13.xml','14.xml','15.xml','16.xml','17.xml',
                                                '18.xml','19.xml','20.xml','22.xml','23.xml','27.xml']

                def testEN(self):
                    process = ProcessMonolingual()
                                    result=0
                                    homeFolder="/home/QTLP_shared/LAB_en"
                                    url="http://www.cngl.ie/panacea-soaplab2-axis/services/panacea.treetagger"
                    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as e:
                        future_to_res = {e.submit(process.process(homeFolder, file, url)): file for file in files}
                        for future in concurrent.futures.as_completed(future_to_res):
                                                res = future_to_res[future]
                                                result+=1
                    print result
                    self.assertEqual(result, 20, "Failed Test")

                def testGR(self):
                    process = ProcessMonolingual()
                    result=0
                                    homeFolder="/home/QTLP_shared/LAB_el"
                                    url="http://nlp.ilsp.gr/soaplab2-axis/typed/services/getstarted.ilsp_nlp?wsdl"
                    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as e:
                        future_to_res = {e.submit(process.process(homeFolder, file, url)): file for file in files}
                        for future in concurrent.futures.as_completed(future_to_res):
                                                res = future_to_res[future]
                                                result+=1
                    print result
                    self.assertEqual(result, 20, "Failed Test")

              def testEL-EN_sentalign(self):
                    process = ProcessAlignment()
                    result=0
                                    homeFolder="/home/QTLP_shared/el_en"
                    sentsplit_url="http://www.cngl.ie/panacea-soaplab2-axis/services/panacea.europarl_sentence_splitter"
                    align_url="http://www.cngl.ie/panacea-soaplab2-axis/services/panacea.berkeley_aligner"
                    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as e:
                        future_to_res = {e.submit(process.process(homeFolder, file, sentsplit_url, align_url)): file for file in files}
                        for future in concurrent.futures.as_completed(future_to_res):
                            res = future_to_res[future]
                            result+=1
                    print result
                    self.assertEqual(result, 20, "Failed Test")
```