ACCEPT

SEVENTH FRAMEWORK PROGRAMME

THEME ICT-2011.4.2(a)

Language Technologies

ACCEPT

Automated Community Content Editing PorTal

www.accept.unige.ch

Starting date of the project: 1 January 2012

Overall duration of the project: 36 months

Browser-based client demonstrator and adapted post-editing environment and evaluation portal prototypes

Workpackage n° 5	Name: Portal Integration
Deliverable n° 5.6	Name: Browser-based client demonstrator and adapted post-editing environment and evaluation portal prototypes
Due date: 31 December 2013	Submission date: 19 December 2013

Dissemination level: PU

Organisation name of lead contractor for this deliverable: SYMANTEC

The research leading to these results has received funding from the European Community's Seventh Framework Programme (*FP7/2007-2013*) under *grant agreement n° 288769.*

SEVENTH FRAMEWORK PROGRAMME

Table of Contents

List of Figures

Foreword

As agreed with the Project Officer on 7 May 2013, the original deliverables D5.4 (Browser-based client demonstrator used to access acrolinx IQ server), D5.5 (Adapted Post-Editing Environment prototype) and D5.6 (Adapted evaluation portal prototype) are being merged into the present, common deliverable (D5.6).

1 Objectives and Structure of the Deliverable

The main objective of this deliverable, which encompasses Tasks 5.1, 5.2 and 5.3, was to refine the prototypes that had been developed in Year 1. Specifically, the main goal of Task 5.1 was to transform the Year 1 checking client prototype into a full-fledged demonstrator in order to meet the requirements of the evaluation work carried out in WP9. The main goal of Task 5.2 was to refine the Year 1 post-editing environment prototype by taking into account the feedback received through the Year 1 User Studies (see deliverables D7.1.1 and D8.1.1) as well as the feedback collected through our Special Interest Group (see Task 10.2). The main goal of Task 5.3 was to refine the Year 1 evaluation framework prototype, by taking into account user requirements originating from Tasks 8.2, 9.2 and 9.3.

The deliverable is structured as follows: a quick summary of the ACCEPT infrastructure is presented before describing all of the ACCEPT components in detail. These components are divided into three main parts: the Pre-Edit components, the Post-Edit components and the Evaluation components. These components are described and discussed in separate sections.

2 Overview of the ACCEPT Infrastructure

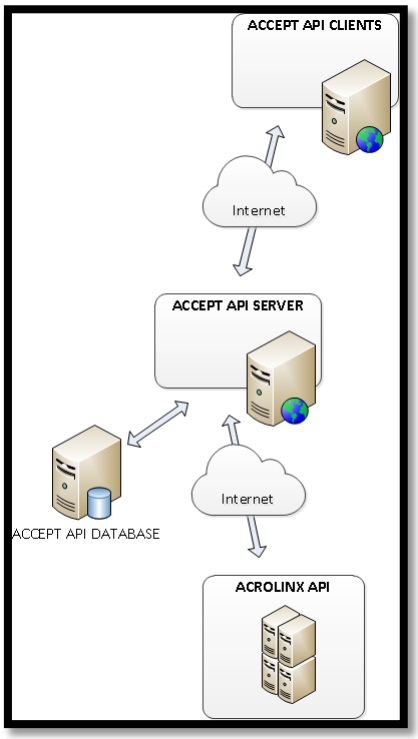


Figure 1: The ACCEPT architecture

The ACCEPT infrastructure comprises three main modules (Pre-Edit, Post-Edit and Evaluation), each of which is made of multiple components (e.g., API, plug-in, portal section). A fourth module is based

on a customised version of the Appraise system and is only available as a portal section. The Appraise system is an open-source tool for manual evaluation of Machine Translation output (Federmann, 2012). Since this module may be used to perform evaluation tasks, it will be covered in the Evaluation section of this deliverable. An instance of the ACCEPT API has been deployed alongside the ACCEPT Portal (www.accept-portal.eu and www.accept-portal.com). The ACCEPT architecture is presented in Figure 1.

While the ACCEPT infrastructure is quite simple, a user who visits the ACCEPT portal for the first time may be overwhelmed by the amount of functionality available, as shown below:

Figure 2: The ACCEPT Portal landing page

In order to help first-time users navigate the ACCEPT Portal, a quick-start guide has been added to the Learn section of the Portal, as shown below:

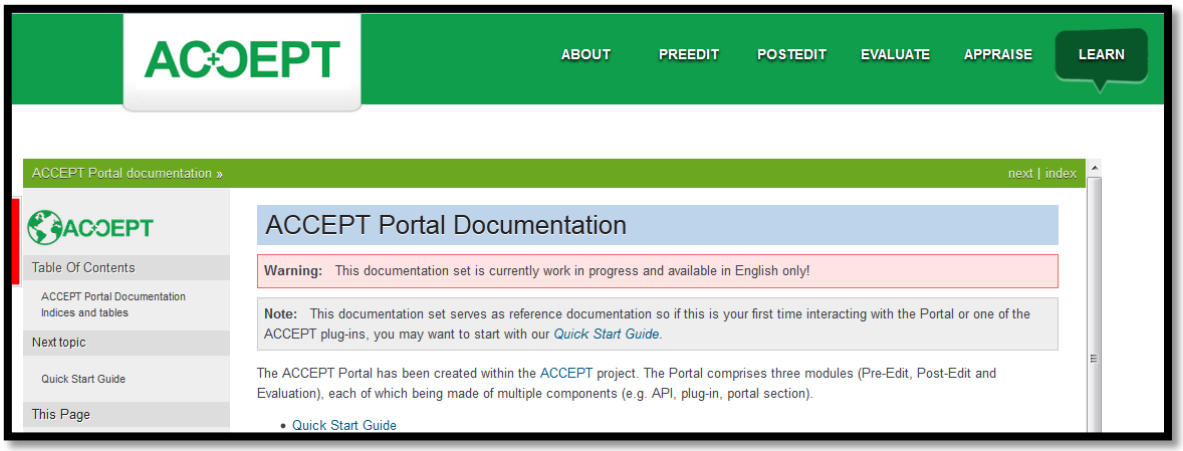


Figure 3: The Learn section of the ACCEPT Portal

From a technical perspective, the content of the Learn section is created using the reStructuredText format and Sphinx. This technology allows the creation of self-contained topics written in a human-readable format, which can then be combined in multiple ways in order to produce various documentation sets (e.g., HTML, PDF, etc). The main points of the quick-start guide are summarized in Appendix 3: First Steps with the ACCEPT Portal and Plug-ins.

3 The ACCEPT Pre-Edit Components

This section presents the new API methods that have been introduced to extend the ACCEPT Pre-Edit functionality and the modifications that have been made to the Pre-Edit plug-in and demo.

3.1 The ACCEPT Pre-Edit API

The ACCEPT Pre-Edit API can be easily described as a piece of software built as a wrapper for the Acrolinx services. The ACCEPT Pre-Edit API combines the main features available within the Acrolinx services in an independent REST (Representational State Transfer) API. This includes the following functionalities: spell checking, grammar checking and style checking. Two new Pre-Edit API methods have been added to retrieve the usage information of a Pre-Edit plug-in instance. The methods are **SimpleGlobalSessionDomain** and **GlobalSessionDomain**.

3.1.1 Overview of the New API Methods

A global session corresponds to what happens between the time when a Pre-Edit client is opened and when it is closed by a user, as detailed in the diagram below:

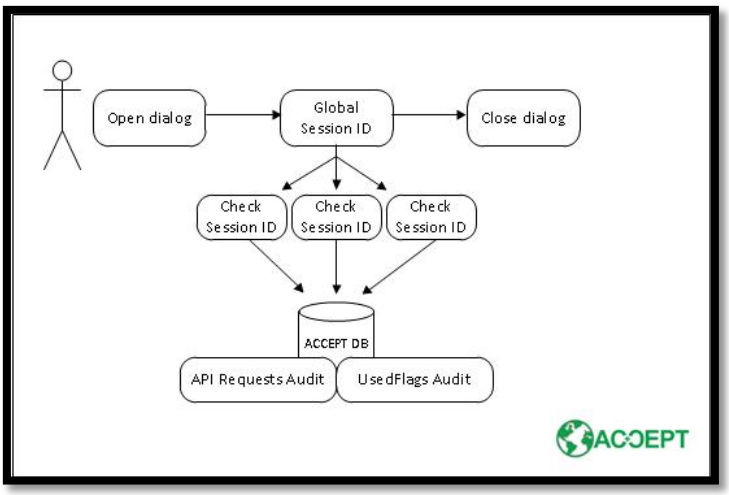


Figure 4: The ACCEPT Pre-Edit session schema

A global session starts when a user triggers a check (when the main window of the Pre-Edit plug-in opens) and ends when the user closes the window. A global session may have more than one child session (e.g. auto-check, manual re-check). Each check is associated with an API request audit trace and with the Acrolinx response. Global session information for a given Pre-Edit client instance can be retrieved from a given date up until the present using the following request (which will return a response in JSON format):

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Core/SimpleGlobalSessionDomain/?id=[instance_id]&start=[date]&end=[date]&userKey=[user_key],
```

where [accept_portal_server] is the name or IP address of the server running the ACCEPT portal and [api_version] is the version of the API (current is 1). [instance_id], [date] and [user_key] correspond to values for the following parameters:

- id: API key used by one or more client instance
- start: starting date for range in UTC format
- end: end date for range in UTC format
- userKey: key of user who created the client instance.

Note: This request may return usage data for multiple languages (e.g. EN, FR or DE) if the API key is shared by multiple client instances. To obtain specific usage data (say, for a given language), filtering on rule set names or language information may be required. To export the usage data in CSV format, the format parameter is specified as follows:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Core/SimpleGloba
lSessionDomain/?id=[instance_id]&start=[date]&end=[date]&userKey=[user_key]
&format=csv
```

To export the data in Excel format, the format parameter is specified as follows:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Core/SimpleGloba
lSessionDomain/?id=[instance_id]&start=[date]&end=[date]&userKey=[user_key]
&format=excel
```

Note: The **SimpleGlobalSessionDomain** method only generates global session data. To get the data including child session information, which is much more detailed since it contains all of the checks that were made in a given global session, the **GlobalSessionDomain** may be used as follows:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Core/GlobalSessi
onDomain/?id=[instance_id]&start=[date]&end=[date]&userKey=[user_key]
```

- The **GlobalSessionDomain** method does not currently accept any format parameter, its only response format being a JSON format, but this may change in the future. Detailed information on the response format is provided in Appendix 4: Overview of the JSON Response Format.

3.2 The ACCEPT Pre-Edit Plug-in

The ACCEPT plug-in uses the CORS (Cross Origin Resource Sharing) mechanism, which confines the Pre-Edit plug-in support to the following browsers:

- Mozilla Firefox 3.6.28+
- Internet Explorer 8+
- Safari 4+
- Opera 12+
- Chrome 20+.

Support for Internet Explorer 8 was added in order to ensure that the most common browsers would be supported.

3.2.1 Installation and Configuration

The plug-in can be installed in a number of ways, such as using a context menu or embedding into a text editor, as detailed in Deliverable 5.1. Another way to achieve the installation is to perform an external call integration. An external call integration allows the developer to use an existing HTML element in the Web page (or even inject a new HTML element) to trigger the plug-in dialog. This is best approach for situations where there is an existing embedded text editor in place and even minor changes in the Web page style are undesirable. A full walkthrough is provided in Appendix 1: External Call Integration for the Pre-Edit plug-in.

3.2.2 Configuration Options

In order to use the ACCEPT plug-in, an API Key is required. To get an API key, a user may log into the Accept Portal and go to: http://www.accept-portal.eu/AcceptPortal/Account/UserProfile.

This page lists a user's current list of applications. To generate an API key, the "Create new Application" button may be clicked and the form filled in. The newly created application is listed in the user's profile details. To get the API key, the application name can be clicked. The API details will be displayed alongside the API Key. The API key will be associated with an IP address.

The ACCEPT Pre-Edit plug-in can be configured to work in many different ways. The supported configuration options are described in Appendix 2: Pre-Edit Plug-in Configuration Options.

3.2.3 New and Updated Functionality

The feedback received throughout Year 1 suggested that the initial prototype should be refined to allow users to:

- Be presented with pre-editing rule violations, suggestions and recommendations in a clear and intuitive manner
- Edit the text while reviewing suggestions and recommendations without introducing conflicts in the original text environment
- Check and edit HTML content
- Trigger manual checks
- Select specific rule sets and avoid being confused with the "ReplaceAll" functionality
- "Learn" words
- Ignore rules
- Provide active feedback to the system
- Get access to clear and concise help on how to use the plug-in.

3.2.3.1 Simplifying the display of rule violations and suggestions

One of the most frequent comments from the usability study conducted in Year 1 was complaints about the excessive amount of tooltips, highlighting and underlining present in the window, as shown below:

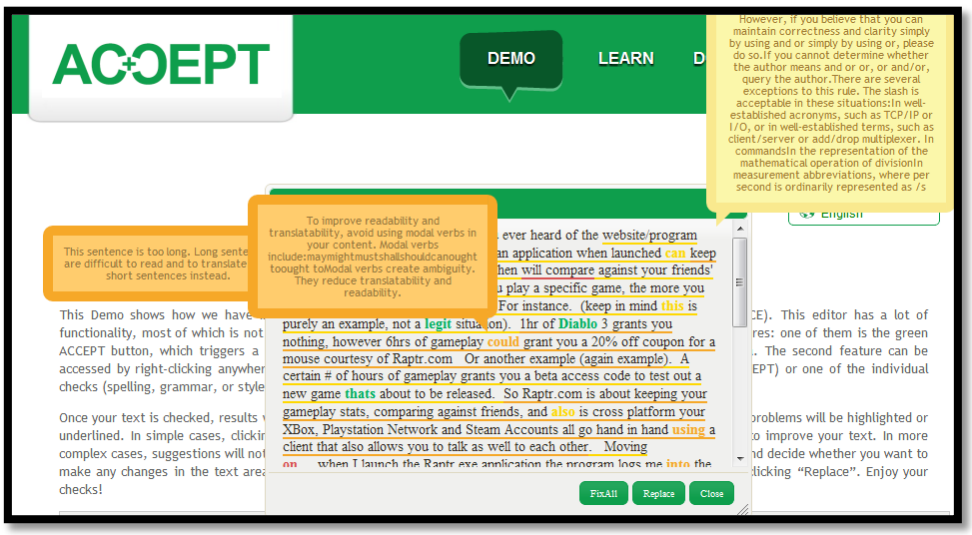


Figure 5: First Pre-Edit prototype

In order to address this problem, the following modifications were made:

- New types of tooltips were introduced (not only to improve the visual display of information but also to better control their position in relation to the text and the plug-in)
- The time spent displaying a tooltip was adjusted in order to:
    - Give users enough time to read the text and interact with the tooltip (using the mouse)
    - Avoid having two tooltips opened at the same time
    - Prevent tooltips from masking drop-down menus
- The amount of recommendation text returned by the language checking provider (Acrolinx) was reduced (e.g. alongside the rule name, a short text description is now returned instead of the complete rule specification)
- The color-coding scheme was simplified
- The amount of underlining was reduced (e.g. instead of underlining all of the words of a long sentence, only the first and last words are now underlined; these first and last words are also highlighted when the user hovers over one of them in order to help them visualize where the long sentence ends).

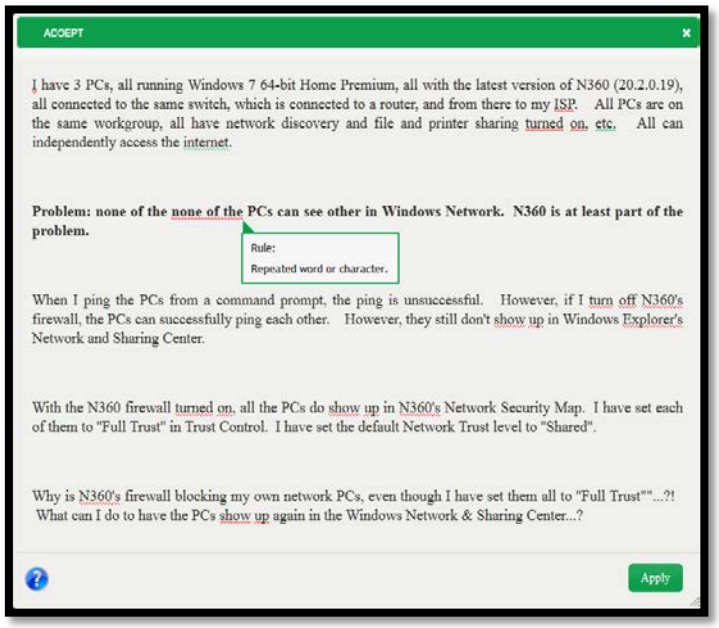


Figure 6: New tooltips with short recommendations

The screenshot above shows that only two colors are now used to underline text: red for general recommendations (e.g. “this sentence is too long”) and green for rules with actual replacement suggestions. While the first prototype used colors to make a distinction between rule types (e.g. spelling vs. style), the new plug-in focuses on the question of whether a replacement can be suggested to the user. After another round of internal usability testing, we decided to switch the use of these colors, since red is often is associated with errors (for which the system should suggest alternatives).

3.2.3.2 Editing text while reviewing suggestions and recommendations

The other common complaint from users was the inability to edit their text while reviewing suggestions. For instance, when they were being advised to reduce the length of a sentence, they had to close the plug-in, edit the original text and check the text again. This workflow was clearly

inefficient, so support for text editing was introduced in the new version of the plug-in, as shown below:

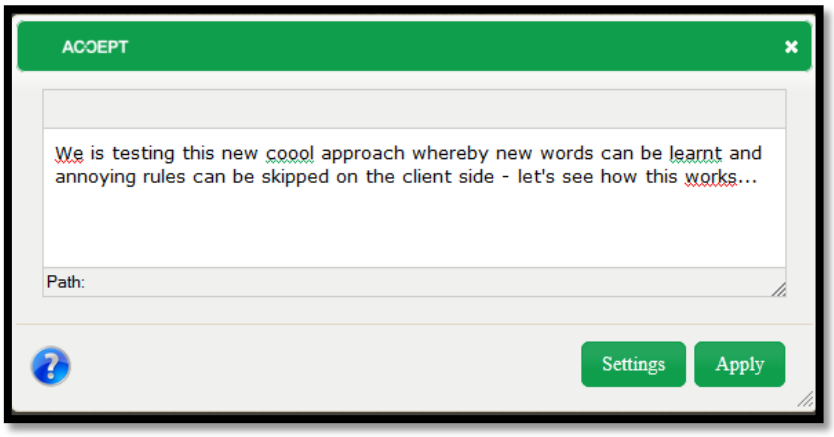


Figure 7: The editable area of the new Pre-Edit plug-in

Besides adding support for inline editing, we also decided to add the **isModal** configuration option to control whether the plug-in window should behave in a modal way (which is now the default). Modal windows are used to display additional content on a new page layer (window) on top of the loaded content (i.e. the original Web page). This was done to ensure that users do not introduce conflicts by modifying text in the plug-in window while also editing their original text in the native environment.

3.2.3.3 Checking HTML content

In addition to allowing users to edit their text while reviewing suggestions, we decided to introduce support for non-text content. The first version of the plug-in only offered users the ability to check text content, thus limiting the potential adoption of the tool in most Web applications. As shown below, HTML content (including pictures, style and hyperlinks) can easily be checked, annotated and edited using the Pre-Edit plug-in. Hyperlinks are actually disabled in the plug-in window but re-enabled once the edited text is applied in the user's native environment.

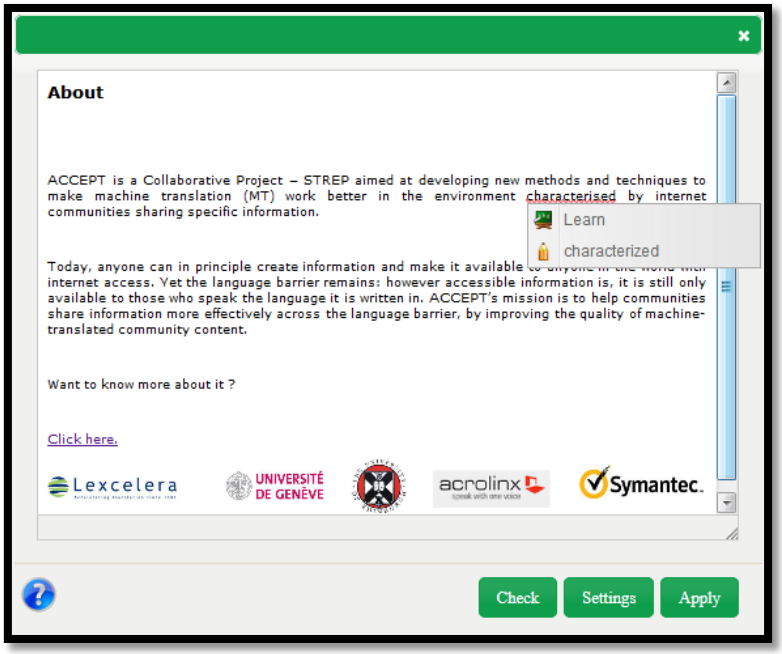


Figure 8: Presenting annotated HTML content

3.2.3.4 Triggering manual checks

As shown in Figure 8: Presenting annotated HTML content, a "Check" button is available in the plug-in window if the **showManualCheck** configuration option is set to True. This button allows users to manually re-check their text whenever they think new modifications require re-checking.

3.2.3.5 Displaying the "Replace All" button

Another source of confusion among users was the "FixAll" button in the first version of the plug-in. This button requested automatic modification of the user's text based on suggestions provided by the language checking tool. Since this replacement approach is automatic, some modifications introduced errors in the text, suggesting that the choice of the phrase "FixAll" was not appropriate. To work around this problem, we decided to rename this option to "Replace All" and make it inactive by default. One possible implementation is therefore to let the user decide whether this option should be shown in the plug-in, as shown in Figure 9.

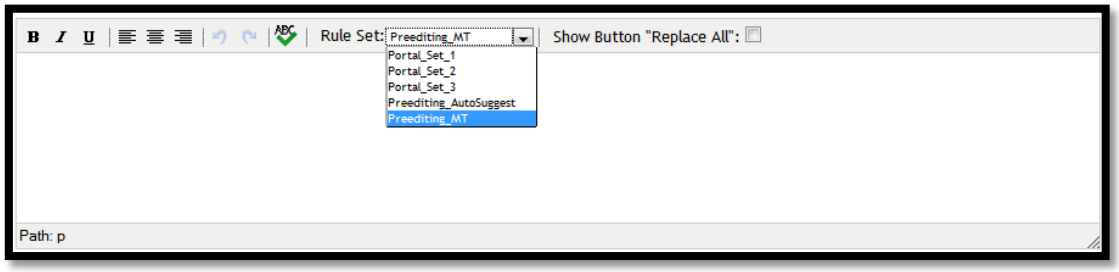


Figure 9: Selection of a rule set and the display of the "Replace All" button

3.2.3.6 Selecting a specific rule set

Figure 9 also shows that this plug-in implementation (which is in place on the Demo section of the ACCEPT portal) lets users decide which rule set should be used to check their text. More information on this option is available in the following section: The ACCEPT Pre-Edit Demo.

3.2.3.7 Learning words

One of the most popular feature requests was the ability to have the system "learn" words so that they would no longer be flagged by the system. While the initial version of the plug-in offered users the ability to ignore flags, these decisions were lost as soon as the user closed the plug-in window. In this version of the plug-in, we decided to use a client-side storage mechanism to make such choices persistent (i.e. using local storage or cookies depending on the browser used). We also decided to provide the user with a way to manage the words they had learnt. As shown in Figure 8: Presenting annotated HTML content, contextual menus with suggestions now contain an extra option (*Learn*), which allows users to inform the system that they wish to consider the form of the flagged word as correct. Once they click on "Learn", the underlining disappears. Users then have to review this choice by clicking the "Settings" button. Once they do so, the plug-in window rotates and displays a list of learnt words, as shown below:

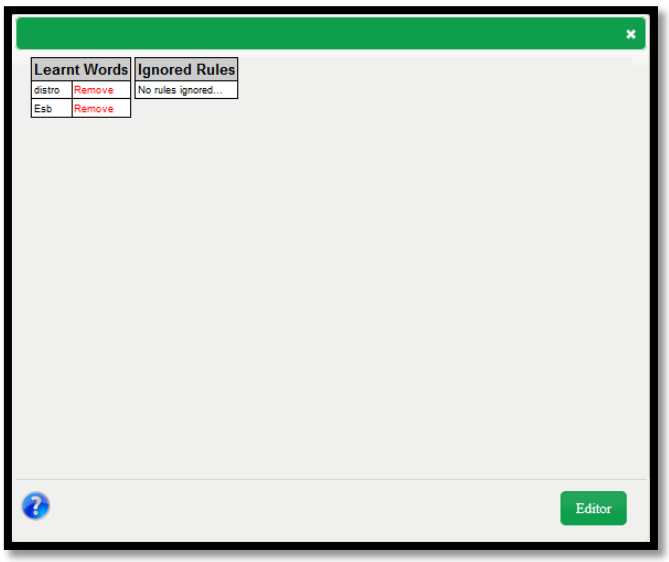


Figure 10: The management of learnt words

Learnt words can be easily deleted by clicking the "Remove" link, the list of words being updated dynamically.

3.2.3.8 Ignoring rules

In the same way that words can be learnt, rules can be ignored (i.e. disabled) by users when they think the rule is not useful. Once a rule is disabled, all violation instances of this rule are ignored and disappear from the plug-in window. Users can revert their decisions at any time by removing rules from the "Ignored rules" table in the "Settings" window.

3.2.3.9 Providing active feedback to the system

In order to collect and export usage information as detailed in Section 3.1.1, some client-side changes were made to ensure that user interactions can be precisely captured in the following situations:

- When a tooltip is displayed but the rule is not ignored
- When a tooltip is displayed and the rule is ignored
- When a drop-down menu is displayed but no option is clicked
- When a drop-down menu is displayed and a suggestion is selected
- When a drop-down menu is displayed and the phrase is learnt.

3.2.3.10 Getting help

The help available to users in the first version of the plug-in was hard to find and extremely text-heavy. In the new plug-in version, we decided to use an easily-recognisable blue icon (symbolized with a question mark) as shown in Figure 8: Presenting annotated HTML content. It was also decided to use the actual buttons from the plug-in in the help dialog to avoid any confusion:

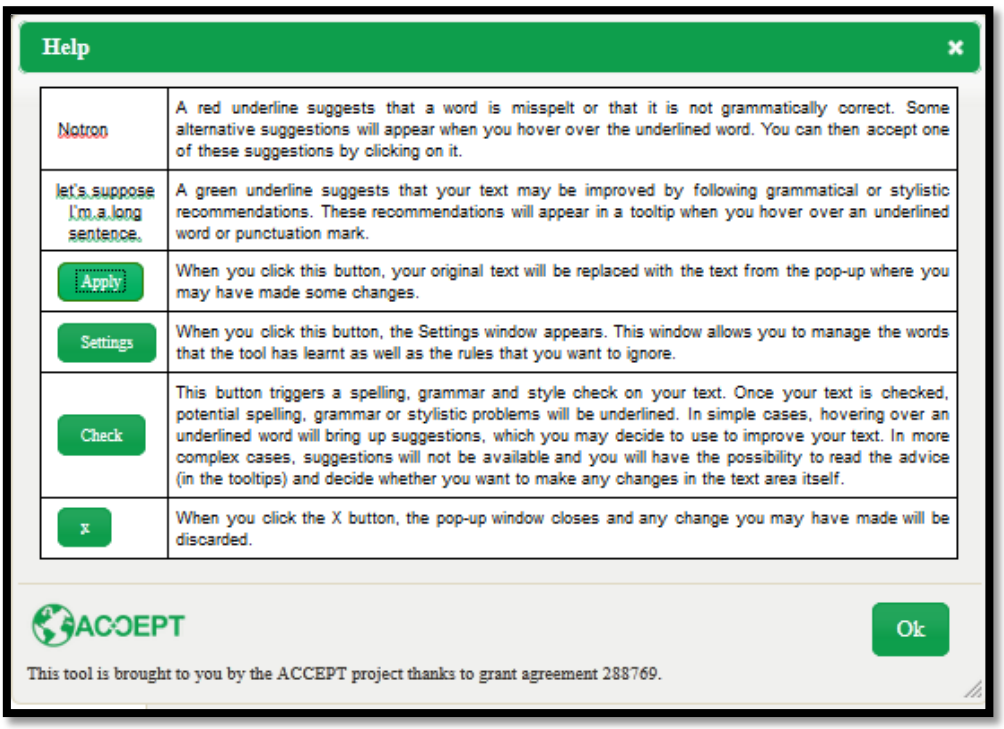


Figure 11: Contextual help dialog

3.2.4 Downloadable Package

The Pre-Edit plug-in can be accessed from the ACCEPT Content Delivery Network (CDN) by including the following references in the head element of an HTML document:

```
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/jquery-1.5.1.min.js" type="text/javascript"></script>
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/jquery-ui-1.8.24.custom.min.js" type="text/javascript"></script>
<link href="http://www.accept-portal.eu/Plugin/v2.0/css/Accept.css" rel="stylesheet" type="text/css" />
<link href="http://www.accept-portal.eu/Plugin/v2.0/css/jquery-ui.css" rel="stylesheet" type="text/css" />
<script src="http://www.accept-portal.eu/Plugin/v2.0/extras/tiny_mce/tiny_mce.js" type="text/javascript"></script>
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/accept-jquery-plugin-2.0.js" type="text/javascript"></script>
```

The plug-in has also been made available as a downloadable package with the contents shown in Table 1.

/	index.html (ACCEPT Pre-Edit plug-in documentation)
/docs	index.html (ACCEPT Pre-Edit plug-in documentation)
/js	jquery-1.5.1.min.js (jQuery core library) jquery-ui-1.8.24.custom.min.js (jQuery UI library) accept-jquery-plugin-2.0.js (ACCEPT plug-in core file)

/css	jquery-ui.css (jQuery UI CSS) accept.css (ACCEPT plug-in CSS)
/examples	Example1.htm (Use case code example.) Example2.htm (Use case code example.)
/extra/tinyMce	tiny_mce.js (TinyMCE plugin core file.)

Table 1: Pre-Edit plug-in package contents

The version of the plug-in described in this document is version 2.0, which can be downloaded from:

http://www.accept-portal.eu/AcceptPortal/en-US/Download/Index

3.3 The ACCEPT Pre-Edit Demo

A Pre-Edit demo is available to any registered user of the ACCEPT portal. This demo shows how the JQuery plug-in works in a TinyMCE environment, by offering source French, English and German language checking. The French demo differs from the English and German demos because the plug-in has been configured to allow users to perform checks in multiple steps, as shown in Figure 12:

Figure 12: The French Pre-Edit demo on the ACCEPT Portal

As shown above, the Pre-Edit plug-in contains three extra buttons next to the Help button. These buttons correspond to specific checks that can be triggered on the text, each based on a separate rule set. Using this type of configuration may be advantageous in situations where the precision of certain rules can be significantly increased if other errors have been corrected first. Instead of grouping all rules in one rule set, it is therefore possible to group rules in smaller rule sets.

Note: This configuration option can be combined with the use of the "ReplaceAll" functionality in order to fully automate the application of suggestions provided by the rule set which has been specified as consisting of automatic rules.

4 The ACCEPT Post-Edit Components

This section presents the new Post-Edit plug-in functionality before describing new characteristics of Post-Edit projects. The new API functionality is presented in this second sub-section. Finally the ACCEPT Post-Edit demos are introduced.

4.1 New Post-Edit Plug-in Functionality

The following functionality has been added in version 2 of the Post-Edit plug-in:

- The ability to customize the display of the target text in the navigation pane on the left.
- The ability to control whether source segments may be shown or hidden by the user
- The ability to record how much time is spent by a user on a specific segment.

4.1.1 Target Text Display

A new functionality was introduced to control the display of target text in the Post-Edit window. This new functionality is fully described in Roturier et al (2013). The following figures show the use of *tgt_templates* to separate each source segment with a line break.

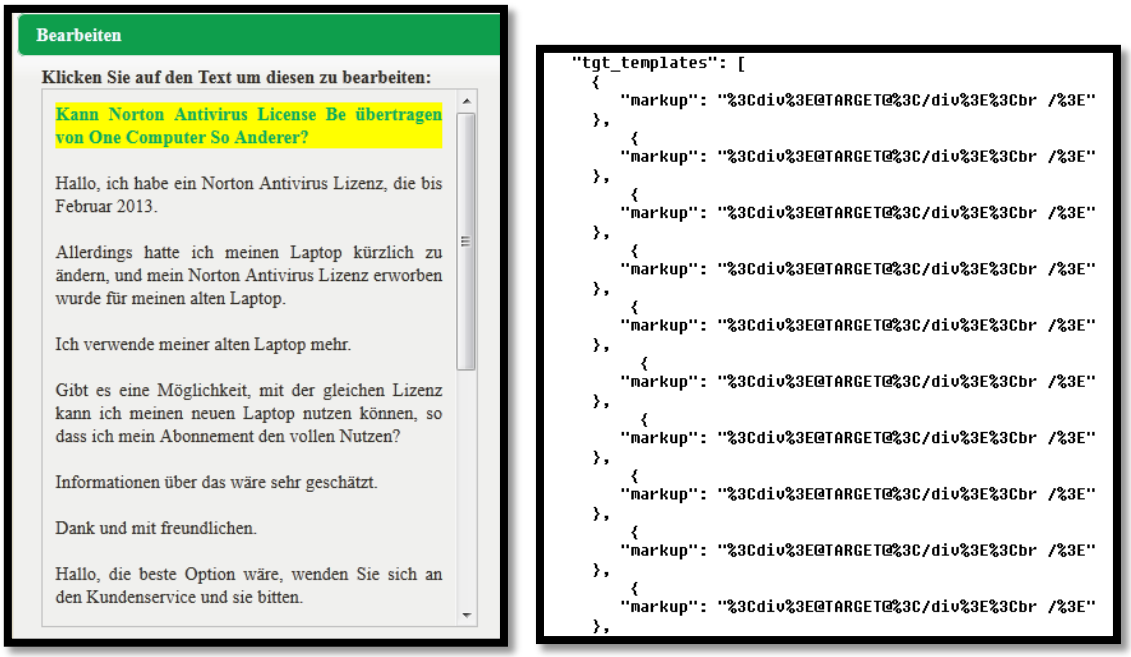


Figure 13: Source text display using target templates

4.1.2 Source Segment Display

Post-editing is traditionally believed to be most successful in a bilingual mode (i.e. post-editing with reference to the source text) so that meaning which may have been lost or distorted in the machine translation process can be retrieved from the source. While research in monolingual post-editing is scarce, especially with regard to domain experts as post-editors rather than linguists/translators, providing the post-editor with the opportunity of choosing the post-editing setup dynamically (i.e. monolingual/bilingual) has been identified as a potential way of minimising or preventing user frustration. This is supported by feedback that has been gathered in internal studies, which indicated

that users were eager to see the source. To illustrate switching between bilingual and monolingual modes, consider what happens if the project default is the monolingual mode. The source will then not be shown in the interface when a task is opened initially. The user can then decide to switch to being shown the original segment for the current segment. Regardless of how many switches are performed per segment, the last state the switch is in is retained for the next segment the user chooses to edit. The button can be toggled at any time. When the editor is closed, the page is refreshed or a new task is selected, the project default is displayed again (in this case the source is not shown).

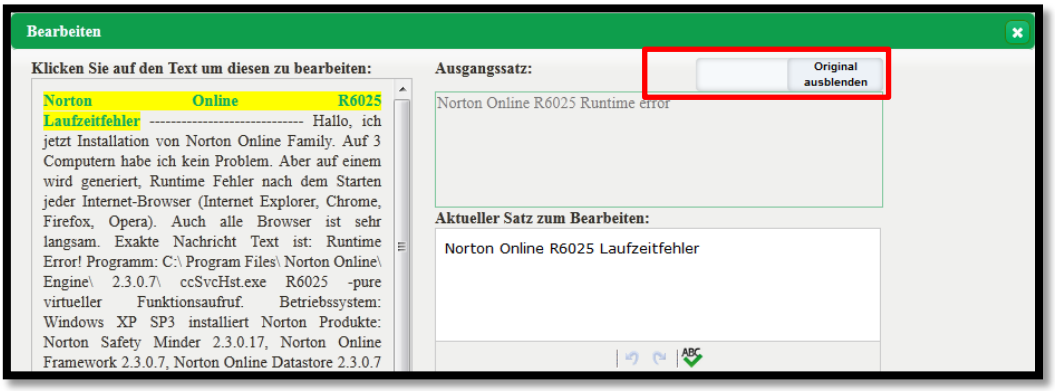


Figure 14: Showing or hiding the source segment

4.1.3 Recording Time

Once users have started working on a Post-Edit project, project administrators can export post-editing activity data at user, document or project level. The data is exported in an XLIFF format.

The following example shows the type of usage data that is captured by the system and made available in the header of the XLIFF report.

```
<header>
<phase-group>
    <phase phase-name="mt_baseline" process-name="Machine Translation" tool="tb"
    toold-id="accept.statmt.org/demo/translate.php" date="2013-08-28T10:28:25.772Z"
    contact-email=""/>
    <phase phase-name="start_pe" process-name="bilingual" tool="ACCEPT Portal"
    toold-id="ACCEPT Post Edit Plug-in 1.0" date="2013-11-27T11:46:45.000Z"
    contact-email="..."/>
    <phase phase-name="r1.1" process-name="bilingual" date="2013-11-27T11:47:16.000Z"
    contact-email="...">
        <note annotates="target" from="user">Yes</note>
        <note annotates="general" from="user">good baseline translation!</note>
    </phase>
    <phase phase-name="t1.1" date="2013-11-27T11:46:45.000Z" contact-email="..."/>
    <phase phase-name="t1.2" date="2013-11-27T11:47:09.000Z" contact-email="..."/>
</phase-group>
<count-group name="1">
    <count phase-name="r1.1" count-type="x-keys" unit="instance">129</count>
    <count phase-name="r1.1" count-type="x-delete-keys" unit="instance">5</count>
    <count phase-name="r1.1" count-type="x-white-keys" unit="instance">5</count>
    <count phase-name="r1.1" count-type="x-nonwhite-keys" unit="instance">31</count>
    <count phase-name="r1.1" count-type="x-arrow-keys" unit="instance">61</count>
    <count phase-name="r1.1" count-type="x-editing-time" unit="x-seconds">55.133</count>
    <count phase-name="r1.1" count-type="x-typing-time" unit="x-seconds">55.133</count>
    <count phase-name="t1.1" count-type="x-think-time" unit="x-seconds">16.099</count>
    <count phase-name="t1.1" count-type="x-start_source_switch" unit="instance">show</count>
    <count phase-name="t1.1" count-type="x-source_switch" unit="instance">2</count>
    <count phase-name="t1.1" count-type="x-source_switch" unit="x-seconds">6.662</count>
    <count phase-name="t1.1" count-type="x-source_switch" unit="x-seconds">5.483</count>
    <count phase-name="t1.2" count-type="x-think-time" unit="x-seconds">6.651</count>
</count-group>
<count-group name="2"/>
<count-group name="3"/>
<count-group name="4"/>
<count-group name="5"/>
</header>
```

Figure 15: Displaying Post-Editing phase information in XLIFF reports

Thanks to the information present in the **header** element, the steps that were taken during the post-editing process can be retraced. The **phase-group** element contains a number of phases corresponding to processes that were used to interact with the target text. The first of these phases (in chronological order) actually occurred before the post-editing process since it corresponds to the automatic translation of the source text using the relevant Machine Translation tool.

The first post-editing phase is the one whose phase-name attribute has a **start_pe** value. The starting time of this phase is indicated in the value of the **date** attribute. The times indicate that the start of this phase coincided with the start of another phase, whose **phase-name** attribute has a **t1.1** value. The syntax of this value is based on the following parts:

- **phase_type**: r (for revision) or t (for thinking)
- translation unit ID: starting at 1
- iteration ID for a given translation unit: starting at 1

The **t1.1** value therefore means that the first iteration of a thinking phase started for the first translation unit of the file. During this phase, some countable events occurred. These events are captured in a number of count elements with a matching **phase-name** attribute value. In this example, the phase lasted just over 16 seconds (as indicated by the element with the **x-think-time** value). The element with the **x-start_source_switch** value indicates that the UI displayed the source text when the phase was started. However, this was changed twice by the user, as indicated by the element with the **x-source_switch** value where the value of unit is instance. Other elements also indicate precisely when the UI was changed.

In short, a thinking phase is used to capture events that happened in the Post-Edit client when these events are not directly related to the editing of the target text. For instance, if the user had commented on the quality of the target text, a **note** child element would have been attached to the phase element.

Based on the times present in the report, it is possible to determine that the **t1.1** phase was not immediately followed by the **t1.2** phase, since the difference between the starting time of the **t1.2** phase and that of the **t1.1** phase is 24 seconds (whereas the **t1.1** phase lasted just over 16 seconds). This means that the user closed the client application and re-opened it 8 seconds later. As soon as the client application was re-opened, the **t1.2** phase started and lasted over 6 seconds. This time, however, the **t1.2** phase was immediately followed by another phase in the client application, the **r1.1** phase (which was a revision phase).

Multiple events occurred during the **r1.1** phase, including the generation of two comments by the user (as indicated by the two child **note** elements) on lines 11 and 12. Other events were captured in count elements, including how long the editing of the target text lasted (as indicated by the **x-editing-time** value). In this example, the **x-editing-time** value is the same as the **x-typing-time value**, indicating that the user started editing the target text using a keyboard key (instead of using a contextual translation option). Various numbers of pressed keys are available for this phase, thus allowing for a detailed analysis of the type of post-editing that was conducted.

x-editing-time and **x-typing-time** values must be interpreted with caution. For unavoidable practical reasons, the calculation of these times may not accurately reflect the reality of the situation. For instance, an **x-typing-time** value of 50 seconds does not necessarily mean that the user was typing

for 50 seconds. It means that the user started typing and that the phase ended 50 seconds later. It is possible, however, that the user typed for 10 seconds, thought for 2 seconds, typed again for 12 seconds, etc. Trying to capture and analyse this data in full detail is extremely challenging, which is why the calculations are currently simplified.

Since an editing revision occurred, the target text is likely to have changed in the first translation unit. This is confirmed when examining the first **trans-unit** element of the **body** element:

```
<body>
    <trans-unit id="1">
        <source>ACCEPT est un projet collaboratif STREP, qui a pour but de développer
            de nouvelles méthodes et techniques visant à améliorer la traduction
            automatique (TA) dans le cadre des communautés Internet partageant des
            informations spécialisées.</source>
        <target phase-name="r1.1">ACCEPT is a collaborative STREP project, which aims
            at developing new methods and techniques to improve machine translation
            (MT) within the framework of Internet communities specialised in sharing
            information.</target>
        <alt-trans phase-name="mt_baseline">
            <target>Accept is a collaborative project STREP, which aims to develop
                new methods and techniques aimed to improve the translation automatic
                (ITA) in the framework of the communities specialised Internet
                sharing information.</target>
        </alt-trans>
    </trans-unit>
```

Figure 16: Displaying Post-Editing revision in XLIFF reports

In this example, it is possible to see that the current target text corresponds to what was produced in the **r1.1** phase, relegating the translation from the **mt_baseline** phase to an **alt-trans** element.

It should be noted that thinking phases do not have any impact on **trans-unit** elements since the target text does not get modified during these phases.

4.2 New Portal-Based Project Functionality

4.2.1 External Projects

While creating post-editing projects within the ACCEPT Portal can be useful to conduct studies or centralize post-editing activities, it can be cumbersome when project participants (i.e. post-editors) are used to working in another environment (e.g. a crowdsourcing platform, an online discussion forum, a content management system, etc.). To address this issue, we decided that the post-edit plug-in should behave in the same way as the pre-edit plug-in and become accessible from outside the ACCEPT portal. To achieve this goal, some modifications had to be made to post-editing projects, by giving project users the possibility to create external projects (i.e. a project where the actual post-editing activity would take place outside of the ACCEPT Portal, while leveraging the ACCEPT API to access translation assistance material and to save any post-editing action).

The following steps can be used to create and manage an external Post-editing environment using the ACCEPT Post-Edit plug-in:

1. Create an external Post-Editing project using the Portal's project creation page.

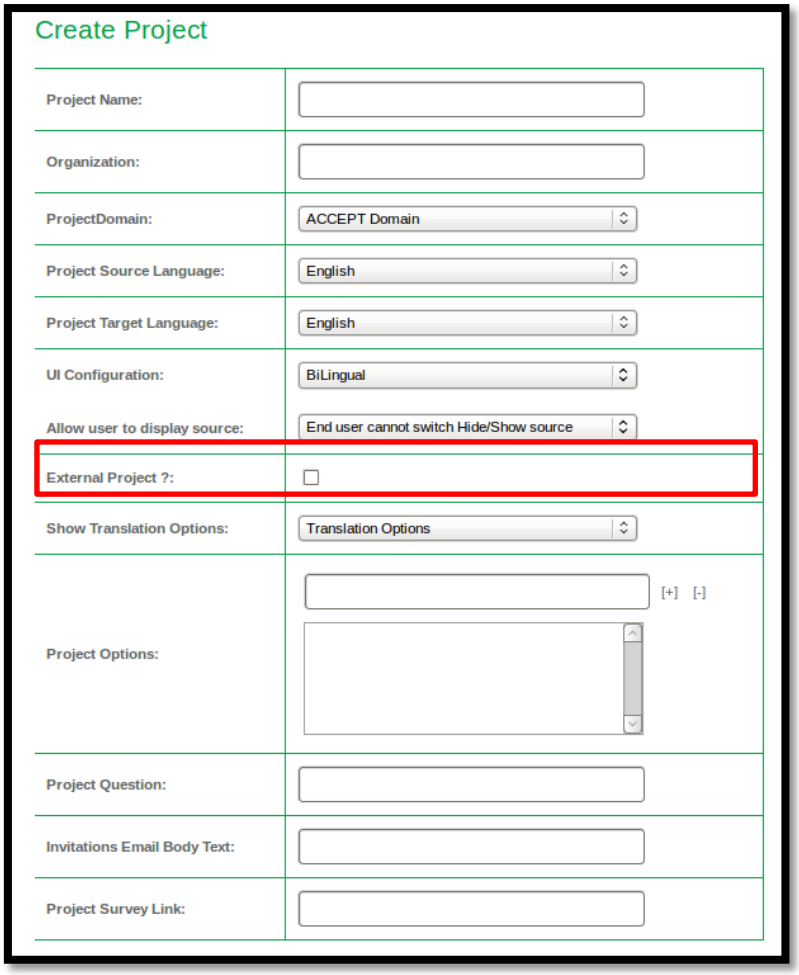


Figure 17: External Post-Edit Project Creation

2. Add Post-Editing tasks to this newly created project using the Portal or the API

While a task upload functionality was already present on the ACCEPT Portal, we decided to expose this functionality via the API to speed up the upload process for projects containing multiple tasks. To achieve this, it was necessary to introduce a private project token to restrict task upload to project owners. This token can then be used by leveraging any HTTP compliant client application (e.g. any HTTP debugging proxy server application such as Fiddler) as follows:

- Set the proper ACCEPT API URL to add tasks to projects. For example: `http://[accept_portal_server]/AcceptApi/Api/v[api_version]/PostEdit/AddDocumentToProject/?token=[token]`
  where [accept_portal_server] is the ACCEPT portal server to target, [api_version] is the version of the API (current is 1), and [token] is the project's admin token
- Add the following HTTP header to the request: **Content-Type : application/json**
- Add the JSON that corresponds to the task in the request body
- Post the request

3. Add external users to this newly created project using the API

To add users to a project, the following POST method can be used:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Admin/AddUserProject
```

where [accept_portal_server] is the ACCEPT portal server you want to target and [api_version] is the version of the API (current is 1).

The request must include the following in the HTTP header: `Content-Type:application/json`

The request must include a JSON body based on the following format:

```
{
    "userName": "…",
    "token": "…"
}
```

where token is the project's admin token and userName the name of the user who will be allowed to work on the task (as specified during the initialisation of the Post-Edit plug-in, as described in next step).

4. Set up an external Post-Editing environment

Since the plug-in is written on top of the jQuery and jQuery UI libraries, these are both a mandatory requirement for deployment in any Web-based environment. The ACCEPT Post-Edit plug-in has the following initialisation options:

Field	Description
dialogHeight	Height for the dialog window where the plug-in will be triggered
dialogWidth	Width for the dialog window where the plug-in will be triggered
leftPaneWidth	Width for the left hand-side navigation pane.
leftPaneHeight	Height for the left hand-side navigation pane.
leftPaneFontSize	Font size for the text in the left hand-side navigation pane.
textAreasWidth	Width for the right hand-side pane.
textAreasHeight	Height for the right hand-side pane.
imagesPath	URL path for all plug-in images
acceptServerPath	URL path for the ACCEPT API
textIdContainer	Attribute name that might contain the Post Editing text identifier
userIdSelector	jQuery selector for the DOM element that contains the user identifier
userIdContainer	Attribute name that might contain the Post Editing user identifier
preEditApiKey	API key for the ACCEPT Pre-Edit plug-in
preEditImagesPath	URL path for the Pre-Edit plug-in images
preEditWith	Width for the Pre-Edit plug-in dialog
preEditHeight	Height for the Pre-Edit plug-in dialog
preEditingLanguageUI	UI language for the Pre-Edit plug-in

Figure 18: Post-Edit plug-in initialisation options

The other plug-in files (CSS and JavaScript) can be referenced using the ACCEPT content delivery network, as shown in the example below.

5. Check project task status

To check the status of a project, the following GET method can be used:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Admin/ProjectTaskStatus?token=token
```

where [accept_portal_server] is the ACCEPT portal server you want to target, [api_version] is the version of the API (current is 1), and [token] is the project's admin token.

The request must include the following in the HTTP header: `Content-Type:application/json`

This method returns a JSON object:

```
"ResponseObject": [
     {
         "TextId": "[UniqueTaskID]",
         "UserId": "ExtDeVUser1",
         "Status": 0
     },
     {
         "TextId": "[UniqueTaskID]",
         "UserId": "ExtDeVUser2",
         "Status": 1
     },
     {
         "TextId": "[UniqueTaskID]",
         "UserId": "ExtDeVUser3",
         "Status": 2
     }
 ]
```

The status values are:

- 0 : task not started by user
- 1 : task started but not finished by user
- 2 : task completed by user

6. Collect recorded data using the Portal

At all times, it is also possible to get project information to keep track of the project's task and users. To get information about a project, the following GET method can be used:

```
http://[accept_portal_server]/AcceptApi/Api/v[api_version]/Admin/ProjectInfo?token=token
```

where [accept_portal_server] is the ACCEPT portal server you want to target, [api_version] is the version of the API (current is 1), and [token] is the project's admin token.

The request must include the following in the HTTP header: `Content-Type:application/json`

This method returns a JSON object whose **ResponseObject** value is composed of two lists: a list of tasks and a list of users.

4.2.2 Projects with Single Revision

While projects with multiple, independent revisions can be extremely useful for studying how different post-editors edit a given target text, they do not address the need to have a single, collaborative revision, which would be required in most real-life deployment scenarios. In order to

meet this requirement, we decided to give project creators the ability to define collaborative projects where post-editors work on a single revision of the text. The advantage of this approach resides in the fact that the modifications made by a post-editor do not have to be repeated when a second post-editor starts working on the same project task.

To implement this functionality, the challenge posed by conflicting edits had to be resolved. Two approaches were considered:

- Having a fully synchronized, real-time editing environment where the changes made by User A are immediately seen by User B (if both User A and User B are working on the same project task at the same time)
- Having a mechanism to prevent users from working on a task if a user is already working on it

The first approach is obviously much more challenging from a technical perspective and it is not clear that it would be suitable for short post-editing tasks, especially at the segment level. We decided to implement the second approach with the following restriction: project creators can decide how long a task can be held by a given user before it can be "claimed" by another one. If users were able to keep working on a task indefinitely, the collaborative aspect of the task would be lost. Project creators can therefore define at project creation the maximum amount of time that should be used to lock a task for a given user, as shown in Figure 19:

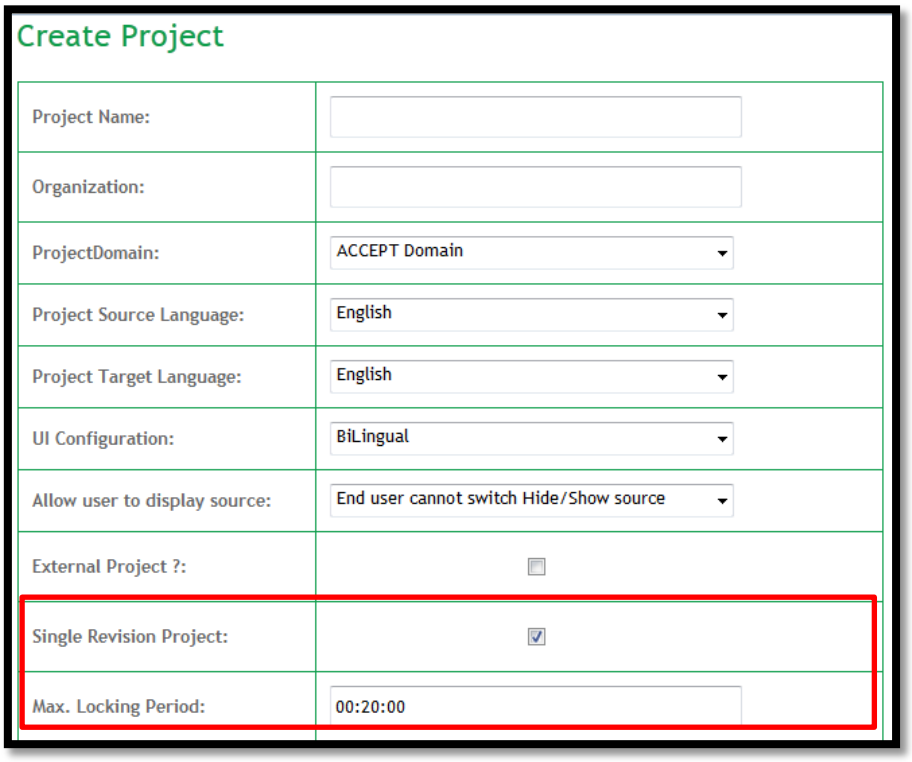


Figure 19: Creation of a single revision project with a 20-minute lock

When a project is configured as shown in Figure 19, it gives users at least 20 minutes to complete a task before another user is able to try to claim it. Once a user has successfully claimed a task, all other users are presented with the warning shown in Figure 20 for the duration of the project's locking period.

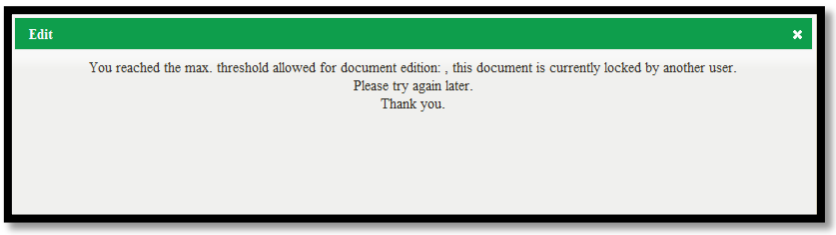


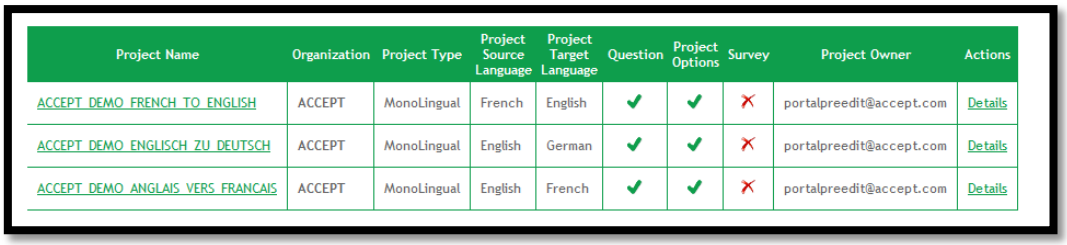
Figure 20: Locking period warning

The use of this configuration will be tested in natural environments (e.g. online forums) to determine whether users in these environments can improve machine-translated texts in a collaborative manner.

4.3 The ACCEPT Post-Edit Demos

In order to showcase the Post-Edit plug-in to users of the ACCEPT portal, three demo projects have been created, one for each of the following language pairs:

- French > English
- English > German
- English > French



Project Name	Organization	Project Type	Project Source Language	Project Target Language	Question	Project Options	Survey	Project Owner	Actions
ACCEPT_DEMO_FRENCH_TO_ENGLISH	ACCEPT	MonoLingual	French	English	✔	✔	✘	portalpreedit@accept.com	Details
ACCEPT_DEMO_ENGLISCH_ZU_DEUTSCH	ACCEPT	MonoLingual	English	German	✔	✔	✘	portalpreedit@accept.com	Details
ACCEPT_DEMO_ANGLAIS_VERS_FRANCAIS	ACCEPT	MonoLingual	English	French	✔	✔	✘	portalpreedit@accept.com	Details

Figure 21: Post-Edit demos on the ACCEPT Portal

Portal users can click on any of these tasks to get access to a small post-editing task, as shown below:

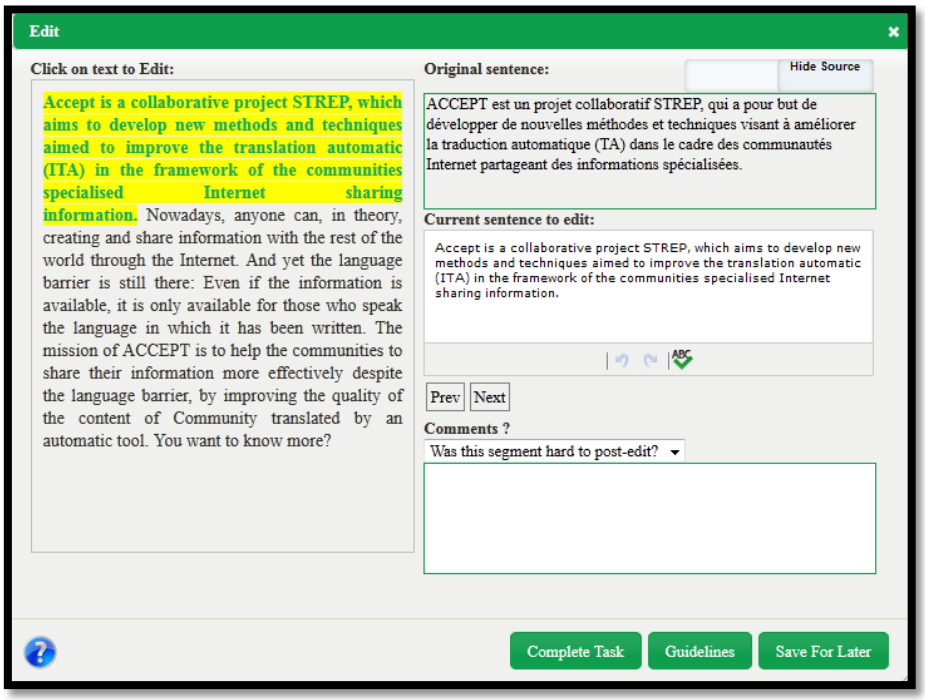


Figure 22: Post-Edit demo task

The figure above shows machine-translated text on the left-hand side of the window, with the first sentence to post-edit highlighted in yellow. On the right-hand side, the same sentence is available for editing in a text area, under the label “Current sentence to edit”. In this screenshot, the user decided to show the source (original) sentence by clicking the switch button on the top right hand-side of the window. This source sentence could be hidden again by clicking “Hide Source”.

5 The ACCEPT Evaluation Components

The ACCEPT Evaluation components are now divided into two parts:

- The ACCEPT Evaluation API and the Evaluate section of the ACCEPT Portal
- The ACCEPT Appraise component

The first component has been updated in Year 2 to address the following shortcomings:

- Evaluation content could not be added to projects, so a client-side mechanism had to be used
- No public method existed to export project data

5.1 Updated ACCEPT Evaluate Project Management Section

Instead of relying on a client-side mechanism to make content available for evaluation, content may be added to an evaluation project. To do so, a JSON file may be uploaded by clicking the Add Content link and selecting a file. The file must comply with the following format:

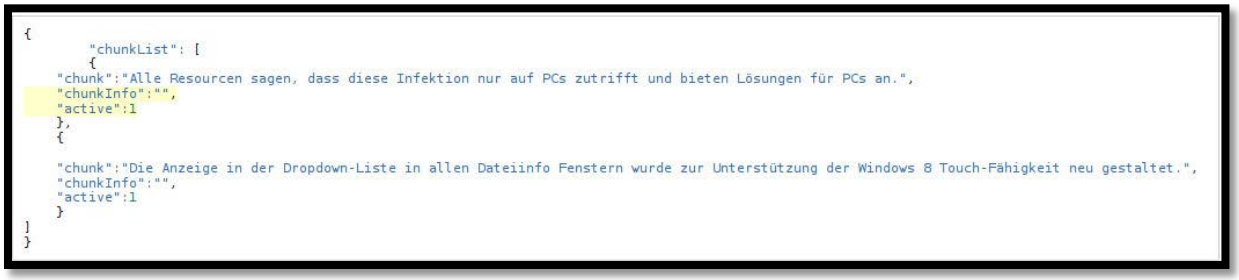


Figure 23: Format of evaluation content

As shown in the example above, any number of sentences (or text chunks) may be uploaded in a **chunkList** array. Each **chunk**, which will be used during the actual evaluation task, must be a UTF-8 string. Additional metadata may be included in **chunkInfo**. The **active** value may be set to 1 (active) or 0 (not active) depending on whether this specific chunk should be considered on the client side.

5.2 Updated ACCEPT Evaluation API

Two new methods have been added: the **ContentChunks** method and the **Scores** method.

The **ContentChunks** method is a GET method. The **ContentChunks** method returns two lists: a list of chunks and list of questions.

```
http://[accept_portal_server]/AcceptApi/api/v[api_version]/Evaluation/ContentChunks/[ID]
```

where [accept_portal_server] is the ACCEPT portal server you want to target, [api_version] is the version of the API (current is 1), and [ID] is the Evaluation project ID. For an evaluation project with an ID of 1 the call would be:

```
http://[accept_portal_server]/AcceptApi/api/v[api_version]/Evaluation/ContentChunks/9?key=21fdc25bebdf456db5c9e0993977bb12
```

The following parameters can be passed on the URL:

Parameters	Details	Example
key	This is a MANDATORY parameter. If this parameter is not passed then the API call will fail. This value can be found on the Project page.	key=21fdc25bebdf456db5c9e0993977bb12
language	This is a MANDATORY parameter. If this parameter is passed then only the questions in the specified language will be returned. Language code is based on RFC 4646. Examples of language codes are: en, fr, en-us, fr_fr	language=en_us
question	This is a MANDATORY parameter. If this parameter is passed then only this question will be returned.	question=1
category	This is a MANDATORY parameter. If this parameter is passed then only the questions in the specified category will be returned.	category=1

The **Scores** method, which can be used to retrieve answer data, is a GET method. The **Scores** method returns a list of answers and associated metadata.

```
http://[accept_portal_server]/AcceptApi/api/v[api_version]/Scores/[ID]
```

where [accept_portal_server] is the ACCEPT portal server you want to target, [api_version] is the version of the API (current is 1), and [ID] is the Evaluation project ID. For an evaluation project with an ID of 1 the call would be:

```
http://[accept_portal_server]/AcceptApi/api/v[api_version]/Scores/9?token=[token_id]
```

The following parameter must be passed on the URL: **token.** This is a MANDATORY parameter. If this parameter is not passed then the API call will fail. This value can be found on the **Project Details** page under **My Project Token**. A response example is shown below, with the actual question answer highlighted in yellow:

Figure 24: Scores method's response

5.3 The ACCEPT Appraise Component

5.3.1 Motivation

One of the original objectives of Task 5.3 was to integrate functionality from an existing evaluation system into the ACCEPT Portal (a system similar to the one deployed at http://eval4all.com). In order to justify such integration, however, this existing system had to be compared against at least another (more recent) evaluation system. One such system is Appraise, which has been used by the SMT community in shared tasks such as WMT 2013. The table below summarizes the main characteristics of both systems.

Functionality/System	Eval4all.com	Appraise
Technology	C#/MS SQL server	Python/Django (DB-agnostic)
Data import (upload)	Yes	Yes
Data import format	XLIFF	XML
Web Data export (download)	No	Yes
Data export format	N/A (Excel after running SQL queries)	XML + TXT
Evaluation Type	Comprehensibility (fixed 5-point scale); Fidelity (fixed yes/no scale)	translation quality checking (fixed 3-pt scale; 3 o5 way ranking of translations; error classification; manual post-editing)
Extensibility to add extra evaluation type	Difficult	Straightforward
Agreement scores generation	No	Yes
Score statistics	Yes (average per task)	Yes (number of score instances per task)
User Progress Status	Yes	Yes
User Management	Yes (including self-service registration)	Yes (through Django interface) but administrator must create user accounts
Localised versions	Yes (French, German, Chinese, Japanese)	No.
License	Unknown	Modified BSD

Table 2: Comparison between Eval4All and Appraise

As this table clearly shows, the Appraise system has more functionality and is easier to extend than its eval4all.com counterpart, especially from a data export perspective. The Appraise system offers project creators the ability to easily export project data using a combination of XML and TXT files (for rating and statistics respectively) by pushing a button. In contrast the eval4all.com system requires an administrative access to the database, where SQL queries have to be run before an Excel file can be exported.

5.3.2 Modifications

The following modifications were performed to add a new evaluation task type to the Appraise system. This new task type was required for the study conducted in WP8, during which multiple versions of post-edited segments were collected.

- An **ACCEPT Ranking** evaluation type choice was added to the **APPRAISE_TASK_TYPE_CHOICES** object. This type choice indicates that a new type of ranking task becomes available to project creators. The objective of this task type is to combine a traditional (system) ranking task with a quality rating task. Ranking tasks may be useful to determine whether a system is better than another system, but it does not quantify the difference that may exist between the two systems. By combining the two approaches, it is possible for a user to indicate that two (or more) translations (produced by MT systems or human translators/post-editors) are ties at a given point on a scale. To simplify things, we decided in this implementation to keep the number of scale points consistent with the number of translations to evaluate. By uploading a file with five translations, an ACCEPT ranking task becomes available using a 5-point scale. Actually, the current implementation defaults to having two questions presented to users (one for fidelity and one for comprehensibility), as shown by the template and the rendered user view below:

```
1    A:
2   {% for translation in translations %}
3    <span onclick="$(this).children('input').attr('checked', 'checked');"><input type="radio" name="rank_{{rank_id}}"
      value="{{forloop.counter}}" /> {% cycle 'nichts' 'wenig' 'vieles' 'das Meiste' 'alles' %}</span>
4   {% endfor %}
5    ||| B:
6   {% for translation in translations %}
7    <span onclick="$(this).children('input').attr('checked', 'checked');"><input type="radio" name="rank2_{{rank_id}}"
      value="{{forloop.counter}}" /> {% cycle 'unverständlich' 'zusammenhangslos' 'nicht muttersprachlich' 'gut' 'perfekt'
      %}</span>
8   {% endfor %}
```

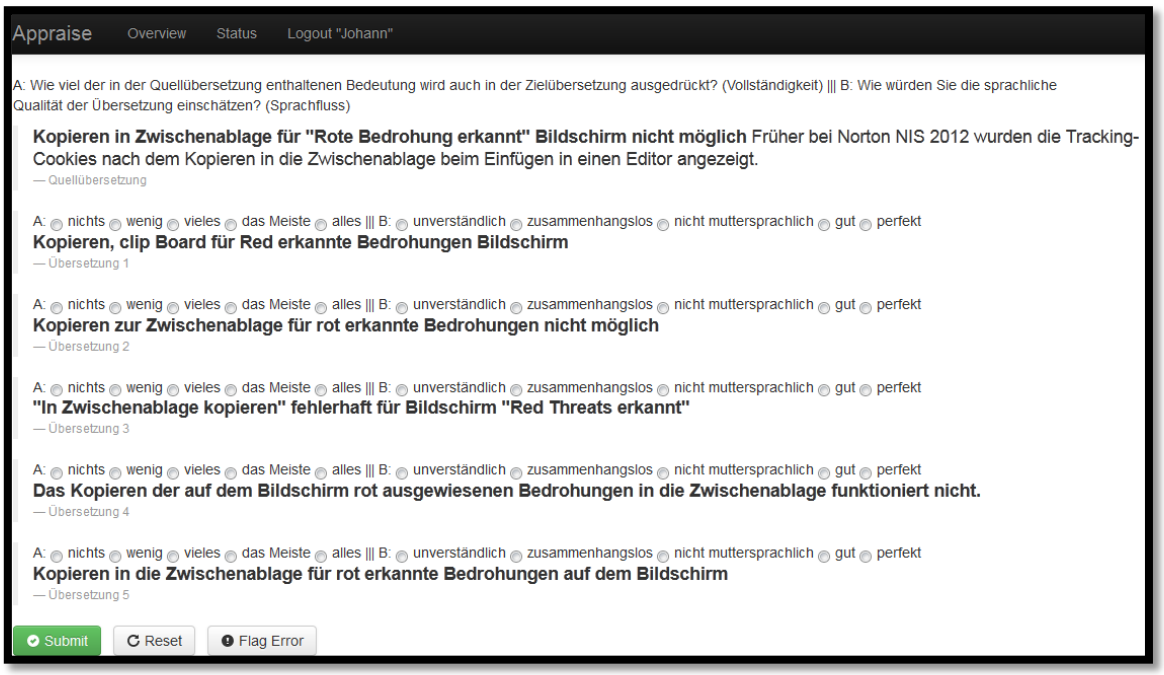


Figure 25: ACCEPT evaluation task using Appraise

- An extra **video_url** field was added to the **EvaluationTask** class. This optional field may be used to embed a video into a user's task page. This can be useful to provide visual evaluation guidelines, instead of relying on text-based guidelines (which are often ignored or misunderstood), as shown below:

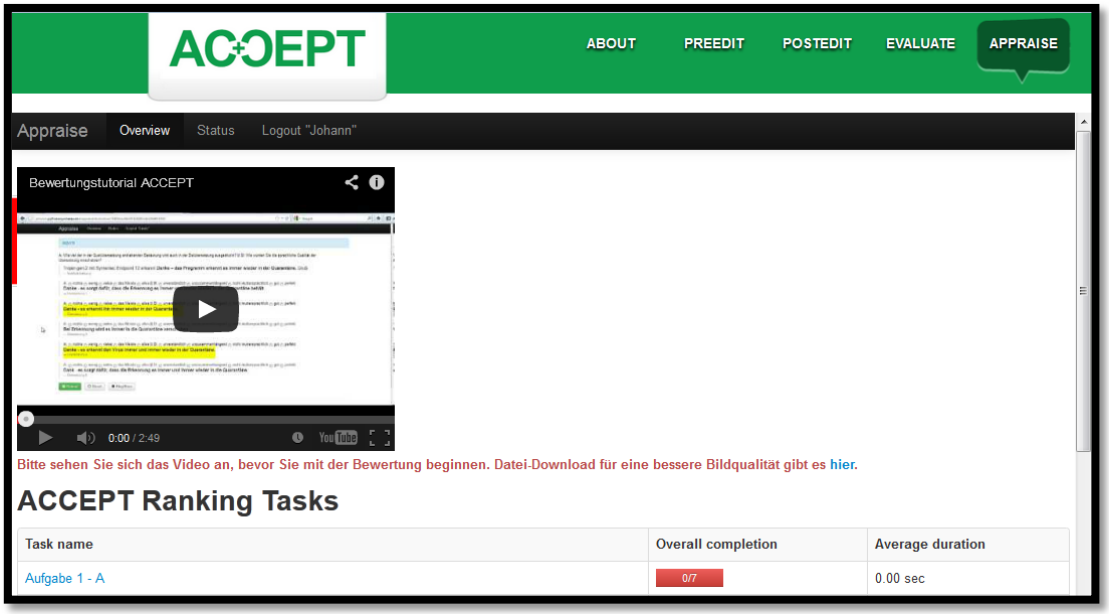


Figure 26: Adding video-based evaluation guidelines to Appraise tasks

- We created a template to export the evaluation results into an XML file:

```
1   {% if skipped %}  <ranking-item{% if attributes%} {{attributes|safe}}{% endif %} duration="{{duration}}" user="{{user}}"
      skipped="true" />
2   {% else %}  <ranking-item{% if attributes%} {{attributes|safe}}{% endif %} duration="{{duration}}" user="{{user}}">
3   {% for attrs, rank, rank2 in translations %}    <translation{% if attrs %} {{attrs|safe}}{% endif %} rank="{{rank}}"
      rank2="{{rank2}}" />
4   {% endfor %}  </ranking-item>
5   {% endif %}
```

5.3.3 Integration

For this initial release, an instance of the modified Appraise system was integrated into the ACCEPT portal using an iframe, as shown below:

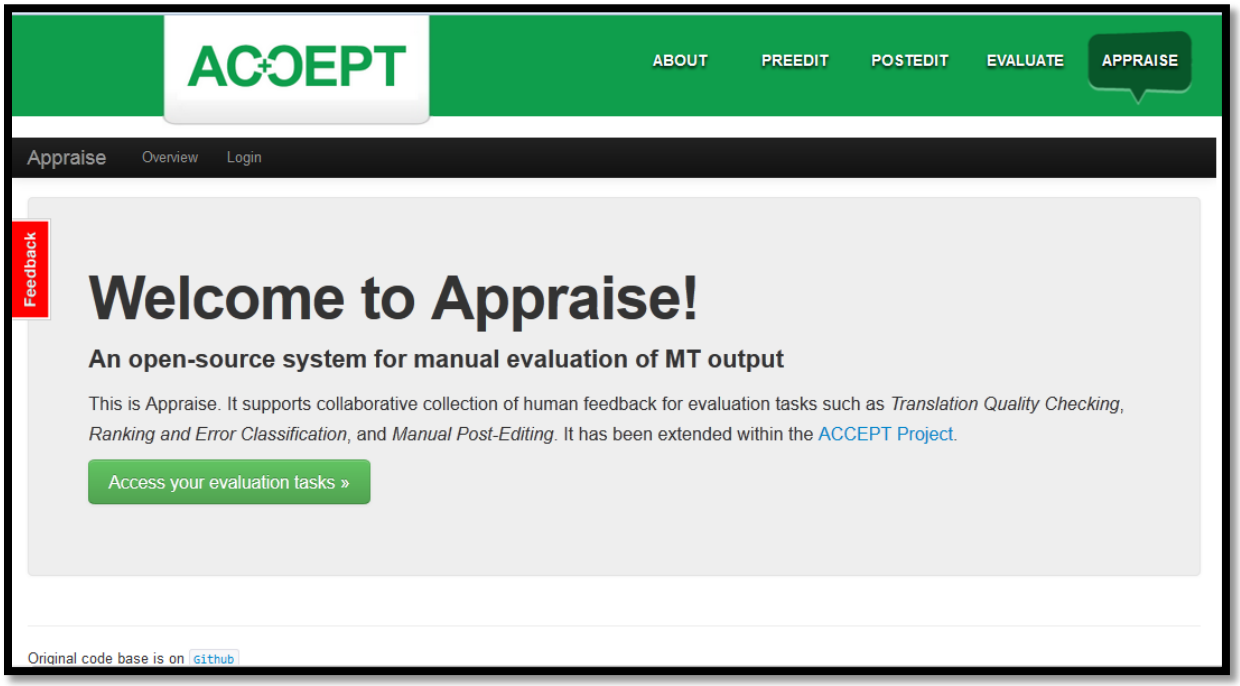


Figure 27: Integrating Appraise within the ACCEPT Portal

This integration may be revisited in the future to determine whether it might be possible to use the ACCEPT credentials to log into the Appraise system.

References

- Christian Federmann:
  Appraise: An Open-Source Toolkit for Manual Evaluation of Machine Translation Output
  In *The Prague Bulletin of Mathematical Linguistics*, volume 98, Prague, Czech Republic, 9/2012.
- Roturier, Johann, Linda Mitchell, David Silva:
  The ACCEPT Post-Editing Environment: a Flexible and Customisable Online Tool to Perform and Analyse Machine Translation Post-Editing.
  In *Proceedings of MT Summit XIV Workshop on Post-editing Technology and Practice*, Nice, France, September 2013.

Appendix 1: External Call Integration for the Pre-Edit plug-in

For this example we are using the simple editor example from the Yahoo! UI 2 download package. Let's suppose this HTML page is actually the environment where we want to integrate the ACCEPT Pre-Edit plug-in.

STEP 1

The first action to perform is always to identify the text source we wish to target. In this example we want the same element used by the Yahoo editor.

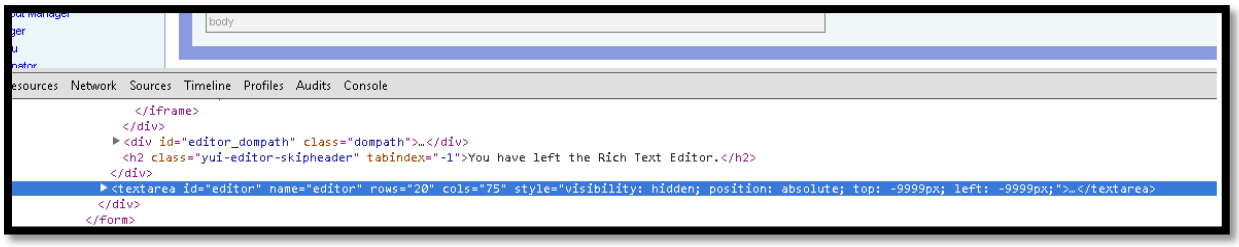


Figure 28: Step 1 of external call example

Now we know that the element with the *editor* ID is the text area where the Yahoo editor is being used and as such the one where we want to implement the ACCEPT Pre-Edit plug-in. The plug-in configuration starts with:

```
$('#editor').Accept({
configurationType: 'externalCall',
...
```

A closer look at the Web page is required to understand where the plug-in dialog could be triggered from (e.g. which HTML button, HTML image, etc.):

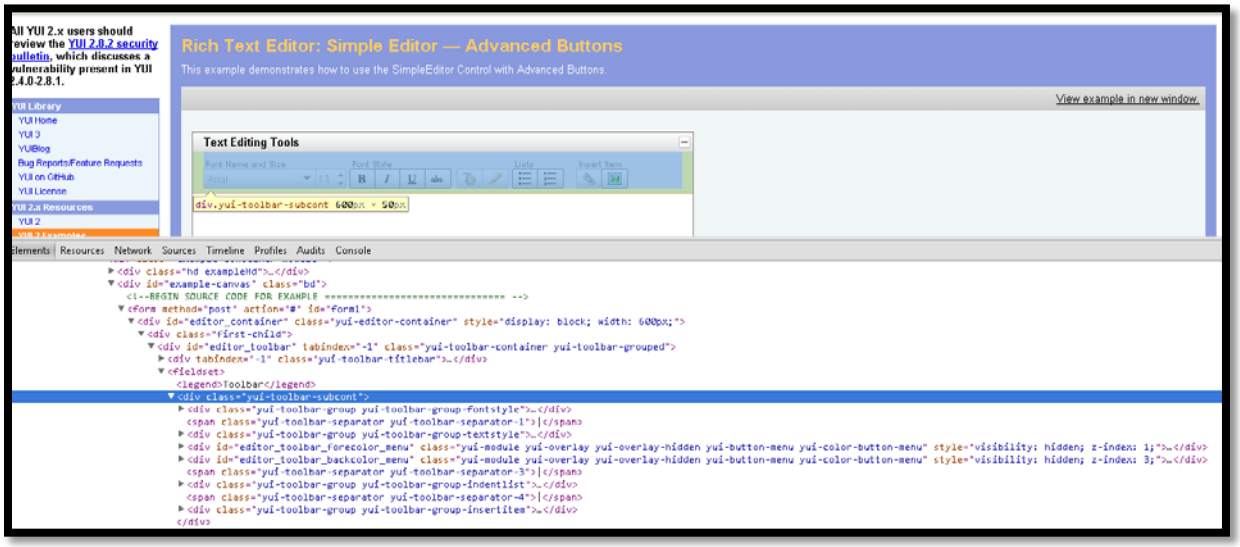


Figure 29: Step 2 of external call example

Looking at the image. it is apparent that a good way to trigger the ACCEPT Pre-Edit dialog would be from a button located in the editor toolbar. Since there are no available buttons, we need to set up the ACCEPT Pre-Edit plug-in to magically create a new HTML element for us.

STEP 2

To add a new HTML element to the toolbar, we need first to identify the HTML element that contains all the toolbar elements.

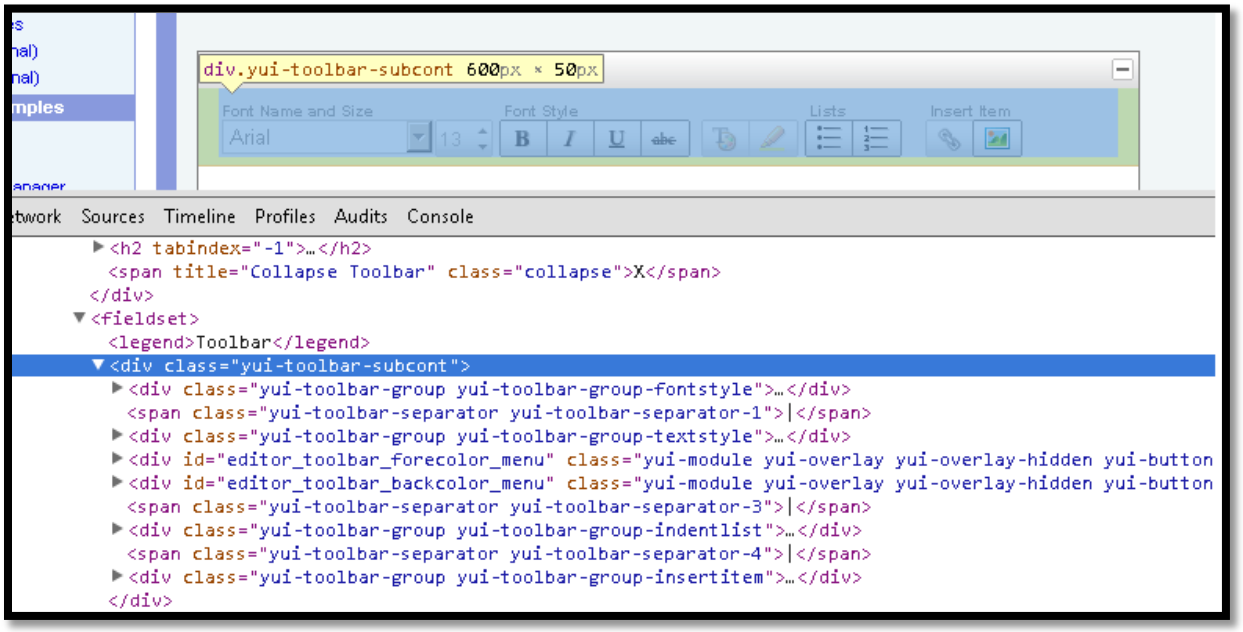


Figure 30: Step 3 of external call example

As shown above, the `div` element selected is indeed the container for all the elements in the editor toolbar. Therefore this `div` element works as a placeholder for the new element. Now that we know what the element is, we also need to find a way to identify it, but the `div` does not have any *id* attribute. This may at first glance look like a problem, but the plug-in can use any valid jQuery selector to find the placeholder. In this case, we know that the CSS class attribute is unique for this element. We can use it to "teach" the ACCEPT Pre-Edit plug-in how to find it:

```
$('#editor').Accept({
configurationType: 'externalCall',
injectSelector: '.yui-toolbar-subcont',
...
```

STEP 3

At this stage we know the text area where the plug-in will be used and the toolbar element where we want to add new HTML content, so now we only need to decide what HTML content we want to add. For this example, we will inject a `div` element containing the ACCEPT *ABC* icon.

```
<div style="float: right;margin-top: 20px;">
<img id="triggerInjectedACCEPTbutton" style="cursor:pointer;height:24px;wid
th:24px;float:right;" src="http://www.accept-portal.eu/Plugin/v2.0/css/imag
es/actions-tools-check-spelling-icon.png">
</div>
```

This needs to be added under the **injectContent** setting as part of the plug-in configuration:

```
$('#editor').Accept({
configurationType:'externalCall',
injectSelector:'.yui-toolbar-subcont',
```

```
injectContent:'<div style="float: right;margin-top: 20px;"><img id="trigger
InjectedACCEPTbutton" style="cursor:pointer;height:24px;width:24px;float:ri
ght;" src="http://www.accept-portal.eu/Plugin/v2.0/css/images/actions-tools
-check-spelling-icon.png"></div>',
...
```

STEP 4

In this step, we need to identify the HTML element from which the click event that displays the ACCEPT dialog is triggered. In STEP 3, we injected code into the page precisely so as to have an extra element to act as the element we are now looking for. Looking carefully at STEP 3 we can see that the *img* element is an ID property we made up for this purpose, so this is the value we want to use for the **triggerCheckSelector** setting, as shown below:

```
$('#editor').Accept({
configurationType:'externalCall',
injectSelector:'.yui-toolbar-subcont',
injectContent:'<div style="float: right;margin-top: 20px;"><img id="trigger
InjectedACCEPTbutton" style="cursor:pointer;height:24px;width:24px;float:ri
ght;" src="http://www.accept-portal.eu/Plugin/v2.0/css/images/actions-tools
-check-spelling-icon.png"></div>',
triggerCheckSelector: '#triggerInjectedACCEPTbutton',
...
```

STEP 5

The main steps are now completed, but in this case they are not sufficient. If we went ahead and tried to run this example with the current configuration, the plug-in would not work because the text content to check is not loaded into the dialog or properly submitted back to the source. The reason is the same as that explained in *STEP 4* of Example 3. Basically the Yahoo! UI editor does not keep the text content in the text area where it was installed but instead within an iframe built during the initialization. This is actually the most common behaviour nowadays in WYSIWYG editors. Let's take a closer look:

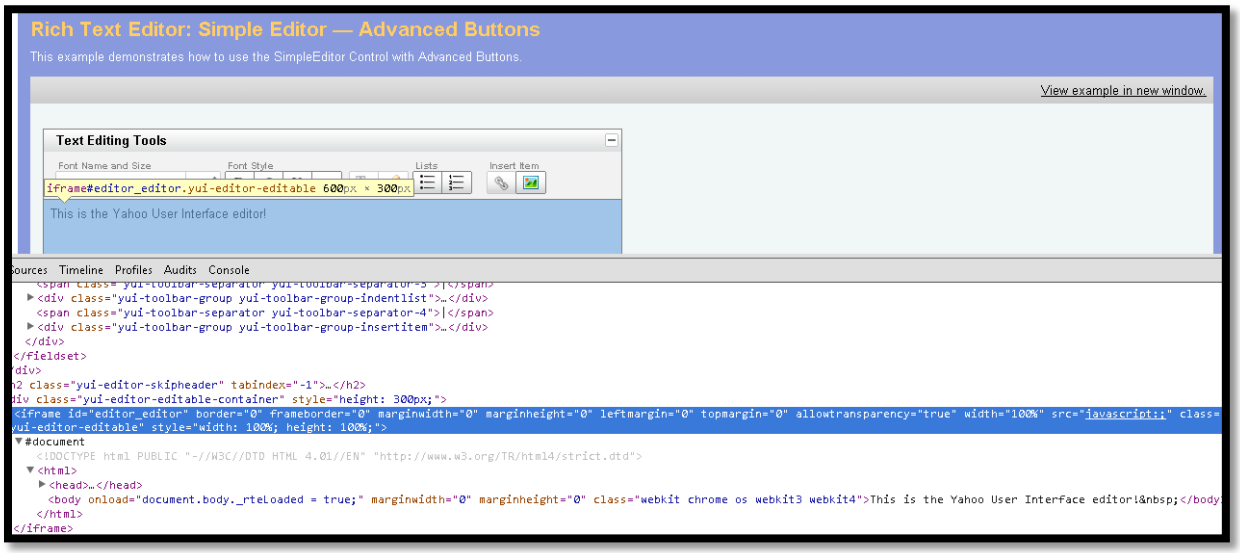


Figure 31: Step 4 of external call example

What do we need to do in order to correct this issue? As part of the plug-in configuration, we need to provide custom methods to get the content into the dialog box and set it back to the source, in this case the iframe with an *editor_editor* ID. Here is the code:

```
$('#editor').Accept({
configurationType:'externalCall',
injectSelector:'.yui-toolbar-subcont',
injectContent:'<div style="float: right;margin-top: 20px;"><img id="trigger
InjectedACCEPTbutton" style="cursor:pointer;height:24px;width:24px;float:ri
ght;" src="http://www.accept-portal.eu/Plugin/v2.0/css/images/actions-tools
-check-spelling-icon.png"></div>',
LoadInputText:function()
{
    return $(document).contents().find('#editor_editor').contents().find('b
ody').html();
},
SubmitInputText:function(text)
{
            $(document).contents().find('#editor_editor').contents().find('
body').html(myContent);
}
```

The ACCEPT Pre-Edit plug-in should now be working! Here is how it should look and below the full code snippet:

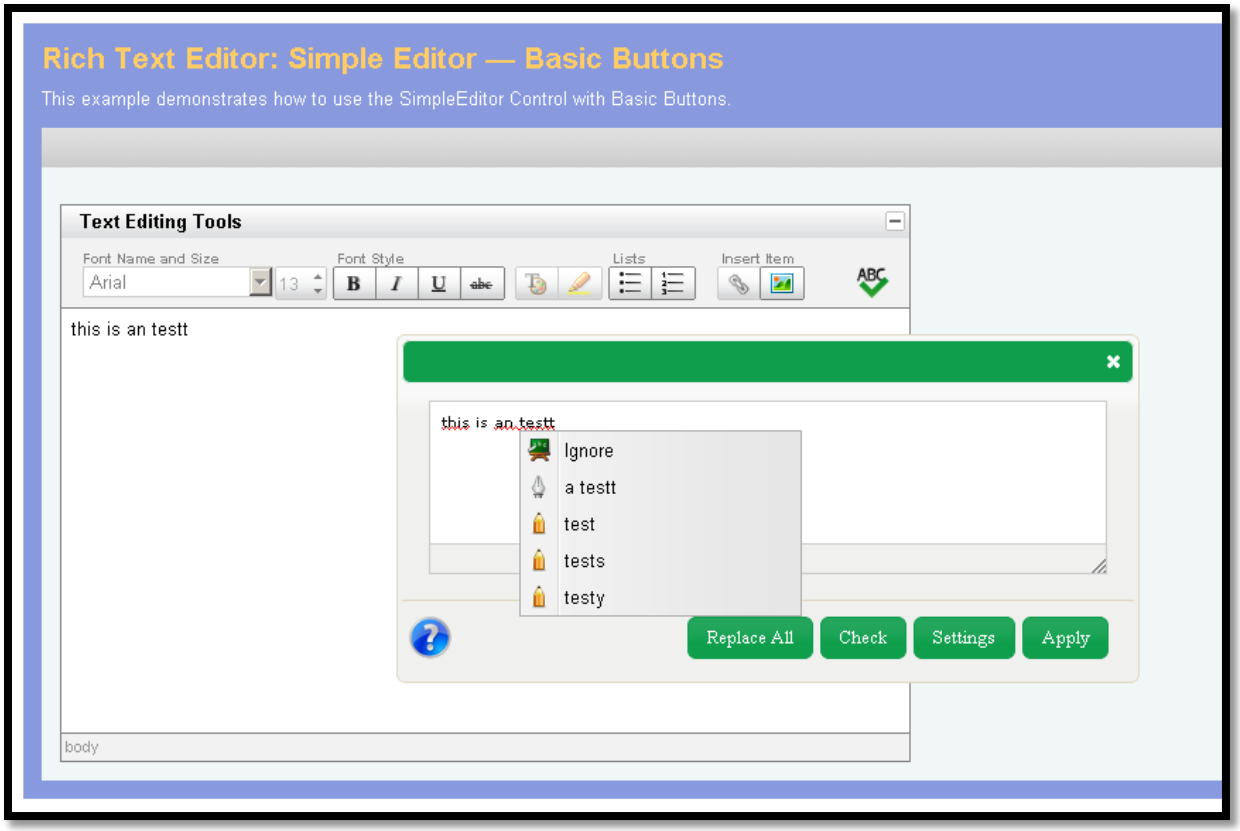


Figure 32: Step 5 of external call example

```
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/jquery-1.5.1.min.js
"></script>
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/jquery-ui-1.8.24.cu
stom.min.js"></script>
<link href="http://www.accept-portal.eu/Plugin/v2.0/css/Accept.css" rel="st
ylesheet" type="text/css" />
<link href="http://www.accept-portal.eu/Plugin/v2.0/css/jquery-ui.css" rel=
"stylesheet" type="text/css" />
<script src="http://www.accept-portal.eu/Plugin/v2.0/extras/tiny_mce/tiny_m
ce.js"></script>
<script src="http://www.accept-portal.eu/Plugin/v2.0/js/accept-jquery-plugi
n-2.0.js?v=9"></script>
<script type="text/javascript">
$(document).ready(function(){
     $('#editor').Accept({
            configurationType:'externalCall',
            injectSelector:'.yui-toolbar-subcont',
            injectContent:'<div style="float: right;margin-top: 20px;"><img
 id="triggerInjectedACCEPTbutton" style="cursor:pointer!important;height:24
px;width:24px;float:right;" src="http://www.accept-portal.eu/Plugin/v2.0/cs
s/images/actions-tools-check-spelling-icon.png"></div>',
            triggerCheckSelector: '#triggerInjectedACCEPTbutton',
            LoadInputText:function()
            {
                return $(document).contents().find('#editor_editor').conten
ts().find('body').html();
            },
            SubmitInputText:function(text)
            {
                $(document).contents().find('#editor_editor').contents().fi
nd('body').html(myContent);
            },
            AcceptServerPath:"http://www.accept-portal.eu/AcceptApiStg/Api/
v1",
            ApiKey:"APIKEY",
            Lang:"en",
            Rule:"Preediting_Forum",
            imagesPath:"http://www.accept-portal.eu/Plugin/v2.0/css/images",
            requestFormat:"HTML",
            languageUi:'en',
            showFixAll:true,
            isModal:false,
            editorWidth:'480px',
            styleSheetPath:'http://www.accept-portal.eu/Plugin/v2.0/css',
            showManualCheck:true
        });
});
```

Appendix 2: Pre-Edit Plug-in Configuration Options

Name	Type	Value
configurationType	String	Default: "contextMenu"

Describes how the plug-in should behave. This parameter can only receive two values, *contextMenu* or *tinyMCEEmbedded*.

Name	Type	Value
AcceptServerPath	String	Default: ""

The URL for the ACCEPT API.

Name	Type	Value
Lang	String	Default: "en"

Language that will be used for the input text. Can be *fr* for French, *en* for English or *de* for German.

Name	Type	Value
imagesPath	String	Default: "../css/images"

The path to the directory that contains all the images used by the plug-in.

Name	Type	Value
tinyMceUrl	String	Default: "extra/tiny_mce/tiny_mce.js"

The path to the tiny MCE JavaScript file. This option is mandatory.

Name	Type	Value
LoadInputText	JavaScript Function	Default: see code below

```
var inputText = "";
inputText = settings.requestFormat == 'TEXT' ? inputText = $("#" + acceptOb
jectId).val() : inputText = $("#" + acceptObjectId).html();
return inputText;
```

Customize the way to load the input text. This parameter is consumed as a function; this means it expected a function to be passed. This function should return the text to check. Example:

```
function () {
   var inputText =  /* INPUT TEXT or HTML */
   return inputText;
}
```

Name	Type	Value
SubmitInputText	JavaScript Function	Default: see code below

```
settings.requestFormat == 'TEXT' ? $('#' + acceptObjectId).val(text) : $('#
' + acceptObjectId).html(text);
```

Customize the way the input text is submitted from the dialog box back into the text editor. This parameter is also consumed as a function, in this case the plug-in expects to pass the text as an input parameter. Example:

```
function (textParameter) { /* SEND THE TEXT OR HTML BACK TO THE TEXT INPUT
AREA */ }
```

Name	Type	Value
languageUi	String	Default: "en"

Language to use for the UI labels. Currently the following languages are supported:

- *en* = English
- *fr* = French
- *de* = German

Name	Type	Value
requestFormat	String	Default: "TEXT"

Format of the text input, which can be *TEXT* for text content or *HTML* for text containing markup language.

Name	Type	Value
Rule	String	Default: The language-specific rule set.

Optional rule set that should be used by the Acrolinx server to check the content. By default, a language-specific rule set will be used.

Name	Type	Value
checkingLevels	String Array	Default: []

Instead of defining the **Rule** setting above it is also possible to define multiple rule sets by using the **checkingLevels** setting. The rule names provided will then be interpreted as checking levels where the first rule name provided is used to perform the first content check. Subsequent checks can then be triggered by clicking the remaining check level buttons.

Name	Type	Value
rightClickEnable	Boolean	Default: false

Indicates whether the right-click context menu should be active.

Name	Type	Value
showFixAll	Boolean	Default: false

Indicates whether the :guilabel:`Replace All` button should be active.

Name	Type	Value
isModal	Boolean	Default: true

Indicates whether the dialog box that shows the results should behave as a modal.

Name	Type	Value
isModal	Boolean	Default: true

Indicates whether the dialog box that shows the results can be dragged.

Name	Type Value	
dialogHeight	Number/String	Default: "auto"

Height (in pixels) of the dialog box that shows the results. Alternatively the value can be set to the string *auto*.

Name	Type Value	
dialogWidth	Number/String	Default: "auto"

Width (in pixels) of the dialog box that shows the results. Alternatively the value can be set to the string *auto*.

Name	Type	Value
placeHolderMaxHeight	Number/String	Default: $(window).height()

Maximum height (in pixels) of the dialog box that shows the results.

Name	Type	Value
placeHolderMaxWidth	Number/String	Default: $(window).width()

Maximum width (in pixels) of the dialog box that shows the results.

Name	Type	Value
placeHolderMinHeight	Number/String	Default: 100

Minimum height (in pixels) of the dialog box that shows the results.

Name	Type	Value
placeHolderMinWidth	Number/String	Default: 380

Minimum width (in pixels) of the dialog box that shows the results.

Name	Type	Value
showManualCheck	Boolean	Default: false

When set to *true*, this button allows the user to manually re-check the content.

Name	Type	Value
styleSheetPath	String	Default: "../css"

Path to the **Accept.css** file. This path needs to be set correctly in order to inject the necessary styles in the tinyMCE iframe (rendered within the dialog).

Name	Type	Value
htmlBlockElements	String	Default: "p ,h1 ,h2 ,h3 ,h4 ,h5 ,h6 ,ol ,ul ,li ,pre ,address ,blockquote ,dl ,div ,fieldset ,form ,hr ,noscript ,table"

List of block level HTML elements the plug-in should consider. A control node is added after each HTML element in this list to simulate a line break in the content to check. These nodes are removed before the content is applied back to the source placeholder.

Name	Type	Value
refreshStatusAttempts	Number	Default: 5

Number of attempts the plug-in will make to check if the response containing the results for the content sent to check are ready. If the attempts' value reaches the threshold limit, subsequent manual triggers may be needed by the user.

Name	Type	Value
editorWidth	Number/String	Default: "380px"

Initial width of the dialog inline editor.

Name	Type	Value
editorHeight	Number/String	Default: "80px"

Initial height of the dialog inline editor.

Name	Type	Value
getSessionUser	JavaScript/Function	Default: function () { return ""; }

It is possible to configure the plug-in to search for end user information (login name, etc...) and attach that info as part of the metadata collected. This can be achieved by writing the necessary code to search and then mandatorily return a string value representative of the desired information.

Name	Type	Value
injectSelector	jQuery Selector/String	Default: null

When the plug-in is configured in **External Call** mode, this setting combined with the **injectContent** setting is used to inject extra HTML code on the page. The idea here is to trigger the content check from one element injected by the combination of these properties. The value expected is a jQuery

selector, the selector being used to find at least one existing DOM element where the new HTML code (provided via the **injectContent** setting value) is injected.

Name	Type	Value
injectContent	HTML/String	Default: null

The HTML content that is injected in the element(s) found by the jQuery selector provided in the setting **injectSelector**.

Name	Type	Value
injectWaitingPeriod	Number	Default: 100

Value in milliseconds the plug-in should wait to inject the HTML content from the **injectContent** setting into the DOM elements matched in the jQuery selector provided by the **injectSelector** setting.

Name	Type	Value
triggerCheckSelector	jQuery Selector/String	Default: null

Expects a jQuery selector. This value is used to find the DOM element from where a mouse click will trigger the content check.

Name	Type	Value
timeoutWaitingTime	Number	Default: 7000

Value in milliseconds that Ajax requests take before falling into a timeout exception.

Appendix 3: First Steps with the ACCEPT Portal and Plug-ins

In order to get started with the ACCEPT Portal and its plug-ins, a user should decide whether their profile best corresponds to:

- An online community member who is interested in authoring textual content that is easier to understand by other community members and easier to translate into other languages;
- An online community member who is interested in helping make previously translated content understandable (e.g. translated textual content that may have been automatically translated is likely to require some editing);
- An online community content or technology owner or manager who is interested in giving community members some assistance in editing textual content (either original or translated content) or who is interested in collecting ratings for translated content.

If the user belongs to the first category and is somewhat proficient in English, French or German, they may want to:

- Go to the Demos section to become familiar with the ACCEPT online Pre-Edit demos, which show how the Pre-Edit plug-in can be used to make their textual content easier to understand and translate;
- Get in touch with their online community manager or community content technology owner to make them aware of the ACCEPT Pre-Edit plug-in (which they can easily download and integrate into the online platform that the user is currently using to contribute community content).

If the user belongs to the second category, they may want to:

- Go to the Demos section to become familiar with the ACCEPT online Post-Edit demos which show how the Post-Edit plug-in can be used to make machine-translated content easier to understand;
- Get in touch with their online community manager to discuss whether some of their community's textual content could be translated into other languages, using a combination of translation suggestions (possibly provided by machine-translation providers) and post-editing. The ACCEPT Portal allows project managers to create Post-Edit tasks, so the user could be assigned some of these small tasks very easily.
- Go to the Working on tasks section to become familiar with the ACCEPT Post-Edit plug-in.

If the user belongs to the third category, they may want to:

- Become familiar with the Pre-Edit, Post-Edit, Evaluation and Portal parts of the documentation. If they are technical, they may want to specifically look into the Plug-in and Managing an external Post-Edit project sections to understand better how the plug-ins can be integrated into their own environment. If they are interested in putting a mechanism in place to collect ratings for translated content, they may also want to take a look at an evaluation Client example.
- Identify community members who may be interested in authoring their textual content with the Pre-Edit plug-in or conducting Post-Edit tasks. They may need to communicate to these members the value of content editing and find ways to motivate them in using these tools.

Appendix 4: Overview of the JSON Response Format

A request made against the **GlobalSessionDomain** method returns a response in JSON format. A response example is shown below.


```
{
    "AcceptSessionCode": "",
    "GlobalSessionId": "",
    "ResponseObject": [
        {
            "ChildSessions": [
                {
                    "ClientResults": [...],
                    "Context": "If is installed, this demo is used as a testt to explain how the report is structred.
                    In this demo we have the possibility to change most, if not all, rules that are used to check the
                    text by selecting an option from the \u201cRule Set\u201d drop-down list.
                    Results does not vary depending on your Linux distro.\n",
                    "ProviderResults": [...],
                    "Results": [...]
                }
            ],
            "GlobalEndTime": "2013-11-26T11:08:08.000Z",
            "GlobalStartTime": "2013-11-26T11:06:25.000Z",
            "Input": "If is installed, this demo is used as a testt to explain how the report is structred.
            In this demo we have the possibility to change most, if not all, rules that are used to check the text by
            selecting an option from the \u201cRule Set\u201d drop-down list.
            Results does not vary depending on your Linux distro.\n",
            "MetaData": {
                "GlobalSessionId": "44f35e3a_40c0acb495fe4cc4aa800c6f951c8c9d",
                "Language": "en",
                "RuleSet": "Preediting_Forum",
                "User": "-55642226"
            },
            "Output": "If it is installed, this demo is used as a test to explain how the report is structured.
            In this demo we have the possibility to change most, if not all, rules that are used to check the text by
            selecting an option from the \u201cRule Set\u201d drop-down list.
            Results do not vary depending on your Linux distro."
        }
    ],
    "ResponseStatus": "OK",
    "TimeStamp": "/Date(1385465000747)/"
}
```


Figure 33: Structure of usage report in JSON format

When a request is successful, the **ResponseObject** value of the response contains the following information:

- When the checking session started, as indicated by the **GlobalStartTime** value
- When the checking session ended, as indicated by the **GlobalEndTime** value
- The original text sent for checking, as indicated by the **Input** value
- The final text at the end of checking sessions, as indicated by the **Output** value
- The language configuration used by the language checking provider, as indicated by the **Language** value in **Metadata**
- The rule set used by the language checking provider, as indicated by the **RuleSet** value in **Metadata**
- A unique anonymised value of the user that triggered the checking session, as indicated by the **User** value in Metadata
- An array of **ChildSessions** containing precise information about at least one specific check performed on the text present in the **Context** value. Three sets of results are available for each check:
    - **ProviderResults**, which correspond to the detailed output generated by the language checking provider
    - **ClientResults**, which correspond to actions performed in the **Settings** window, as explained below.
    - **Results**, which correspond to actions performed in the main window, as explained below.

```
"ClientResults": [
    {
        "Action": "remove_learn_word",
        "ActionValue": "",
        "FlagContext": "structred",
        "Name": "spelling_flag"
    },
    {
        "Action": "remove_ignore_rule",
        "ActionValue": "",
        "FlagContext": "",
        "Name": "use_comma_after_introductory_phrase"
    }
],
```


Figure 34: ClientResults usage information


```
"Results": [
    {
        "Action": "learn_word",
        "ActionValue": "",
        "FlagContext": "structred",
        "IndexEnd": 84,
        "IndexStart": 75,
        "Name": "spelling_flag"
    },
    {
        "Action": "learn_word",
        "ActionValue": "",
        "FlagContext": "distro",
        "IndexEnd": 300,
        "IndexStart": 294,
        "Name": "spelling_flag"
    },
    {
        "Action": "ignore_rule",
        "ActionValue": "",
        "FlagContext": "In this demo we have the possibility to change most, if not all, " +
    "rules that are used to check the text by selecting an option from the \u201cRule Set\u201d drop-down list",
        "IndexEnd": 246,
        "IndexStart": 86,
        "Name": "sentence_too_long"
    },
    {
        "Action": "ignore_rule",
        "ActionValue": "",
        "FlagContext": "In",
        "IndexEnd": 88,
        "IndexStart": 86,
        "Name": "use_comma_after_introductory_phrase"
    },
    {
        "Action": "display_tooltip",
        "ActionValue": "",
        "FlagContext": "If is",
        "IndexEnd": 5,
        "IndexStart": 0,
        "Name": "wrong_sequence_of_words"
    },
    {
        "Action": "accept_suggestion",
        "ActionValue": "Results do",
        "FlagContext": "Results does",
        "IndexEnd": 260,
        "IndexStart": 248,
```


Figure 35: Results usage information

In the example shown in Figure 35, the **ClientResults** section of the report allows us to understand that the user:

- Removed the word *structred* that they had previously learnt
- Removed the *use_comma_after_introductory_phrase* rule that they had previously decided to ignore.

In the example above, the Results section of the report allows us to understand that the user (among other things):

- Decided to learn the 6-letter word *distro*, which had been flagged between indexes 294 and 300 (but not including index 300) as a spelling error
- Decided to ignore the *sentence_too_long* rule, based on an instance starting at index 86

- Displayed the recommendation tooltip for the *wrong_sequence_of_words* rule at index 0 but decided not to ignore the rule
- Decided to accept the suggestion ‘Results do’ for ungrammatical phrase ‘Results does’ at index 248.