User Manual

PET

Wilker Aziz — University of Wolverhampton
Lucia Specia — University of Sheffield

September 30, 2012

Contents

I know the code is still a bit messy and poorly documented. I am working on that, at least once a week I am giving support to this code.

If you can think of a useful feature please let me know. By the way, please do report bugs!

Wilker Aziz.

Chapter 1

Running

1.1 Downloading

You can download PET already built and with a bunch of example files from `http://pers-www.wlv.ac.uk/~in1676/pet`. This version is usually slightly customized and if just want to try PET out this is probably what you are looking for. After downloading you can start using it almost directly (check Section 1.2).

You can also git clone the code, specially if you intend to contribute:

```
git clone https://github.com/wilkeraziz/PET.git
```

In this case you will need to build it before using (see Section 1.3).

1.2 Requirements

PET only requires a recent version of the Java Virtual Machine (`1.6.0_26` or superior), which is likely to be already installed if you keep your machine up to date.

If you have downloaded this from `http://pers-www.wlv.ac.uk/~in1676/pet`, no building is required, keep following this section then go to Section 1.4. If you git cloned the code, check Section 1.3, then proceed with this one.

On Microsoft Windows it is important to check whether the PATH to the Java binary files is properly set. Again, this may be already set. If you try to run the tool **??** and it does not work, follow the guidelines in Section 1.2.1 to set the path.

1.2.1 Windows

1. Do I have the right JVM?

   (a) Open a terminal

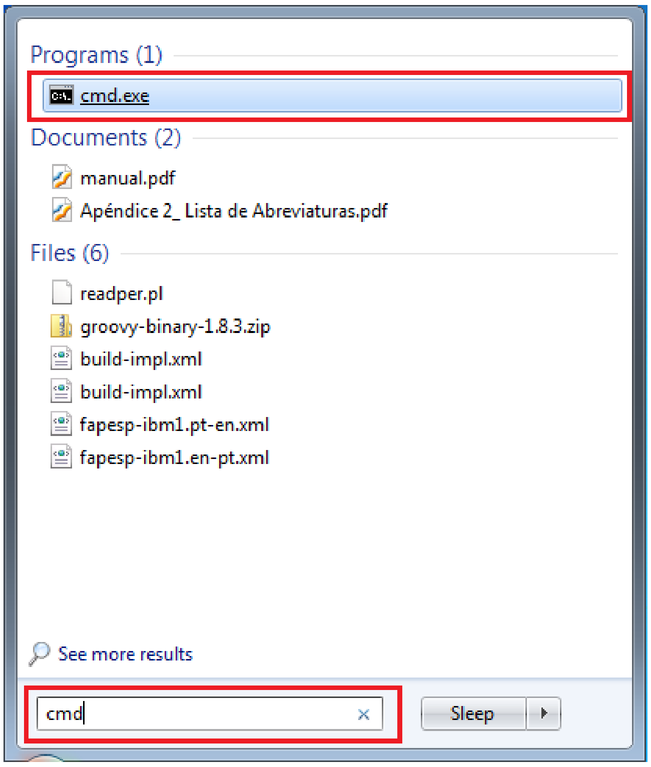
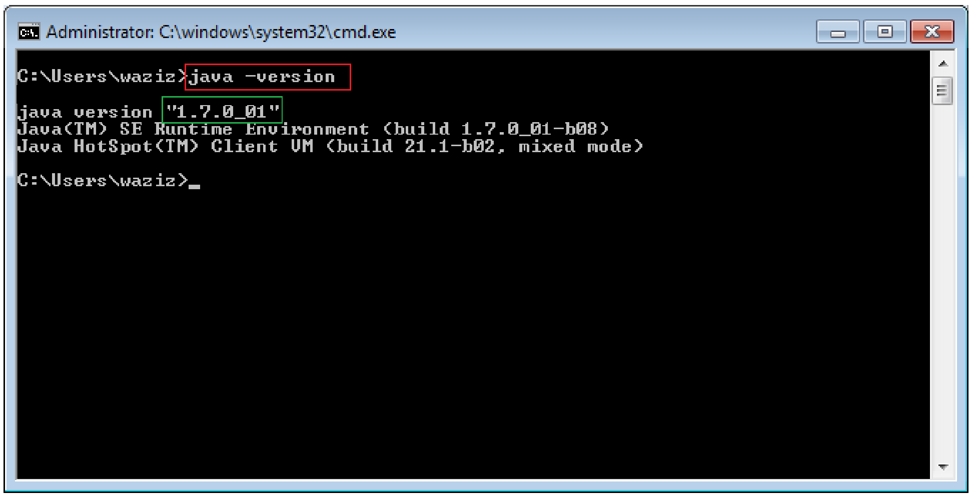


Figure 1.1: Terminal

(b) Type in `java -version`



```
C:\Users\waziz>java -version

java version "1.7.0_01"
Java(TM) SE Runtime Environment (build 1.7.0_01-b08)
Java HotSpot(TM) Client VM (build 21.1-b02, mixed mode)

C:\Users\waziz>_
```

Figure 1.2: Java version

2. What if my computer does not recognize the command `java`?

   (a) Make sure you have downloaded and installed the JVM(`http://java.com/en/download/index.jsp`).

3. I am sure I have installed the JVM, yet I don't get it working, what can I do?

   (a) Open your computer's properties

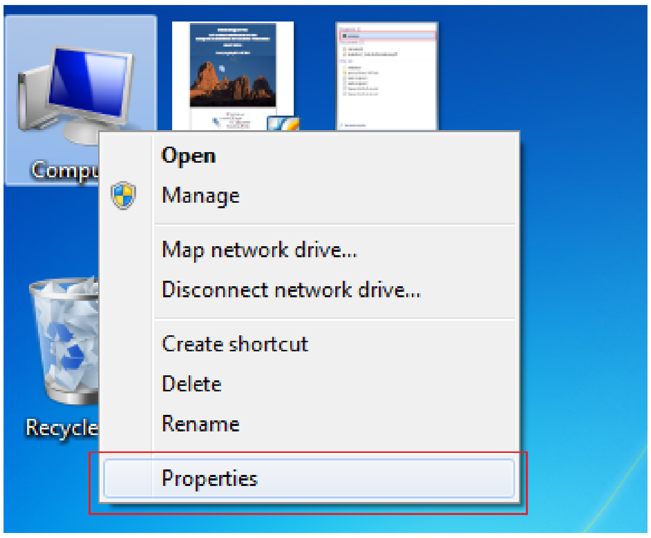
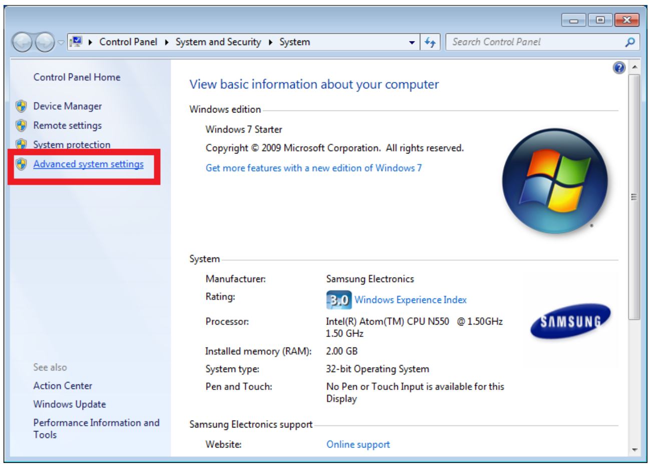


Figure 1.3: Properties

(b) Open the ‘Advanced Settings’



Control Panel ▸ System and Security ▸ System

Search Control Panel

Control Panel Home

Device Manager

Remote settings

System protection

Advanced system settings

See also

Action Center

Windows Update

Performance Information and Tools

View basic information about your computer

Windows edition

Windows 7 Starter

Copyright © 2009 Microsoft Corporation. All rights reserved.

Get more features with a new edition of Windows 7

System

Manufacturer: Samsung Electronics

Rating: 3.0 Windows Experience Index

Processor: Intel(R) Atom(TM) CPU N550 @ 1.50GHz 1.50 GHz

Installed memory (RAM): 2.00 GB

System type: 32-bit Operating System

Pen and Touch: No Pen or Touch Input is available for this Display

Samsung Electronics support

Website: Online support

Figure 1.4: Advanced settings

(c) Go to ‘Environment variables’

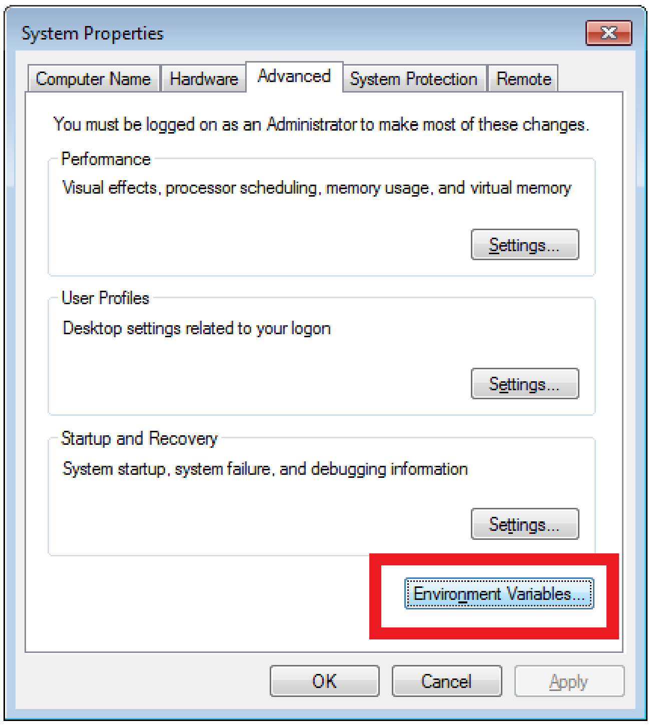
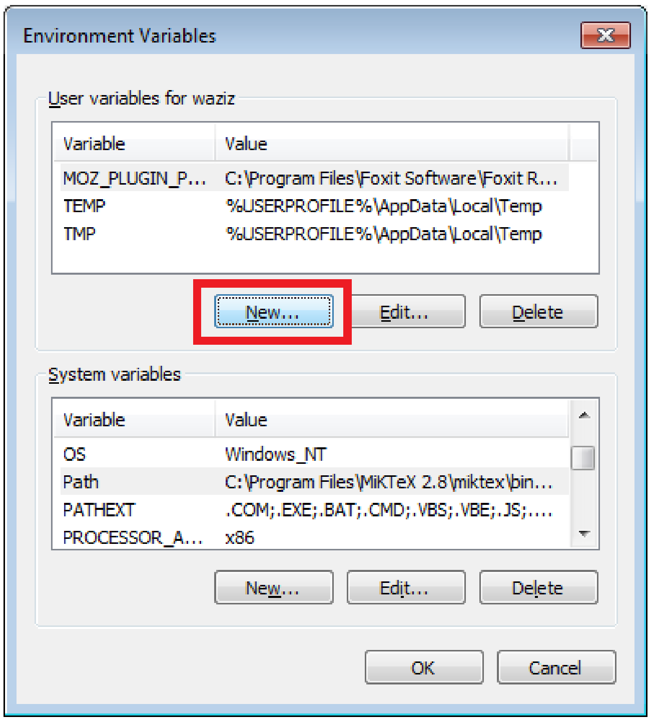

System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

Figure 1.5: Environment variables

(d) Add a new variable

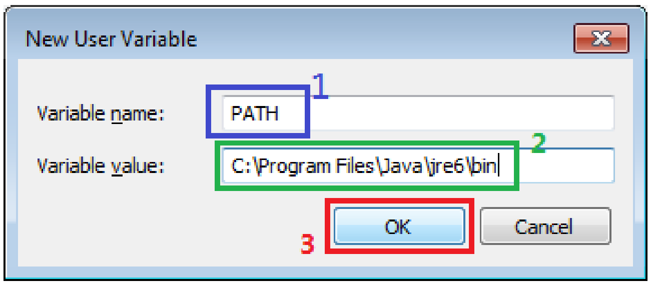


Variable	Value
MOZ_PLUGIN_P...	C:\Program Files\Foxit Software\Foxit R...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
OS	Windows_NT
Path	C:\Program Files\MiKTeX 2.8\miktex\bin...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	x86

Figure 1.6: New user variable

(e) Set variable's name and value to 'PATH' and '<path-to-java>' respectively

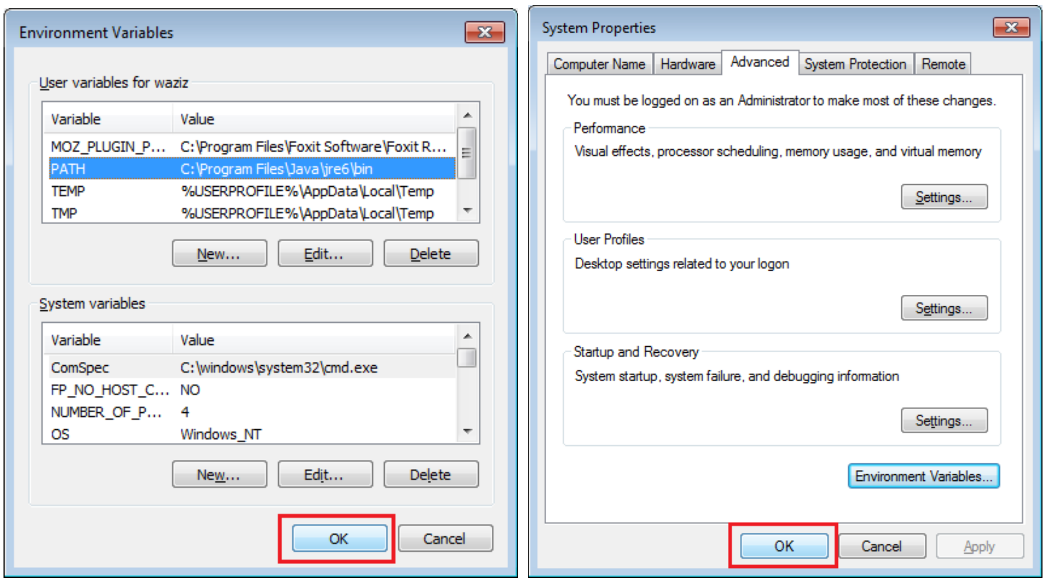


Variable name: PATH

Variable value: C:\Program Files\Java\jre6\bin

Figure 1.7: Java's path

(f) Conclude



User variables for waziz

Variable	Value
MOZ_PLUGIN_P...	C:\Program Files\Foxit Software\Foxit R...
PATH	C:\Program Files\Java\jre6\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	4
OS	Windows_NT

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

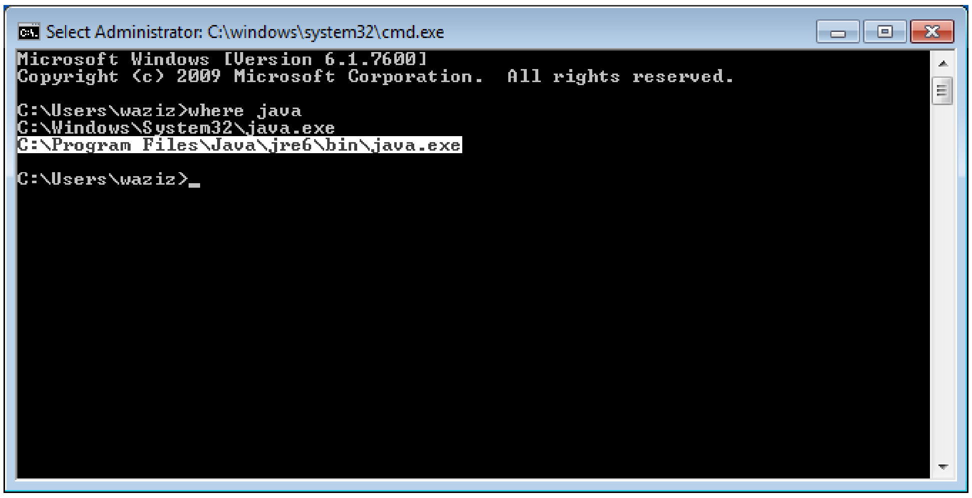
Startup and Recovery
System startup, system failure, and debugging information

Figure 1.8: Conclude

4. How do I find the path to Java binaries?

   (a) Open a terminal (see Figure 1a)

   (b) Type in `where java`



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\waziz>where java
C:\Windows\System32\java.exe
C:\Program Files\Java\jre6\bin\java.exe

C:\Users\waziz>_
```

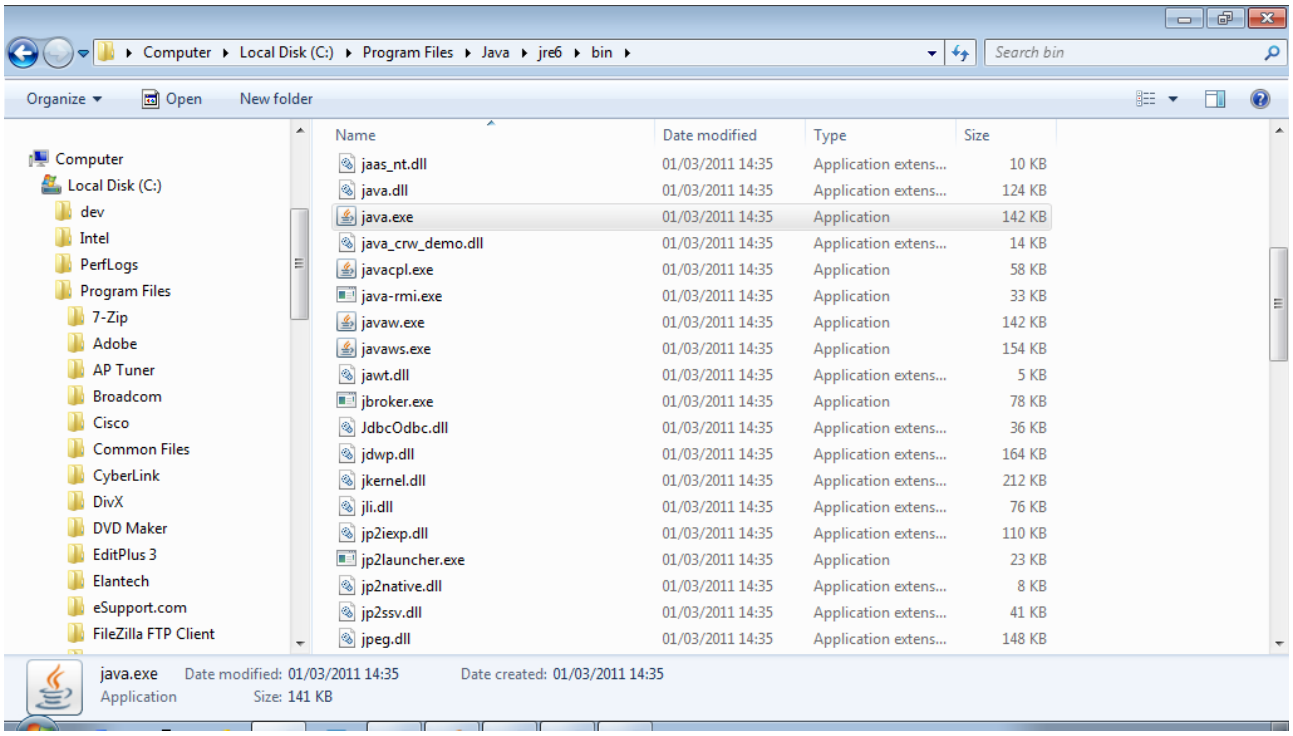
Figure 1.9: Finding Java

If this does not work you may try the following:

(c) Open the Windows Explorer (e.g. double click 'Computer')

(d) Find the directory where you have installed Java (usually `C:\Program Files\Java`)

(e) Go to `jre6\bin`

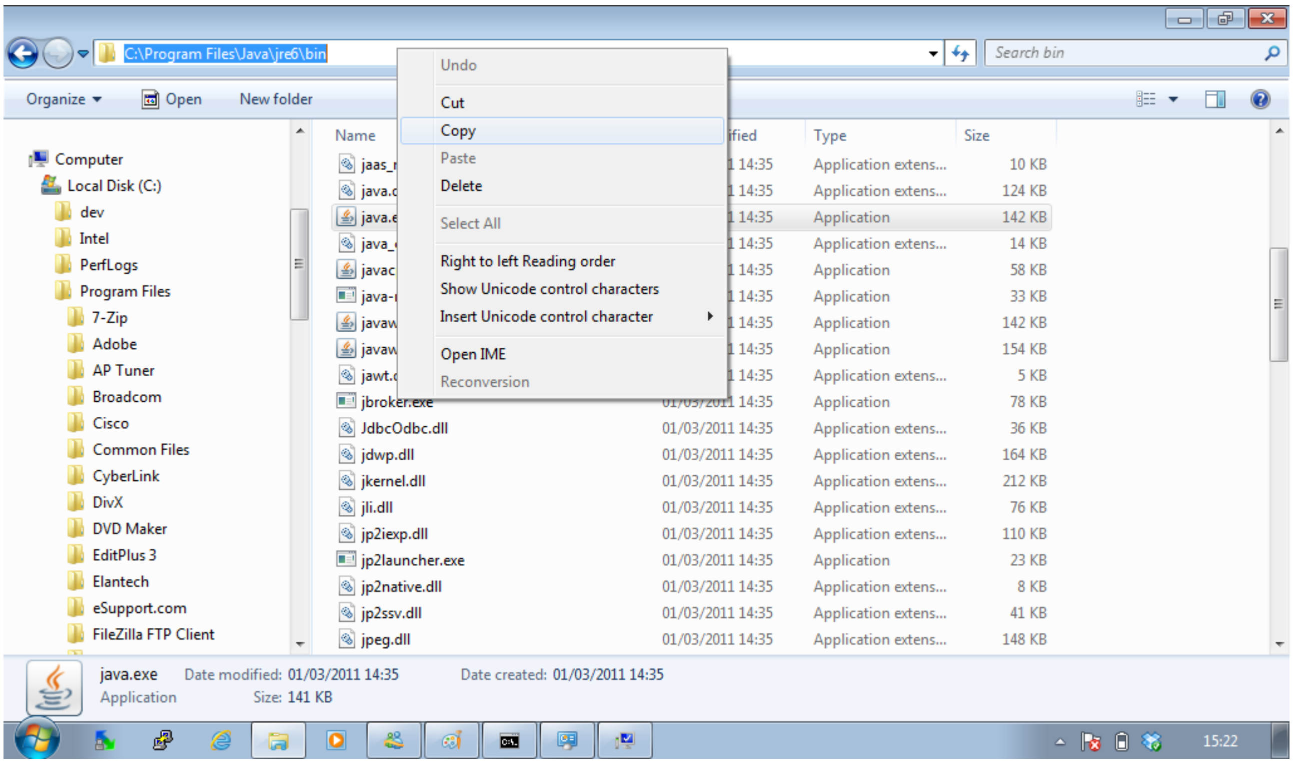


Name	Date modified	Type	Size
jaas_nt.dll	01/03/2011 14:35	Application extens...	10 KB
java.dll	01/03/2011 14:35	Application extens...	124 KB
java.exe	01/03/2011 14:35	Application	142 KB
java_crw_demo.dll	01/03/2011 14:35	Application extens...	14 KB
javacpl.exe	01/03/2011 14:35	Application	58 KB
java-rmi.exe	01/03/2011 14:35	Application	33 KB
javaw.exe	01/03/2011 14:35	Application	142 KB
javaws.exe	01/03/2011 14:35	Application	154 KB
jawt.dll	01/03/2011 14:35	Application extens...	5 KB
jbroker.exe	01/03/2011 14:35	Application	78 KB
JdbcOdbc.dll	01/03/2011 14:35	Application extens...	36 KB
jdwp.dll	01/03/2011 14:35	Application extens...	164 KB
jkernel.dll	01/03/2011 14:35	Application extens...	212 KB
jli.dll	01/03/2011 14:35	Application extens...	76 KB
jp2iexp.dll	01/03/2011 14:35	Application extens...	110 KB
jp2launcher.exe	01/03/2011 14:35	Application	23 KB
jp2native.dll	01/03/2011 14:35	Application extens...	8 KB
jp2ssv.dll	01/03/2011 14:35	Application extens...	41 KB
jpeg.dll	01/03/2011 14:35	Application extens...	148 KB

java.exe Date modified: 01/03/2011 14:35 Date created: 01/03/2011 14:35
Application Size: 141 KB

Figure 1.10: Java installation folder

(f) Copy the path from the address bar



Name	Date modified	Type	Size
jbroker.exe	01/03/2011 14:35	Application	78 KB
JdbcOdbc.dll	01/03/2011 14:35	Application extens...	36 KB
jdwp.dll	01/03/2011 14:35	Application extens...	164 KB
jkernel.dll	01/03/2011 14:35	Application extens...	212 KB
jli.dll	01/03/2011 14:35	Application extens...	76 KB
jp2iexp.dll	01/03/2011 14:35	Application extens...	110 KB
jp2launcher.exe	01/03/2011 14:35	Application	23 KB
jp2native.dll	01/03/2011 14:35	Application extens...	8 KB
jp2ssv.dll	01/03/2011 14:35	Application extens...	41 KB
jpeg.dll	01/03/2011 14:35	Application extens...	148 KB

java.exe Date modified: 01/03/2011 14:35 Date created: 01/03/2011 14:35
Application Size: 141 KB

Figure 1.11: Path to binaries

1.3 Building

This is only necessary if you git cloned the code. Use ant to build PET. After cloning the code, enter its directory and run:

```
ant compile jar javadoc
```

This will produce the directories `build`, `dist` and `dist/javadoc`. You are probably ready to run PET now (see Section 1.4).

1.4 How to run

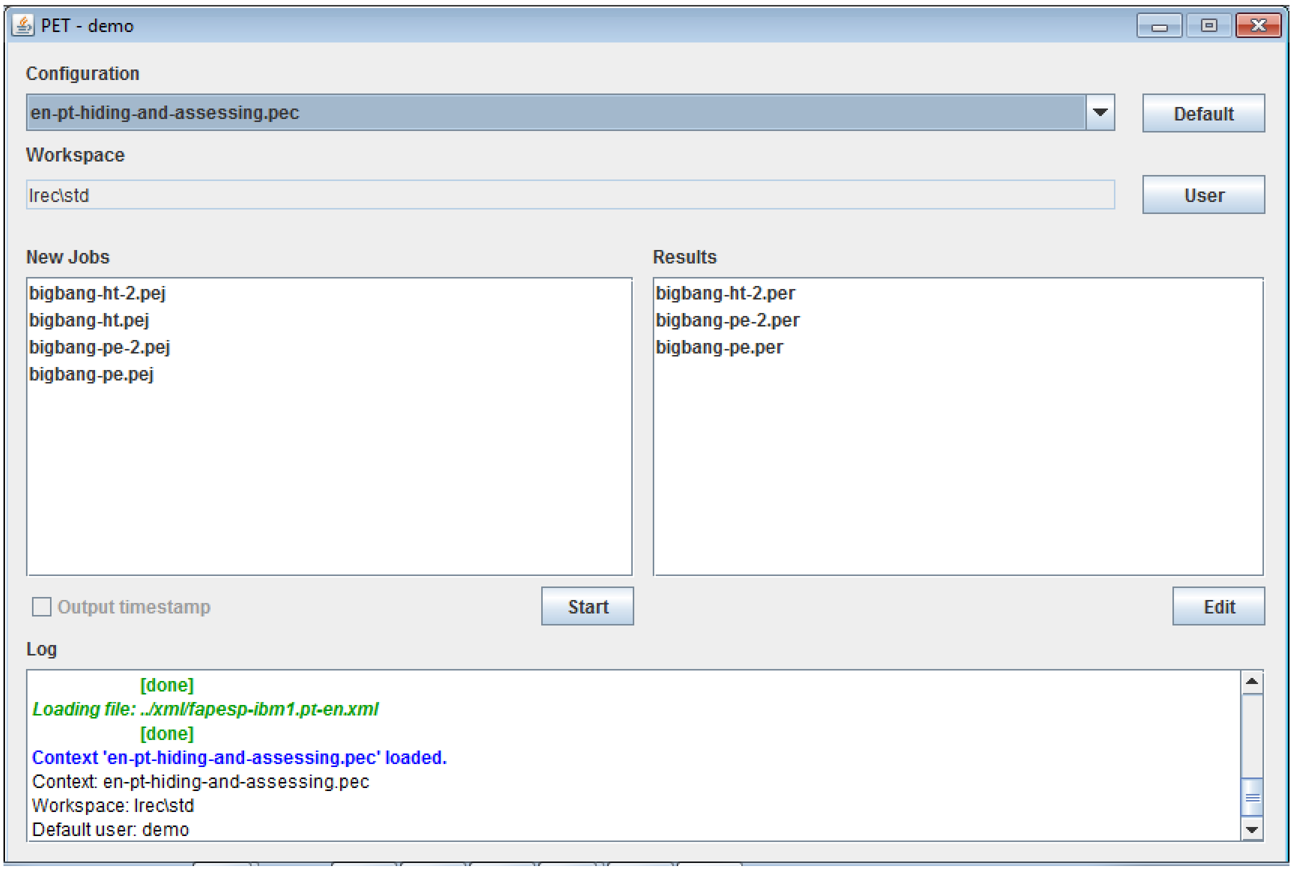
Simply execute the file `run.bat`. On Windows double clicking the file will be enough. On Linux or Mac if the “executable permission” is set, double-clicking the file then selecting “run on terminal” will be enough. If not, set it under the tab “Permissions” right-clicking the file, or open a terminal and type `sh run.bat`.

Chapter 2

Interface

2.1 Main

The main page (see Figure 2.1) is the page that you see when PET is loaded. Observe that the title displays the default *user* of the tool ("demo" in this example), as set in the configuration file.



en-pt-hiding-and-assessing.pec

lrec\std

New Jobs:
bigbang-ht-2.pej
bigbang-ht.pej
bigbang-pe-2.pej
bigbang-pe.pej

Results:
bigbang-ht-2.per
bigbang-pe-2.per
bigbang-pe.per

```
                    [done]
Loading file: ../xml/fapesp-ibm1.pt-en.xml
                    [done]
Context 'en-pt-hiding-and-assessing.pec' loaded.
Context: en-pt-hiding-and-assessing.pec
Workspace: lrec\std
Default user: demo
```

Figure 2.1: PET's main page

Figures 2.1 shows:

`Configuration` allows the selection of a configuration file, also referred to as *context*, where all the job and interface options are set;

`Workspace` displays the *workspace* defined by the *context*, that is, where the files with the jobs to be performed are stored, and where the edited jobs will be saved;

`Jobs` list of available *jobs*, where jobs can be a collection of segments to post-edit, revise or translate, or any combination of these;

`Results` list of (partial) *results*, that is, the output of jobs that have been started (completed or ongoing);

`Log` lists the progress of actions, such as loading a new *context*, loading dictionaries, etc.;

`Output timestamp` if set, adds timestamps the *results* files; if not set, a single output file with the same name as the input file will be created, with a different extension;

`Default` (re-)loads the *default context*, if set in a meta file (see 3.1);

`User` switches to a different *user*;

`Start` starts a *job* from zero;

`Edit` continues a (partial) *result*;

Once one loads the tool using `run.bat` the main page is displayed and a default context is loaded (if specified in a meta file called `pec.meta`). Loading a context may take some time and one can visualize the loading progress in the `Log`. A context (see 3.2) is a set of parameters that sets PET to a certain configuration. A valid context is a file ending in `.pec` that necessarily defines a workspace and may define a default user.

A workspace is a directory in the file system that contains all relevant input and output files. Each `user` will have a sub-directory in the workspace tree, so that input and output files are grouped by user. If a default user is defined, PET will load the jobs (`.pej` files) and results (`.per` files) available to that user. One can change the user to any of the available ones in the workspace. As for now, PET does not display the known users, so its up to the user knowing its own username and inputing it correctly when queried. If a valid username is given, PET will update the list of jobs and results according to that user.

2.2 Annotation Page

The annotation page (see Figure 2.2) is the page that you see when a job is loaded. The exact appearance of the annotation page may change due to several aspects of the task (set in the context file).

Figure 2.2 illustrates some important features:

`Labels (1/light blue)` at the top of the window you will find some useful information, such as the status of the current *unit*, the number of revisions and the time spent on that unit. Additional information may be displayed depending on what is set in the context;

`Progress (2/green)` on the top-right hand side the status of the current job is shown in units done out of the total units in the file (in red); the status of the current jobs is also shown in number of units saved to disk; besides, the timestamp of the last time the progress was saved to disk is shown;

`Top box (3/orange)` displays a fixed text (normally the source text) while the active unit is being post-edited; the exact text displayed can be chosen by the user by right-clicking on the box. The top box can also used to preview alternative source text, reference text or machine translations;

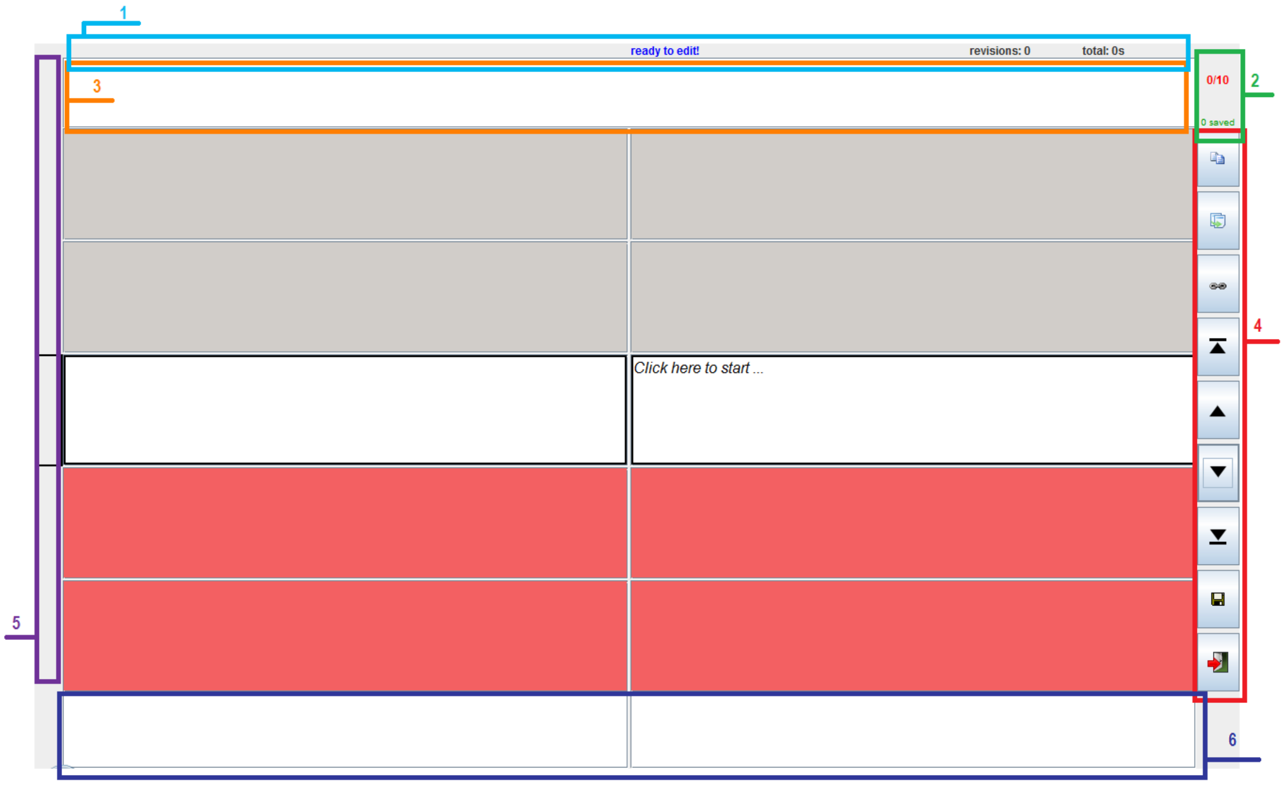


Figure 2.2: PET's annotation page

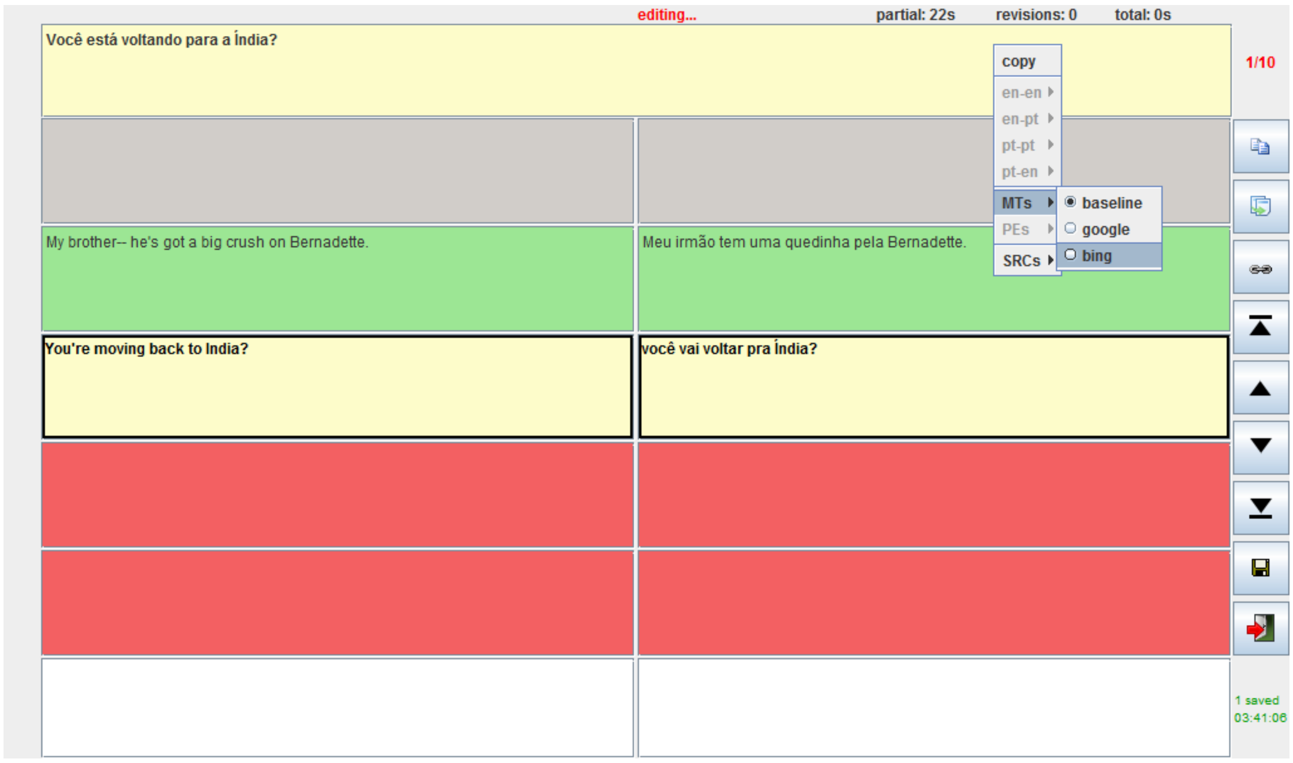
`Tool box (4/red)` placed on the right-hand-side, it displays action buttons explained in Section 2.2.1;

`Id box (5/purple)` placed on the left-hand-side, it is an empty column that may display information such as an identifier or a sequential count for the units on screen;

`Bottom box (6/blue)` displays two panes that render additional information relevant to the active unit; the left pane displays information about the source text and the right pane displays information about the target text (unless PET's standard orientation is changed via context file);

`Units` the center of the page contains a grid of n lines (configured via context file) and two columns (i.e for source and translation, unless it is a monolingual revision job). The line in the center of the screen (highlighted with dark borders) is the active unit, as explained in Section 2.2.2.

The top half of the `units` area shows already edited units, while the bottom half shows units to come (see Figure 2.2). As the user progresses, some units are displayed in green, some are displayed in red. Green stands for "done", that is, units that have been post-edited/translated/revised, while red stands for "to be done". The unit in white is the active unit (always in the central area), and it turns yellow to indicate that the editing has started. Note that the label at the top will change from "ready to edit" to "editing". The labels at the top also displays the time spent on the current revision ("partial"), the number of complete revisions ("revisions") and the total time spent on revisions that are complete ("total").


editing... partial: 22s revisions: 0 total: 0s

Você está voltando para a Índia?

My brother-- he's got a big crush on Bernadette.

Meu irmão tem uma quedinha pela Bernadette.

You're moving back to India?

você vai voltar pra Índia?

1 saved 03:41:06

Figure 2.3: PET's annotation page - progress

2.2.1 Tool box

The context of the tool box (right-hand side box in Figure 2.2 with the tool's action buttons) is also configurable via context file. It generally displays:

`Copy` copy from the top box to the active unit (target side);

`Revert` revert the active unit to its last revision;

`Bind` binds source and target scrolling (useful for long segments);

`Previous undone` searches (backward) for the first undone unit;

`Previous` backs one unit;

`Next` advances one unit;

`Next undone` searches (forward) for the first undone unit;

`Save` saves the current progress to disk;

`Close` closes the job (asks for confirmation and offers a chance to save the progress);

By click anywhere on the tool box, it will be focused. Once focused, scrolling down the mouse wheel will move forward and scrolling up will move backward among the units. When the tool box is focused some shortcuts are available. Placing the pointer over a button for a second will show a tool tip text and information about shortcuts for that button.

Shortcuts are not case-sensitive, some of them are:

`I` enables the *insert* mode, that is, it changes the active unit to the "editing" state;

↓ moves to next unit (if in scrolling state);

↑ moves to previous unit (if in scrolling state);

<END> moves forward until an undone unit is found;

<HOME> moves backward until an undone unit is found;

B binds source and target scrolling;

<F10> saves;

<ALT>+<F4> closes the tool;

2.2.2 Active unit

The active unit is the central line in the units area. It is highlighted by a dark border and is the only unit whose target side is editable. By default, the active unit is in the state "ready to edit", when its target box receives the focus the unit will go to the state "editing" (the box turns yellow), which means that effort indicators will start to be logged. The active unit in the "editing" state will be generally referred to as the *editing box*.

The *editing box* offers some shortcuts, some of them are:

<ALT>+↓ finalizes the current unit and moves to next unit;

<ALT>+N finalizes the current editing and moves forward starting the next unit;

<ALT>+↑ finalizes the current unit and moves to previous unit;

<ALT>+P finalizes the current unit and moves backward starting the previous unit;

<ALT>+B binds source and target scrolling;

<CTRL>+Z undo editing;

<CTRL>+Y redo editing;

<CTRL>+C copy selection;

<CTRL>+X cut selection;

<CTRL>+V paste;

<CTRL>+R replace/insert selection;

<CTRL>+I insert/replace selection;

<CTRL>+D delete selection;

<CTRL>+S shift selection;

In case a unit is in the state "editing" an attempt to close the job will result in a prompt to finalize or discard the modifications done to that active unit.

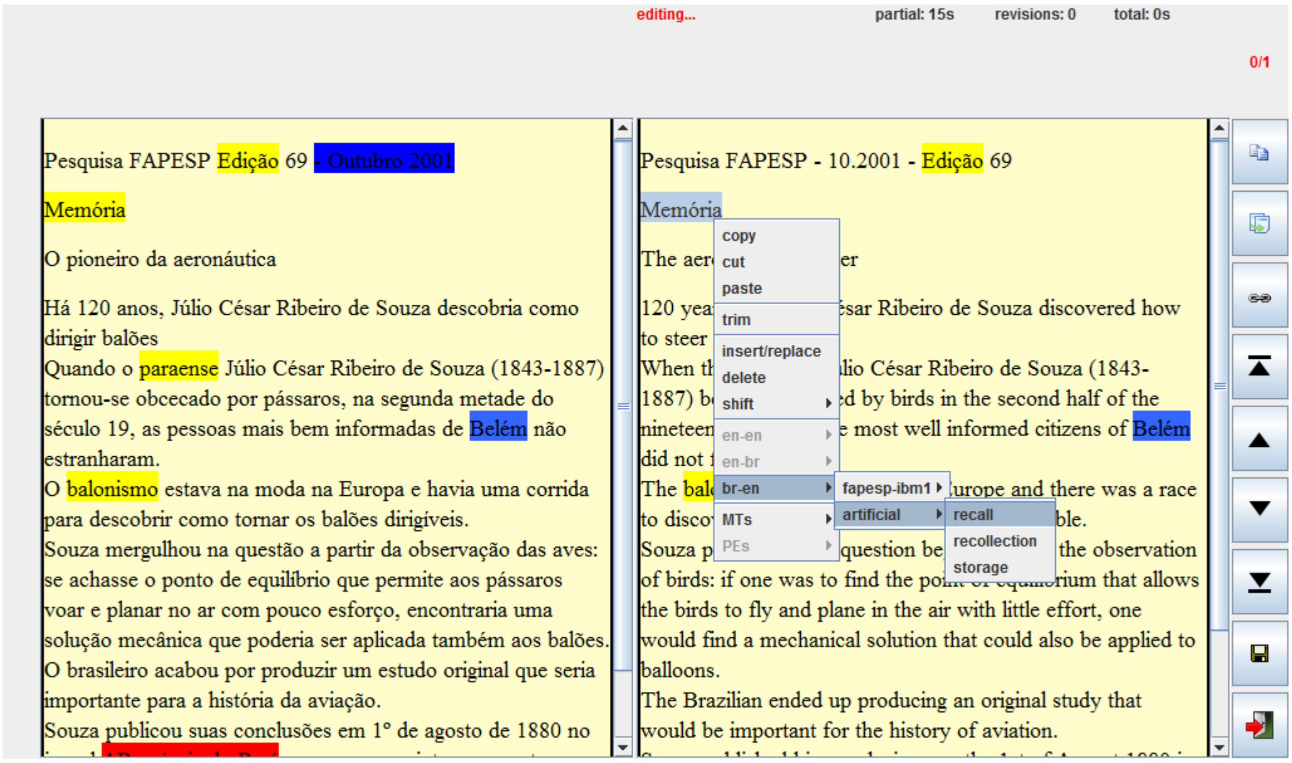
2.2.3 Drop-down menu

By right-clicking the several text boxes one gets a drop-down menu with a few options. All the boxes, except for the active unit, are read-only, therefore they will simply display read-only options, such as "copy" and dictionary lookup. The options available will depend on what is given to the tool as input for the job (dictionaries, alternative translations, etc.).

Available dictionaries are shown using the languages of the current job, for instance: source-source or target-target (monolingual dictionaries), source-target or target-source (bilingual dictionaries). Several dictionaries can be loaded and lookup operations are grouped by dictionary (see Figure 2.2.3). In order to use dictionaries, it is necessary to right-click a portion of the text, as opposed to simply clicking the unit box. In the *editing box*, selecting an option from a dictionary will replace the selected text by the selected option.

Alternative text can be displayed for the active unit in the "editing" state if such alternative text is given in the `.pej` file 3.3. Alternative text includes: i) multiple sources, ii) multiple references and iii) multiple translations. Past revisions of the unit (PEs) can also be shown and used, if they are available for the job. The drop down menu displays these alternative texts.

Moving the mouse over the options will display a preview on the top box, while selecting one of the options will cause the alternative text to replace the content of the current box. One may replace a source by an alternative source, the top box by any of the available alternative text and the target unit by an available translation (MT or PE).



Pesquisa FAPESP Edição 69 - Outubro 2001

Memória

O pioneiro da aeronáutica

Há 120 anos, Júlio César Ribeiro de Souza descobria como dirigir balões
Quando o paraense Júlio César Ribeiro de Souza (1843-1887) tornou-se obcecado por pássaros, na segunda metade do século 19, as pessoas mais bem informadas de Belém não estranharam.
O balonismo estava na moda na Europa e havia uma corrida para descobrir como tornar os balões dirigíveis.
Souza mergulhou na questão a partir da observação das aves: se achasse o ponto de equilíbrio que permite aos pássaros voar e planar no ar com pouco esforço, encontraria uma solução mecânica que poderia ser aplicada também aos balões.
O brasileiro acabou por produzir um estudo original que seria importante para a história da aviação.
Souza publicou suas conclusões em 1° de agosto de 1880 no

Pesquisa FAPESP - 10.2001 - Edição 69

Memória

The aer er

120 yea ésar Ribeiro de Souza discovered how to steer
When th lio César Ribeiro de Souza (1843-1887) b ed by birds in the second half of the nineteen e most well informed citizens of Belém did not
The bal urope and there was a race to disco ble.
Souza p question be the observation of birds: if one was to find the po rium that allows the birds to fly and plane in the air with little effort, one would find a mechanical solution that could also be applied to balloons.
The Brazilian ended up producing an original study that would be important for the history of aviation.

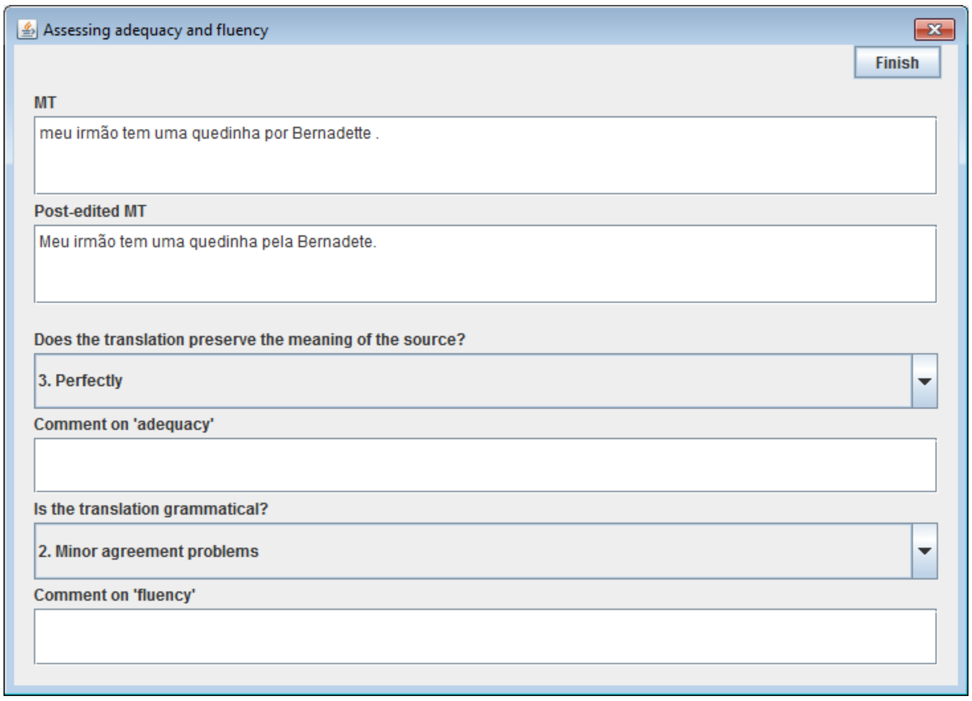
Figure 2.4: PET's annotation page - dictionaries

2.2.4 Drag and drop

Text can be dragged from any text box (source, target, etc.) and dropped into the active unit's target box (if "editing"). Text can also be moved around within the editing box by using drag and drop.

2.3 Assessment Page

An optional assessment page can be displayed after every active unit is completed. The exact type of assessment can be configured via context file. Figures 2.3 and 2.3 show examples, the latter displays a query that accepts multiple answers.



MT

meu irmão tem uma quedinha por Bernadette .

Post-edited MT

Meu irmão tem uma quedinha pela Bernadete.

Does the translation preserve the meaning of the source?

3. Perfectly

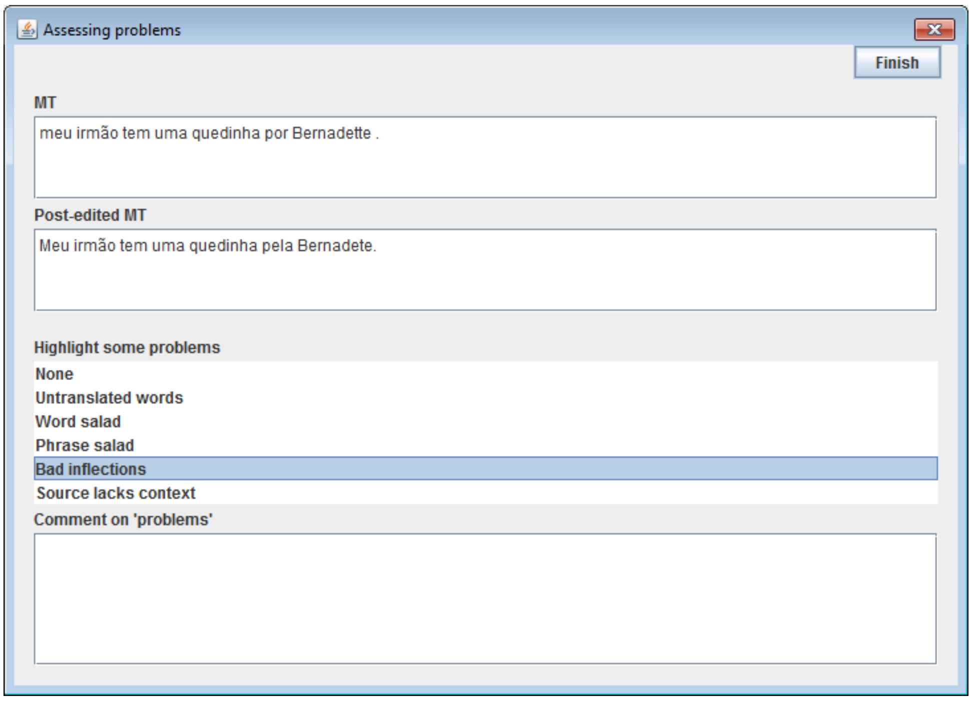
Comment on 'adequacy'

Is the translation grammatical?

2. Minor agreement problems

Comment on 'fluency'

Figure 2.5: PET's assessment page - single selection



MT

meu irmão tem uma quedinha por Bernadette .

Post-edited MT

Meu irmão tem uma quedinha pela Bernadete.

Highlight some problems

None
Untranslated words
Word salad
Phrase salad
Bad inflections
Source lacks context

Comment on 'problems'

Figure 2.6: PET's assessment page - multiple selection

Chapter 3

Files

3.1 Meta

When started, PET will look for the `pec.meta` file in its main directory. If present, this meta configuration file can be optionally used to set up to two global variables (others may be created in future):

> `dir=`*path* sets the path to the directory from where the tool should be run. PET will run considering it is placed in that directory: it will look there for `.pec` configuration files and paths such as the workspace and external files will be relative to it;
>
> `default=`*pec* sets a default *context* to be loaded when the tool starts.

If the file is not present or it is present but all or some of the variables are not defined, PET will assume default values/behaviour:

> `dir` its default value is the tool's base directory;
>
> `default` if not set the tool will not load any context by default, it will expect the user to select a context to be loaded;

3.2 PEC

A context in PET is defined in a `.pec` file. A `.pec` file has a simple declarative syntax to specify different attributes. The main attributes are described in what follows.

3.2.1 Defining a workspace

The workspace is the directory where files are read from and stored, and therefore it has to be set. It is a path relative to the tool's folder unless 'dir' is set in `pec.meta` (see Section 3.1), in which case the workspace will be relative to that 'dir' path.

> `workspace=`*path* sets the workspace to 'path';

To create a user space, create a folder *username* in your workspace and place `.pej` files under this folder.

You may set a default user, that is, a user space to be loaded when the tool starts.

> `user=`*username* sets the default user space to 'username';

3.2.2 Controlling how units get displayed

1. You can set how the active unit is rendered:

   In some experiments you may want to prevent annotators from inspecting the units before they start working on them, as this could affect the results (specially the time-based indicators). You can set when the active unit (before it starts being edited) should be hidden.

   `hideIfNotEditing=`*always*|*undone*|*never* sets when active units are hidden;

   - *always* always hides the active unit;
   - *undone* hides the active unit unless it has been edited before;
   - *never* always shows the active unit;

   (a) You can apply the rule selected by `hideIfNotEditing` to all the units, as opposed to applying it only to the active unit:

   `applyHideIfNotEditingToAll`

   Setting `hideIfNotEditing=`*undone* and `applyHideIfNotEditingToAll` will make hidden all the units on the screen that have not yet been edited;

2. You can set what is shown in the unit box when the unit is hidden:

   `editableMessageUndone=`*message* sets a *message* to be shown for an undone unit, for instance *Click here to start...*;

   `editableMessageDone=`*message* sets a *message* to be shown for a previously done unit, for instance *Click here to redo...*;

3. You can hide panes:

   `hideBottomPane` hides the pane containing external information 3.2.6;

   `hideTopPane` hides the pane containing sources and/or references and/or alternative translations;

   `blindPE` hides the source text, useful for monolingual post-editing;

4. You can swap the source and target panes:

   `displayTS` displays the target pane on the left-hand side and the source pane on the right-hand side;

5. For the active unit you can show reference translations:

   `showReference` displays reference translations at the top pane;

6. You can display the identifiers of the units on the left-hand side of the screen:

   `showSentenceId` displays the `id` attribute of the units;

7. You can block the editing (in read-only tasks or tasks aimed at gathering assessments only):

   `blockEditing` makes the units non-editable;

8. You can display HTML-formatted text

   `renderHTML` interprets text as HTML;

3.2.3 Fitting small screens

1. You can change font and size of text:

   `standardFont=`*font,size* sets the underlying font and size of the tool;

   `editingFont=`*font,size* sets the font and size of the editing unit;

   `editableFont=`*font,size* sets the font and size of any editable unit;

   `idFont=`*font,size* sets the font and size of the units' identifiers;

2. You can change how much context is displayed to the annotator:

   `sentencesByPage=`*number* displays a *number* of units per page (the default is 11);

3.2.4 Collecting effort indicators

PET automatically collects some effort indicators, such as post-editing time, but some of them need to be set.

1. You can collect keystrokes:

   `keystrokes` enables 4 indicators: number of keys pressed (*keystrokes*), how many of those where space-like keys (*white-keystyped*), how many of those where non-white keys (*nonwhite-keystyped*) and how many of those were control keys (*iso-keystyped*).

2. You can flag PEs that match their respective MTs exactly (i.e., translations that did not require any editing):

   `enableUnchanged`

3. You can enable the user to automatically accept MTs:

   `enableAutoAccept` enables a button in the interface and whenever this button is used there will be an indicator (*autoaccept*) in your `.per` file;

   (a) You can skip the collection of explicit assessments on auto-accept:
   `skipAssessmentOnAutoAccept`

   These two options give the user a quicker way of accepting an MT. Besides, the use of these options are logged, so that one can easily recognize the MTs that were accepted as they were presented. If the auto accept button is hit after some edits were performed, those edits will be completely disregarded.

4. You can enable the user to discard a segment, that is, tag it as "impossible" to edit:

   `enableDiscard` enables a button in the interface and whenever this button is used there will be an indicator (*impossible*) in your `.per` file;

   (a) You can skip the collection of explicit assessments on discard (collecting them is the default):
   `skipAssessmentOnDiscard`

5. You may want to log all the changes performed by the annotator:

   `trackChanges` logs timestamped insertions and deletions, also tries to infer substitutions and shifts.

3.2.5 Gathering assessments

PET also allows explicit, more subjective assessments to be collected after every unit is completed.

1. You can set assessment questions for both translation and post-editing jobs in a context file:

   `postEditingAssessment=`*id*|*message*|*max*|*list* sets a post-editing assessment question

   or

   `translationAssessment=`*id*|*message*|*max*|*list* sets a translation assessment question

   - *id* is the assessment identifier, for instance *Accuracy*;
   - *message* is the message to the user, for instance *Highlight the most glaring problems*;
   - *max* is the maximum number of answers or * for no constraint;
   - *list* is a vertical-bar-separated list of options (answers/categories) to be displayed, for instance *OOV terms*|*Word order*|*Semantic ambiguity*;

2. In addition, every assessment question can have a box for comments:

   `disableCommentOnAssessment` hides a comments box in the assessment page;

3. You can enter as many assessment options as you want, they will be displayed one by page, unless you set how many assessments are displayed by page:

   `assessmentsByPage=`*number* specifies the maximum number of assessments by assessment page;

4. You can hide all assessment question if the job does not require them:

   `disableAssessment`

3.2.6 Additional information

1. You can show the annotator additional information about the active unit. Additional information is given via the unit's attributes (multiple attributes are allowed) given to the tool as part of the input file and will be shown as labels at the top of the annotation page.

   `generalInfo=`*attribute[,color]* displays an attribute with a specific color (the color is optional), for instance *duration,blue* and *subtype,green* and *lines*;

   `generalInfoFont=`*font,size* sets the font and size of the general info;

2. You can choose to show external information that refers to n-grams in the active unit using the two boxes at the bottom pane of the annotation page:

   `externalSourceInfo=`*file* displays external info for the source text;

   `externalTargetInfo=`*file* displays external info for the target text;

You can specify as many files as necessary. You can implement readers and printers for these files (to be detailed in the API). PET comes with a default reader that reads XML format (see Section 3.5) and a default printer that prints values associated to n-grams from the source (for `externalSourceInfo`) and from the target (for `externalTargetInfo`) of the active unit. Setting a customized reader and a customized printer as part of these parameters is reserved for future use.

3. You can specify some parameters that control how external information is displayed:

   `externalSourceInfoMinOrder=`*number* sets the minimum order of the n-grams to be matched in the source side;

   `externalSourceInfoMaxOrder=`*number* sets the maximum order of the n-grams to be matched in the source side;

   `externalSourceInfoMinLength=`*number* sets the minimum length (in characters) of the source text to be matched;

   `externalSourceInfoMaxLength=`*number* sets the maximum length (in characters) of the source text to be matched;

   `externalSourceInfoNoLonger` prevents displaying information that is longer than the matched source text;

   The same options apply to the target language information:

   `externalTargetInfoMinOrder=`*number*

   `externalTargetInfoMaxOrder=`*number*

   `externalTargetInfoMinLength=`*number*

   `externalTargetInfoMaxLength=`*number*

   `externalTargetInfoNoLonger`

3.2.7 Dictionaries

PET can display options from one or more monolingual and bilingual dictionaries. In order to make the interface more intuitive you should set the language parameters:

`source=`*acronym* sets an acronym for the source language

`target=`*acronym* sets an acronym for the target language

1. You can add one or monolingual dictionaries (repeat the commands below for each new dictionary):

   `s2s=`*file* adds a source-source dictionary

   `t2t=`*file* adds a target-target dictionary

2. You can add one or more bilingual dictionaries:

   `s2t=`*file* adds a source-target dictionary

   `t2s=`*file* adds a target-source dictionary

3.2.8 Constraints

The tool is aimed to allow for constraints that are specific to a given job to be defined. These still requires some work (and documentation), but one example that is already implemented and can be used will give an example of the type of constraints that could be added. This refers to attempting to guide translators to restrict the length of the target unit according to a suggested threshold (as given in the input file). It was defined for experiments with post-editing/translating subtitles, which need to follow strict length conventions to fit screens and people's reading speed.[1]

1. You can activate length constraints:

   `lengthConstraints=`*ideal,preferable,max* enables the length constraint

   - *ideal* the unit's attribute that specify an "ideal length";
   - *preferable* the unit's attribute that specify a "preferable length";
   - *max* the unit's attribute that specify a "maximum allowed length".

   The target text in the active unit will be rendered differently according to the values of those attributes, and will change dynamically as the translators change the text, adding or removing characteres. For example, translations beyond the maximum length will be shown in red. This behaviour is implemented via PET's API. We will soon provide more documentation for that. For now, you can try it using the default implementation.

3.2.9 Output files

1. PET can save the progress of a job automatically:

   `autoSave` saves the job unit by unit and also generates some additional files that are used to restore data (in case of unexpected failure);

2. You can stamp the output file with the time it was created:

   `outputTimeStamp` timestamps the name of the output file;

3. There is a check box on PET's main page that allows the user decide whether or not output files are timestamped. You may want to disable this option:

   `hideOutputTimeStampCheckBox`

3.3 PEJ

PET's main input file is an XML file. The XML scheme is not yet available, but the examples provided with this distribution should make creating new files fairly intuitive. A file is called a "job" and it is identified by an "id" field (see Listing 3.1), which will also be the stem of its output file.

```xml
<job id="identifier">
 <!-- ... -->
</job>
```

Listing 3.1: PET's input file - a "job"

[1]See our EAMT-2012 paper for more details: "Cross-lingual Sentence Compression for Subtitles": `http://hltshare.fbk.eu/EAMT2012/html/Papers/32.pdf`

A job is made of "units[2]" that can be either for human translation (HTs) or for post-editing (PEs) (see Listing 3.2). A unit is identified by a unique number and it must be either a translation from scratch (ht) or a post-editing (pe). In PE units at least one translation (MT) is expected. Multiple sources (S), references (R) and machine translations (MT) may be given. They should be assigned "producers" to identify the origin of the text. They can be assigned scores (for example, confidence/quality estimation scores).

```xml
<!-- A PE task -->
<unit id="1" type="pe">
 <S producer="manual-transcription">
  <!-- ... -->
 </S>
 <S producer="speech-to-text">
  <!-- ... -->
 </S>
 <R producer="gold">
  <!-- ... -->
 </R>
 <MT producer="system1" score="1">
  <!-- ... -->
 </MT>
 <MT producer="system2" score="5">
  <!-- ... -->
 </MT>
</unit>
<!-- A HT task -->
<unit id="2" type="ht">
 <S producer="manual-transcription">
  <!-- ... -->
 </S>
</unit>
```

Listing 3.2: PET's input file - "units"

Jobs, units and sentences can have additional attributes that may or may not be used depending on advanced features of the tool (Listing 3.3).

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<job source="/home/waziz/experiments/opus/baseline/data/greys.08x06.v1-v1.en-br.en"
     first="1"
     target="/home/waziz/experiments/opus/google/mt/greys.08x06.v1-v1.en-br.br"
     last="50"
     id="greys.08x06.v1-v1.en-br.s2.1"
     meta="/home/waziz/experiments/opus/google/mt/greys.08x06.v1-v1.en-br.original-meta">
  <unit ratio="17.3333"
        max="80"
        lines="1"
        duration="1.5100"
        subtype="1/1"
        sub="1"
        id="1"
        type="pe"
        ideal="26"
        preferable="26">
    <S producer="opus">That's it, Fran. One more push.</S>
    <MT producer="s2" score="0.6">E isso ai, Fran. Mais um empurrao.</MT>
  </unit>
  <unit ratio="16.3636"
        max="80"
        lines="1"
        duration="2.0000"
        subtype="1/1"
        sub="2"
        id="2"
```

[2]In older versions of PET the XML tag used to declare a unit was "task". This mismatch is being corrected both in the code and in the input/output files. We use "unit" throughout this document as it is clearer. For while PET will recognize the tags "task" and "unit" as synonyms.

```
        type="ht"
        ideal="30"
        preferable="24">
    <S producer="opus">As babies, we were easy.</S>
  </unit>
  <!-- ... -->
```

Listing 3.3: PET's input file - "attributes"

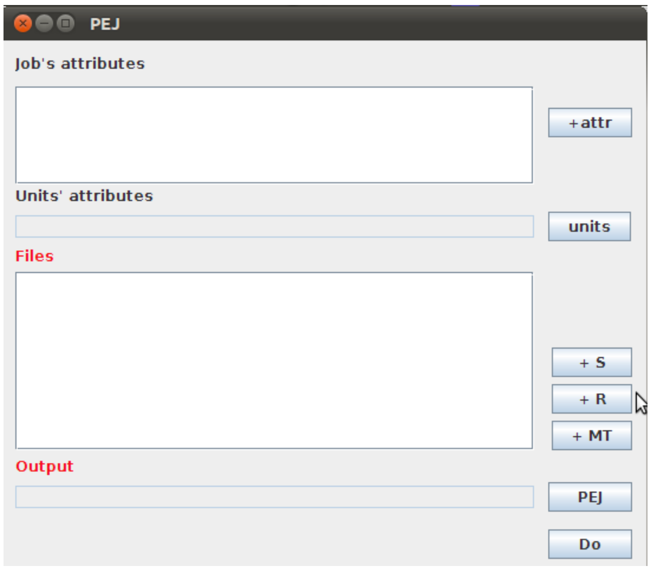


Figure 3.1: PEJ- Auxiliary tool for generating `.pej` files

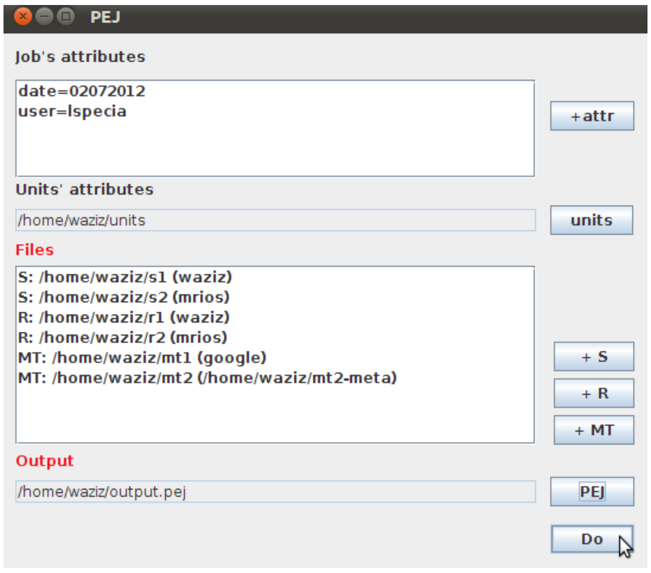
PEJ (Figure 3.3) can format plain text files into `.pej` files for you: run `pej.bat` and a simple interface will be loaded. PEJ expects plain text files containing one segment per line. You may input several files for each type of segment (i.e. source, reference, MTs). Each segment must be assigned a producer, therefore the interface will ask either for i) a file-level producer, or ii) a meta file containing attribute-value pairs (one list per line) from where the producer of each segment will be parsed.

Figure 3.2 exemplifies input files for PEJ. These settings will generate a `.pej` file containing two units each one having one source and two machine translations. The first column is an optional file that may be used to set attributes to each unit. If present this file must specify the attribute `type` of every unit. The second column is an example of source file, 'opus' is the producer of the file, PEJ will query you for that information. The third column is an example of machine-translated file, 'google' would be the producer. The fourth column is another example of machine-translated file and the fifth column is an example of a meta file containing attributes for the segments in `MT2.txt`. If present this file must set the producer of every segment.

Figure 3.3 shows an examples of use of PEJ.

`Units-attr.txt`	`S1.txt (opus)`	`MT1.txt (google)`	`MT2.txt`	`MT2-attr.txt`
type=pe show=bigbang max=25	Ela bateu na minha cara.	She slapped me.	She beat me.	producer=model1
type=pe show=dexter max=20	Eu nao tenho culpa.	I am not to blame.	Not my fault.	producer=model2

Figure 3.2: PEJ: example of input files



date=02072012
user=lspecia

/home/waziz/units

S: /home/waziz/s1 (waziz)
S: /home/waziz/s2 (mrios)
R: /home/waziz/r1 (waziz)
R: /home/waziz/r2 (mrios)
MT: /home/waziz/mt1 (google)
MT: /home/waziz/mt2 (/home/waziz/mt2-meta)

/home/waziz/output.pej

Figure 3.3: PEJ- settings

3.4 PER

PET's output file (.per) is a .pej file extended with more information - see the example in Listing 3.4. In a .per file, the job is tagged with its status and progress, units are also annotated with their status and if a unit has been completed, it will contain a collection of annotations.

```
<job id="bigbang-pe-2" progress="1/10" status="GOING_ON">
 <unit id="1" status="FINISHED" type="pe">
 <S producer="en-opus">My brother— he's got a big crush on Bernadette.</S>
 <R producer="pt-opus">Meu irmao... esta apaixonado por Bernadette.</R>
 <MT producer="baseline">meu irmao tem uma quedinha por Bernadette .</MT>
 <MT producer="google">Meu irmao – ele tem uma grande paixao por Bernadette.</MT>
 <MT producer="bing">Meu irmao – ele tem uma grande paixao por Bernadette.</MT>
 <annotations revisions="2">
  <annotation r="1">
   <PE producer="demo.baseline">Meu irmao tem uma quedinha pela Bernadette.</PE>
   <!-- effort indicators ... -->
  </annotation>
  <annotation r="2">
   <PE producer="demo.baseline">Meu irmao tem uma quedinha pela Bernadete.</PE>
   <!-- effort indicators ... -->
  </annotation>
 </annotations>
 </unit>
 <!-- ... -->
</job>
```

Listing 3.4: PET's output file

Each revision for each unit contains: i) the human translation (HT) or post-edited text (PE) according to the type of task, with the information on the user who performed the task (e.g. "demo" in Listing 3.4) and the translation chosen to be edited (e.g. "baseline") and depending on the options set in the context file: ii) implicit effort indicators, iii) explicit assessments, and iv) comments.

Listing 3.5 shows the output with explict and implicit effort indicators. Note the timestamped primitive edits (i.e. insertion and deletion) sorted by time and wrapped into more meaningful operations (e.g substitution and shift) when possible.

```
<job id="bigbang-pe-2" progress="1/10" status="GOING_ON">
  <unit id="1" status="FINISHED" type="pe">
  <S producer="en-opus">My brother-- he's got a big crush on Bernadette.</S>
  <R producer="pt-opus">Meu irmao... esta apaixonado por Bernadette.</R>
  <MT producer="baseline">meu irmao tem uma crush por Bernadette .</MT>
  <MT producer="google">Meu irmao - ele tem por Bernadette uma grande paixao.</MT>
  <MT producer="bing">Meu irmao - ele tem uma grande paixao por Bernadette.</MT>
  <annotations revisions="1">
    <annotation r="1">
      <!-- output -->
      <PE producer="demo.baseline">Meu irmao tem uma quedinha por Bernadette.</PE>
      <!-- explicit indicators -->
      <assessment id="adequacy">
        <score>2. Preserves the core</score>
      </assessment>
      <assessment id="fluency">
        <score>2. Minor agreement problems</score>
      </assessment>
      <assessment id="problems">
        <score>Phrase salad</score>
      </assessment>
      <!-- implicit indicators -->
      <indicator id="editing" type="time">50s,2</indicator>
      <indicator id="assessing" type="time">41s,2</indicator>
      <indicator id="keystrokes" type="count">38</indicator>
      <indicator id="white-keystyped" type="count">0</indicator>
      <indicator id="nonwhite-keystyped" type="count">8</indicator>
      <indicator id="iso-keystyped" type="count">8</indicator>
      <indicator id="unchanged" type="flag">false</indicator>
      <!-- timestamped edits -->
      <!-- baseline is shown -->
      <indicator id="target" t0="0,70" type="sysselection">baseline</indicator>
      <indicator elapsed=",0" id="assignment" length="40" offset="0" t0=",71" type="change">meu irmao tem uma crush por Bernadette .</indicator>
      <!-- baseline is replaced by google -->
      <indicator id="target" t0="7s,374" type="sysselection">google</indicator>
      <indicator id="substitution" type="wrap">
        <action elapsed=",0" id="deletion" length="40" offset="0" t0="7s,375" type="change">meu irmao tem uma crush por Bernadette .</action>
        <action elapsed=",0" id="insertion" length="53" offset="0" t0="7s,375" type="change">Meu irmao - ele tem por Bernadette uma grande paixao.</action>
      </indicator>
      <indicator elapsed="2s,260" id="deletion" length="6" offset="10" t0="15s,617" type="change">- ele </indicator>
      <indicator id="shift" type="wrap">
        <action elapsed=",0" id="insertion" length="15" offset="47" t0="29s,623" type="change"> por Bernadette</action>
        <action elapsed="1s,183" id="deletion" length="15" offset="14" t0="34s,157" type="change">por Bernadette </action>
      </indicator>
      <indicator id="substitution" type="wrap">
        <action elapsed=",0" id="deletion" length="14" offset="18" t0="44s,398" type="change">grande paixo.</action>
        <action elapsed="1s,563" id="insertion" length="8" offset="18" t0="44s,399" type="change">quedinha</action>
      </indicator>
      <indicator elapsed=",0" id="insertion" length="1" offset="41" t0="48s,848" type="change">.</indicator>
    </annotation>
  </annotations>
</unit>
```

Listing 3.5: PET's output file

3.5 External Information

XML files are used to bring external information to PET. External information can be displayed using the bottom boxes shown in Figure 2.2 (referred to as *passive info*) or using the drop-down menus shown in Figure 2.2.3 (referred to as *active info*). Either way, the principle is fairly simple, a database of external information is a collections of key-value pairs.

For *passive info* keys are n-grams in the active unit and values are HTML content that will be offered as extra information at the bottom pane. PET will list in the bottom pane all the information matching n-grams (following the scpecs in the configuration file) of the active unit text at the beginning of the editing. For *active info* keys are n-grams selected by the user and queried via the drop-down menu (for while PET only renders dictionaries in this way).

PET does not handle fuzzy matches yet, so the keys are strings to be matched exactly, furthermore the matching of keys is case insensitive.

Listing 3.6 shows an example typical of a monolingual dictionary. Values are actively queried by the users when using the drop-down menu.

```xml
<db alias="cambridge">
  <entry>
    <phrase>moving</phrase>
    <paraphrase score="0.36873065">changing place</paraphrase>
  </entry>
  <entry>
    <phrase>moving back</phrase>
    <paraphrase score="0.36873065">going back</paraphrase>
    <paraphrase score="0.36873065">returning</paraphrase>
  </entry>
  <entry>
    <phrase>looks like</phrase>
    <paraphrase score="0.5">appears</paraphrase>
    <paraphrase score="0.2">seems</paraphrase>
  </entry>
</db>
```

Listing 3.6: PET's database file - active

The entries in Listing 3.7 are definitions and links to external resources (e.g. wikipedia) that are retrieved by PET when ones starts the editing of a unit.

```xml
<db alias="cambridge">
  <entry>
    <phrase>Cholera</phrase>
    <paraphrase>infection in the small intestine caused by the bacterium Vibrio
        cholerae</paraphrase>
    <paraphrase><a href="http://en.wikipedia.org/wiki/Cholera">wikipedia</a></paraphrase>
  </entry>
  <entry>
    <phrase>Leprosy</phrase>
    <paraphrase>Hansen's disease</paraphrase>
    <paraphrase><a href="http://en.wikipedia.org/wiki/Leprosy">wikipedia</a></paraphrase>
  </entry>
  <entry>
    <phrase>Hansen's disease</phrase>
    <paraphrase>leprosy</paraphrase>
    <paraphrase><a href="http://en.wikipedia.org/wiki/Leprosy">wikipedia</a></paraphrase>
  </entry>
</db>
```

Listing 3.7: PET's database file - passive

In Listings 3.6 and 3.7 an entry has a key 'phrase' and a list of values each one identified by the tag 'paraphrase'. A score may be also given. For example, if multiple paraphrases are available, a score reflecting their length could be of use in experiments where the length of the translation is important. Frequency or confidence scores can also be useful.

Chapter 4

Examples

4.1 LREC/EAMT-demo version

If you download the LREC/EAMT-2012 workspace you will find a few examples of `.pec` and `.pej` files.

You may place the workspace `lrec` and the meta file `pec.meta` in the tool's root directory or you can place the workspace `lrec` wherever you prefer and set the variable `dir` in your tool's `pec.meta` to the location you chose.

If you use the provided `pec.meta` you will notice that it sets:

`dir=`*lrec*

`default=`*en-pt-typical.pec*

The directory `lrec` contains 5 examples of context files:

1. `en-pt-typical.pec` comes with a typical setup in which future units are hidden;
2. `en-es-wmt.pec` mixes HT and PE tasks and it's focused on gathering post-editing time, no explicit assessment is requested;
3. `fapesp.pec` renders the bitext as HTML and presents the whole text in a single unit;
4. `fapesp-mono.pec` renders only the target text (as HTML) in a single unit;
5. `subs.pec` sets a task with general and external info as well as length constraints;

The directory `lrec` also contains 4 users (sub-directories) which contain examples of input files (`.pej`):

1. `fapesp` check this one for an example with HTML
2. `std`
3. `subs` check this one for examples with length constraints and additional attributes for units
4. `wmt`

Finally there is a directory called `xml` that contains XML databases with external information.

Chapter 5

API

PET's javadocs are not great, but they are getting better. If you have not built the internal documentation you can do it using ant:

```
ant javadoc
```

This should create the directory `dist/javadoc`, where you can find `index.html`.

5.1 PEJ

Look for the javadoc for the class `pet.pej.PEJBuilder`.