hwTools

Release 0.0.1

Gaël Cousin

May 25, 2023

CONTENTS

1 Indices and tables **1**
- 1.1 The hwTools package . . . 1
- 1.2 The parser module . . . 1
- 1.3 The ui_manager module . . . 15
- 1.4 The data_manager module . . . 19
- 1.5 the training_data_script module . . . 25

Python Module Index **27**

Index **29**

CHAPTER ONE

INDICES AND TABLES

- genindex
- modindex
- search

1.1 The hwTools package

This python package provides tools for the manipulation of handwritten text in view of handwritten text recognition, including cursive writing.

The parser module is oriented toward the extraction of the various lines, words and single characters from a given scanned document.

In view of further statistical treatment, an interactive procedure is proposed for the user to control character extraction and matching with a transcribed version of the text.

It is the main procedure of the training_data_script module.

The ui_manager module deals with UI aspects of this interaction and the data_manager module deals with the I/O aspects.

The parser module is conceived with the cultural biases of a french author. We hope it works fine for most of western languages.

We plan to enlarge this package with algorithms that would exploit enough collected data to parse and transcribe new handwritten text automatically, in a single user based perspective: having the same writing for the labeled data and the new text.

1.2 The parser module

Gather facilities for manipulation of handwritten text images: extraction of lines, words, characters, etc.

Functions:

horizontal(img, centers, thickness[, as_list])	Return horizontal lines with prescribed thickness and centers.
bounding_box_slices(image)	Return the slices that define the bounding box of image.
bounding_box(image)	Return the bounding box of a 2d image.
comp_size(labels, val)	Return the size of a component in a labeled image.
hand_made_binary_dilation(img, footprint)	Naive binary dilation, cheaper than skimage's for sparse images.

Classes:

PageExtract	A class for page extracts, to retain context information.
Line	The class for the lines of the treated scanned documents.
Word	A subclass of Line that deals with images of handwrittent words.
Page	The class for the pages of the treated scanned documents.
MatchedGlyph	The class of matched glyphs i.e. pairs (glyph,string).
WordMatcher	A class to match glyphs with strings.
CutInfo	The class for information on tentative cuts, good or wrong ones.
CutParser	The class of objects that parse the user's cut choice.

`hwtools.parser.`**`horizontal`**(*img*, *centers*, *thickness*, *as_list=False*)

Return horizontal lines with prescribed thickness and centers.

Parameters

- **`img`** (*`np.ndarray`*) – The image on which the lines should be overlayed.
- **`centers`** (*`list[int]`*) – the heights at which the lines must appear.
- **`thickness`** (*`int`*) – the prescribed thickness for the lines.
- **`as_list`** (*`bool, optional`*) – If the lines are beeing returned in a single image. Defaults to False.

Returns

A boolean image that contains all the horizontal lines if as_list==False, else a list of such images, each containing a unique line.

Return type

np.ndarray() | list[np.ndarray]

`hwtools.parser.`**`bounding_box_slices`**(*image*)

Return the slices that define the bounding box of image.

Parameters

`image` (*`np.ndarray`*) – a 2d ndarray

Raises

- **`ValueError`** – "The input image is not 2-d"
- **`ValueError`** – "The given array is null, its bounding box is not well defined."

Returns

(s_y,s_x) the pair of slices, s_y for
the vertical axes, s_x for the horizontal one.

Return type

tuple[slice,slice]

`hwtools.parser.`**`bounding_box`**(*image*)

Return the bounding box of a 2d image.

Parameters

`image` (*`np.ndarray`*) – A np.ndarray with image.ndim==2. It represents a grayscale or binary image.

Returns

The smallest rectangular subimage that contains all
non-zero pixels.

Return type
np.ndarray

`hwtools.parser.`**`comp_size`**(*labels*, *val*)

Return the size of a component in a labeled image.

The size is understood as the number of pixels.

Parameters

- **labels** (*np.ndarray*) – an integer valued 2d ndarray
- **val** (*int*) – the value whose pixels we want to count.

Returns

The number of occurences of value among the pixels of
labels.

Return type
int

`hwtools.parser.`**`hand_made_binary_dilation`**(*img*, *footprint*)

Naive binary dilation, cheaper than skimage's for sparse images.

Parameters

- **img** (*np.ndarray*) – The binary image to which the dilation must be applied
- **footprint** (*np.ndarray*) – The footprint for the dilation algorithm.

Returns

a binary image obtained by binary dilation of img wrt
footprint.

Return type
np.ndarray

`class` `hwtools.parser.`**`PageExtract`**

A class for page extracts, to retain context information.

It has strong relations with the class Page.

Methods:

__init__(image, char_height, char_thickness, ...)	Instantiate self.
in_page()	Put the page extract in its original page.
__getitem__(positions)	Mimick behaviour of __getitem__ for np.ndarray.
bounding_box()	Return the bounding box of self.
changed_image(new_image)	Return copy of self with image attribute changed to new_image.

`__init__`(*image*, *char_height*, *char_thickness*, *char_width*, *corner_y*, *corner_x*, *parent_page_shape*)

Instantiate self.

Parameters

- `image` (*np.ndarray*) – The image of the page extract
- `char_height` (*int*) – An estimate of the characters' height.
- `char_thickness` (*int*) – An estimate of the characters' thickness.
- `char_width` (*int*) – An estimate of the characters' width.
- `corner_y` (*int*) – the top-left corner y coordinate within the parent page.
- `corner_x` (*int*) – the top-left corner x coordinate within the parent page.
- `parent_page_shape` (*tuple[int,int]*) – The shape of the parent
- `page.` –

`in_page()`

Put the page extract in its original page.

Returns

a page extract of the whole ambient page dimensions, with the input PageExtract at its position.

Return type

PageExtract

`__getitem__`(*positions*)

Mimick behaviour of __getitem__ for np.ndarray.

Parameters

- `positions` (*tuple[slice | int, slice | int]*) – tuple of
- `slices.` (*indices or*) –

Returns

The page extract corresponding to the sliced
self.image seen within parent_page.

Return type

PageExtract

`bounding_box()`

Return the bounding box of self.

Returns

The smallest rectangular subimage that contains
all non-zero pixels of self.

Return type

np.ndarray

`changed_image`(*new_image*)

Return copy of self with image attribute changed to new_image.

Parameters

`new_image` (*np.ndarray*) – The new image

Returns

Copy of self with image attribute changed to
new_image.

Return type
PageExtract

class `hwtools.parser.`**`Line`**

The class for the lines of the treated scanned documents.

Methods:

`__init__`(page_extract, line_height)	Instantiate Line objects.
`words`([min_width, max_gap, margin, ...])	Return the Words extracted from the line.

Attributes:

`image`	Return self.page_extract.image.
`complete_line`	Return self.page_extract.image != 0.
`basic_line`	Return self.page_extract.image == 1.
`satellites`	Return the 'satellite' components in self.image.
`char_thickness`	Return self.page_extract.char_thickness.
`char_height`	Return self.page_extract.char_height.
`char_width`	Return self.page_extract.char_width.

`__init__`(*page_extract*, *line_height*)

Instantiate Line objects.

Parameters

- **`page_extract`** (`PageExtract`) – The underlying PageExtract
- **`line_height`** (*`int`*) – the line height within page_extract.parent_page.

`property image`

Return self.page_extract.image.

`property complete_line`

Return self.page_extract.image != 0.

`property basic_line`

Return self.page_extract.image == 1.

`property satellites`

Return the 'satellite' components in self.image.

Parameters

`as_list` (*`bool, optional`*) – Whether to return the satellites together as a unique binary image or as a list of distinct connected ones. Defaults to False.

Returns

the satellites as a (list of)
binary images.

Return type
_list[np.ndarray] |np.ndarray

`property char_thickness`

Return self.page_extract.char_thickness.

property char_height

Return self.page_extract.char_height.

property char_width

Return self.page_extract.char_width.

words(*min_width=None*, *max_gap=None*, *margin=None*, *line_thickness=None*)

Return the Words extracted from the line.

If a parameter is received as None, it will be given a default integer value, in function of self. Afterwards, the algorithm is as follows: a thickline at line_height is drawn 'over' line.image, with thickness determined by line_thickness. Every connected component of line that meets this thick line is considered as participating to the basic_line of the line. (The other components are termed satellites.) Then, if the connected components are grouped as the equivalence relation obtained by saturation of the relation 'the horizontal projections are far from at most max_gap'.

Then, the groups with horizontal projection width less than min_width are discarded, considered as noise.

For each of the remaining groups, we consider the smallest interval that contains its horizontal projection, enlarge it on both side by margin. The front pixels that lie over these enlarged intervals define the various Words (one per interval).

Parameters

- **min_width** (*int | None, optional*) – _description_. Defaults to None.
- **max_gap** (*int | None, optional*) – _description_. Defaults to None.
- **margin** (*int | None, optional*) – _description_. Defaults to None.
- **line_thickness** (*int | None, optional*) – _description_. Defaults to None.

Returns
The calculated Words from self.

Return type
list[Words]

class hwtools.parser.**Word**

A subclass of Line that deals with images of handwrittent words.

Methods:

__init__(page_extract, line_height)	Instantiate Line objects.
words([min_width, max_gap, margin, ...])	Return self and print a warning.
word_part(sub_image)	Return a copy of self with self.image replaced by subimage.
cut(shapes)	Return a copy of self, but cut, removing the pixels in shapes.
find_word_components([sat_margin, ...])	Find the components of a Word.
_touches(p, comp[, side])	Decide if p touches comp on the right, the left or any side.
_inbetween_piece(p, q, cut_comp[, ...])	Return the piece of cut_comp which lies between p and q.
_creates_too_small_comp(shape, ...)	Check if cutting at shape creates too small pieces.
cutting_shapes([height_tolerance, ...])	Find where to cut self to separate its letters.
_normalized_image([char_height, char_thickness])	Normalize word.

Attributes:

cutting_shape_thickness	Prescribe the thickness of cutting shapes for word cutting.

`__init__`(*page_extract*, *line_height*)

Instantiate Line objects.

Parameters

- `page_extract` (`PageExtract`) – The underlying PageExtract
- `line_height` (*int*) – the line height within page_extract.parent_page.

`words`(*min_width=None*, *max_gap=None*, *margin=None*, *line_thickness=None*)

Return self and print a warning.

`word_part`(*sub_image*)

Return a copy of self with self.image replaced by subimage.

Parameters

`sub_image` (*np.ndarray*) – the replacement image for self.image.

`cut`(*shapes*)

Return a copy of self, but cut, removing the pixels in shapes.

Parameters

`shapes` (*list[np.ndarray]*) – regions of self.image to turn white. (cutting shapes)

Returns

Cut version of self.

Return type

Word

`find_word_components`(*sat_margin=None*, *line_thickness=None*)

Find the components of a Word.

These components are calculated as the connected components of self.basic_line (as defined for find_words, in terms of line_thickness) plus some satellites of Word, that are considered as belonging to the component if any pixel of them lie over the sat_margin enlarged projection of the component.

Parameters

- `sat_margin` (*int | None, optional*) – The tolerance for satellites attribution. Defaults to None. If None is passed a value is fixed automatically.
- `line_thickness` (*int | None, optional*) – The thickness that determines the basic_line computation. Defaults to None. If None is passed a value is fixed automatically.

Returns

The list of components of self, as subWords.

Return type

list[*Word*]

property cutting_shape_thickness

Prescribe the thickness of cutting shapes for word cutting.

Returns

(3 * self.char_thickness) // 2

Return type
int

static `_touches`(*p*, *comp*, *side=None*)

Decide if p touches comp on the right, the left or any side.

Parameters

- **p** (*tuple[int, int]*) – A point in an image, coords (y,x)
- **comp** (*np.ndarray*) – A set of pixels in this image, given as a boolean array.
- **side** (*str | None, optional*) – The side to consider: None, 'right' or 'left'. Defaults to None. If None is passed, any side is considered valid.

Returns

True if comp is on the side-hand of p,
False otherwise. If side is None, True if and only if p touches comp.

Return type
bool

static `_inbetween_piece`(*p*, *q*, *cut_comp*, *side_sensitive=False*)

Return the piece of cut_comp which lies between p and q.

Parameters

- **p** (*tuple[int, int]*) – The left-hand point, in the form (y,x).
- **q** (*tuple[int, int]*) – The right-hand point, in the form (y0,x0). We suppose x0>x.
- **cut_comp** (*np.ndarray*) – A binary image.
- **side_sensitive** (*bool*) – whether one takes into account the side on which p and q are touched by the sought component.

Returns

if side_sensitive is True,

a connected component of cut_comp that

touches p on the right-hand side of p and q on the left-hand of q; provided it exists. In this case, it is given as a binary image. If such a component does not exist, None is returned.

If side_sensitive is False p or q may touch the sought component on any side. If such a component does not exist, None is returned.

Return type
np.ndarray | None

`_creates_too_small_comp`(*shape*, *too_small_comp_radius*, *min_size*)

Check if cutting at shape creates too small pieces.

Parameters

- **shape** (*np.ndarray*) – a cutting shape
- **window_radius** (*int*) – The radius of the window around shape center in which we perform the size check.
- **min_size** (*int*) – the minimal acceptable size for the cut components.

Returns

True if a too small component would be created,
False otherwise.

Return type
bool

`cutting_shapes`(*height_tolerance=None*, *pixel_tolerance=None*, *window_radius=None*, *thickness=None*, *too_small_comp_radius=None*, *min_size=None*, *recompute=False*)

Find where to cut self to separate its letters.

We explain below the aspects of the algorithm that are tunable through the function parameters. For this discussion, we suppose None of the parameter is passed as None.

A cut/cutting shape is a 1 pixel width rectangular region in self.image. The parameter of the algorithm that fixes the height of the sought cutting shapes is 'thickness'. A cutting shape should enjoy the property that once it is removed from self.basic_component, the latter's number of connected components augments.

We actually check a slightly different property: the shape must disconnect the restriction of self.basic_line to a square window of radius 'window_radius', we also require that no pixel of self.basic_line remains in that window on the vertical line that contains the cut.

For each component of self, as calculated as self.find_components(), the algorithm determines a first set of cut candidates that respects the conditions above (among others). The candidates are ordered from left to right. We add Still componentwise, the leftmost and rightmost pixel of self.basic_line to the list. This list is filtered from left to right, to ensure some kind of event happens between two succesive points: either height events or pixel events.

- **We consider some height event happens between two**
  elements of the list if the part of self.basic_component that links them has at least a height variation of 'height_tolerance'.
- **We consider some pixel event happens between two**
  elements of the list if the part of self.basic_component that links them has at least a variation of 'pixel_tolerance' for the number of pixel in its 1-pixel wide vertical slices.

Any element of the list is eliminated if no such event happens between it and the preceding element (of the updated list).

[. . .]

Finally, We want to remove the cuts that would create too small pieces. This is tested in restriction to a square window of radius 'too_small_comp_radius', the newly created components in that window must be bigger or equal than 'min_size' for the cut to be acceptable.

Parameters

- **`height_tolerance`** (*`int | None, optional`*) – Tolerance for height events. Defaults to None. If None is passed a value is fixed automatically.
- **`pixel_tolerance`** (*`int | None, optional`*) – Tolerance for pixel events. Defaults to None. If None is passed a value is fixed automatically.
- **`window_radius`** (*`int | None, optional`*) – Window size for disconnection test. Defaults to None.If None is passed, a value is fixed automatically.
- **`thickness`** (*`int | None, optional`*) – The thickness for the cutting shapes. Defaults to None. If None is passed a value is fixed automatically.
- **`too_small_comp_radius`** (*`int | None, optional`*) – Window size for new component size test. Defaults to None. If None is passed a value is fixed automatically.
- **`min_size`** (*`int | None, optional`*) – The minimal acceptable size in new component size test. Defaults to None. If None is passed a value is fixed automatically.

- **recompute** (*int | None, optional*) – If the cuts should be recomputed. Defaults to False. To spare computation, it is prefered to store the output after the first call of this method. When testing new parameter values, use recompute=True.

Returns
The list of cutting shapes as binary images of the same shape as self.image and the list of the centers of these shapes in the coordinates of self.image.

Return type
tuple[list[np.ndarray],tuple[int,int]]

_normalized_image(*char_height=None*, *char_thickness=None*)

Normalize word.

Normalize word to a given char_height and char_thickness and arrange for line_height to be half of the new self.image.shape[0].

Parameters

- **char_height** (*int*) – the sought char_height
- **char_thickness** (*int*) – the sought char_thickness

Returns
the normalized Word

Return type
Word

class hwtools.parser.**Page**

The class for the pages of the treated scanned documents.

Methods:

_black_pixels_to_front(img)	Converts image to a boolean array where black pixels become True.
__init__(img, char_height, char_thickness, ...)	Instantiate Page Object.
extract(sub_image)	Give the PageExtract of self associated to sub_image.
extract_line(sub_image, line_height)	Convert subimage and line height to a line.
line_heights([delta, ...])	Checks if the line heights are already computed.
_find_line_heights([delta, ...])	Calculates the line heights in terms of certain options, see line_heights docstring.
lines_number()	The number of detected lines in self.
what_line_heights(subimage, line_heights, ...)	Determine the line heights relevant to a subimage of a Page.
lines([line_thickness])	Separate the Lines of self.

static **_black_pixels_to_front**(*img*)

Converts image to a boolean array where black pixels become True.

Parameters
img (*np.ndarray*) – The image to treat.

Returns :
A np.ndarray of 'bool' type where the black pixels of img are converted to True.

`__init__`(*img*, *char_height*, *char_thickness*, *char_width*)

Instantiate Page Object.

Parameters

- **`img`** (*`np.ndarray`*) – The image of a scanned handwritten text.
- **`char_height`** (*`int`*) – An estimate of the height of the character, measured in pixels. The value should be around the height of an upper case character.
- **`char_thickness`** (*`int`*) – The thickness of the characters, in pixels.
- **`char_width`** (*`int`*) – The approximate width of a lower case character, in pixels.

`extract`(*sub_image*)

Give the PageExtract of self associated to sub_image.

Parameters

`sub_image` (*`ndarray`*) – an ndarray with the same shape as self.

Returns

The bounding box of the front (non zero) pixels of sub_image as a PageExtract.

Return type

PageExtract

`extract_line`(*sub_image*, *line_height*)

Convert subimage and line height to a line.

Parameters

- **`sub_image`** (*`ndarray`*) – Has the same shape as self and has
- **`line.`** (*`front pixels corresponding to a given`*) –
- **`line_height`** (*`int`*) – A y coordinate in self for the
- **`height.`** (*`line's`*) –

Returns

The he resulting line in the bounding box of sub_image.

Return type

Line

`line_heights`(*delta=None*, *smoothing_window_width=None*, *noise_removal=None*)

Checks if the line heights are already computed. If yes, these are returned. Else they are computed. If one of the optional parameter values is specified, the line heights are recomputed with the prescribed parameter values.

The algorithm is based on finding the local maxima of the function height->pixel count of the 1-pixel-high vertical line with this height. However, before analyzing the function, it is smoothened by replacing the function's values by homogeneous means over intervals of width 'smoothing_window_width'.

Some local maxima are considerd irrelevant and discarded if the value fo the smoothened function is not over 'noise_removal' at these local maxima.

Actually, The notion of a local maximum may be discretized in various manners. Ours is the following, defined in terms of the parameter 'delta': a point p is considered a local maximum if the values at p+delta and p-delta are smaller or equal than the value at p.

Parameters

- **`delta`** (*`int | None, optional`*) – Defines the used notion of local maximum. Defaults to None. If None is passed, a value is fixed automatically.

- **smoothing_window_width** (*int | None, optional*) – Defines the sized of the used smoothing window. Defaults to None. If None is passed, a value is fixed automatically.
- **noise_removal** (*int | None, optional*) – Define the pixel number threshold under which the line is ignored. Defaults to None. If None is passed, n value is fixed automatically.

Returns
The relevant line heights.

Return type
list[int]

_find_line_heights(*delta=None*, *smoothing_window_width=None*, *noise_removal=None*)

Calculates the line heights in terms of certain options, see line_heights docstring.

lines_number()

The number of detected lines in self.

static **what_line_heights**(*subimage*, *line_heights*, *line_thickness*)

Determine the line heights relevant to a subimage of a Page.

Returns the elements h of line_heights such that a thickline of thickness line_thickness centered at height h meets some non 0 pixel of subimage.

Parameters

- **subimage** (*np.ndimage*) – The subimage (a grayscale or booelan image)
- **line_heights** (*list[int]*) – The line heights to be tested
- **line_thickness** (*int*) – The thickness parameter of the test.

Returns
List of heights that pass the test.

Return type
list[int]

lines(*line_thickness=None*)

Separate the Lines of self.

The used algorithm separates the lines in the following manners. For each height in self.line_heights(), imagine a thick horizontal line centerd at this height, whith thickness given by the 'line_thickness' parameter.

If no connected component of self.image meets two of these thick lines, one wishes to put together all the components that touch a given thickline to form the basic line attributed to the corresponding height. Afterwards, the still non attributed components are considered as satellites and attached to some basic lines under certain criteria.

When some connected component of self.image meets several of these thick lines, a cutting procedure is used to separate the various batches.

Parameters
line_thickness (*int| None, optional*) – The thickness parameter of the algorithm. Defaults to None. If None is passed, this parameter is set automatically.

Returns

The list of the constructed Lines, one per element
in self.line_height.

Return type
list[Lines]

`class hwtools.parser.`**`MatchedGlyph`**

The class of matched glyphs i.e. pairs (glyph,string).

Here we mainly consider glyphs as Words formed by a single handwritten character. However, if needed we might accept also the generalized concept of the iamge representation of a piece of text that can be extracted from a page as a Word object.

Methods:

`__init__`(glyph, string)	Instantiate MatchedGlyph, matching pairs (glyph,string).

`__init__`(*glyph*, *string*)

Instantiate MatchedGlyph, matching pairs (glyph,string).

Parameters

- **`glyph`** (`Word`) – a Word object formed around the image representation of a string
- **`string`** (*`str`*) – The string represented in the Word.

`class hwtools.parser.`**`WordMatcher`**

A class to match glyphs with strings.

Methods:

`__init__`(word, text[, punctuation])	Instantiate the glyph matcher.
`punctuation_split`()	Separate self.text in word blocks and punctuation blocks.
`match`()	Try to match self.Word with self.text as MatchedGlyphs.

Attributes:

`has_punctuation`	Decide if self.text contains punctuation.

`__init__`(*word*, *text*, *punctuation=None*)

Instantiate the glyph matcher.

Parameters

- **`word`** (`Word`) – a suitably cut Word, that may include punctuation.
- **`text`** (*`str`*) – the text to be matched with the Word punctuation (str, optional): The set of characters to be considered as punctuation. If not passed, it will be deduced from the punctuation variable of the string package.

`punctuation_split`()

Separate self.text in word blocks and punctuation blocks.

Returns

the list of its blocks, by order of appearance.
The list always starts with a non punctuation block, possibly empty, so that odd index blocks correspond exactly to punctuation blocks.

Return type
list[str]

`property has_punctuation`

Decide if self.text contains punctuation.

Returns
True if self.text contains punctuation, False otherwise.

Return type
bool

`match()`

Try to match self.Word with self.text as MatchedGlyphs.

Raises
`ValueError` – "The number of components in self.Word do not match self.text's length."

Returns
The deduced list of MatchedGlyphs.

Return type
list[*MatchedGlyph*]

`class hwtools.parser.CutInfo`

The class for information on tentative cuts, good or wrong ones.

Methods:

__init__(subword, cutting_shape_chars, quality)	Instantiate CutInfo

`__init__`(*subword*, *cutting_shape_chars*, *quality*)

Instantiate CutInfo

Parameters

- **subword** (`Word`) – the surrounding Word of a tentative cut, extracted between the two neighboring tentative cuts. (The end and beginning of the parsed Word are considered neighboring tentative cuts in the definition of the surrounding Word)
- **cutting_shape_chars** (*tuple[int,int,int]*) – The cutting shape described as a tuple (y,x,height) where (y,x) are the coordinates of the center of the cutting vertical segment and height is its height.
- **quality** (*bool*) – True if the user accepted the cut, False otherwise.

`class hwtools.parser.CutParser`

The class of objects that parse the user's cut choice.

Methods:

__init__(word, cutting_shapes, ...)	Instantiate CutParser.
subword_from_cutting_position(i)	Find the subword between the neighboring tentative cuts.
get_cut_infos()	Return the list of CutInfos from our cut choices.

`__init__`(*word*, *cutting_shapes*, *cut_positions*, *cut_choices*)

Instantiate CutParser.

Parameters

- **word** (`Word`) – The Word beeing cut.
- **cutting_shapes** (*list[np.ndarray]*) – The cutting shapes as boolean images of the same shape as Word.image.
- **cut_positions** (*list[tuple[int,int]]*) – The coordinate tuples of the centers of the cutting shapes in the form (y,x).
- **cut_choices** (*list[bool]*) – The information of which cutting_shapes where chosen. True for chosen, False otherwise.

`subword_from_cutting_position`(*i*)

Find the subword between the neighboring tentative cuts.

Parameters

i (*int*) – The index of the considered cut in self.cut_positions.

Returns

The sought Word, extracted from self.word.

Return type

Word

`get_cut_infos`()

Return the list of CutInfos from our cut choices.

Returns

The list of CutInfo objects associated

cutting self.Word according to self.cut_choices among self.cutting_shapes.

Return type

list[*CutInfo*]

1.3 The ui_manager module

Deal with UI aspects.

Classes:

Ui	The abstract class for user interfaces.
LocalUi	A type of user interface, designed for local use of the parser.

class `hwtools.ui_manager.`**`Ui`**

The abstract class for user interfaces.

Methods:

say(text)	Inform the user, saying text.
get_author()	Ask the user to choose an already known author or a new one.
get_language()	Ask the user to provide language details.
get_document()	Get document data in view of its parsing.
get_char_width()	Ask for character width, in pixels.
get_char_height()	Ask for character height, in pixels.
get_char_thickness()	Ask for character thickness, in pixels.
choose_cutting_shapes_and_match(word, ...[, ...])	Select cutting shapes for word and match it to text extract.

abstract say(*text*)

Inform the user, saying text.

Parameters
text (*str*) – The text to forward to the user.

abstract get_author()

Ask the user to choose an already known author or a new one.

Returns
identifier for the author

Return type
str

abstract get_language()

Ask the user to provide language details.

Returns
the locale code for the language and region.

Return type
str

abstract get_document()

Get document data in view of its parsing.

Returns
(input_path,transcription_path,doc_title)

Return type
tuple[str,str,str]

abstract get_char_width()

Ask for character width, in pixels.

Returns
character width

Return type
int

abstract get_char_height()

Ask for character height, in pixels.

Returns
character height

Return type
int

abstract get_char_thickness()

Ask for character thickness, in pixels.

Returns
character thickness

Return type
int

abstract choose_cutting_shapes_and_match(*word*, *shapes*, *text*, *index*, *daltonism=False*)

Select cutting shapes for word and match it to text extract.

Parameters

- **word** (Word) – The Word to be parsed
- **shapes** (*list[np.ndarray]*) – A list of cutting shape proposals to choose from.
- **text** (*str*) – The typeset version of the text from which Word was extracted
- **index** (*int*) – an estimate of the starting point of the typeset version of Word in text.
- **daltonism** (*bool*) – Decides if a color-blind-friendly color scheme is to be used.

Returns

A 2d tuple containing the list of
booleans that retain which cutting shapes where chosen as first entry and the typeset version of word as second entry.

Return type
tuple[list[bool], str]

class hwtools.ui_manager.LocalUi

A type of user interface, designed for local use of the parser.

The graphical user interaction is provided using the npGUI package.

All the methods imposed by the parent abstract class Ui.

Methods:

say(text)	Inform the user, saying text.
get_author()	Ask the user to choose an already known author or a new one.
detect_language()	Detects the OS UI language.
get_language()	Ask the user to provide language details.
get_document()	Get document data in view of its parsing.
get_char_width()	Ask for character width, in pixels.
get_char_height()	Ask for character height, in pixels.
get_char_thickness()	Ask for character thickness, in pixels.
_augmented_region(region, handle_radius)	Augment a region by adding a circular handle and other pixels.
choose_cutting_shapes_and_match(word, ...[, ...])	Select cutting shapes for word and match it to text extract.

`say`(*text*)

Inform the user, saying text.

Parameters
`text` (*str*) – The text to forward to the user.

`get_author`()

Ask the user to choose an already known author or a new one.

Returns
identifier for the author

Return type
str

`detect_language`()

Detects the OS UI language.

Returns
the locale code for the language and region.

Return type
str

`get_language`()

Ask the user to provide language details.

Returns
the locale code for the language and region.

Return type
str

`get_document`()

Get document data in view of its parsing.

Returns
(input_path,transcription_path,doc_title)

Return type
tuple[str,str,str]

`get_char_width`()

Ask for character width, in pixels.

Returns
character width

Return type
int

`get_char_height`()

Ask for character height, in pixels.

Returns
character height

Return type
int

`get_char_thickness()`

Ask for character thickness, in pixels.

Returns
character thickness

Return type
int

`static _augmented_region(`*region*, *handle_radius*`)`

Augment a region by adding a circular handle and other pixels.

Parameters

- **`region`** (*`np.ndarray`*) – Binary image with the region as the front pixels.
- **`handle_radius`** (*`int`*) – the radius of the added handle.

Returns
new binary image, with added front pixels.

Return type
np.ndarray

`choose_cutting_shapes_and_match(`*word*, *shapes*, *text*, *index*, *daltonism=False*`)`

Select cutting shapes for word and match it to text extract.

Parameters

- **`word`** (`Word`) – The Word to be parsed
- **`shapes`** (*`list[np.ndarray]`*) – A list of cutting shape proposals to choose from.
- **`text`** (*`str`*) – The typeset version of the text from which Word was extracted
- **`index`** (*`int`*) – an estimate of the starting point of the typeset version of Word in text.

Returns

A 2d tuple containing the list of
booleans that retain which cutting shapes where chosen as first entry and the typeset version of word as second entry.

Return type
tuple[list[bool], str]

1.4 The data_manager module

Manage data input and output.

Functions:

`_create_directory`(directory)	Create local directory if it does not exist.
`cook_storing_directory`(data_path, author, ...)	Cook up a storing directory path from author and doc_title.

Classes:

DataManager	The class that stores and retrieves parsed data and documents.

`hwtools.data_manager.`**`_create_directory`**(*directory*)

Create local directory if it does not exist.

Parameters
`directory` (*`str`*) – Desired absolute path to the directory.

`hwtools.data_manager.`**`cook_storing_directory`**(*data_path*, *author*, *doc_title*)

Cook up a storing directory path from author and doc_title.

It is a preparatory function, to define DataManagers.

Parameters

- **`data_path`** (*`str`*) – application's user data directory.
- **`author`** (*`str`*) – The parsed document's author.
- **`doc_title`** (*`str`*) – The desired title for the document.

Returns
An absolute path.

Return type
str

`class` `hwtools.data_manager.`**`DataManager`**

The class that stores and retrieves parsed data and documents.

Methods:

`__init__`(storing_directory[, binary_extracts])	Instantiate DataManager and create its directory.
`store_file`(source, rel_target_path)	Store source file to rel_target_path within self.directory.
`store_scan`(source)	Store scanned document in self.directory.
`store_transcription`(source)	Store transcribed document in self.directory.
`image_from_scan`([page_number])	Return given page of the scanned document as a numpy 2darray.
`store_image`(image, name[, check_contrast])	Store image in self.directory as a file.
`retrieve_image`(name)	Retrieve stored image from its name.
`store_image_with_labels`(image, name[, ...])	Store labeled image in self.directory as a file.
`retrieve_image_with_labels`(name)	Retrieve stored labeled image from its name.
`store_image_binarily`(image, name[, ...])	Convert image to a binary one and store it in self.directory.
`retrieve_binary_image`(name)	Retrieve stored binary image from its name.
`store_glyph`(matched_glyph, name[, ...])	Store the matched_glyph on disk.
`retrieve_glyph_metadata`(name)	Retrieve MatchedGlyph metada from name.
`store_cut_info`(cut_info, name)	Store the CutInfo on disk.
`retrieve_cut_info_metadata`(name)	Retrieve CutInfo metada from name.
`set_data_counters`([matched_glyph_counter, ...])	Set the counters for stored MatchedGlyphs and CutInfos.
`store_data`(matched_glyphs, cut_infos)	Store the passed lists of objects?

Attributes:

compacted_text	Return self.text with new lines and blank spaces removed.

Classes:

GlyphMetadata	GlyphMetadata(name, string, char_height, char_width, char_thickness, line_height)
CutInfoMetadata	CutInfoMetadata(name, quality, cutting_shape_chars, char_height, char_width, char_thickness, line_height)

`__init__`(*storing_directory*, *binary_extracts=True*)

Instantiate DataManager and create its directory.

Parameters

- **`storing_directory`** (*`str`*) – The absolute path to the directory where the data manager will store its information.
- **`binary_extracts`** (*`bool`*) – Defaults to True. If True all the
- **`Otherwise`** (*`stored text extracts will be binary images.`*) –

:param : :param the labeling that separates the basic_line and the various: :param satellites is preserved: :param but the images stored on disk are: :param less human readable.:

`store_file`(*source*, *rel_target_path*)

Store source file to rel_target_path within self.directory.

Parameters

- **`source`** (*`str`*) – The absolute path to the source file.
- **`rel_target_path`** (*`str`*) – The target path, relative to self.directory.

Returns

Absolute path to stored file.

Return type

str

`store_scan`(*source*)

Store scanned document in self.directory.

Parameters

`source` (*`str`*) – absolute path to the document.

`store_transcription`(*source*)

Store transcribed document in self.directory.

Parameters

`source` (*`str`*) – absolute path to the transcribed document.

`image_from_scan`(*page_number=0*)

Return given page of the scanned document as a numpy 2darray.

In this first version the scan is supposed to have a unique page, so that only 0 is a meaningful input.

Parameters
`page_number` (*int, optional*) – The page number. Defaults to 0.

`property compacted_text`

Return self.text with new lines and blank spaces removed.

Returns:
str: self.text with ‘

‘s and ‘ ‘s removed.

`store_image`(*image*, *name*, *check_contrast=False*)

Store image in self.directory as a file.

self.binary_extracts determines if the image is stored as a binary image or if its labels are preserved.

Parameters

- `image` (*np.ndarray*) – The image as an ndarray
- `name` (*str*) – The name for the storing file, its extension will be self.image_ext.
- `check_contrast` (*bool, optional*) – Display contrast warnings. Defaults to False.

`retrieve_image`(*name*)

Retrieve stored image from its name.

self.binary_extracts determines if the image is considered as a binary image or a labeled image. Hence, the DataManager that stores the image and the one that retrieves the image must have the same binary_extracts attribute.

Parameters
`name` (*str*) – The name used when the image was stored.

Returns
The image in its original format.

Return type
np.ndarray

`store_image_with_labels`(*image*, *name*, *check_contrast=False*)

Store labeled image in self.directory as a file.

Parameters

- `image` (*np.ndarray*) – The image as an ndarray
- `name` (*str*) – The name for the storing file, its extension will be self.image_ext.
- `check_contrast` (*bool, optional*) – Display contrast warnings. Defaults to False.

`retrieve_image_with_labels`(*name*)

Retrieve stored labeled image from its name.

Parameters
`name` (*str*) – The name used when the image was stored.

Returns
The image in its original format.

Return type
np.ndarray

`store_image_binarily`(*image*, *name*, *check_contrast=False*)

Convert image to a binary one and store it in self.directory.

Parameters

- `image` (*np.ndarray*) – The image as an ndarray
- `name` (*str*) – The name for the storing file, its extension will be self.image_ext.
- `check_contrast` (*bool, optional*) – Display contrast warnings. Defaults to False.

`retrieve_binary_image`(*name*)

Retrieve stored binary image from its name.

Parameters
`name` (*str*) – The name used when the image was stored.

Returns
The image in its original format.

Return type
np.ndarray

`store_glyph`(*matched_glyph*, *name*, *with_satellites=True*)

Store the matched_glyph on disk.

`class GlyphMetadata`

GlyphMetadata(name, string, char_height, char_width, char_thickness, line_height)

Methods:

__new__(_cls, name, string, char_height, ...)	Create new instance of GlyphMetadata(name, string, char_height, char_width, char_thickness, line_height)

Attributes:

char_height	Alias for field number 2
char_thickness	Alias for field number 4
char_width	Alias for field number 3
line_height	Alias for field number 5
name	Alias for field number 0
string	Alias for field number 1

`static __new__`(*_cls*, *name*, *string*, *char_height*, *char_width*, *char_thickness*, *line_height*)

Create new instance of GlyphMetadata(name, string, char_height, char_width, char_thickness, line_height)

`char_height`

Alias for field number 2

`char_thickness`

Alias for field number 4

`char_width`

Alias for field number 3

`line_height`

Alias for field number 5

`name`

Alias for field number 0

`string`

Alias for field number 1

`retrieve_glyph_metadata`(*name*)

Retrieve MatchedGlyph metada from name.

Parameters
`name` (`str`) – name of the stored MatchedGlyph

Returns
The named tuple with the required entries.

Return type
GlyphMetadata

`store_cut_info`(*cut_info*, *name*)

Store the CutInfo on disk.

`class CutInfoMetadata`

CutInfoMetadata(name, quality, cutting_shape_chars, char_height, char_width, char_thickness, line_height)

Methods:

`__new__`(_cls, name, quality, ...)	Create new instance of CutInfoMetadata(name, quality, cutting_shape_chars, char_height, char_width, char_thickness, line_height)

Attributes:

`char_height`	Alias for field number 3
`char_thickness`	Alias for field number 5
`char_width`	Alias for field number 4
`cutting_shape_chars`	Alias for field number 2
`line_height`	Alias for field number 6
`name`	Alias for field number 0
`quality`	Alias for field number 1

`static __new__`(*_cls*, *name*, *quality*, *cutting_shape_chars*, *char_height*, *char_width*, *char_thickness*, *line_height*)

Create new instance of CutInfoMetadata(name, quality, cutting_shape_chars, char_height, char_width, char_thickness, line_height)

`char_height`

Alias for field number 3

char_thickness
Alias for field number 5

char_width
Alias for field number 4

cutting_shape_chars
Alias for field number 2

line_height
Alias for field number 6

name
Alias for field number 0

quality
Alias for field number 1

retrieve_cut_info_metadata(*name*)
Retrieve CutInfo metada from name.

Parameters
name (*str*) – name of the stored CutInfo

Returns
The named tuple with the required entries.

Return type
CutInfoMetadata

set_data_counters(*matched_glyph_counter=0*, *cut_info_counter=0*)
Set the counters for stored MatchedGlyphs and CutInfos.

Parameters

- **matched_glyph_counter** (*int, optional*) – The new value for the MatchedGlyphs counter. Defaults to 0.
- **cut_info_counter** (*int, optional*) – The new value for the CutInfos counter. Defaults to 0.

store_data(*matched_glyphs*, *cut_infos*)
Store the passed lists of objects?

Parameters

- **matched_glyphs** (*list[MatchedGlyph]*) – A list of MatchedGlyphs to store.
- **cut_infos** (*list[CutInfo]*) – A list of CutInfos to store.

1.5 the training_data_script module

Provide the script for interactive parsing.

Functions:

treat_word(my_ui, my_data_manager, text, ...)	Parse interactively a Word.
main()	Offer the terminal command.

`hwtools.training_data_script.`**`treat_word`**(*my_ui*, *my_data_manager*, *text*, *word*, *start_index*, *max_errors_count*, *daltonism*)

Parse interactively a Word.

This function is mainly meant to appear in a loop, when we parse all the Words (parser.Word) of a page. By parsing, we mean:

- choosing relevant cuts in the word to separate its various letters, these cuts are chosen among a certain number of cut proposals
- matching these letters with the transcribed text,
- storing the corresponding parser.MatchedGlyphs on disk,
- storing the information of which cuts were relevant, together with an image extract to retain in which context they appeared (parser.CutInfos).

Parameters

- **`my_ui`** (`Ui`) – The UI being used.
- **`my_data_manager`** – The data manager being used.
- **`text`** (*`str`*) – The compacted transcribed version of the text being parsed. Compacted means spaces and lineskips were removed.
- **`word`** (`Word`) – The Word to be parsed.
- **`start_index`** (*`int`*) – the index of text at which the word transcription is supposed to start.
- **`max_errors_count`** (*`int`*) – The maximum number of parsing errors before we decide to do nothing for (skip) this word.
- **`daltonism`** (*`bool`*) – Defaults to False. If False the default color scheme is used, the ability to distinguish usual green and red is necessary. If set to True a color-blind-friendly color scheme is used.

Returns
The start index for the next Word in the loop.

Return type
int

`hwtools.training_data_script.`**`main`**()

Offer the terminal command.

PYTHON MODULE INDEX

h

hwtools, 1
hwtools.data_manager, 19
hwtools.parser, 1
hwtools.training_data_script, 25
hwtools.ui_manager, 15

INDEX

Symbols

`__getitem__()` (*hwtools.parser.PageExtract method*), 4
`__init__()` (*hwtools.data_manager.DataManager method*), 21
`__init__()` (*hwtools.parser.CutInfo method*), 14
`__init__()` (*hwtools.parser.CutParser method*), 14
`__init__()` (*hwtools.parser.Line method*), 5
`__init__()` (*hwtools.parser.MatchedGlyph method*), 13
`__init__()` (*hwtools.parser.Page method*), 10
`__init__()` (*hwtools.parser.PageExtract method*), 3
`__init__()` (*hwtools.parser.Word method*), 7
`__init__()` (*hwtools.parser.WordMatcher method*), 13
`__new__()` (*hwtools.data_manager.DataManager.CutInfoMetadata static method*), 24
`__new__()` (*hwtools.data_manager.DataManager.GlyphMetadata static method*), 23
`_augmented_region()` (*hwtools.ui_manager.LocalUi static method*), 19
`_black_pixels_to_front()` (*hwtools.parser.Page static method*), 10
`_create_directory()` (*in module hwtools.data_manager*), 20
`_creates_too_small_comp()` (*hwtools.parser.Word method*), 8
`_find_line_heights()` (*hwtools.parser.Page method*), 12
`_inbetween_piece()` (*hwtools.parser.Word static method*), 8
`_normalized_image()` (*hwtools.parser.Word method*), 10
`_touches()` (*hwtools.parser.Word static method*), 8

B

`basic_line` (*hwtools.parser.Line property*), 5
`bounding_box()` (*hwtools.parser.PageExtract method*), 4
`bounding_box()` (*in module hwtools.parser*), 2
`bounding_box_slices()` (*in module hwtools.parser*), 2

C

`changed_image()` (*hwtools.parser.PageExtract method*), 4
`char_height` (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 24
`char_height` (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 23
`char_height` (*hwtools.parser.Line property*), 5
`char_thickness` (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 24
`char_thickness` (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 23
`char_thickness` (*hwtools.parser.Line property*), 5
`char_width` (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 25
`char_width` (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 23
`char_width` (*hwtools.parser.Line property*), 6
`choose_cutting_shapes_and_match()` (*hwtools.ui_manager.LocalUi method*), 19
`choose_cutting_shapes_and_match()` (*hwtools.ui_manager.Ui method*), 17
`comp_size()` (*in module hwtools.parser*), 3
`compacted_text` (*hwtools.data_manager.DataManager property*), 22
`complete_line` (*hwtools.parser.Line property*), 5
`cook_storing_directory()` (*in module hwtools.data_manager*), 20
`cut()` (*hwtools.parser.Word method*), 7
`CutInfo` (*class in hwtools.parser*), 14
`CutParser` (*class in hwtools.parser*), 14
`cutting_shape_chars` (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 25
`cutting_shape_thickness` (*hwtools.parser.Word property*), 7
`cutting_shapes()` (*hwtools.parser.Word method*), 9

D

`DataManager` (*class in hwtools.data_manager*), 20
`DataManager.CutInfoMetadata` (*class in hwtools.data_manager*), 24

DataManager.GlyphMetadata (*class in hwtools.data_manager*), 23
detect_language() (*hwtools.ui_manager.LocalUi method*), 18

E

extract() (*hwtools.parser.Page method*), 11
extract_line() (*hwtools.parser.Page method*), 11

F

find_word_components() (*hwtools.parser.Word method*), 7

G

get_author() (*hwtools.ui_manager.LocalUi method*), 18
get_author() (*hwtools.ui_manager.Ui method*), 16
get_char_height() (*hwtools.ui_manager.LocalUi method*), 18
get_char_height() (*hwtools.ui_manager.Ui method*), 16
get_char_thickness() (*hwtools.ui_manager.LocalUi method*), 18
get_char_thickness() (*hwtools.ui_manager.Ui method*), 17
get_char_width() (*hwtools.ui_manager.LocalUi method*), 18
get_char_width() (*hwtools.ui_manager.Ui method*), 16
get_cut_infos() (*hwtools.parser.CutParser method*), 15
get_document() (*hwtools.ui_manager.LocalUi method*), 18
get_document() (*hwtools.ui_manager.Ui method*), 16
get_language() (*hwtools.ui_manager.LocalUi method*), 18
get_language() (*hwtools.ui_manager.Ui method*), 16

H

hand_made_binary_dilation() (*in module hwtools.parser*), 3
has_punctuation (*hwtools.parser.WordMatcher property*), 14
horizontal() (*in module hwtools.parser*), 2
hwtools
    module, 1
hwtools.data_manager
    module, 19
hwtools.parser
    module, 1
hwtools.training_data_script
    module, 25
hwtools.ui_manager
    module, 15

I

image (*hwtools.parser.Line property*), 5
image_from_scan() (*hwtools.data_manager.DataManager method*), 21
in_page() (*hwtools.parser.PageExtract method*), 4

L

Line (*class in hwtools.parser*), 5
line_height (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 25
line_height (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 24
line_heights() (*hwtools.parser.Page method*), 11
lines() (*hwtools.parser.Page method*), 12
lines_number() (*hwtools.parser.Page method*), 12
LocalUi (*class in hwtools.ui_manager*), 17

M

main() (*in module hwtools.training_data_script*), 26
match() (*hwtools.parser.WordMatcher method*), 14
MatchedGlyph (*class in hwtools.parser*), 13
module
    hwtools, 1
    hwtools.data_manager, 19
    hwtools.parser, 1
    hwtools.training_data_script, 25
    hwtools.ui_manager, 15

N

name (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 25
name (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 24

P

Page (*class in hwtools.parser*), 10
PageExtract (*class in hwtools.parser*), 3
punctuation_split() (*hwtools.parser.WordMatcher method*), 13

Q

quality (*hwtools.data_manager.DataManager.CutInfoMetadata attribute*), 25

R

retrieve_binary_image() (*hwtools.data_manager.DataManager method*), 23

retrieve_cut_info_metadata() (*hwtools.data_manager.DataManager method*), 25
retrieve_glyph_metadata() (*hwtools.data_manager.DataManager method*), 24
retrieve_image() (*hwtools.data_manager.DataManager method*), 22
retrieve_image_with_labels() (*hwtools.data_manager.DataManager method*), 22

S

satellites (*hwtools.parser.Line property*), 5
say() (*hwtools.ui_manager.LocalUi method*), 17
say() (*hwtools.ui_manager.Ui method*), 16
set_data_counters() (*hwtools.data_manager.DataManager method*), 25
store_cut_info() (*hwtools.data_manager.DataManager method*), 24
store_data() (*hwtools.data_manager.DataManager method*), 25
store_file() (*hwtools.data_manager.DataManager method*), 21
store_glyph() (*hwtools.data_manager.DataManager method*), 23
store_image() (*hwtools.data_manager.DataManager method*), 22
store_image_binarily() (*hwtools.data_manager.DataManager method*), 23
store_image_with_labels() (*hwtools.data_manager.DataManager method*), 22
store_scan() (*hwtools.data_manager.DataManager method*), 21
store_transcription() (*hwtools.data_manager.DataManager method*), 21
string (*hwtools.data_manager.DataManager.GlyphMetadata attribute*), 24
subword_from_cutting_position() (*hwtools.parser.CutParser method*), 15

T

treat_word() (*in module hwtools.training_data_script*), 25

U

Ui (*class in hwtools.ui_manager*), 15

W

what_line_heights() (*hwtools.parser.Page static method*), 12
Word (*class in hwtools.parser*), 6
word_part() (*hwtools.parser.Word method*), 7
WordMatcher (*class in hwtools.parser*), 13
words() (*hwtools.parser.Line method*), 6
words() (*hwtools.parser.Word method*), 7