Harkkapoliisi_0.3.0

API Documentation

June 21, 2011

Contents

1 Package src

1.1 Modules

- **controller**: The module forwards files to the proper inspector.
  (Section 2, p. 4)
- **gui** *(Section 3, p. 6)*
  - **email_sender**: The module contains the function for sending email.
    (Section 4, p. 7)
  - **gui**: The module provides the Mod_Python graphical user interface for the software.
    (Section 5, p. 8)
  - **gui_DOM**: The module provides the Mod_Python graphical user interface for the software.
    (Section 6, p. 9)
- **inspectors** *(Section 7, p. 10)*
  - **common_methods**: The module contains common methods to access DOM tree.
    (Section 8, p. 11)
  - **conversions**: The module is for conversions.
    (Section 9, p. 13)
  - **docx_inspector**: The module provides the methods for inspecting docx files.
    (Section 10, p. 15)
  - **mso_meta_inspector**: The module is for getting Microsoft Office file formats' docx, pptx, and xlsx meta informations.
    (Section 11, p. 25)
  - **odp_inspector**: The module provides the scratch for inspecting opd files.
    (Section 12, p. 26)
  - **odt_inspector**: The module provides the methods for inspecting odt files.
    (Section 13, p. 27)
  - **ooo_meta_inspector**: The module provides the methods for getting OpenOffice.org file formats' odt, odp, and ods meta informations.
    (Section 14, p. 34)
- **requirements**: The module contains the classes for reading and storing the data of a requirements XML file.
  (Section 15, p. 36)
- **word_processing**: The module makes the comparisons between the office document properties and requirements specified for each user.
  (Section 16, p. 38)

1.2 Variables

Name	Description
__package__	**Value:** `None`

2 Module src.controller

The module forwards files to the proper inspector.

After the inspection it collects the feedback to be shown on the user interface.

See Also: Function beginInspection().

Author: Vili Auvinen, Olli Kauppinen, Juho Tammela

2.1 Functions

checkZipFiles(*zipFile*, *fileName*, *feedback*, *requirements*)

Checks if the file is a zip file and forwards the document files to the proper inspector function.

Recursively checks if files inside a zip file are zip files. If the file is a zip file and it's in Microsoft Office or OpenOffice.org file format, gives the file forward.

stringToParagraph(*s*)

Converts a string to html paragraph.

resultsToDom(*fileName*, *results*)

Converts with DOM the results of an inspection to the HTML format.

Parameters

`fileName:` The name of the file where the results are saved.

`results:` The results of the inspection.

Return Value

Results as HTML format.

getFileExtension(*fileName*)

Gets the file extension from the file name string.

Splits the given string at the character . and returns the string on the right side of it.

Parameters

`fileName:` the ile name as a string.

Return Value

The file extension or the original string if the character . is not found.

printResults(*resultsDict*)

beginInspection(*sentfile*, *filename*, *feedback*, *email*)

Begins the inspection of a file. Gets the requirements.xml using the email-address as a folder name.

Parameters

`sentFile:` the file to be inspected, can be Microsoft Office docx file, OpenOffice.org odt file or a zip file containing them.

`filename:` the name of the file.

`feedback:` empty list where the feedback of the inspection is stored.

`email:` the email address of the user.

Return Value

An error string as a HTML paragraph if the requirements file is not found.

2.2 Variables

Name	Description
__package__	**Value:** `'src'`

3 Package src.gui

3.1 Modules

- **email_sender**: The module contains the function for sending email.
  (Section 4, p. 7)
- **gui**: The module provides the Mod_Python graphical user interface for the software.
  (Section 5, p. 8)
- **gui_DOM**: The module provides the Mod_Python graphical user interface for the software.
  (Section 6, p. 9)

3.2 Variables

Name	Description
__package__	**Value:** `None`

4 Module src.gui.email_sender

The module contains the function for sending email.

Author: Juho Tammela

4.1 Functions

index(*req*)

The main function is called by a JavaScript file asynchronously. It sends email to the user.

Email address and the message body is stored in Request object.

Parameters

`req`: Mod_Python request object holding the information given in the Ajax call.

Return Value

The string to indicate success or failure in sending the email. The string can be parsed as XML because of Internet Explorer.

Internet Explorer wants XML as response, else it thinks something went wrong when it tries to parse xml out of the response and doesn't succeed!

4.2 Variables

Name	Description
__package__	**Value:** `'src.gui'`

5 Module src.gui.gui

The module provides the Mod_Python graphical user interface for the software.

Author: Juho Tammela

5.1 Functions

firstPage(*name*, *nameError*, *email*, *emailError*, *fileDiskError*, *fileUrl*, *fileUrlError*)

Gets the form page of the interface as an HTML string.

feedBackPage(*feedback*, *name*, *email*)

Gets the feedback page of the interface.

index(*req*)

5.2 Variables

Name	Description
__package__	**Value:** `'src.gui'`

6 Module src.gui.gui_DOM

The module provides the Mod_Python graphical user interface for the software.

Author: Juho Tammela

6.1 Functions

getHtmlDocument(*scriptName*)

firstPage(*name*, *nameError*, *email*, *emailError*, *fileDiskError*, *fileUrl*, *fileUrlError*)

feedBackPage(*feedback*, *name*, *email*)

index(*req*)

6.2 Variables

Name	Description
__package__	**Value:** `'src.gui'`

7 Package src.inspectors

7.1 Modules

- **common_methods**: The module contains common methods to access DOM tree.
  (Section 8, p. 11)
- **conversions**: The module is for conversions.
  (Section 9, p. 13)
- **docx_inspector**: The module provides the methods for inspecting docx files.
  (Section 10, p. 15)
- **mso_meta_inspector**: The module is for getting Microsoft Office file formats' docx, pptx, and xlsx meta informations.
  (Section 11, p. 25)
- **odp_inspector**: The module provides the scratch for inspecting opd files.
  (Section 12, p. 26)
- **odt_inspector**: The module provides the methods for inspecting odt files.
  (Section 13, p. 27)
- **ooo_meta_inspector**: The module provides the methods for getting OpenOffice.org file formats' odt, odp, and ods meta informations.
  (Section 14, p. 34)

7.2 Variables

Name	Description
__package__	**Value:** `None`

8 Module src.inspectors.common_methods

The module contains common methods to access DOM tree.

Author: Vili Auvinen, Olli Kauppinen, Juho Tammela

8.1 Functions

getTextContent(*element*)

Returns recursively all the text content from the given element. The function can be used both in, odt_inspector and docx_inspector modules.

Parameters

`element:` the element to be checked.

Return Value

All text content from its element and descendants.

getAttributeContent(*element*)

Return recursively all the attribute's content from the given element and it's children. The function can be used both in, odt_inspector and docx_inspector modules.

Parameters

`element:` the element to be checked.

Return Value

All attribute's content from element and descendants.

checkStringFromContent(*element*, *wantedString*)

Check if given element contains given string. Encode to utf-8.

Return Value

True if the element contains given string, False if not.

checkIfEmailAddress(*element*)

Checks if the element contains an e-mail address. @-char is identifier.

Return Value

An e-mail address, None if the element contains no e-mail.

getDescendants(*element*, *elementList*)

Gets elements all descendants.

Parameters

`element:` is the node whose descendants are returned.

`elementList:` is an empty list when function have called from outside. There is elements in list when function calls itself.

Return Value

The list of all descendants of the specified element.

checkParentNode(*element*, *parentTag*)

Checks if the element have the given parentTag as parent

Parameters

`element:` is the element to be checked

`parentTag:` is the tag which the returned parents have.

Return Value

True if the given element have parentTag as parent

8.2 Variables

Name	Description
__package__	**Value:** `None`

9 Module src.inspectors.conversions

The module is for conversions. There is functions for convert the measure to right dimension.

Author: Vili Auvinen, Olli Kauppinen, Juho Tammela

9.1 Functions

convertCmOrInToPt(*value*)

Converts a value in centimeters or inches to points.

Parameters

`value`: the string contains a value centimeter or inches with 'cm' or 'in' after the value.

Return Value

converted value in points (pt).

(type=string)

Note: XML-example: <style:paragraph-properties fo:margin-top="0.39cm" </style:paragraph-properties>

convertPtToCm(*pt*)

convertPtToTwip(*pt*)

convertTwipToPt(*twips*)

convertCmToTwip(*cm*)

convertTwipToCm(*twips*)

convertTwipToInch(*twips*)

convertInchToTwip(*inches*)

convertPercentToDecimal(*percent*)

Converts a percent to a decimal format.

Return Value

The value in decimal format.

convertCmOrInToString(*value*, *decimals*=1)

Converts the value in centimeters or inches to the string format.

Parameters

`value:` the string contains centimeter or inches with 'cm' or 'in' after value.

`decimals:` the number of the decimals (default 1).

Return Value

The value in centimeters.

(type=string)

convertCmOrInDictToString(*convertedDict*, *decimals*=1)

Converts dictionary's value in centimeters or inches to the string format.

Parameters

`convertedDict:` The dictionary contains centimeter or inches with 'cm' or 'in' as values.

`decimals:` the number of the decimals (default 1).

Return Value

Converted dictionary in centimeters.

9.2 Variables

Name	Description
__package__	**Value:** `None`

10 Module src.inspectors.docx_inspector

The module provides the methods for inspecting docx files.

Author: Vili Auvinen, Juho Tammela

10.1 Functions

getStyle(*document*, *requirementStyleName*)

Gets all definitions of a style from document dictionary.

Converts twips to centimeters.

Return Value

A dict with all the style definitions of the one style with the translated keys to match return value odt_inspector's getStyle(). False, if the style was not found.

checkHeadersAndFooters(*document*)

Checks that the headers and footers of a document are made correctly.

Assumes that the document has three sections:

1. cover section
2. table of contents section or toc section
3. actual content section or text section

Return Value

Findings in the errorIds-dict as key-boolean pairs as described above.

See Also: checkSections method must pass in order to run this method

Notes:

- Places findings in the errorIds-dict as key-boolean pairs:
  'frontPage': was there headers or footers in the cover section.
  'tocPageNumbering': is there a page numbering in the toc section.
  'differentPageNumbering': is the page numbering different in the cover and text sections.
  'nameInToc': is the last modifiers name in toc section header or footer.
  'nameInText': is the last modifiers name in text section header or footer.
  'pageNumbering': is there a page numbering in the text section.
  'tocNumStart': does the toc section page numbering start at 1.
  'textNumStart': does the text section page numbering start at 1.
  'titlePg': is the Microsoft Office setting "Different first page" on.
- XML example:
  <w:pgNumType w:fmt="lowerRoman" w:start="1"/>)
  <w:pgNumType w:start="1"/>

getParagraphElementsBySections(*docXml*, *sectionName*)

Get paragraph elements of the wanted section. The page breaking section break elements changes section, continuous section brake elements don't change section.

The first list of the section elements is the cover section. The second list of the section elements is the table of contents-section. The third list of the section elements is the text section. The document has to have at least 3 sections.

Parameters

`docXml:` The document.xml file as a DOM tree.

`sectionName:` The wanted section can be 'cover', 'toc' or 'text'.

Return Value

The list of the section elements.

getSectionElementsBySections(*docXml*, *index*=`None`)

Gets all the w:sectPr elements of a document or optionally the w:sectPr elements of a specific section.

w:sectPr elements are stored in a two dimensional list. Continuous section breaks are appended to current outer list index. The page breaking section raises the outer list index.

Parameters

`index:` The index of the outer pageSections list that is get. None by default.

Return Value

The two dimensional list of all w:sectPr elements if index is None. Otherwise returns the list at the given index.

checkSections(*document*, *errorList*)

Goes through the section elements in the document checking that the sections are done properly.

There must be at least three sections in the document. The cover page and the table of the contents cannot be in the same section. Also checks that the Microsoft Office Word setting "Different first page" is off.

Return Value

True if everything went well, False if something went terribly wrong or error list if an error was found and the checking could be completed.

getPageMarginals(*document*)

Gets the document page marginals sizes.

Return Value

False if the marginals are not coherent, otherwise a dictionary containing the marginal sizes.

getPageSize(*document*)

Gets the document page sizes.

Return Value

False if the page sizes are not coherent, otherwise a dictionary containing the page width and length.

checkTocContent(*document*)

Checks if all of the headings created in the document are listed in the table of contents.

Return Value

True if toc matches the headings content, False otherwise.

checkTOC(*document*)

Check if table of contents is done correctly. It has to have a page break before (and after) it.

Return Value

True if toc is made correctly, False otherwise.

See Also: checkTocContent – calls the method if there's a table of contents to be found.

Note: XML example:

<w:p w:rsidR="004A16ED" w:rsidRDefault="004A16ED" w:rsidP="006158B0">

<w:pPr>

<w:pStyle w:val="Otsikko"/>

</w:pPr>

<w:r w:rsidRPr="006158B0">

<w:lastRenderedPageBreak/>

<w:t>SISALLYSLUETTELO</w:t>

</w:r>

</w:p>

<w:p w:rsidR="002274FC" w:rsidRDefault="00FA6E61">

<w:pPr>

<w:pStyle w:val="Sisluet1"/>

checkCoverPage(*document*)

Checks if the front page is done correctly

Return Value

coverPageText dictionary containing True or False values.

getRelsTargetByRId(*rId*, *rels*)

Returns the value of Target attribute of a Relationship element with the given id in a given rels file. The value of Target attribute can be for example a relative path to local XML files or images. It can also be a hyperlink.

Parameters

`rId:` Id attribute value of a Relationship element.

`rels:` rels file as a DOM tree.

Return Value

The value of Target attribute if found.

getParentParagraph(*element*, *tag*=`'w:p'`)

Returns the parent <w:p>-element of a given element if there is one.

Parameters

`element:` The element whose parent <w:p> element is searched for.

`tag:` The parent tagname, defaults to 'w:p'.

Return Value

The parent element, or None if no parent is found.

checkImages(*document*)

Check if there is an image in the document.

Return Value

True if even one image is found, False otherwise.

getImagePaths(*document*)

Gets the image paths or the file names of the images used in the document.

Return Value

The image targets as strings in a list.

checkImageCaptions(*document*)

Checks if the next paragraph after a picture paragraph uses the caption style.

Also checks that the caption contains an automatic field. Goes through all picture paragraphs.

Return Value

True if all images have captions, False otherwise.

checkStyleUsage(*document*, *errorIdsAndPositions*)

Checks that text paragraphs are using styles and that no manual style definitions are made.

Goes through all paragraph-elements in a document looking for <w:pStyle>-elements. Gets the style definitions to see if there are manual changes.

Parameters

`errorIdsAndPositions:` A dict for error strings. Should contain keys 'manualChanges' and 'styleNotUsed'.

Return Value

True if nothing was found, False if even one error was found.

Note: Exception:

Automatically generated table on contents can contain "manual" style definitions. The <w:sectPr> elements within paragraph elements are skipped also.

checkEndnotesAndFootnotes(*document*)

Checks if there is an endnote or a footnote in the document.

Looks for w:endnoteReference and w:footnoteReference elements.

Return Value

True if an endnote or a footnote is found, False otherwise.

checkCrossRefrenceToImageCaption(*document*)

Goes through images' captions looking for a reference. Then checks if the caption is referenced somewhere.

Return Value

True if a cross reference is found, False otherwise.

checkHeadingNumbering(*document*, *errorIdsAndPositions*)

Checks the headings in the document.

Goes through the heading styles used in the document checking that they use a multilevel numbering, the numbering is done correctly using styles and that the numbering is connected to other heading styles.

Gets all the heading styles used in the document. Searches for the heading's numbering definition reference in styles.xml. Next searches the associated numbering definition in numbering.xml. Next searches the correct numbering level definition associated to the heading. Checks that the numbering is multilevel and done correctly using the heading styles.

Parameters

`errorIdsAndPositions`: A dict for appending errors in key - stringlist pairs. Should contain the following keys:

- 'manualNumbering' – numbering is done manually somehow.
- 'styleNotUsed' – an expected heading style is not used.
- 'differentNumbering' – some heading style is using different numbering than some other heading styles.
- 'notMultilevel' – the numbering is not multilevel.
- 'outlineLvl' – the outline of a heading style is not correct.
- 'numStart' – the numbering doesn't start at 1.
- 'numWrong' – the numbering is somehow not done with styles.
- 'numFormat' – the numbering format is not correct.
- 'notSequential' – heading styles are not used correctly in a row for example heading 3 is used after heading 1.

Note: XML example:

styles.xml:

<w:style w:type="paragraph" w:styleId="Heading2"> - Heading 2 style definition

<w:name w:val="heading 2"/>

<w:pPr>

<w:numPr>

<w:ilvl w:val="1"/> - Numbering Level Reference

<w:numId w:val="1"/> - Numbering Definition Instance Reference

</w:numPr>

<w:outlineLvl w:val="1"/>

</w:pPr>

</w:style>

numbering.xml:

<w:abstractNum w:abstractNumId="0"> - Abstract Numbering Definition

<w:multiLevelType w:val="multilevel"/> - Abstract Numbering Definition Type

<w:lvl w:ilvl="0"> - </w:lvl> - Numbering Level Definition

<w:lvl w:ilvl="1"> - Numbering Level Definition

checkIndex(*document*)

Checks that the document has an automatically made index.

Return Value

False if an index is missing, '2' if index is not automatically made and True if everything was OK.

checkIndexContent(*document*)

Checks that the document has a index that is not empty, and that the index entries are referenced somewhere in the document.

First gets all the index styles' definitions from styles.xml and finds paragraphs using the styles in the document.xml. Checks that there is a field code element indicating that the index is generated automatically. Collects the content of the index and checks it isn't empty. Finds references to the index entries and matches them to the index content.

Return Value

'3' if the index is empty, '4' if the content does not match with the document and True if everything went OK.

Note: XML example:

Index example:

<w:p w:rsidR="002F2A09" w:rsidRDefault="00CA51D5">

<w:r>

<w:fldChar w:fldCharType="begin"/>

</w:r>

<w:r>

<w:instrText xml:space="preserve"> INDEX \c "2" \z "1035" </w:instrText>

</w:r>

<w:r>

<w:fldChar w:fldCharType="separate"/>

</w:r>

</w:p>

<w:p w:rsidR="002F2A09" w:rsidRDefault="002F2A09">

<w:pPr>

<w:pStyle w:val="Index1"/>

<w:tabs>

<w:tab w:val="right" w:leader="dot" w:pos="3950"/>

</w:tabs>

</w:pPr>

<w:r>

<w:t>Index entry level 1</w:t>

</w:r>

</w:p>

Reference example:

<w:r w:rsidR="00B27B47">

<w:instrText xml:space="preserve"> XE "</w:instrText>

</w:r>

<w:r w:rsidR="00B27B47" w:rsidRPr="00B27B47">

<w:instrText>Level 1 entry</w:instrText>

</w:r>

<w:r w:rsidR="00B27B47" w:rsidRPr="00B27B47">

<w:instrText>:</w:instrText>

</w:r>

checkDoubleWhitespaces(*document*)

Checks double whitespaces in the document.

Return Value

The amount of occurrences of the double whitespaces found in the document, False otherwise.

checkAsterisk(*document*)

Checks the *-character in the document.

Return Value

The amount of occurrences of the asterisks found in the document, False otherwise.

checkStringFromDocument(*docXml*, *string*)

Checks if a string is found in the text content of the document (in the w:t-elements). If string is found, returns how many occurences were found in a paragraph.

Return Value

The amount of occurrences of the string is found in the document, False otherwise.

checkTabs(*document*)

Checks if the tabulator is used in the document.

Return Value

The amount of the tabulator occurrences found in the document, False if none was found.

Note: Exceptions:

- automatically generated table of contents and index contain tabulators.
- before an automatically generated index there is a paragraph-element with <instrText>-element and a <tab>-element.

isParagraphEmpty(*p*, *styleXml*)

Checks if a paragraph is empty.

Parameters

`p:` The paragraph element under inspection.

Return Value

False if the paragraph is not empty, True if it is empty.

Note: Expections:

Picture in the document produces an empty paragraph. Empty table cell produces an empty paragraph. A table produces an empty paragraph right after the table. Objects and graphics produce an empty paragraph. ...

checkEmptyParagraphs(*document*)

Finds all empty paragraphs in the document.

Return Value

amount of empty paragraph occurrences in the document, False if none was found.

Note: Expections:

Picture in the document produces an empty paragraph. Empty table cell produces an empty paragraph. A table produces an empty paragraph right after the table. ...?

checkList(*document*, *listName*=`'List'`)

Goes through all paragraph elements in the document looking for paragraphs that use some list style.

Parameters

`listName`: The list stylename we want to check. Defaults to 'List', which finds list styles such as 'List', 'List Bullet', 'List Numbered'.

Return Value

True, if a list style is used in the document, False otherwise.

checkSpreadsheetChart(*document*)

Checks that the document has a chart copied from a spreadsheet document. The Chart must be pasted as a link.

checkSpreadsheetTable(*document*)

Checks that the document has a table copied from a spreadsheet document. For now checks that the table is pasted as a link.

checkPresentationGraphicsChart(*document*)

Checks that the document contains a chart pasted from PowerPoint as a vector graphics picture or as an object. Doesn't really know if the picture or object is actually from PowerPoint!

10.2 Variables

Name	Description
__package__	**Value:** `'src.inspectors'`

11 Module src.inspectors.mso_meta_inspector

The module is for getting Microsoft Office file formats' docx, pptx, and xlsx meta informations. The module is not currently used, but it will be in future development.

Author: Juho Tammela

11.1 Functions

getElementValue(*elementTagName*, *coreXml*)

Returns the text content of the given element.

elementTagName – the tag name of the element.

Return Value

Thetext content of the element. If no text content is found, returns the error message.

getCreator(*coreXml*)

Return the creator of the document.

getLastModifier(*coreXml*)

Return the last modifier

getCreateDate(*coreXml*)

getLastModifiedDate(*coreXml*)

getRevision(*coreXml*)

11.2 Variables

Name	Description
__package__	**Value:** `'src.inspectors'`

12 Module src.inspectors.odp_inspector

The module provides the scratch for inspecting opd files.

Author: Olli Kauppinen

12.1 Variables

Name	Description
filename	**Value:** `'sampleFiles/odp/yliopisto_opiskelu.odp'`
__package__	**Value:** `'src.inspectors'`

13 Module src.inspectors.odt_inspector

The module provides the methods for inspecting odt files.

Author: Vili Auvinen, Juho Tammela, Olli Kauppinen

13.1 Functions

getPageMarginals(*documentDict*)

Get the page marginals. Searches for only from used master pages.

Return Value

The page marginals. If the marginals are different between the used pages, then return false.

See Also: convertCmOrInDictToString

getPageSize(*documentDict*)

Get the page size.

Return Value

The converted page size. If the size is different between the used pages, then returns False.

See Also: convertCmOrInDictToString

checkEmptyParagraphs(*documentDict*)

Checks the empty paragraphs from document. getDocumentPararaphs method gets all paragraphs to be checked for. An empty paragraph is permitted after the table of content and in page break elements.

Return Value

The number of the empty paragraphs if efound, otherwise returns False.

checkDoubleWhitespaces(*documentDict*)

Checks double spaces. Checks if the document has text:s tag.

Return Value

The amount of the double spaces.

Note: XML example:

<text:s text:c=”2”/> –> 3 spaces

<text:s/> –> 2 spaces

checkTabs(*documentDict*)

Checks tabulators from the document. getDocumentPararaphs method gets the all paragraphs to be checked for.

Return Value

The number of the tabulators if found, otherwise returns False.

checkAsterisk(*documentDict*)

Checks asterisk from the document. getDocumentPararaphs method gets all paragraphs to check for.

Return Value

The number of the asterisks if found, otherwise returns False.

checkTocContent(*documentDict*)

Compares document headings to the TOC entries.

Return Value

True if all entries matches otherwise returns an error message.

checkTOC(*documentDict*)

Checks if the document contains the table of contents.

Return Value

True if there is the table of content, otherwise returns False.

checkIndex(*documentDict*)

Checks if the document have the alphabetical index.

Return Value

True if the alphabetical index exists otherwise returns False.

checkIndexContent(*documentDict*)

Compares the document marked texts to the alphabetical index entries.

Return Value

True if all entries matches otherwise returns an error code.

checkTable(*documentDict*)

Checks if the document has a table.

Return Value

True if there is a table and False if not.

checkPageNumberFromFooterAndHeader(*documentDict*, *masterPageElement*, *element*)

Checks page number format by given element and master page element.

Parameters

`masterPageElement:` the master page element to look for.

`element:` a footer or a header element.

Return Value

The number format if it exists, otherwise returns False.

The number format is optionally in the element (footer or header). If the number format is not in the element then the page-layout element defines number format.

getAuthorAndPageNumberFormat(*documentDict*, *masterPageElement*)

Gets the author and the number format from the header and the footer.

Parameters

`masterPageElement:` the master page element to look for.

Return Value

The dictionary which contains the author and the page number format.

checkHeadingNumbering(*documentDict*, *errorIdsAndPositions*)

Checks the outline style. Level is highest used headings outline level. Normally Heading 1 should be 1 and Heading 2 should be 2.

Return Value

True if ok, False if not.

Note: XML example:

<text:outline-style style:name="Outline">

<text:outline-level-style text:level="1" style:num-format="1">

<style:list-level-properties text:list-level-position-and-space-mode="label-alignment">

<style:list-level-label-alignment text:label-followed-by="listtab" text:list-tab-stop-position="0.762cm" fo:text-indent="-0.762cm" fo:margin-left="0.762cm"/>

</style:list-level-properties>

</text:outline-level-style>

<text:outline-level-style text:level="2" style:num-format="1" text:display-levels="2">

<style:list-level-properties text:list-level-position-and-space-mode="label-alignment">

<style:list-level-label-alignment text:label-followed-by="listtab" text:list-tab-stop-position="1.016cm" fo:text-indent="-1.016cm" fo:margin-left="1.016cm"/>

</style:list-level-properties>

</text:outline-level-style>

<text:outline-level-style text:level="3" style:num-format="">

<style:list-level-properties text:list-level-position-and-space-mode="label-alignment">

<style:list-level-label-alignment text:label-followed-by="listtab" text:list-tab-stop-position="1.27cm" fo:text-indent="-1.27cm" fo:margin-left="1.27cm"/>

</style:list-level-properties>

</text:outline-level-style>

...

</text:outline-style>

checkImages(*documentDict*)

Checks if the document contains an image.

Return Value

True if there is an image, otherwise False.

checkList(*documentDict*)

Checks if the document contains a list.

Return Value

True if there is a list, otherwise False.

printLists(*documentDict*)

Prints the lists of the document.

To Do: getListContent

getObjectPaths(*documentDict*)

Gets objects paths. Searches if the document have an image.

Return Value

The object path list if founds an image, otherwise an error message

getStyle(*documentDict*, *styleName*)

Get style defination attributes by given style name. parentStyleList is for executing the inheritation of styles.

Return Value

The style defination dictionary.

Notes:

- Inheritation of the styles:

  default paragraph -style-> standard-style -> style(Text body) -> P-style -> T-style

- XML example (styles.xml):

  <style:style style:name="Standard" style:family="paragraph" style:class="text">

  <style:paragraph-properties fo:orphans="2" fo:widows="2" style:writing-mode="lr-tb"/>

  <style:text-properties style:use-window-font-color="true" style:font-name="Courier New" fo:font-size="10pt" fo:language="fi" fo:country="FI" style:font-name-asian="Times New Roman" style:font-size-asian="10pt" style:font-name-complex="Times New Roman" style:font-size-complex="10pt" style:language-complex="ar" style:country-complex="SA"/>

  </style:style>

  <style:style style:name="Text_20_body" style:display-name="Text body" style:family="paragraph" style:parent-style-name="Standard" style:class="text" style:master-page-name="">

  <style:paragraph-properties fo:margin-left="1cm" fo:margin-right="0cm" fo:margin-top="0.247cm" fo:margin-bottom="0.247cm" fo:text-indent="0cm" style:auto-text-indent="false" style:page-number="auto" fo:break-before="auto" fo:break-after="auto"/>

  <style:text-properties style:font-name="Tahoma"/>

  </style:style

- XML example (content.xml):

  <style:style style:name="P2" style:family="paragraph" style:parent-style-name="Text_20_body">

  <style:paragraph-properties fo:text-align="start" style:justify-single-word="false"/>

  </style:style>

checkEndnotesAndFootnotes(*documentDict*)

Checks the end- and the footnotes.

Return Value

True if there is endnote or footnote in the document, otherwise False.

checkImageCaptions(*documentDict*)

Checks the caption and the reference of the image.

Return Value

True if the document images have caption and reference, otherwise False.

checkCoverPage(*documentDict*)

Checks that the front page is done correctly

Parameters

`title:` True if the title in cover page is the same as in the document meta.

`name:` True if the cover page contains the same author name as in the document meta.

`email:` True if the cover page contains e-mail address.

Return Value

The cover definitions in a dictionary.

getPageNumberFormatAndAuthor(*documentDict*, *section*)

Gets the page number format and the author name from the document.

Parameters

`section:` can have a value 'cover', 'toc' or 'text'.

Return Value

The dictionary which contains the author and the page number information.

checkHeadersAndFooters(*documentDict*)

Checks that the headers and the footers of the document are made correctly.

Assumes that the document has three sections:

1. the cover section,
2. the table of contents section or the toc section and
3. the actual content section or the text section.

See Also: checkSections method must pass in order to run this method

Places findings in the headerAndFooterDict as key-boolean pairs:

- 'frontPage' was there headers or footers in the cover section.
- 'tocPageNumbering' is there a page numbering in the toc section.
- 'differentPageNumbering' is the page numbering different in the cover and text sections.
- 'nameInToc' is the last modifiers name in toc section header or footer.
- 'nameInText' is the last modifiers name in text section header or footer.
- 'pageNumbering' is there a page numbering in the text section.
- 'tocNumStart' does the toc section page numbering start at 1.
- 'textNumStart' does the text section page numbering start at 1.

checkSections(*documentDict*, *errorList*)

Checks that the document sections have been made correctly. If the amount of the section breaks is not over 3 then return the error message list.

Return Value
True if the sections are ok, return errorList if not ok.

getMetaAuthor(*documentDict*)

Gets the author, who have last modified the document.

Return Value
The last modified author.

getMetaTitle(*documentDict*)

Gets document title from the meta information.

Return Value
The title which have defined in meta information.

getMetaEdited(*documentDict*)

Gets the last modified date and time from the meta.

Return Value
The last modified date in ISO 8601 standard (yyyy-mm-ddThh:mm:ss)

checkStyleUsage(*documentDict*, *errorIdsAndPositions*)

Goes through all the elements in the document which have used any style. Checks that elements are using the correct styles (i.e. not Standard or Default style) and that no manual style definitions are made (like T1).

13.2 Variables

Name	Description
__package__	**Value:** `'src.inspectors'`

14 Module src.inspectors.ooo_meta_inspector

The module provides the methods for getting OpenOffice.org file formats' odt, odp, and ods meta informations.

Author: Olli Kauppinen

14.1 Functions

getMetaInformation(*documentDict*, *elementTagName*)

Gets the meta information by the given elementTagName from odt,odp and ods documents.

Return Value
The meta information.

getDocumentStatistic(*documentDict*, *attributeName*)

Gets the document statistic by the given attributeName.

Return Value
The statistic value.

getMeta(*documentDict*)

Gets all the document meta information.

Return Value
The meta information in the dictionary. Tag names are the key values in the dictionary.

getOdtStatistic(*documentDict*)

Gets all the odt document statistic information.

Return Value
The statistic information in the dictionary. Attribute names are the key values in the dictionary.

getOdsStatistic(*documentDict*)

Gets all the ods document statistic information.

Return Value
The statistic information in the dictionary. Attribute names are the key values in the dictionary.

printOdpStatistic()

Get all the odp document statistic information.

Return Value
The statistic information in the dictionary. Attribute names are the key values in the dictionary.

14.2 Variables

Name	Description
__package__	**Value:** `'src.inspectors'`

15 Module src.requirements

The module contains the classes for reading and storing the data of a requirements XML file.

Requirements object contains a list of Requirement objects.

Author: Vili Auvinen, Olli Kauppinen, Juho Tammela

15.1 Variables

Name	Description
__package__	**Value:** `None`

15.2 Class Requirements

The class is used for reading a requirements file and storing it in a list of Requirement objects.

15.2.1 Methods

__init__(*self*, *requirementsFile*)

Parameters

`requirementsFile:` The requirements file as a DOM-tree.

getRequirements(*self*, *requirementsFile*)

Loops the requirement elements and stores them in a list as Requirement objects.

Return Value

The list of Requirement objects.

15.3 Class Requirement

The class is used for storing the information of one requirement.

15.3.1 Methods

__init__(*self*, *requirement*)

Parameters

`requirement:` The requirement as a DOM element.

getErrorMessage(*self*, *errorValue*=`'DEFAULT'`)

Gets a error message from the requirement.

Parameters

`errorValue:` The errorvalue of the error message, defaults to 'DEFAULT'.

getExpectedValues(*self*, *requirement*)

Gets all the expected values of a requirement.

Parameters

`requirement:` The requirement as a DOM element.

getFeedbacks(*self*, *requirement*)

Gets all the error feedback messages.

Parameters

`requirement:` The requirement as a DOM tree.

16 Module src.word_processing

The module makes the comparisons between the office document properties and requirements specified for each user.

Author: Vili Auvinen, Olli Kauppinen, Juho Tammela

To Do: Change the name of the module to word_inspector.

16.1 Functions

makeDocumentDict(*documentFile*, *fileExtension*)

Creates a document dict which contains the XML files from a document file by the given fileExtension.

Parameters

`fileExtension:` can be docx or odt or odp etc.

Return Value

The document dictionary.

Note: Code example: docXml = document['word/document.xml']

See Also: _checkers

processRequirements(*inspector*, *document*, *requirements*, *results*)

Processes requirements by looping through checkers dict which contains the method names described in the XML requirement file.

Parameters

`inspector:`	defines the given inspector.
`document:`	defines document dictionary which contains the XML files.
`requirements:`	defines Requirements object which contains the given requirements in the XML requirement file.
`results:`	defines the given errors in the results dictionary.

printResults(*resultsDict*)

For testing.

checkBooleanRequirement(*function*, *requirement*, *document*, *results*)

Checks the boolean requirements. The inspector methods return a boolean.

Return Value

True if expectedValue is returned from the inspector method, False otherwise.

See Also: processRequirements for parameters.

checkDictRequirement(*function*, *requirement*, *document*, *results*)

Checks the dictionary requirement. The inspector methods return a dictionary of which the values are compared to the ones in XML requirement file.

See Also: processRequirements for parameters.

checkRequirementEndNoteFootNote(*inspector*, *requirement*, *document*, *results*)

checkRequirementSections(*inspector*, *requirement*, *document*, *results*)

Checks the requirement sections. The inspector method takes an empty list as an argument. If its length is not zero after the inspector method, it means that errors were founded.

Return Value

out from the method if inspectorData return False.

See Also: processRequirements for parameters.

checkRequirementMargins(*inspector*, *requirement*, *document*, *results*)

Checks the requirement margins by calling the checkDictRequirement.

See Also: processRequirements for parameters and checkDictRequirement for the actual method.

checkRequirementPageSize(*inspector*, *requirement*, *document*, *results*)

checkRequirementCoverPage(*inspector*, *requirement*, *document*, *results*)

checkRequirementHeadingNumbering(*inspector*, *requirement*, *document*, *results*)

Checks the heading numbering requirement.

Error ids and positions are defined in a dict which is then used in the method inspector.checkHeadingNumbering(document, errorIdsAndPositions). If the keys in the dict are not None, errors have been appended.

See Also: docx_inspector.checkHeadingNumbering(document, errorIdsAndPositions)., processRequirements for parameters.

checkRequirementStyles(*inspector*, *requirement*, *document*, *results*)

Checks the style requirements. Compares the style requirements described in the XML file to the document properties defined by the user. Appends a default error if inspector.getStyle returns False.

Return Value

Nothing if inspector.getStyle returns False.

See Also: processRequirements for parameters.

checkRequirementTOC(*inspector*, *requirement*, *document*, *results*)

Checks if the table of the contents exists. If it does not exist, appends a default error message.

If it exists, checks if the table of contents is correctly made. If not, append an error message.

See Also: processRequirements for parameters.

checkRequirementImages(*inspector*, *requirement*, *document*, *results*)

Checks if there are images in the document. Calls the checkBooleanRequirement function.

See Also: checkBooleanRequirement(function, requirement, document, results).

checkRequirementEmptyParagraphs(*inspector*, *requirement*, *document*, *results*)

Checks empty paragraphs from the document. Appends an error message if there are some to be found.

See Also: processRequirements for parameters.

checkRequirementList(*inspector*, *requirement*, *document*, *results*)

Checks if there are lists in the document. Calls the checkBooleanRequirement function.

See Also: processRequirements for parameters.

checkRequirementStyleUsage(*inspector*, *requirement*, *document*, *results*)

Checks the style usage. An error dict (below) with two key-value pairs is used in the method inspector.checkStyleUsage. The inspector method returns the text paragraphs where manual changes have been made or style has not been used at all.

errorIdsAndPositions = {'styleNotUsed': [], 'manualChanges': []}

See Also: processRequirements for parameters.

checkRequirementTabs(*inspector*, *requirement*, *document*, *results*)

Checks if the tabs have been used in the document. Does nothing if tabs are not found, otherwise append an error message.

See Also: processRequirements for parameters.

checkRequirementDoubleWhitespace(*inspector*, *requirement*, *document*, *results*)

Checks if double whitespaces are found in the document. Does nothing if double whitespaces are not found, otherwise append an error message.

See Also: processRequirements for parameters

checkRequirementAsterisk(*inspector*, *requirement*, *document*, *results*)

Checks if asterisks are found in the document. Does nothing if asterisks are not found, otherwise append an error. A special print formatting is used here. Could be useful in other methods as well.

See Also: processRequirements for parameters.

checkRequirementImageCaptions(*inspector*, *requirement*, *document*, *results*)

Checks if image captions are used.

See Also: checkBooleanRequirement, processRequirements for parameters.

checkRequirementHeadersAndFooters(*inspector*, *requirement*, *document*, *results*)

Checks the headers and the footers requirement.

The method can be run only if checkRequirementSections goes through. The function is hard to implement in a smart way.

See Also: checkSections, processRequirements for parameters.

checkRequirementIndex(*inspector*, *requirement*, *document*, *results*)

Checks if index is found in the document. If index is correctly made, checks the index content.

See Also: processRequirements for parameters.

inspect(*documentFile*, *requirements*, *fileExtension*)

Inspects a document by the given file extension which is either odt or docx.

Parameters

`documentFile:` docx or odt file.

`requirements:` the requirements specified in the XML requirement file.

`fileExtension:` docx or odt file.

16.2 Variables

Name	Description
inspectorDict	**Value:** {'docm': <module 'src.inspectors.docx_inspector' from '/h...
__package__	**Value:** 'src'

Index