```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
#
#The MIT License
#
#Copyright (c) 2011
#
#Permission is hereby granted, free of charge, to any person obtaining a copy
#of this software and associated documentation files (the "Software"), to deal
#in the Software without restriction, including without limitation the rights
#to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
#copies of the Software, and to permit persons to whom the Software is
#furnished to do so, subject to the following conditions:
#
#The above copyright notice and this permission notice shall be included in
#all copies or substantial portions of the Software.
#
#THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
#IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
#AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
#OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
#THE SOFTWARE.
#
#Authors:
#   Vili Auvinen (vili.k.auvinen@jyu.fi)
#   Olli Kauppinen (olli.kauppinen@jyu.fi)
#   Juho Tammela (juho.i.tammela@jyu.fi)

'''
The module provides the methods for inspecting odt files.

@author: Vili Auvinen, Juho Tammela, Olli Kauppinen
'''

import conversions
import string
import ooo_meta_inspector
import common_methods

def _getStyleElementByStyleName (documentDict, styleName):
    '''Gets style element by the given style name.
    It searches first from the file content.xml if doesn't find then searches from the file style.xml.

    @return: The style element if the style exists, otherwise returns None.
    '''
    styleElements = documentDict['content.xml'].getElementsByTagName('style:style')
    for element in styleElements:
        if element.getAttribute('style:name') == styleName:
            return element

    styleElements = documentDict['styles.xml'].getElementsByTagName('style:style')
    styleElements += documentDict['styles.xml'].getElementsByTagName('text:list-style')
    for element in styleElements:
        if element.getAttribute('style:name') == styleName:
            return element
    return None


def _getStyleElementByDisplayName (documentDict, styleName):
    '''Gets the style element by the given style name.
    It searches first from content.xml if doesn't find then searches from style.xml.

    @note: XML:
    <style:style style:name="Text_20_body" style:display-name="Text body">
    <style:style style:name="tutkielma">

    Display name --> style name
    " "-->"_20_"
    "_"-->"_5f_"

    @return: The element of the style display name if it exists, otherwise returns None.
    '''
    styleElements = documentDict['content.xml'].getElementsByTagName('style:style')
    for element in styleElements:
        if element.getAttribute('style:display-name').lower() == styleName.lower():
            return element

    styleElements = documentDict['styles.xml'].getElementsByTagName('style:style')
    for element in styleElements:
        if element.getAttribute('style:display-name').lower() == styleName.lower():
            return element
    return None

def _getStyleDisplayNameByStyleName (documentDict, styleName):
    '''Gets the style display name by the given style name.
    It uses _getStyleElementByStyleName method to find the style.
    If the style doesn't have display-name attribute then the display name is just the style name.

    @note: XML example:

    <style:style style:name="Text_20_body" style:display-name="Text body">

    <style:style style:name="tutkielma">

    Display name --> style name
    " "-->"_20_"
    "_"-->"_5f_"

    @return: The style display name.
    '''
```

```
    element = _getStyleElementByStyleName(documentDict, styleName)
    if element.hasAttribute('style:display-name'):
        return element.getAttribute('style:display-name')
    else:
        return styleName

def _getMasterPageStyleElement (documentDict, masterPageStyleName):
    '''Get the master page style element by the given master page style name.

    @return: The master page element.
    '''
    masterPageStyles = documentDict['styles.xml'].getElementsByTagName ('style:master-page')
    for element in masterPageStyles:
        if masterPageStyleName == element.getAttribute('style:name'):
            return element
    return None

def _getPageLayoutElement(documentDict, pageLayoutName):
    '''Gets page layout element by the given page layout name.

    @return: The page layout.
    '''
    pageLayoutElements = documentDict['styles.xml'].getElementsByTagName ('style:page-layout')
    for element in pageLayoutElements:
        if pageLayoutName == element.getAttribute('style:name'):
            return element
    return None


def getPageMarginals(documentDict):
    '''Get the page marginals.
    Searches for only from used master pages.

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

    @see: convertCmOrInDictToString
    '''
    margins = {'top': None, 'bottom': None, 'right': None, 'left': None}
    usedMasterPages = _getUsedMasterPageElements(documentDict)

    for element in usedMasterPages:
        pageLayoutName = element.getAttribute('style:page-layout-name')
        layoutElement = _getPageLayoutElement(documentDict, pageLayoutName)
        for key in margins.keys():
            margin = layoutElement.getElementsByTagName('style:page-layout-properties')[0].getAttribute('fo:margin-' + key)

            if margins [key] is None:
                margins [key] = margin
            elif margins [key] != margin:
                return False



    return conversions.convertCmOrInDictToString(margins)

def getPageSize(documentDict):
    '''Get the page size.

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

    @see: convertCmOrInDictToString
    '''
    pageSize = {'height': None, 'width': None }

    usedMasterPages = _getUsedMasterPageElements(documentDict)

    for element in usedMasterPages:
        pageLayoutName = element.getAttribute('style:page-layout-name')
        layoutElement = _getPageLayoutElement(documentDict, pageLayoutName)
        for key in pageSize.keys():
            size = layoutElement.getElementsByTagName('style:page-layout-properties')[0].getAttribute('fo:page-' + key)

            if pageSize [key] is None:
                pageSize [key] = size
            if pageSize [key] != size:
                return False

    return conversions.convertCmOrInDictToString(pageSize)

def _getUsedMasterPageElements (documentDict):
    ''' Get the all master page elements which are used in document.
    'Standard' master page style used if there is no other definitions.

    @return: The list of the used master page elements.
    '''
    usedMasterPageElements = []
    usedMasterPageDict = _getAllStyleNamesWithDifferentMasterPage(documentDict)
    for masterPageName in usedMasterPageDict.values():
        masterPageElement = _getMasterPageStyleElement(documentDict, masterPageName)
        usedMasterPageElements.append(masterPageElement)

    if len(usedMasterPageElements) == 0:
        usedMasterPageElements.append(_getMasterPageStyleElement(documentDict, 'Standard'))
    return usedMasterPageElements

def _getDefaultStyleElement(documentDict, styleFamily):
    '''Get the default style element by the given style family.

    @note:
```

```
    Every style is based on style family.

    <style:style style:name="Heading_20_1" style:display-name="Heading 1" style:family="paragraph">

    @param styleFamily: gets wanted default style.

    Style family can be paragraph, graphic, table or table-row.

    @return: default style element.
    '''
    defaultStyleElements = documentDict['styles.xml'].getElementsByTagName('style:default-style')
    for element in defaultStyleElements:
        if element.getAttribute ('style:family') == styleFamily:
            return element
    return None


def _getMasterPageStyleName(documentDict, styleElement):
    '''Get the master page style name by the given style element.

    @return: The master page name, if master page is '' then return 'Standard'.
    '''
    masterPageName = styleElement.getAttribute ('style:master-page-name')
    if masterPageName == '':
        return 'Standard'
    else:
        return masterPageName

def 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: The number of the empty paragraphs if efound, otherwise returns False.
    '''
    pageBreakStyles = _getPageBreakStyleNames(documentDict)
    paragraphs = _getDocumentParagraphs(documentDict)
    emptyParagraphs = 0
    for element in paragraphs:
        if not element.childNodes and element.previousSibling.tagName != 'text:table-of-content' and element.getAttribute('text:style-name')not in s
            emptyParagraphs = emptyParagraphs + 1

    if emptyParagraphs == 0:
        return False
    return emptyParagraphs

def checkDoubleWhitespaces(documentDict):
    '''Checks double spaces.
    Checks if the document has text:s tag.

    @note: XML example:

    <text:s text:c="2"/> --> 3 spaces

    <text:s/> --> 2 spaces

    @return: The amount of the double spaces.
    '''
    doubleSpaces = documentDict['content.xml'].getElementsByTagName('text:s')

    if len(doubleSpaces) == 0:
        return False

    return len(doubleSpaces)

def checkTabs(documentDict):
    '''Checks tabulators from the document.
    getDocumentPararaphs method gets the all paragraphs to be checked for.

    @return: The number of the tabulators if found, otherwise returns False.
    '''
    paragraphs = _getDocumentParagraphs(documentDict)
    tabs = 0
    for element in paragraphs:
        elementListLength = len(element.getElementsByTagName('text:tab'))
        if  elementListLength != 0:
            tabs = tabs + elementListLength
    if tabs == 0:
        return False

    return tabs

def checkAsterisk(documentDict):
    '''Checks asterisk from the document.
    getDocumentPararaphs method gets all paragraphs to check for.

    @return: The number of the asterisks if found, otherwise returns False.
    '''
    paragraphs = _getDocumentParagraphs(documentDict)
    asterisk = 0
    for element in paragraphs:
        if common_methods.checkStringFromContent(element, '*') is True:
            asterisk += 1
    if asterisk == 0:
        return False
    return asterisk

def _getDocumentParagraphs(documentDict):
    '''Gets all the paragraphs from the document.
    Including all text-p(text paragraphs) and text-h (headings) elements.
    It is used in checkTabs and checkEmptyParagraphs.

    @return: The list of the used elements.
```

```
    '''
    #FIXME: must append more paragraphs (like text:list) and deeper (all descendants)
    elementList = []
    officeTextElement = documentDict['content.xml'].getElementsByTagName ('office:text')[0]
    elements = officeTextElement.childNodes
    for element in elements:
        if element.nodeName == 'text:p' or element.nodeName == 'text:h':
            elementList.append(element)
    return elementList

def _getListOfUsedStyleElements (documentDict):
    '''Gets all used style elements.
    In the file content.xml the element office:body contains the used styles.

    @return: The element list of the used styles.
    '''

    usedStyleElements = []

    bodyElements = common_methods.getDescendants(documentDict['content.xml'].getElementsByTagName('office:body')[0], [])

    for element in bodyElements:
        if element.nodeType is not element.TEXT_NODE and element.hasAttribute('text:style-name'):
            if len(common_methods.getTextContent(element)) != 0:
                usedStyleElements.append(element)

    return usedStyleElements

def _getListOfUsedStyleNames (documentDict):
    '''Gets all used style names.
    Gets the parent style of PI-style (I is integer value) like P1 is Heading_20_1.

    @return: The list of all the style names.
    '''
    usedStyleNames = []
    for element in _getListOfUsedStyleElements (documentDict):
        styleName = element.getAttribute('text:style-name')
        if styleName [0] == 'P' and styleName[1].isdigit():
            styleElement = _getStyleElementByStyleName(documentDict, styleName)
            if styleElement.hasAttribute('style:parent-style-name'): # if P-style doesn't have parent-style-name, it's not real style.
                usedStyleNames.append(styleElement.getAttribute('style:parent-style-name'))
        else:
            usedStyleNames.append(element.getAttribute('text:style-name'))

    usedStyleNames = list(set(usedStyleNames)) #deletes duplicates from list
    usedStyleNames.sort()

    return usedStyleNames

def _checkPageBreakStyleElement (styleElement):
    '''Checks if the given style element contains the page break.

    @return: The style element if contains the page break, otherwise returns False.
    '''

    hasParagraphProperties = styleElement.getElementsByTagName('style:paragraph-properties')
    if hasParagraphProperties:
        if hasParagraphProperties[0].getAttribute('fo:break-before') == 'page':
            return styleElement
    return False

def _getPageBreakStyleNames(documentDict):
    '''Gets the names of the styles which contains the page break.

    @return: The list of page break style names.
    '''
    pageBreakStylesNames = []
    styleElements = documentDict['content.xml'].getElementsByTagName('style:style')
    styleElements += documentDict['styles.xml'].getElementsByTagName('style:style')
    for element in styleElements:
        pageBreak = _checkPageBreakStyleElement(element)
        if pageBreak is not False:
            pageBreakStylesNames.append(element.getAttribute('style:name'))

    return pageBreakStylesNames

def _getAllStyleNamesWithDifferentMasterPage(documentDict):
    '''Gets all the style names which changes the master page.
    The master page will change when a style has master-page-name attribute
    and its is nonempty. If is empty ("") then master page is standard and
    if has no attribute with same as previous master page.

    @note: masterPageDict: contains a key as a style name and value as a master page name.

    @return: The dictionary of the styles which changes master page.
    '''
    styleElements = documentDict['content.xml'].getElementsByTagName('style:style')
    styleElements += documentDict['styles.xml'].getElementsByTagName('style:style')

    masterPageDict = {}

    for element in styleElements:
        if element.getAttribute('style:master-page-name') != '':
            masterPageDict[element.getAttribute('style:name')] = element.getAttribute('style:master-page-name')
    return masterPageDict

def _getSectionBreakElements(documentDict):
    '''Gets section break elements from the document.
    Finds all the elements (including text, list, heading...) which chance the section.

    @return: The list of the elements which changes the section.
    '''
    usedStyleElementsInDocument = _getListOfUsedStyleElements(documentDict)
```

```
    masterPagesDict = _getAllStyleNamesWithDifferentMasterPage(documentDict)
    sectionBreakElements = []
    for element in usedStyleElementsInDocument:
        if masterPagesDict.has_key(element.getAttribute('text:style-name')):
            sectionBreakElements.append(element)
    return sectionBreakElements

def _getTOC(documentDict):
    '''Gets table of content.
    Each TOC entry is own entry in tocList.

    @return: The list of the elements in table of content.
    '''
    tocList = []
    if checkTOC(documentDict) is True:
        toc = documentDict['content.xml'].getElementsByTagName('text:table-of-content')
        tocTextElements = toc[0].getElementsByTagName ('text:p')
        for element in tocTextElements:
            if element.parentNode.nodeName == 'text:index-title':
                tocTitle = common_methods.getTextContent(element)
            else:
                tocList.append(common_methods.getTextContent(element))
        return tocList

def checkTocContent(documentDict):
    '''Compares document headings to the TOC entries.

    @return: True if all entries matches otherwise returns an error message.
    '''
    #FIXME: add better error messages (not False)
    tocList = _getTOC(documentDict)
    headingList = _getHeadingList(documentDict)['headings']

    if len(tocList) != len(headingList):
        return False#"Number of document headings doesn't match with number of TOC entries "
    i = 0
    while i < len(tocList):
        #if string.find (tocList[i], headingList[i]) == -1:
        if headingList[i] not in tocList[i]:
            return False #headingList[i-1]+" -heading doesn't exist in table of content"
        i += 1
    return True



def checkTOC(documentDict):
    '''Checks if the document contains the table of contents.

    @return: True if there is the table of content, otherwise returns False.
    '''
    toc = documentDict['content.xml'].getElementsByTagName('text:table-of-content')
    if len(toc) == 0:
        return False
    else:
        return True

def checkIndex(documentDict):
    '''Checks if the document have the alphabetical index.

    @return: True if the alphabetical index exists otherwise returns False.
    '''
    aIndex = documentDict['content.xml'].getElementsByTagName('text:alphabetical-index')
    if len(aIndex) == 0: return False
    return True

def _getIndexContentFromDocument (documentDict):
    '''Gets marked alphabetical index entries from the document.

    @return: The content list of the alphabetical index entries.
    '''
    indexContentList = []
    indexMarks = documentDict['content.xml'].getElementsByTagName ('text:alphabetical-index-mark-start')
    for element in indexMarks:
        indexContentList.append (element.nextSibling.nodeValue)
    return indexContentList

def _getIndexContent(documentDict):
    '''Gets the alphabetical index content.
    Each alphabetical index entry is an own entry in the list.

    @return: The list of the alphabetical index content.
    '''
    alphabeticalIndexList = []
    if checkIndex(documentDict) is True:
        aIndex = documentDict['content.xml'].getElementsByTagName('text:alphabetical-index')[0]
        aIndexTextElements = aIndex.getElementsByTagName('text:p')
        for element in aIndexTextElements:
            alphabeticalIndexList.append(common_methods.getTextContent(element))
    return alphabeticalIndexList

def checkIndexContent(documentDict):
    '''Compares the document marked texts to the alphabetical index entries.

    @return: True if all entries matches otherwise returns an error code.
    '''
    indexList = _getIndexContent(documentDict)
    markedList = _getIndexContentFromDocument(documentDict)
    if len(indexList) == 0:
        return '3' # errorcode 3 = Index is empty

    for markedItem in markedList:
        for indexItem in indexList:
            found = False
```

```
            #if string.find (indexItem, markedItem) != -1:
            if markedItem in indexItem:
                found = True
                break
        if found is False:
            return '4' #errorcode 4 = marked item not found from index

    return True

def _getHeadingList(documentDict):
    '''Gets all headings from the document and the used outline level.
    Each heading is an own entry in the list.

    @return: The dictionary ['headings'] contains a list of headings and ['level'] contains the value of the highest used heading outline level.
    '''
    headingList = []
    headingOutlineLevel = 0
    headings = documentDict['content.xml'].getElementsByTagName('text:h')
    for element in headings:
        heading = common_methods.getTextContent(element)
        headingList.append(heading)
        if headingOutlineLevel < element.getAttribute('text:outline-level'):
            headingOutlineLevel = element.getAttribute('text:outline-level')
    return {'headings':headingList, 'level':headingOutlineLevel}

def checkTable (documentDict):
    '''Checks if the document has a table.

    @return: True if there is a table and False if not.
    '''
    table = documentDict['content.xml'].getElementsByTagName('table:table')
    if len(table) == 0:
        return True # "There is no table at all."
    else:
        return True


def _getTableDict (documentDict):
    '''Gets tables in dictionary.
    Every table is own entry in tablesDict (key = table1,table2...)
    Every tableDict has table's cell address as key (A1,A2...) and cell value as dictionary's value.

    @return: The dictionary of the table dictionaries.
    '''
    tablesDict={}
    tableNumber = 0
    if checkTable (documentDict) is True:
        tableElements = documentDict['content.xml'].getElementsByTagName('table:table')
        for tableElement in tableElements:
            tableDict = {}

            tableRowElements = tableElement.getElementsByTagName('table:table-row')
            rowIndex = 0
            for row in tableRowElements:
                rowIndex += 1
                rowCellElements = row.getElementsByTagName('table:table-cell')
                columnIndex = ord('A') - 1
                for cell in rowCellElements:
                    columnIndex += 1
                    index = chr(columnIndex) + str(rowIndex)
                    tableDict[index] = common_methods.getTextContent(cell)
            tableNumber +=1
            tablesDict['table'+str(tableNumber)]=tableDict
        return tablesDict
    return False #"There is no table at all"



def checkPageNumberFromFooterAndHeader(documentDict, masterPageElement, element):
    '''Checks page number format by given element and master page element.

    @param masterPageElement: the master page element to look for.
    @param element: a footer or a header element.

    @return: 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.
    '''
    pageLayoutElement = _getPageLayoutElement(documentDict, masterPageElement.getAttribute('style:page-layout-name'))
    pageNumberElements = element.getElementsByTagName('text:page-number')
    if pageNumberElements:
        if pageNumberElements[0].hasAttribute('style:num-format'):
            numFormat = pageNumberElements[0].getAttribute('style:num-format')
        else:
            numFormat = pageLayoutElement.getElementsByTagName('style:page-layout-properties')[0].getAttribute('style:num-format')
        return numFormat

    return False

def getAuthorAndPageNumberFormat(documentDict, masterPageElement):
    '''Gets the author and the number format from the header and the footer.

    @param masterPageElement: the master page element to look for.

    @return: The dictionary which contains the author and the page number format.
    '''

    meta = ooo_meta_inspector.getMeta(documentDict)
    footer = masterPageElement.getElementsByTagName ('style:footer')
    header = masterPageElement.getElementsByTagName ('style:header')

    authorAndNumberDict = {'headerPageNumber':None, 'headerAuthor': None, 'footerPageNumber': None, 'footerAuthor': None}
```

```
    if footer:
        if checkPageNumberFromFooterAndHeader(documentDict, masterPageElement, footer[0]) is not False:
            authorAndNumberDict['footerPageNumber'] = checkPageNumberFromFooterAndHeader(documentDict, masterPageElement, footer[0])
        if common_methods.checkStringFromContent(footer[0], meta['dc:creator']):
            authorAndNumberDict['footerAuthor'] = meta['dc:creator']

    if header:
        if checkPageNumberFromFooterAndHeader(documentDict, masterPageElement, header[0]) is not False:
            authorAndNumberDict['headerPageNumber'] = checkPageNumberFromFooterAndHeader(documentDict, masterPageElement, header[0])

        if common_methods.checkStringFromContent(header[0], meta['dc:creator']):
            authorAndNumberDict['headerAuthor'] = meta['dc:creator']

    return authorAndNumberDict


def 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.

    @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-lef

    </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-lef

    </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=

    </style:list-level-properties>

    </text:outline-level-style>

    ...

    </text:outline-style>

    @return: True if ok, False if not.
    '''

    outlineStyleElements = documentDict['styles.xml'].getElementsByTagName('text:outline-style')

    if len(outlineStyleElements) == 0:
        outlineStyleElement = _getStyleElementByStyleName(documentDict, 'Outline')
        outlineLevels = outlineStyleElement.getElementsByTagName('text:list-level-style-number')
    else:
        outlineStyleElement = outlineStyleElements[0]
        outlineLevels = outlineStyleElement.getElementsByTagName('text:outline-level-style')


    level = _getHeadingList(documentDict)['level']
    index = 0
    for element in outlineLevels:
        index = index + 1
        if index > int(level):
            return True
        if int(element.getAttribute('text:level')) == index and element.getAttribute('style:num-format') != "":
            "ok"#index"-level heading numbering is correctly made"
        else:
            return False#"numbering is not correct on "+ str(level)+ "-level"


def _getImagePaths(documentDict):
    '''Gets the image paths.
    Checks if the document have an image.
    Images are located in the picture folder.

    @return: The founded paths of the images in the list, otherwise returnsFalse.
    '''
    imagePathList = []
    if checkImages(documentDict) is True:
        imageElements = documentDict['content.xml'].getElementsByTagName ('draw:image')
        for element in imageElements:
            imagePath = element.getAttribute ('xlink:href')
            imagePathList.append(imagePath)

    if len(imagePathList)==0:
        return False
    return imagePathList
```

```
def checkImages(documentDict):
    '''Checks if the document contains an image.

    @return: True if there is an image, otherwise False.
    '''
    imageElements = documentDict['content.xml'].getElementsByTagName ('draw:image')
    if len(imageElements) == 0:
        return False
    else:
        return True

def checkList(documentDict):
    '''Checks if the document contains a list.

    @return: True if there is a list, otherwise False.
    '''
    listElements = documentDict['content.xml'].getElementsByTagName ('text:list')
    if len(listElements) == 0:
        return False

    return True


def printLists(documentDict):
    '''Prints the lists of the document.

    @todo: getListContent
    '''
    if checkList(documentDict) is True:
        listElements = documentDict['content.xml'].getElementsByTagName ('text:list')
        for element in listElements:
            print 'Lista tehty', element.getAttribute('text:style-name'), '-tyylillä, lista:'
            listContent = element.getElementsByTagName ('text:list-item')
            for text in listContent:
                print "-", common_methods.getTextContent(text)


def getObjectPaths(documentDict):
    '''Gets objects paths.
    Searches if the document have an image.

    @return: The object path list if founds an image, otherwise an error message
    '''
    objectPathList = []
    objectElements = documentDict['content.xml'].getElementsByTagName ('draw:object')
    if objectElements:
        for element in objectElements:
            objectPath = element.getAttribute ('xlink:href')
            objectPathList.append(objectPath)
        return objectPathList
    else:
        return False #error: there is no objects at all"

def _isStyleUsed(documentDict, styleName):
    '''Checks if the given style is used in the document.

    @return: True if used, otherwise return False.
    '''
    usedStyleList = _getListOfUsedStyleNames (documentDict)
    if styleName in set(usedStyleList):
        return True
    return False

def getStyle(documentDict, styleName):
    '''Get style defination attributes by given style name.
    parentStyleList is for executing the inheritation of styles.

    @note: Inheritation of the styles:

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

    @note: 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" sty

    </style:style>

    <style:style style:name="Text_20_body" style:display-name="Text body" style:family="paragraph" style:parent-style-name="Standard" style:class="t

    <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" s

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

    </style:style

    @note: 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>

    @return: The style defination dictionary.

    '''
    originalStyleName = styleName
    styleNameLower = styleName.lower()
    #if string.find(styleNameLower, 'body') != -1:
```

```
    if 'body' in styleNameLower:
        styleName = 'Text_20_body'
    if _getStyleElementByStyleName(documentDict, styleName) is None:
        if _getStyleElementByDisplayName(documentDict, styleName) is None:
            return False
        else: styleName = _getStyleElementByDisplayName(documentDict, styleName).getAttribute ('style:name')

    if _isStyleUsed(documentDict, styleName) is False:
        return False

    defaultStyleElement = _getDefaultStyleElement(documentDict, 'paragraph')

    fontName = defaultStyleElement.getElementsByTagName('style:text-properties')[0].getAttribute('style:font-name')
    fontSize = defaultStyleElement.getElementsByTagName('style:text-properties')[0].getAttribute('fo:font-size')
    language = defaultStyleElement.getElementsByTagName('style:text-properties')[0].getAttribute('fo:language')


    styleAttributeDict = {
    'style:name':styleName,
    'style:parent-style-name':None,
    'fo:language':language,
    'style:font-name':fontName,
    'fo:font-size':fontSize,
    'fo:text-transform':None,
    'fo:text-indent':'0',
    'fo:line-height':'100%',
    'fo:margin-left':0,
    'fo:margin-right':0,
    'fo:margin-top':0,
    'fo:margin-bottom':0,
    'fo:keep-with-next':False,
    'fo:text-align': 'start',
    'fo:orphans':None,
    'fo:widows': None,
    'fo:font-style': False,
    'fo:font-weight':False}

    translateDict = {
    'style:name': 'styleName',
    'style:font-name':'fontName',
    'fo:font-size':'fontSize',
    'fo:text-transform':'transform',
    'fo:margin-left':'indentLeft',
    'fo:margin-right':'indentRight',
    'fo:text-indent':'indentFirstLine',
    'fo:line-height':'linespacing',
    'fo:margin-top':'spacingBefore',
    'fo:margin-bottom':'spacingAfter',
    'fo:keep-with-next':'keepWithNext',
    'fo:text-align': 'alignment',
    'fo:font-style': 'italic',
    'fo:font-weight':'bold',
    'widowControl':'widowControl'
    }
    styleDict = {}

    parentStyleList = _getParentStyleList (documentDict, styleName)

    #Get style attribute values
    while parentStyleList:
        styleAttributeDict['style:name'] = parentStyleList.pop()
        styleAttributeDict = _getStyleAttributes (documentDict, styleAttributeDict)

    #Conversions:
    styleAttributeDict['fo:margin-top'] = conversions.convertCmOrInToPt(styleAttributeDict['fo:margin-top'])
    styleAttributeDict['fo:margin-bottom'] = conversions.convertCmOrInToPt(styleAttributeDict['fo:margin-bottom'])
    styleAttributeDict['style:name'] = originalStyleName #_getStyleDisplayNameByStyleName(documentDict, styleAttributeDict['style:name']).lower()
    styleAttributeDict['fo:line-height'] = conversions.convertPercentToDecimal(styleAttributeDict['fo:line-height'])
    styleAttributeDict['fo:font-size'] = str(round(float(styleAttributeDict['fo:font-size'].split('pt')[0]), 1))
    styleAttributeDict['fo:text-indent'] = conversions.convertCmOrInToString(styleAttributeDict['fo:text-indent'])
    styleAttributeDict['fo:margin-left'] = conversions.convertCmOrInToString(styleAttributeDict['fo:margin-left'])
    styleAttributeDict['fo:margin-right'] = conversions.convertCmOrInToString(styleAttributeDict['fo:margin-right'])
    if styleAttributeDict['fo:font-weight'] == 'bold': styleAttributeDict['fo:font-weight'] = True
    elif styleAttributeDict['fo:font-weight'] == 'normal': styleAttributeDict['fo:font-weight'] = False
    if styleAttributeDict['fo:font-style'] == 'italic': styleAttributeDict['fo:font-style'] = True
    elif styleAttributeDict['fo:font-style'] == 'normal': styleAttributeDict['fo:font-style'] = False
    if styleAttributeDict['fo:keep-with-next'] == 'always': styleAttributeDict['fo:keep-with-next'] = True
    if styleAttributeDict['fo:text-align'] == 'justify': styleAttributeDict['fo:text-align'] = 'both'
    elif styleAttributeDict['fo:text-align'] == 'start': styleAttributeDict['fo:text-align'] = 'left'
    elif styleAttributeDict['fo:text-align'] == 'end': styleAttributeDict['fo:text-align'] = 'right'


    if styleAttributeDict['fo:widows'] >= 2 and styleAttributeDict['fo:orphans'] >= 2:
        styleAttributeDict['widowControl'] = True #same in docx
    else:
        styleAttributeDict['widowControl'] = False #same in docx


    #Translate keys
    for key in translateDict.keys():
        styleDict[translateDict[key]] = styleAttributeDict[key]

    return styleDict

def _getStyleAttributes (documentDict, styleAttributeList):
    '''Searches if the style have the wanted attribute if it have then replace attribute value, otherwise keep old value.
    The style attribute list contains all the relevant style information.

    @return: The list of the styles attributes.
    '''
    styleElement = _getStyleElementByStyleName(documentDict, styleAttributeList['style:name'])

    if styleElement:
```

```
        paragraphPropertiesElement = styleElement.getElementsByTagName('style:paragraph-properties')
        textPropertiesElement = styleElement.getElementsByTagName('style:text-properties')

        for attribute in styleAttributeList.keys():
            if styleElement.hasAttribute (attribute):
                styleAttributeList[attribute] = styleElement.getAttribute (attribute)

        if paragraphPropertiesElement:
            for attribute in styleAttributeList.keys():
                if paragraphPropertiesElement[0].hasAttribute (attribute):
                    styleAttributeList[attribute] = paragraphPropertiesElement[0].getAttribute (attribute)

        if textPropertiesElement:
            for attribute in styleAttributeList.keys():
                if textPropertiesElement[0].hasAttribute (attribute):
                    if attribute == 'fo:font-size' and textPropertiesElement[0].getAttribute(attribute).endswith('%'):
                        styleAttributeList[attribute] = str(int(styleAttributeList[attribute].split('pt')[0]) * int(textPropertiesElement[0].getAttr
                    else:
                        styleAttributeList[attribute] = textPropertiesElement[0].getAttribute(attribute)

    return styleAttributeList


def _getParentStyleList (documentDict, styleName):
    '''Gets the parent style list for the given style name.

    @return: The list of parent styles (lists first entry is style itself).
    '''
    parentStyleList = [styleName]

    while _checkParentStyle(documentDict, styleName):
        styleName = _checkParentStyle(documentDict, styleName)
        parentStyleList.append(styleName)

    return parentStyleList


def _checkParentStyle (documentDict, styleName):
    '''Checks if the style have a parent style.

    @return: The parent style name.
    '''
    try:
        return _getStyleElementByStyleName(documentDict, styleName).getAttribute('style:parent-style-name')
    except:
        return None

def checkEndnotesAndFootnotes(documentDict):
    '''Checks the end- and the footnotes.

    @return: True if there is endnote or footnote in the document, otherwise False.
    '''
    noteElements = documentDict['content.xml'].getElementsByTagName('text:note')
    if len(noteElements) == 0:
        return False
#        for element in noteElements:
#            if element.getAttribute ('text:note-class')=='endnote':
#                return True
#            elif element.getAttribute ('text:note-class')=='footnote':
#                return True
    return True


def checkImageCaptions(documentDict):
    '''Checks the caption and the reference of the image.

    @return: True if the document images have caption and reference, otherwise False.
    '''
    caption = False
    reference = False
    imageElements = documentDict['content.xml'].getElementsByTagName ('draw:image')
    if imageElements:
        for element in imageElements:
            captionNode = element.parentNode.parentNode
            if len(common_methods.getTextContent(captionNode)) > 0:
                caption = True
            imagesReferenceElements = captionNode.getElementsByTagName('text:sequence')
            if imagesReferenceElements:
                reference = True

    if caption is False and reference is False:
        return False
    elif caption is False:
        return False #could replace with caption error message
    elif reference is False:
        return False #could replace with reference error message
    return True


def checkCoverPage(documentDict):
    '''Checks that the front page is done correctly

    @return: The cover definitions in a dictionary.

    @keyword title: True if the title in cover page is the same as in the document meta.
    @keyword name: True if the cover page contains the same author name as in the document meta.
    @keyword email: True if the cover page contains e-mail address.

    '''
    elementList = _getSectionElements(documentDict, 'cover')
    if elementList is None:
        return False
    meta = ooo_meta_inspector.getMeta(documentDict)
```

```
    cover = {
    'title':False,
    'name':False,
    'email' :False
    }
    for element in elementList:
        if common_methods.checkIfEmailAddress (element):
            cover['email'] = True
        if common_methods.checkStringFromContent(element, meta['dc:creator']):
            cover['name'] = True
        if common_methods.checkStringFromContent(element, meta['dc:title']):
            cover['title'] = True
    return cover



def getPageNumberFormatAndAuthor (documentDict, section):
    '''Gets the page number format and the author name from the document.

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

    @return: The dictionary which contains the author and the page number information.
    '''
    sectionList = _getSectionBreakElements(documentDict)
    sectionBrakeElements = {'cover':sectionList[0], 'toc':sectionList[1], 'text':sectionList[2]}

    styleName = sectionBrakeElements[section].getAttribute('text:style-name')

    styleElement = _getStyleElementByStyleName(documentDict, styleName)

    masterPageStyleName = _getMasterPageStyleName(documentDict, styleElement)
    masterPageStyleElement = _getMasterPageStyleElement (documentDict, masterPageStyleName)
    authorAndNumberDict = getAuthorAndPageNumberFormat(documentDict, masterPageStyleElement)


    authorAndNumberDict['numStart'] = styleElement.getElementsByTagName('style:paragraph-properties')[0].getAttribute('style:page-number')
    return authorAndNumberDict


def 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: 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.
    '''
    headerAndFooterDict = {'frontPage': False, 'tocPageNumbering': False, 'differentPageNumbering': False,
            'nameInToc': False, 'nameInText': False, 'pageNumbering': False, 'tocNumStart': False,
            'textNumStart': False}

    cover = getPageNumberFormatAndAuthor (documentDict, 'cover')
    toc = getPageNumberFormatAndAuthor (documentDict, 'toc')
    text = getPageNumberFormatAndAuthor (documentDict, 'text')

    if cover['footerAuthor'] or cover['headerAuthor'] or cover['headerPageNumber'] or cover['footerPageNumber'] is not None:
        headerAndFooterDict ['frontPage'] = True
    else: headerAndFooterDict ['frontPage'] = False

    if toc['footerAuthor'] or toc['headerAuthor'] is not None:
        headerAndFooterDict ['nameInToc'] = True
    if toc['headerPageNumber'] or toc['footerPageNumber'] is not None:
        headerAndFooterDict['tocPageNumbering'] = True

    if text['footerAuthor'] or text['headerAuthor'] is not None:
        headerAndFooterDict ['nameInText'] = True
    if text['headerPageNumber'] or text['footerPageNumber'] is not None:
        headerAndFooterDict['pageNumbering'] = True

    #TODO: tarkastus myös headerista (mitä jos on useita sivunumeroita???)
    if toc['footerPageNumber'] != text['footerPageNumber']:
        headerAndFooterDict['differentPageNumbering'] = True

    if toc['numStart'] == '1':
        headerAndFooterDict ['tocNumStart'] = True

    if text['numStart'] == '1':
        headerAndFooterDict ['textNumStart'] = True
    return headerAndFooterDict
```

```
def _getSectionElements (documentDict, section):
    '''Gets the elements of the wanted section.
    The section break elements changes the section.
    Searches trough the whole document.
    Adds each element to right section in sectionElements dictionary.
    When finds section break element then changes the dictionary to next section.
    First list elements to cover-section.
    Second list elements to toc-section.
    And last list element to text-section.
    Document have to have atleast 3 sections.

    @return: The section elements in the list.
    '''
    sectionBreakElementList = _getSectionBreakElements(documentDict)
    sectionElements = {'cover':None, 'toc':None, 'text':None}
    sectionList = []
    officeBodyElement = documentDict['content.xml'].getElementsByTagName ('office:body')[0] #always exact 1 element
    documentElementList = officeBodyElement.firstChild.childNodes
    i = 0
    k = len(documentElementList)
    if len(sectionBreakElementList) < 3:
        return None
    while not documentElementList[i].isSameNode(sectionBreakElementList[1]):
        sectionList.append(documentElementList[i])
        i += 1
    sectionElements ['cover'] = sectionList
    sectionList = []
    while not documentElementList[i].isSameNode(sectionBreakElementList[2]):
        sectionList.append(documentElementList[i])
        i += 1
    sectionElements ['toc'] = sectionList
    sectionList = []
    while i < k:
        sectionList.append(documentElementList[i])
        i += 1
    sectionElements ['text'] = sectionList

    return sectionElements[section]


def 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: True if the sections are ok, return errorList if not ok.
    '''
    toc = False
    cover = True
    sections = len(_getSectionBreakElements (documentDict))
    if sections < 3:
        return False
    coverElements = _getSectionElements(documentDict, 'cover')
    for coverElement in coverElements:
        if coverElement.nodeName == 'text:table-of-content':
            cover = False
            errorList.append('cover')

    tocElements = _getSectionElements(documentDict, 'toc')
    for tocElement in tocElements:
        if tocElement.nodeName == 'text:table-of-content':
            toc = True
            break
    if toc is False:
        errorList.append('toc')

    if toc and cover is True:
        return True
    else:
        return errorList


def _getMeta(documentDict):
    '''Gets all the meta information.

    @see: ooo_meta_inspector.getMeta

    @return: All the meta in the dictionary.
    '''
    return ooo_meta_inspector.getMeta(documentDict)

def getMetaAuthor (documentDict):
    '''Gets the author, who have last modified the document.

    @return: The last modified author.
    '''

    metaDict = ooo_meta_inspector.getMeta(documentDict)
    return metaDict ['dc:creator']

def getMetaTitle (documentDict):
    '''Gets document title from the meta information.

    @return: The title which have defined in meta information.
    '''
    metaDict = ooo_meta_inspector.getMeta(documentDict)
    return metaDict ['dc:title']

def getMetaEdited (documentDict):
    '''Gets the last modified date and time from the meta.

    @return: The last modified date in ISO 8601 standard (yyyy-mm-ddThh:mm:ss)
    '''
    metaDict = ooo_meta_inspector.getMeta(documentDict)
```

```
    return metaDict ['dc:date']

#FIXME: koko kappaleeseen käsin tehdyt muutokset tulevat P-tyyleihin (ei T), joten nämä jäävät huomioimatta/
#pitäisi tehdä tarkastus, jolla lailliset P-tyylit erotetaan laittomista
def 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).

    '''

    illegalStyles = []
    for styleElement in _getListOfUsedStyleElements(documentDict):
        styleName = styleElement.getAttribute('text:style-name')
        if styleName[0] == 'T' and styleName[1].isdigit():
            illegalStyles.append(styleName)

        elif styleName[0] == 'P' and styleName[1].isdigit():
            styleElement = _getStyleElementByStyleName(documentDict, styleName)
            for child in styleElement.childNodes:
                if child.hasAttribute('fo:font-style')or child.hasAttribute('fo:text-align')or child.hasAttribute('fo:font-weight'):
                    illegalStyles.append(styleName)

    for element in _getListOfUsedStyleElements(documentDict):
        if element.getAttribute('text:style-name') in set(illegalStyles):
            errorIdsAndPositions['manualChanges'].append(common_methods.getTextContent(element)[:30])
        elif element.getAttribute('text:style-name') == 'Standard':
            errorIdsAndPositions['styleNotUsed'].append(common_methods.getTextContent(element)[:30])
```