Article
Henry Pereira · Jan 16, 2023 14m read

Create and Read MS Word documents with ePython

In this article, I will show you how one can easily create and read Microsoft Word documents using InterSystems IRIS with the leverage power of embedded Python.

Setup

First things first, let ’ s install the Python module called python-docx. There are a lot of modules to write MS Word files in Python. However, this one is the easiest one to use.

Just execute the following command on the terminal:

```
!pip3 install python-docx
```

If you are working with Docker, like I do, just add the following line to a Dockerfile

```
ENV PIP_TARGET=${ISC_PACKAGE_INSTALLDIR}/mgr/python RUN pip3 install python-docx
```

Let ’ s Get Started!

Now that you have installed the module, let ’ s create a very simple document. It will be a skeleton for a more complex document afterward.
To create an empty file, we use the following code:

```
ClassMethod CreateDocument(path as %String) [ Language = python ]
{

from docx import Document

document = Document()

document.save(path + "document.docx")

}
```

If you execute it, you ’ ll see that it will create an empty document.

Quick note:

Since **python-docx** creates .docx files, you don ’ t have to use MS Word. BothGoogle Docs and LibreOffice are free alternatives that support .docx files, and they are just as good as the MS Office suite.

After creating an empty document, let ’ s add headings and subheadings to its structure our document. To do that we will use the **addheading()** method.

The **addheading()** method accepts two arguments: the first one is the text, and the second one determines the style by *level*. There are 10 levels: 0 is the biggest one, while 9 is the smallest.

```
ClassMethod CreateDocument(path as %String) [ Language = python ]
{

from docx import Document

document = Document()
document.add_heading('The Lord of the Donuts', 0)
document.add_heading('best donut from middle earth', 2)
document.save(path + "document.docx")

}
```

The document.docx until now should look like this:

Adding Images

To add an image to a word document we will use **addpicture()** method.

The path to the image is passed as the first parameter.

```
ClassMethod Create(path As %String) [ Language = python ]
{

from docx import Document
document = Document()

document.add_heading('The Lord of the Donuts', 0)

document.add_heading('best donut from middle earth', 2)

document.add_picture(path + "donut.png")

document.save(path + "document.docx")

}
```

Also, you can specify the width and height of the image. To resize the image, you might have to recalculate the values using dots-per-inch(dpi) format. However, python-docx has an auxiliary class called docx.shared that helps you to convert inches, centimeters, and millimeters.

```
document.add_picture(path + "donut.png", width=docx.shared.Inches(5), height=docx.sha
red.Inches(7))
```

Writing Paragraphs

To begin writing paragraphs, you can use the **add_paragraph()** method, as we did with headings.

```
ClassMethod Create(path As %String) [ Language = python ]
{
    from docx import Document

    document = Document()
    document.add_heading('The Lord of the Donuts', 0)
    document.add_heading('best donut from middle earth', 2)
    document.add_picture('/irisrun/repo/assets/donut.png')

    document.add_paragraph('One donut to rule them all \nOne donut to find them \nOne
 donut to bring them all \nAnd in the darkness bind them')

    document.save(path + "document.docx")

}
```

Great, but how do we change the font size, style, and color?

I ' m glad you asked. We are going to need to add a run.

Run in Word is a sequence of characters, where all of them share the same character formatting.

So, let ' s add our paragraph object to a variable, and we add a run to that variable using**add_run().**

The first parameter is the text, and the second parameter is the style.

You can use such styles as bold, italic, subscript, underline, strike, double_strike, emboss, etc.

Let ' s break our paragraph into parts and apply a bold font to all words " One " , whereas for " find them " we will choose italics.

```
ClassMethod Create(path As %String) [ Language = python ]
{
    from docx import Document

    document = Document()
    document.add_heading('The Lord of the Donuts', 0)
    document.add_heading('best donut from middle earth', 2)
    document.add_picture(path + 'donut.png')

    paragraph = document.add_paragraph('One donut to rule them all, \n')
    paragraph.add_run('One').bold = True
    paragraph.add_run(' donut to ')
    paragraph.add_run('find them,').italic = True
    paragraph.add_run(' One').bold = True
    paragraph.add_run(' donut to bring them all, \nAnd with sugar bind them\n')

    document.save(path + "document.docx")
}
```

Great! As you can see **Run**s are small blocks to stylize.
To change the font, you just need to modify the **font.name** property from the run.

```
ClassMethod Create(path As %String) [ Language = python ]
{
    from docx import Document

    document = Document()
    document.add_heading('The Lord of the Donuts', 0)
    document.add_heading('best donut from middle earth', 2)
    document.add_picture(path + 'donut.png')

    paragraph = document.add_paragraph('One donut to rule them all, \n')
    paragraph.add_run('One').bold = True
    paragraph.add_run(' donut to ')
    paragraph.add_run('find them,').italic = True
    paragraph.add_run(' One').bold = True
    paragraph.add_run(' donut to bring them all, \nAnd with sugar bind them\n')

    run = paragraph.add_run('In the Land of Sprinkles where the sweetness lie')
    run.font.name = 'Aharoni'

    document.add_heading('How to order', 2)
    document.add_paragraph('Start with a Coating,', style='List Number')
    document.add_paragraph('Pick a Topping,', style='List Number')
    document.add_paragraph('Choose a Drizzle', style='List Number')

    document.save(path + "document.docx")
}
```

As you can see, the last line of the paragraph is written in a different font. Great, but there is a better way to improve the paragraph. All you need is to create a custom font style. To specify the style type, we need to import the WD_STYLE_TYPE enum from python-docx. Also, remember to import “ Pt ” class from shared to use it for the font size. After that, you should call the **add_style** and give a name to this new style. When adding the run to the paragraph, include the custom style name as the second parameter.

```
ClassMethod Create(path As %String) [ Language = python ]
{
    from docx import Document
    from docx.enum.style import WD_STYLE_TYPE
    from docx.shared import Pt
    from docx.enum.text import WD_ALIGN_PARAGRAPH

    document = Document()
    document.add_heading('The Lord of the Donuts', 0)
    document.add_heading('best donut from middle earth', 2)
    document.add_picture(path + 'donut.png')

    paragraph = document.add_paragraph('One donut to rule them all, \n')
    paragraph.add_run('One').bold = True
    paragraph.add_run(' donut to ')
    paragraph.add_run('find them,').italic = True
    paragraph.add_run(' One').bold = True
    paragraph.add_run(' donut to bring them all, \nAnd with sugar bind them\n')

    font_styles = document.styles
    font_charstyle = font_styles.add_style('customStyle', WD_STYLE_TYPE.CHARACTER)
    font_object = font_charstyle.font
    font_object.size = Pt(15)
    font_object.name = 'Book Antiqua'
    run = paragraph.add_run('In the Land of Sprinkles where the sweetness lie','custo
mStyle')

    document.save(path + "document.docx")
}
```

At this point the document should look like this:

Creating Lists

If you want to create an ordered list, just add multiple paragraphs and select the style for the **List Number** as we did before.

```
        font_styles = document.styles
    font_charstyle = font_styles.add_style('customStyle', WD_STYLE_TYPE.CHARACTER)
    font_object = font_charstyle.font
    font_object.size = Pt(15)
    font_object.name = 'Book Antiqua'
    run = paragraph.add_run('In the Land of Sprinkles where the sweetness lie','custo
mStyle')

    document.add_heading('How to order', 2)
    document.add_paragraph('Start with a Coating,', style='List Number')
    document.add_paragraph('Pick a Topping,', style='List Number')
    document.add_paragraph('Choose a Drizzle', style='List Number')

    document.save(path + "document.docx")
```

It ’ s the same thing as Unordered Lists, but the style must use here will be **List Bullet**

```
        font_styles = document.styles
    font_charstyle = font_styles.add_style('customStyle', WD_STYLE_TYPE.CHARACTER)
    font_object = font_charstyle.font
    font_object.size = Pt(15)
    font_object.name = 'Book Antiqua'
    run = paragraph.add_run('In the Land of Sprinkles where the sweetness lie','custo
mStyle')

    document.add_heading('How to order', 2)
    document.add_paragraph('Start with a Coating,', style='List Number')
    document.add_paragraph('Powndered Sugar', style='List Bullet')
    document.add_paragraph('Glazed', style='List Bullet')
    document.add_paragraph('Chocolate Icing', style='List Bullet')

    document.add_paragraph('Pick a Topping,', style='List Number')
    document.add_paragraph('Sprinkles Rainbow', style='List Bullet')
    document.add_paragraph('Chopped Peanuts', style='List Bullet')

    document.add_paragraph('Choose a Drizzle', style='List Number')
    document.add_paragraph('Hot fudge', style='List Bullet')
    document.add_paragraph('Marshmalow', style='List Bullet')
    document.add_paragraph('Salted Caramel', style='List Bullet')

    document.save(path + "document.docx")
```

Before moving on, we will require a fast refactoring of the code to make it look a little bit more "**pythonish**".

```
        font_styles = document.styles
    font_charstyle = font_styles.add_style('customStyle', WD_STYLE_TYPE.CHARACTER)
    font_object = font_charstyle.font
    font_object.size = Pt(15)
    run = paragraph.add_run('In the Land of Sprinkles where the sweetness lie','custo
mStyle')
    run.font.name = 'Book Antiqua'

    steps = dict(coating = ['Powndered Sugar', 'Glazed', 'Chocolate Icing'],
        topping = ['Sprinkles Rainbow','Chopped Peanuts'],
        drizzle = ['Hot fudge', 'Marshmalow', 'Salted Caramel']
    )

    document.add_heading('How to order', 2)
    document.add_paragraph('Start with a Coating,', style='List Number')
    for coat in steps['coating']:
        document.add_paragraph(coat, style='List Bullet')

    document.add_paragraph('Pick a Topping,', style='List Number')
    for top in steps['topping']:
        document.add_paragraph(top, style='List Bullet')

    document.add_paragraph('Choose a Drizzle', style='List Number')
    for drizzle in steps['drizzle']:
        document.add_paragraph(drizzle, style='List Bullet')

    document.save(path + "document.docx")
```

With this latest modification, we can now see how the document has changed so far.

Working with tables

Working with tables is a little bit different than everything else we have done until now.
Before we start working on tables, let ’ s get the data.

Just import IRIS and pandas to the method and execute a simple SELECT.

```
from docx import Document
from docx.enum.style import WD_STYLE_TYPE
from docx.shared import Pt
import iris
import pandas as pd
#
#
#
rs = iris.sql.exec('SELECT id, img, name, "desc" FROM dc_docx_sample.Donuts')
```

Now we need to add a table to the document by calling a method **add_table()** and defining how many columns we will require with the cols parameter and adding only one row for the header.

You can define a style for the table by setting style property. For this example, let ’ s add a grid with the ‘**Table Grid** ’ style.

```
        table = document.add_table(rows=1, cols=4)
    table.style = "Table Grid"
```

The table object has an array of rows with a sequence of table cells in each row. For each cell, we can merge them with other cells or add paragraphs with runs as mentioned before. We can also add another table (subtable), or simply add text. Let ’ s populate the header row:

```
        table = document.add_table(rows=1, cols=4)
    table.style = "Table Grid"
    heading_row = table.rows[0].cells
    heading_row[0].text = 'ID'
    heading_row[1].text = 'Image'
    heading_row[2].text = 'Product'
    heading_row[3].text = 'Description'
```

Finally, using a “ for loop ” , populate each row with the data that we get. The trick is to add a paragraph and add_picture for the img property on the Donuts example class.

```
        rs = iris.sql.exec('SELECT id, img, name, "desc" FROM dc_docx_sample.Donuts')
    df = rs.dataframe()
    for idx in df.index:
        row_cells = table.add_row().cells
        row_cells[0].text = str(df['id'][idx])
        image_cell = row_cells[1].add_paragraph('')
        image_cell.add_run().add_picture(path + 'img/' + df['img'][idx])
        row_cells[2].text = str(df['name'][idx])
        row_cells[3].text = str(df['desc'][idx])
```

You can now output the document with the following table:

Look's very good!

Reading MS Word Documents

Last but not least, it ’ s time to read some MS Word files!

In the assets folder on the repository, you can find three docx files. The first one, called dogstale.docx, is just a part of chapter one of a short story, written by Mark Twain.

In the same way, we instantiate the object in the document when creating it. We are going to do it now. The only difference will be the file path and name on the constructor argument.

The paragraphs property is an array of paragraphs in the file in the document order.

Let ’ s fetch all the paragraphs from the dogstale.docx and then display the total number of paragraphs:

```
ClassMethod DogsTale(path As %String) [ Language = python ]
{
    from docx import Document
    doc = Document(path + "dogs_tale.docx")

    print(len(doc.paragraphs))
}
```

Nice!

As we have seen before, a Paragraph has a collection of runs. So, let ’ s dissect the third paragraph of this file:

```
ClassMethod DogsTale(path As %String) [ Language = python ]
{
    from docx import Document
    doc = Document(path + "dogs_tale.docx")

    print(len(doc.paragraphs))

    p = doc.paragraphs[2]
    for run in p.runs:
        aux = ''
        if (run.bold): aux = '// has Bold style'
        if (run.italic): aux = '// has Italic style'
        print(run.text, aux + '\n')
}
```

Note that we can get direct access to the third paragraph by index (remember that Python array of elements starts at 0)

Finally, we can get the complete text of the document and print it as an array:

```
ClassMethod DogsTale(path As %String) [ Language = python ]
{
    from docx import Document
    doc = Document(path + "dogs_tale.docx")

    print(len(doc.paragraphs))

    p = doc.paragraphs[2]
    for run in p.runs:
        aux = ''
        if (run.bold): aux = '// has Bold style'
        if (run.italic): aux = '// has Italic style'
        print(run.text, aux + '\n')

    fulltext = []
    for p in doc.paragraphs:
        fulltext.append(p.text)
    print(fulltext)
}
```

The last thing we can do to spice it up a little bit is to read the table on order.docx

```
ClassMethod Read(path As %String) [ Language = python ]
{
    from docx import Document
    doc = Document(path + "order.docx")

    fulltext = []
    for p in doc.paragraphs:
        fulltext.append(p.text)
    print(fulltext)

    tables = doc.tables
    data = {}
    cols = {}
    for table in tables:
        key = None
        for i, row in enumerate(table.rows):
            for col, cell in enumerate(row.cells):
                text = cell.text
                if i == 0:
                    cols[col] = text
                    data[text] = []
                    continue
                data[cols[col]].append(text)
    print(data)
}
```

Let ’ s make a template

It would be a fun exercise to practice everything we have learned so far by making a template. A template is a document that contains a "boilerplate" text that does not change. The input parameter for the information you want to replace will be in the following format: {{input_parameter}}

In the assets folder on the repository, you can find the third file called template.docx

As we did before, let's instantiate the document by passing the file with the path in the constructor argument.

Then, let's define the dictionary, where we will select the parameter names and values.

```
ClassMethod Template(path As %String) [ Language = python ]
{
    from docx import Document


    dict = {"company_name": "Mordor co",
        "name": "Saruman the White",
        "employee_name": "Radagast the Brown",
        "employee_job_title": "magician",
        "job_title": "wizard",
        "quality": "wise",
        "recipient_name": "Gandalf the Gray"
        }
    regex1 = re.compile(r"\{\{(.*?)\}\}")
    doc = Document(path + "template.docx")
}
```

Now create a Python function to search a pattern into paragraphs and runs to replace it with the dictionary when found.

```
ClassMethod Template(path As %String) [ Language = python ]
{
    import re
    from docx import Document


    def docx_replace_regex(doc_obj, regex, dict):
        for p in doc_obj.paragraphs:
            for r in p.runs:
                arr = regex.findall(r.text)
                for word in arr:
                    r.text = r.text.replace('{{'+word+'}}', dict[word])

    dict = {"company_name": "Mordor co",
        "name": "Saruman the White",
        "employee_name": "Radagast the Brown",
        "employee_job_title": "magician",
        "job_title": "wizard",
        "quality": "wise",
        "recipient_name": "Gandalf the Gray"
        }
    regex1 = re.compile(r"\{\{(.*?)\}\}")
    doc = Document(path + "template.docx")
    docx_replace_regex(doc, regex1, dict)
    doc.save(path + 'result1.docx')
}
```

Let me add a quick explanation to those who are new to Python. This little 'r' before the quote means "raw string literal", in which a backslash, for example, means "just a backslash". You don ’ t need any escape sequences to represent a new line, tabs, etc.

Conclusion

The article covers the process of writing MS Word files, adding a paragraph, runs, headers, and images to MS Word documents. Finally, how to read paragraphs and runs has been explained too.

If you want to go deep, I strongly recommend taking a look at the official documentation of python-docx

Please do reach out to me with all the suggestions, comments, and feedback you may have, I'll be glad to answer you.

Github link to this code

Thanks for reading.

#Embedded Python #InterSystems IRIS

Source URL:https://community.intersystems.com/post/create-and-read-ms-word-documents-epython