Paper CT12

Take Notepad++ to the Next Level with the Python Script Plugin

John Bielinski, Galderma, La Tour-de-Peilz, Switzerland

ABSTRACT

More and more SAS® programmers are discovering the advantages of incorporating the Notepad++ text editor into their workflow. This simple editor offers a huge range of functionalities that can aid in developing and debugging programs. You can expand its capabilities even more with the Python Script plug-in. This tool puts the full power of the Python programming language at your disposal, enabling you to customise Notepad++ as you see fit. There is almost no limit to what you can do - automating complex tasks, creating your own menu items, interacting with external files and programs, and much more besides. This paper will give a brief overview of the use of this tool, introducing some of the basic concepts behind programming in Python within this environment. We will then demonstrate a few simple scripts in order to give a taste of the wide range of tasks that can be performed.

INTRODUCTION

In addition to the built-in SAS editor, it can often be useful to use a dedicated text editor to perform certain tasks while developing and debugging SAS programs. One of the most widely-used text editors is Notepad++ which offers a wealth of useful features. Its already impressive functionality can be further expanded by installing third-party plugins.

The Python Script plugin allows the user to take advantage of the powerful Python language to write scripts which can be run directly within Notepad++. Its benefit lies in the fact that, through the use of special objects and methods, almost any action that can be performed with the keyboard or mouse can be incorporated into a script, thereby giving the user almost unlimited scope to expand and customise the capabilities of their Notepad++ session

This paper will provide an overview on the installation and use of this plugin. While some familiarity with the Notepad++ editor and the Python language is beneficial, the paper will not go deeply into either of these. The focus will instead be on the practicalities of installing and using the plugin, and on demonstrating the interaction between Notepad++ and Python by means of a few simple examples.

OVERVIEW OF SOFTWARE

Notepad++

Notepad++ is a free, open-source text editor for Windows developed in 2003 by Don Ho [1].
Used alongside the SAS editor, it can be an extremely useful tool to help with developing and debugging SAS programs. See reference [2] for a detailed overview of several of its features, such as multi-line editing and regex-enabled search-and-replace, that can be particularly valuable to SAS programmers. It can recognise keywords and syntax for over 70 programming languages; although SAS is not one of these, it can easily be added using the User Defined Language interface [3,4].

Python

Python probably does not need much introduction. Although almost 30 years old, its popularity has exploded over the past 4 or 5 years, particularly in the fields of data analysis and machine learning. At its heart however, it is quite a simple language with an intuitive, easy-to-read syntax which makes it ideal for scripting and for automating routine tasks [5]. Sometimes referred to as the "Swiss Army knife" of coding languages, it has a vast built-in library of functions and methods to perform almost any task imaginable.

Python Script plugin

Developed by Dave Brotherstone in 2010, this is a standard Notepad++ plugin which allows the user to develop and run Python programs which can interact directly with their Notepad++ session.
At the time of writing, the current version is v1.5.4.0. [6]
This version uses Python 2.7.18 rather than the now ubiquitous Python 3. (An alpha version compatible with Python 3 is available though the developer's github repository. However, downloading this version is not recommended.)

GETTING STARTED

Installation

The best way to install is via the Notepad++ menu:

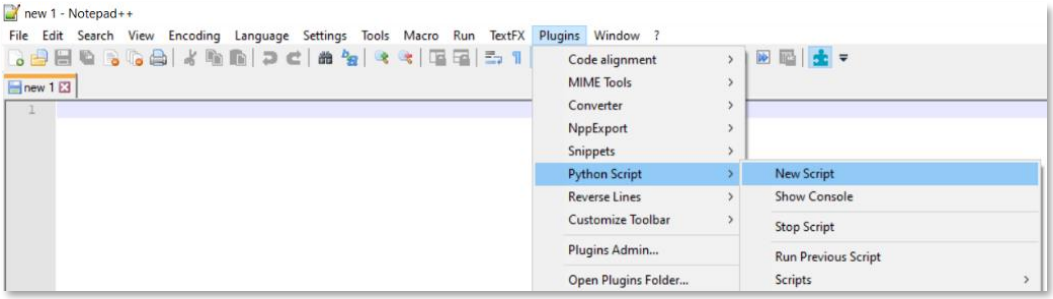
1. Select **Plugins** > **Plugins Admin…** from the menu
2. Look for Python Script under the **'Available'** tab and tick the checkbox.

3. Select **Install**
4. The plugin should be available in the Plugins tab the next time you restart Notepad++

Documentation
Once the plugin is installed, the best source of documentation is the 'Context-help' file which can be accessed directly within Notepad++ itself (**Plugins>Python Script>Context-help**)

CREATING AND RUNNING SCRIPTS

Creating a script
A new script can be created via the sequence of menu commands as shown:



A new Python (.py) file will open in Notepad++ to which any standard Python commands can be included. However, in order to fully interact with our Notepad++ session, we need to use some additional objects and methods that are available to us. These are the `notepad`, `editor` and `console` objects.

The notepad object
This refers to the Notepad++ app itself. You can use it for things like opening, saving and closing files or getting information about the current Notepad++ session

The editor object
Refers to the file in the current editor window. Generally any commands related to editing files can be performed via methods on the `editor` object.

Most scripts will contain several commands relating to these two objects. Some examples of common editing tasks are:

`notepad.new()`	Open a new .txt file in a new tab
`notepad.getCurrentFilename()`	Get the full path of the file in the currently active editor window
`editor.getText()` `editor.getSelText()`	Read the entire contents of the file into a Python string variable Read the currently selected text into a Python string variable

`editor.documentStart()`
`editor.documentEnd()`
`editor.gotoLine(`*`n`*`)`
	Move the cursor to the start of the document.
	Move the cursor to the end of the document
	Move the cursor to the specified line (N.B. line references are zero-indexed, `editor.gotoLine(0)` will go to the first line in the document)

`editor.addText(`*`text`*`)` `editor.appendText(`*`text`*`)`	Add text at the current cursor position Add text to the end of the document without moving the cursor.
`editor.replaceSel(`*`text`*`)` `editor.setText(`*`text`*`)`	Replace the current selection with the specified text Replace the entire contents of the file with the specified text

N.B. Remember that names in Python are case-sensitive. Most `editor` and `notepad` methods follow a `lowerCamelCase` naming convention.

There are literally hundreds of methods to perform more or less any imaginable task within your Notepad++ session. The best way to discover them is to refer to the documentation (see above) and try them out. Be aware the documentation for some of the methods is quite sparse or non-existent, so you will probably need to spend some time experimenting to see how they work.
CAUTION- since things like deleting text, saving and closing files, etc. can all be done with just a few lines of code, it's advisable to practise in a safe setting where you can't do too much damage to any important files.

The console object
This is of limited use, and it mostly relates to the Console window which concerns the Python session itself. Error messages appear in the console window so you will definitely find yourself using it while developing and debugging your scripts.
Some commands you may wish to use are:
`console.show(), console.hide()` – display or hide the console window.
`console.clear()` – clear the contents of the console window
`console.write(`*`text`*`)` – write text to the console window (useful for debugging)

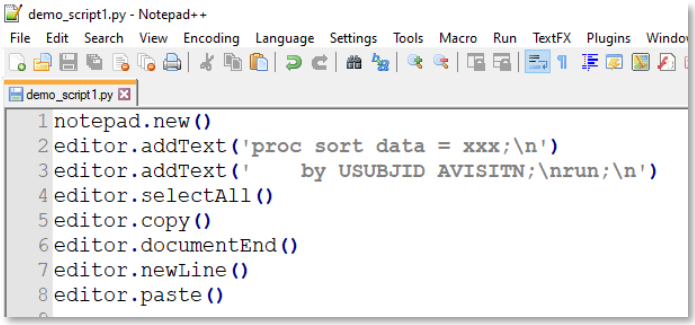
Running a script
Once saved, a script is immediately available to be used at any time – there is no need to submit it or compile it, nor does it need to be open.
To run a script, you can just select it from the Notepad++ like any other menu item:
Plugins>Python Script>Scripts>…
and then select the script you want to run from the list available.

The scripts themselves are located in
C:\Users\<user name>\AppData\Roaming\Notepad++\plugins\config\PythonScript\scripts
You can create subfolders within this area, allowing you to sort and group your scripts by category if you wish.

SOME EXAMPLES

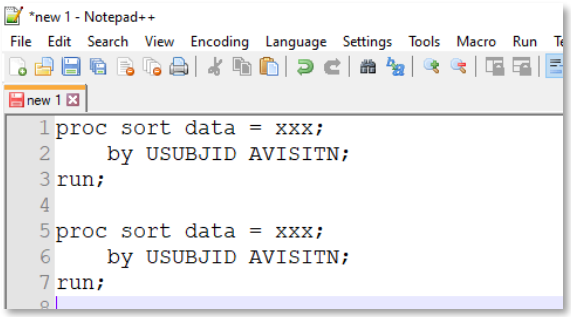
Below are some examples to demonstrate how it all works in practice. The main purpose of these demo scripts is to highlight certain concepts rather than to solve real-world problems, so they may seem a bit contrived. Moreover, I don't claim to be an experienced Python programmer, so these scripts should certainly not be regarded as the most elegant or efficient way of doing things. Hopefully, however, they give an idea of the great versatility that these tools offer, and the range of tasks that can be performed.

Example 1: Automating a sequence of editing commands.
Here is a basic script consisting entirely of `notepad` and `editor` commands which will automatically create a new text file containing some generic proc sort statements which could be used as a starting point for a SAS program.
All of the commands correspond to standard editing tasks, and most should be fairly self-explanatory.



1. Create a new text file
2-3. Add some text
4-5. Copy entire contents to clipboard
6. Move cursor to end of document
7. Add a blank line
8. Paste the clipboard contents

If we run the script via the Plugins>Python Script menu, the newly created file will look like this:

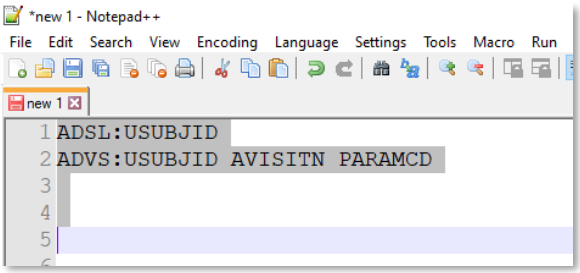


Now of course we could easily achieve the same thing simply by using the Macro function in Notepad++ to record the sequence of commands, or we could create an Abbreviation within the SAS editor itself.
However, this simple example helps to demonstrate how we can use a script to control our Notepad++ session right down to the level of individual keystrokes or mouse clicks. The real power of this environment lies in the ability to incorporate these editing tasks into a fully-functioning Python program.

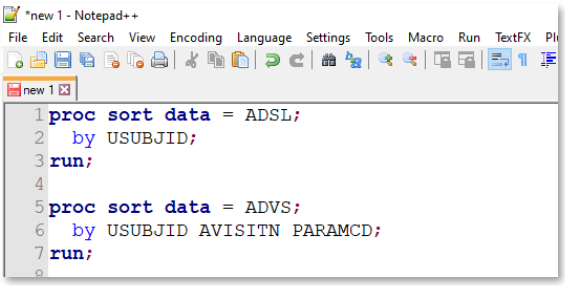
Example 2: Converting lines of text into SAS code
The next script builds on the simple one shown above, but is a bit more interactive. It works by reading in some basic text from the editor window which is then converted it into valid SAS code.

The figure below demonstrates how the script works. In the left window the user has created a new text file and added some lines of text in the format DATASET:VARIABLE(S). They can then select these lines with the mouse and run the script via the Notepad++ menu. Once run, the text is automatically 'translated' into SAS Proc Sort statements as shown in the right hand window.

Before running script



After running script



The script itself looks like this:

```
1  def text2Sort_main():
2
3  ## Read in selected text, remove leading and trailing blanks and convert to list
4        inputText = editor.getSelText()
5        lineList  = inputText.strip().splitlines()
6
7  ## Create SAS code based on the contents of each line
8        sasCode = ''
9        for ds,sortvars in [(line.strip().split(':')) for line in lineList]:
10          sasCode += 'proc sort data = {0};\n'.format(ds)
11          sasCode += '  by {0};\n'.format(sortvars)
12          sasCode += 'run;\n\n'
13
14
15 ## Replace the selected text with the SAS code
16       editor.replaceSel(sasCode)
17
18 ## Set the programming language to SAS
19 ## (N.B. Will only work if you have added SAS as a User-Defined Language)
20       notepad.runMenuCommand('Language','SAS')
21
22
23 text2Sort_main()
```

Notice that the script has been created in the form of a *function* called `text2Sort_main()` which is then called at the very end of the script (line 23). There is no obligation to do this, nor is there any restriction on the name you choose for the function. However it is good practice, as it ensures that any variables created within the script are local in scope and cannot be inadvertently used by another script.

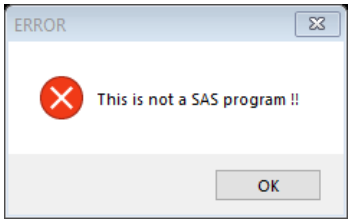
A more complex version of this script is shown in Appendix 1. This is designed to automatically generate SAS code to sort and merge datasets based on information provided by the user. The basic principle is the same; the user 'submits' a line of text by selecting it in the editor window and running the script. The user then provides additional information about the sorting/merging variables by means of the `notepad.prompt()` command, and the required SORT and MERGE lines of SAS code are generated.

Example 3: Summarising information from a SAS program

In addition to writing text to a file, we may sometimes want to obtain information about the contents of a document. This too can easily be done with a Python script.
Appendix 2 shows an example of such a script together with some sample input (a SAS program) and output (a pop-up screen showing summary information). In this example, the script provides a count of all the PROC statements found in the program. Such a script might be useful if you want to get a quick overview of a long SAS program without having to manually scan through it.

When run, the script first checks that the current file is indeed a SAS program.
If the file does not have a '.sas' extension, an error message is displayed and the script is aborted.

```
progName = notepad.getCurrentFilename()
if not (progName.endswith('.sas')):
    notepad.messageBox( 'This is not a SAS program !!'
                       ,'ERROR'
                       , MESSAGEBOXFLAGS.ICONERROR)
    return None
```



When the message box containing the summary information is displayed it gives the user the option to write this information to a file. This is done by using the optional MESSAGEBOXFLAGS.YESNO flag, The default response is set to 'No' by the MESSAGEBOXFLAGS.DEFBUTTON2 flag.

```
newYN = notepad.messageBox(msgBoxText
                           ,'Report'
                           ,MESSAGEBOXFLAGS.YESNO|MESSAGEBOXFLAGS.DEFBUTTON2)

     ## If user selects 'Yes', open new file and write summary text
        if newYN == MESSAGEBOXFLAGS.RESULTYES:
            notepad.new()
            editor.addText(summTxt)
```

As an aside, the command `from __future__ import division` in the first line of the program has been added to allow floating point division to be performed (line 30). In Python 2.7.x, if two integers are divided the result is truncated so that only the integer part is returned as demonstrated below, so this line must be added to simulate the behaviour of Python 3.

```
>>> 3/2
1
>>> float(3)/2
1.5
>>> from __future__ import division
>>> 3/2
1.5
```

TIPS

Hopefully these examples have given some idea of a few of the ways Python scripts can be incorporated into your Notepad++ session. Below are a few things to consider before starting on a new script.

Do you even need a script? - Before you even start writing your script, you should first check if there is already a built-in menu command in Notepad++ that can do some of the work for you. Failing that, there may well be a plugin that you could use. Some plugins that can be very useful when editing programs are "TextTX, "Code Alignment" and "Snippets". Note that it is possible to include plugin commands within a Python script. For example, if you have the Code Alignment plugin installed, you could execute its 'Align by equals' command with the following line in your script:

```
notepad.runPluginCommand('Code alignment', 'align by equals')
```

Keep your scripts simple – It's worth taking a step back to look at why we want to create a Python script in the first place. Presumably our main goal is to produce a working SAS program, and we are just using Notepad++ as a tool to assist us in this task. By adding Python scripts into the mix we now have two sets of programs to maintain, and we need to decide whether the additional time spent writing and debugging our scripts is a worthwhile investment.

Keep automation to a minimum – As we've seen, more or less any Notepad++ command can be incorporated into a script. It is very easy to include code to, say, automatically save a file [`notepad.save()`] or to close it [`notepad.close()`]. A script that contained these commands would silently execute them without any interaction from the user, which may not always be desirable. Moreover, since a script can be run with just a couple of mouse clicks, it is just as easy to accidentally run the *wrong* script as it is to run the right one. For these reasons, any scripts that make major changes to a file should be avoided, or at least should only be used if the changes can easily be undone.

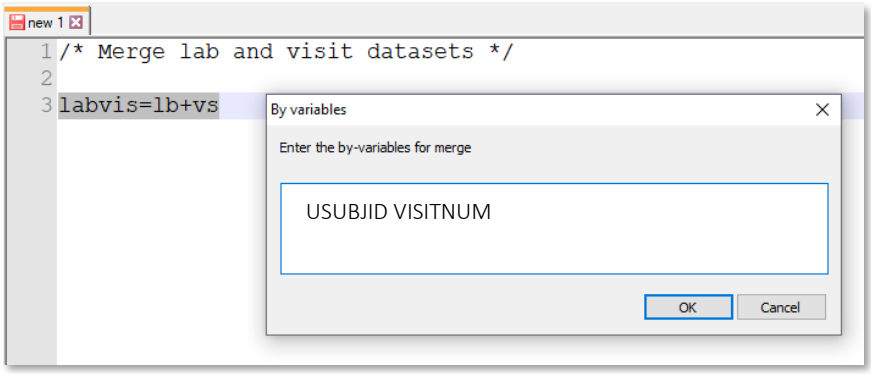
CONCLUSION

Notepad++ is a great addition to any SAS programmer's toolkit. The Python Script plugin can be easily and seamlessly integrated into your Notepad++ environment, allowing you to customise and expand its already impressive range of features almost without limit.

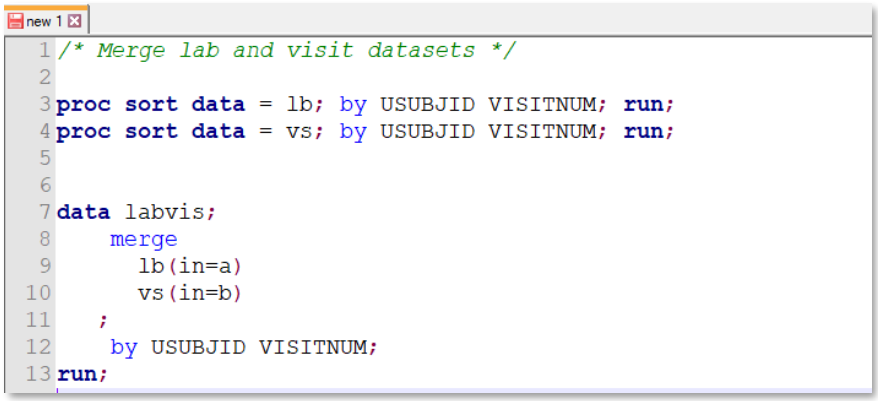
APPENDIX 1. SCRIPT TO CONVERT TEXT INTO SAS SORT/MERGE STATEMENTS

```
def text2Merge_main():

## Read in text in the form "XXX=a+b+..." where a,b,.. are the datasets to be
## merged (any number are allowed) and XXX is the dataset to be created.
## By-variables for sorting/merging are provided by user input.

    inText=editor.getSelText().strip()

    newDS  = inText.split('=')[0]
    inList = inText.split('=')[1].split('+')


## Get sorting/merging variable(s)
    byVars=notepad.prompt('Enter the by-variables for merge','By variables',' ')

## Generate SAS code
    sortText,mergeText,SASCode = ('','','')
    for i,ds in enumerate(inList):
        sortText += 'proc sort data = {0}; by {1}; run;\n'.format(ds,byVars)
        mergeText+= '\n      {0}(in={1})'.format(ds,chr(i+97))


    SASCode += sortText + '\n\n'
    SASCode += 'data ' + newDS + ';\n'
    SASCode += '    merge ' + mergeText + '\n   ;\n'
    SASCode += '    by ' + byVars + ';\n'
    SASCode += 'run;\n'


## Replace the selected text with the SAS code
    editor.replaceSel(SASCode)
    notepad.runMenuCommand('Language','SAS')


text2Merge_main()
```

1. User selects line of text, runs script and enters variable names into dialog box



2. Text is automatically replaced with SAS code

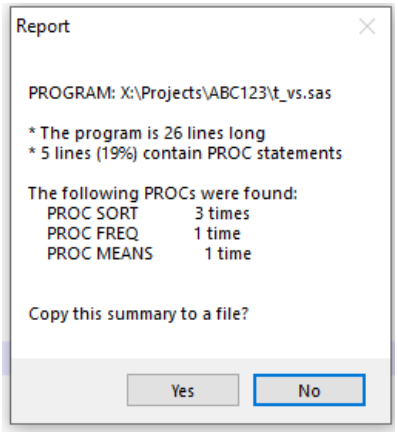


APPENDIX 2. SCRIPT TO SHOW SUMMARY INFORMATION FOR A SAS PROGRAM

```
from __future__ import division
import re

def countProcs_main():

## Check if current file is a SAS program. If not, show error message and exit
     progName=notepad.getCurrentFilename()
     if not(progName.endswith('.sas')) :
              notepad.messageBox( 'This is not a SAS program !!'
                                 ,'Error' ,MESSAGEBOXFLAGS.ICONERROR)
              return None


## Search for the word 'PROC' at the beginning of a line,
##  followed by one or more word characters.
     pattern=re.compile(r'^\s*\bPROC\b\s+(\w+)', re.IGNORECASE|re.MULTILINE)
     procsFound = pattern.findall(editor.getText())

## Count number of times each proc occurs
     procCount = dict()
     for p in [i.strip().upper() for i in procsFound]:
        procCount[p] = procCount.get(p,0) +1

## Get total number of lines in program, and total number of PROC statements
     lc,pc = editor.getLineCount(), sum(procCount.values())

## Create summary text
     summTxt  = 'PROGRAM: ' + progName + '\n\n'
     summTxt += '* The program is {0} lines long\n'.format(lc)
     summTxt += '* {0} lines ({1:.0%}) contain PROC statements'.format(pc,pc/lc)
     summTxt += '\n'*2
     summTxt += 'The following PROCs were found:\n'
     for k,v in sorted(procCount.items(),key=lambda item: item[1],reverse=True):
           summTxt += '    PROC {0:12} {1:4} time{2}\n'.format(k,v,'s'*(v>1))


## Display message box containing summary text, plus option to save to file
     msgBoxText = summTxt + ('\n'*2 + 'Copy this summary to a file?')
     newYN = notepad.messageBox(msgBoxText
                               ,'Report'
                               ,MESSAGEBOXFLAGS.YESNO|MESSAGEBOXFLAGS.DEFBUTTON2)

## If user selects 'Yes', open new file and write summary text
     if newYN == MESSAGEBOXFLAGS.RESULTYES:
         notepad.new()
         editor.addText(summTxt)

countProcs_main()
```

t_vs.sas

```
proc sort data = adsl; by STUDYID USUBJID; run;
proc sort data = advs; by STUDYID USUBJID; run;

data allvs;
    merge adsl(in=a) advs(in=b);
    by STUDYID USUBJID;
run;
Proc Sort data = allvs;
    by paramcd avisitn;
run;

PROC FREQ data = allvs;
    by paramcd avisitn avisit;
    tables avalcat1 / out= freq1;
run;


proc means data = allvs;
     var aval;
```



REFERENCES

1. Notepad++ homepage: https://notepad-plus-plus.org/

2. Purkess, S. “Improving your SAS code without SAS: Using Notepad++ to super-charge your coding”, PHUSE EU Connect, 2019, CT08
https://www.phusewiki.org/docs/2019%20Amsterdam/Papers_presentations/CT/CT%20Final%20Papers/CT08.pdf

3. Notepad++ user-defined language interface:
https://npp-user-manual.org/docs/user-defined-language-system/

4. Hemedinger, C. "Using Notepad++ as your SAS code editor"
https://blogs.sas.com/content/sasdummy/2017/08/25/npp-with-sas/

5. Sweigart, A. "Automate the Boring Stuff with Python"
https://automatetheboringstuff.com/

6. Python Script github page: https://github.com/bruderstein/PythonScript/releases/tag/v1.5.4

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
John Bielinski
Galderma SA
Rue d'Entre-deux-Villes, 10
1814 La Tour-de-Peilz
Switzerland

john.bielinski@galderma.com