Paper SM15

Automate your Metadata: Reading and Modifying Information from PDF Files using Python

Salomo Sparna, Student, Freiburg, Germany
Renate Scheiner-Sparna, Alira Health, München, Germany

ABSTRACT

eCRF forms are normally exported from the data capture systems in PDF, in this format they are annotated according to CDISC SDTM standard. The annotations are meta-information that can be further processed for programming specifications or other purposes. This presentation shows how to export the annotations to a pre-structured Microsoft Excel file using Python. In addition to exporting the annotation fields to Excel rows, information can be added, e.g. for standard variables that are always present in CDISC SDTM standard. Reading and modifying data from PDF files is also useful for cleaning messy files that contain more than one generation of annotations. Some older annotations are no more in text boxes but embedded in the PDF body. This can be harmonized using Python. Modified information can also be written back to the PDF file. The code will be available on GitHub (https://github.com/ssparna/PHUSE-EU-Connect-SM15)

INTRODUCTION

In this paper a concept for improving the handling of metadata will be presented. The general idea is to use a collection of Python programs for simple automations by processing annotations from PDF files. This is not a ready-for-use solution, but an effort to enrich metadata handling.

IMPORTANCE OF AUTOMATED METADATA PROCESSING IN CLINICAL STUDIES

In the process of a clinical study, transparent documentation and metadata are essential. The steps between the different levels of data need standardized and automized processes as far as possible. There are at least the following reasons for this:

1. Automized processes are less error prone. You are sure you don't miss any information, and you get exactly the variables with names and properties that are specified.
2. The amount of work can be controlled. Over time, lots of hours can be saved by any automated step.
3. Processes are transparent and better maintainable when they are managed by programs and not manually.

We are looking at a concept that covers the generation of metadata on CDISC SDTM data level. Coming from the raw data level, the process is:

- We have an empty version of the eCRF.
- This is annotated with the SDTM variables.
- From the SDTM variables, the SDTM programming specification is generated.

While the annotation is still a manual step, we want to show that it is possible to automate the following step. The information from the annotation text boxes can be read and further processed by a Python program.

Figure 1 illustrates the steps in the process and the step we want to improve.

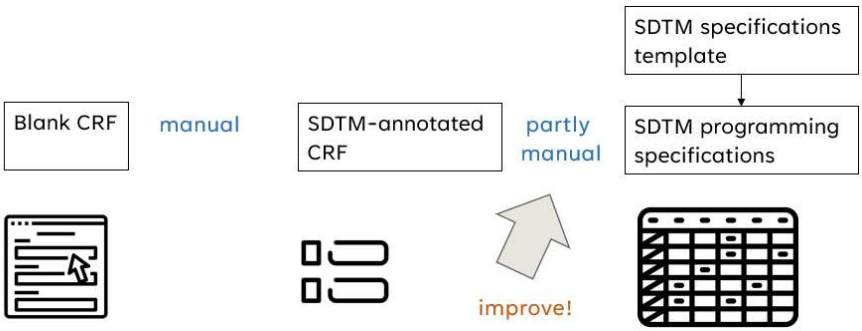

Figure 1 The process from the blank CRF to the specifications

TRANSFERRING METADATA FROM ECRF SDTM ANNOTATION TO SDTM SPECIFICATION

We need different kinds of metadata on CDISC SDTM data level. First, we need the eCRF annotations, which are done manually on a blank version of the eCRF. Second, we need a set of programming specifications for the SDTM domains. The annotations are part of the information that we need for the programming specification. The idea is to draw and re-use the metainformation from annotated eCRF.
The action point in the process is defined by the following: We have received the recent version of the eCRF which has been annotated with the SDTM variable names and destinations. Now it is the goal to generate SDTM programming specifications.
While the annotation of the blank eCRF is still a manual step, we want to show that it is possible to automate the following step. The information from the annotation text boxes can be read and further processed by a Python program.

There are many ways in the industry to produce a template of the SDTM specifications, mainly from the CDISC standards library. But the link to the information on the annotated eCRF is important. We need the information which of the required variables for an SDTM domain are already present in the eCRF and which are missing. We can check whether the annotations correspond to the requirements for standard variables and that the ones for non-standard variables are exactly as annotated.

INPUTS

Due to the nature of the data we are working with, we will use example inputs for an eCRF and an Excel template.

The CRF is an anonymized example of a common eCRF, though shorter than usual. An excerpt of it can be seen in figure 2.

The template, a snippet of which can be seen in figure 3, is a table with expected variables and datasets according to the CDISC STDM standard.

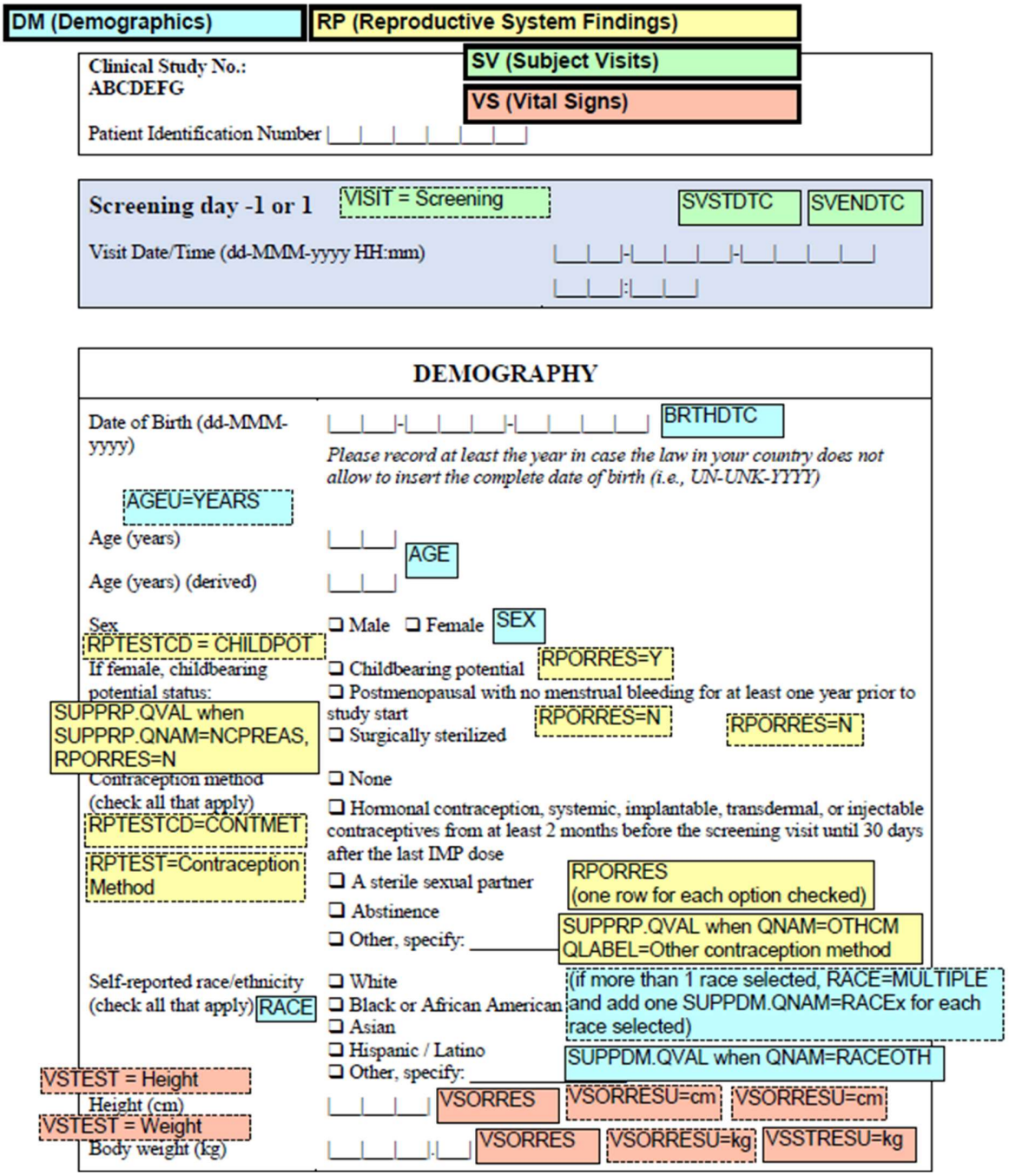

Figure 2 Annotated eCRF

	A	B	C	D	E	F	G	H	I	J
1	Order	Dataset	Variable	Label	DataType	Length	SignificantI	Format	Mandatory	Core
1754	1	DM	STUDYID	Study Identifier	text	200			Yes	Req
1755	2	DM	DOMAIN	Domain Abbreviation	text	200			Yes	Req
1756	3	DM	USUBJID	Unique Subject Identifier	text	200			Yes	Req
1757	4	DM	SUBJID	Subject Identifier for the Study	text	200			Yes	Req
1758	5	DM	RFSTDTC	Subject Reference Start Date/Time	datetime	19		ISO 8601	No	Exp
1759	6	DM	RFENDTC	Subject Reference End Date/Time	datetime	19		ISO 8601	No	Exp
1760	7	DM	RFXSTDTC	Date/Time of First Study Treatment	text	200			No	Exp
1761	8	DM	RFXENDTC	Date/Time of Last Study Treatment	text	200			No	Exp
1762	9	DM	RFCSTDTC	Date/Time of First Challenge Agent Admin	text	200			No	Perm
1763	10	DM	RFCENDTC	Date/Time of Last Challenge Agent Admin	text	200			No	Perm

Figure 3 Variables sheet of template

PREREQUISITES

As this is a Python program, Python 3.8 or later as well as the Packages “PyPDF2”, “openpyxl” and “FreeSimpleGUI” are required. Additionally using a virtual environment with the packages is advised.

RUNNING THE PROGRAM

1. Run main.py
2. Select input files
3. Select output folder
4. Select additional options
5. Press the EXPORT ANNOTATIONS button

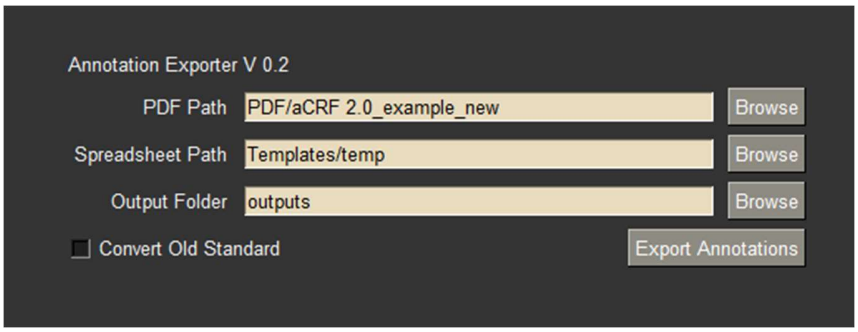

Figure 3 The program GUI

An example of the output where present variables have been marked can be seen below:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	Order	Dataset	Variable	Label	DataType	Length	Significant	Format	Mandatory	Core	Codelist	Origin	Pages	Method	Present in aCRF
1757	4	DM	SUBJID	Subject Identifier for the Study	text	200			Yes	Req		CRF	2		Present
1758	5	DM	RFSTDTC	Subject Reference Start Date/Time	datetime	19		ISO 8601	No	Exp					
1759	6	DM	RFENDTC	Subject Reference End Date/Time	datetime	19		ISO 8601	No	Exp					
1760	7	DM	RFXSTDTC	Date/Time of First Study Treatment	text	200			No	Exp					
1761	8	DM	RFXENDTC	Date/Time of Last Study Treatment	text	200			No	Exp					
1762	9	DM	RFCSTDTC	Date/Time of First Challenge Agent Admin	text	200			No	Perm					
1763	10	DM	RFCENDTC	Date/Time of Last Challenge Agent Admin	text	200			No	Perm					
1764	11	DM	RFICDTC	Date/Time of Informed Consent	text	200			No	Exp		CRF	2		Present

Figure 3 Output with variables marked Present in the “Present in aCRF” column

Adding variables which are not present in the template

If there are non-standard variables annotated in the CRF they will be appended at the bottom of the variable sheet. Additionally, supplementary datasets and variables will be automatically added with all standard variables in case the annotations show they are needed, as figure 4 shows.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Order	Dataset	Variable	Label	DataType	Length	Significant	Format	Mandatory	Core	Codelist	Origin	Pages	Method	Present in aCRF
1909		DS	DSSDTC			200						CRF	2		Present
1910		SUPPDS	QVAL			200						CRF	2		Present
1911		SUPPDS	QNAM			200						CRF	2		Present
1912		SUPPDS	QLABEL			200						CRF	2		Present
1913		SUPPDM	QVAL			200						CRF	3		Present
1914		SUPPDM	QNAM			200						CRF	3		Present
1915		SUPPDM	QLABEL			200						CRF	3		Present
1916		SUPPRP	QVAL			200						CRF	3		Present
1917		SUPPRP	QNAM			200						CRF	3		Present
1918		SUPPRP	QLABEL			200						CRF	3		Present
1919			RACE			200						CRF	3		Present

Figure 4 SUPP variables are appended to the bottom of the variables sheet just like not recognized variables.

PROGRAM FLOW

The program aims to output the modified template with all variables and datasets marked, as well as a csv version of those results for further processing.

FILE MAIN.PY

The file main.py contains mostly boilerplate for the GUI, parses the input strings and makes sure the file endings are correct. Additionally, it calls the function *export_annotations()* from the AnnotExport class with the input strings as parameters.

FILE PAGE.PY

This file defines the Page class which keeps track of the datasets found on each page with their respective page number.

FILE ANNOT_EXPORT.PY

This is the main file. Firstly, the template and PDF are opened by their respective packages, which creates python usable objects. We then loop through each page of the PDF, create a page object for it and look for any annotations

on the page. If there are none, we skip to the next page otherwise we loop through every annotation on the page. We verify it is a text annotation, that has a color and has some content. If it is identified as a dataset it is added to the list of datasets on the Page object.

The following assumptions are applied to identify domains and variables in the annotations:

- Domains are two alphabetical characters at the start of the annotation to the left of an open Bracket with the corresponding name. In case of the older standard the open bracket is replaced with an equal sign.
- Variables are any number of alphabetical characters to the left of an equal sign, sometimes separated by a separator.
- SUPP datasets are annotations that start with “SUPP”.
- Domain mapping is done by color, currently the color has to match exactly but adding some sort of distance calculation between colors would be thinkable.

Every annotation is then added to the template as either a dataset or variable, marked as present and if necessary appended at the end. Variables of the respective domains are checked against the variables sheet of the template and either marked as present or appended at the end. Variables without a corresponding domain annotation are also appended with an empty domain cell. Additionally variables have the page numbers on which they are found, the length, origin and domain added if they are not already present.

Annotations on domains are checked against the dataset sheet of the template, marked as present and if not found in the template appended at the end. Also, if requested at program start, the old CDISC standard syntax for annotations can be converted to the new one, generating an additional PDF file as an output.

Afterwards the csv is generated, and the modified template is saved to the output folder.

HOW TO PROCEED WITH THE METADATA

The following points are some consequences from the generated metadata information for the process.

1. Missing annotations in the eCRF can be identified. A Python program like this can be run iteratively until the eCRF annotation is completed.
2. The step from an SDTM template to the ready-to-use SDTM specification is more automated and transparent. The variable lines are clearly representing the annotations. Required variables can be easily identified if present or not to complete the annotation of the eCRF. This prevents later delays, as programming can start with specifications of good quality.
3. Non-standard variables are often defined for information that could also be mapped to permissible CDISC variables. When you have all the permissible variables in the SDTM specification template and add the annotation records, this can be easily cleaned manually by an experienced CDISC user and the annotations in the eCRF can be updated. This way, the data structure will adhere more to the CDISC standard.

Besides the outlined workflow, we found that the Python program can be useful for such as migrating eCRFs to a new version of the CDISC standard. Especially when the project is an integrated safety or efficacy summary (ISS, ISE) which often contains many eCRFs, it can be worth to set up a program and re-use it for all of them.

IDEAS FOR ADDITIONAL USAGE

In addition to the base use the program can easily be modified to take up additional tasks such checking whether all requested variables are present. Additionally annotations that are embedded within the PDF file itself can be converted to FreeText annotations.

These uses could be further expanded to include things like automating the creation of annotations from a list of variables and datasets, verifying the correctness of the current annotation syntax for a specific version of the standard or even further processing of the resulting excel file. Examples could be checking the font size, the presence of an equal sign or brackets being in the right place.

CONCLUSION

With this presentation we show only one small example for applying some features of Python to one defined PDF document. The general idea of reading information from a PDF document may lead to more ideas that could enhance the process of a clinical study.

Using these improvements will maximize worker productivity and leave them time to tend to other higher level tasks while reducing error prone, boring manual work.

REFERENCES

Packages

- Openpyxl (https://pypi.org/project/openpyxl/)
- PyPDF2 (https://pypi.org/project/PyPDF2/)

- FreeSimpleGUI (https://pypi.org/project/FreeSimpleGUI/)

ACKNOWLEDGMENTS

Many thanks to Jörg Sahlmann for helping me with his expertise in Python and helping me create the documentation for the Github page.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Salomo Sparna
Company: Student
Work Phone: +49 1575 2998 087
Email: SalomoSparna@gmail.com
Website: Not yet
Github: https://github.com/ssparna/PHUSE-EU-Connect-SM15