PharmaSUG 2020 - Paper SS-159

Automating CRF Annotation using Python

Hema Muthukumar, Fred Hutchinson Cancer Research Center;
Kobie O'Brian, Fred Hutchinson Cancer Research Center;
Julie Stofel, Fred Hutchinson Cancer Research Center;

ABSTRACT

When data are submitted to the FDA, an Annotated Case Report Form (aCRF) is to be provided as a PDF document, to help reviewers find the origin of data included in submitted datasets. These annotations should be simple, clean, and should respect appearance and format (color, font) recommendations. aCRFs are traditionally produced manually. This involves using a text editor in PDF and working variable by variable across many pages. This is a time-consuming process that can take many hours. In addition, maintaining consistency across pages requires substantial effort. This paper talks about an effective way to automate the entire aCRF process using Python. This approach automatically annotates the variables on the CRF next to their related questions on the appropriate pages.

INTRODUCTION

In this method we use the following:

- a Study Design Specification (SDS) which is an Excel workbook version of the study database build XML document exported from the Electronic Data Capture (EDC) system MediData Rave
- an SDTM mapping specification (also an Excel workbook) created by Data Standards analysts
- the study case report forms (CRF) in PDF format.

The output for this method is a Forms Data Format (FDF) file, which is called in automatically to annotate the CRF. This method significantly reduces the time and effort required to create an aCRF while eliminating inconsistent annotations. This approach is very useful since it does not require a global dictionary for annotations or a previously annotated PDF. This makes it very flexible, as it can be implemented to annotate CRFs for different types of trials and organizations.

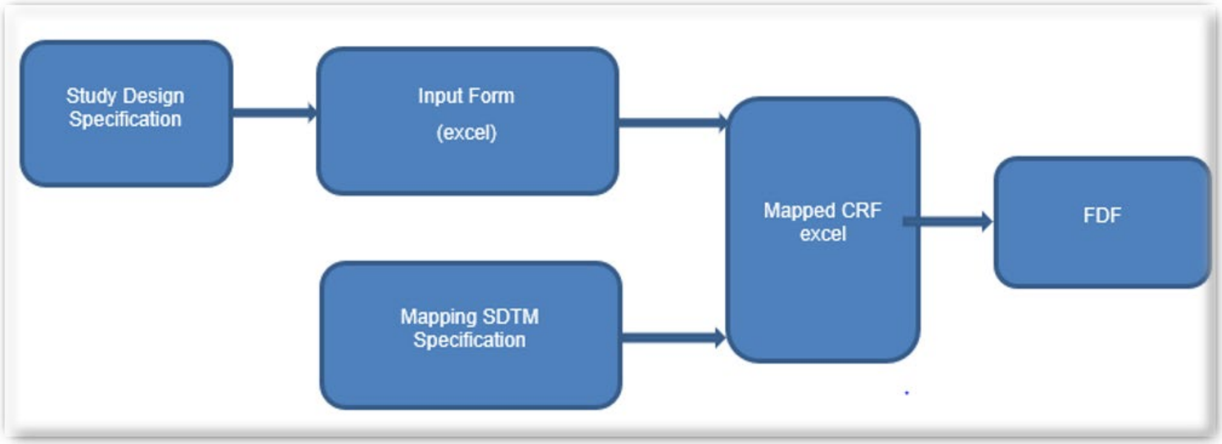


Figure 1. Process flow for the annotation. The Study Design Specification (SDS) is provided by the EDC system, where fields are listed in the order as they appear on the CRF pages. The Mapping SDTM Specification has one sheet per SDTM dataset. It includes multiple columns of metadata attributes, and it describes the data mapping required.

ADVANTAGES OF USING PYTHON

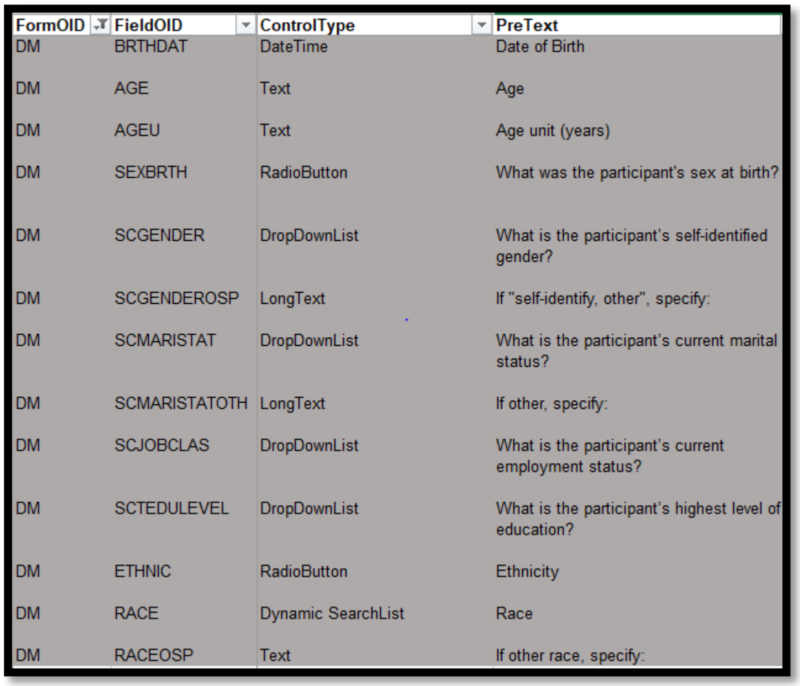
With its origin as an open source scripting language, Python usage has grown over time. Today, it supports many libraries and functions for almost any statistical operation or model building you may want to do. Since the introduction of pandas, it has become very powerful for operations on structured data. It is known for simplicity and clear syntax which in turn increases readability. The main challenge in automating annotation is getting the text location where the annotation needs to be placed. We overcame this challenge by using the Python module PDFquery, which provided the text location from the case report form, making it easy for us to place the annotation next to the required fields.

REQUIRED INPUTS FOR THE ANNOTATION

- Case Report Form (PDF)
- Study Design Specification (Excel)
- SDTM Mapping Specification (Excel)

STUDY DESIGN SPECIFICATION

The Study Design Specification is an output from the EDC system as an Excel spreadsheet. The columns which will be used for the automation are FormOID, FieldOID, ControlType and PreText. FormOID has the dataset name, FieldOID has the variable names, ControlType will specify type of control used for the field such as CheckBox, RadioButton etc., and PreText has the exact question as on the eCRF. ControlType plays an important role in deciding the placement of the annotation.

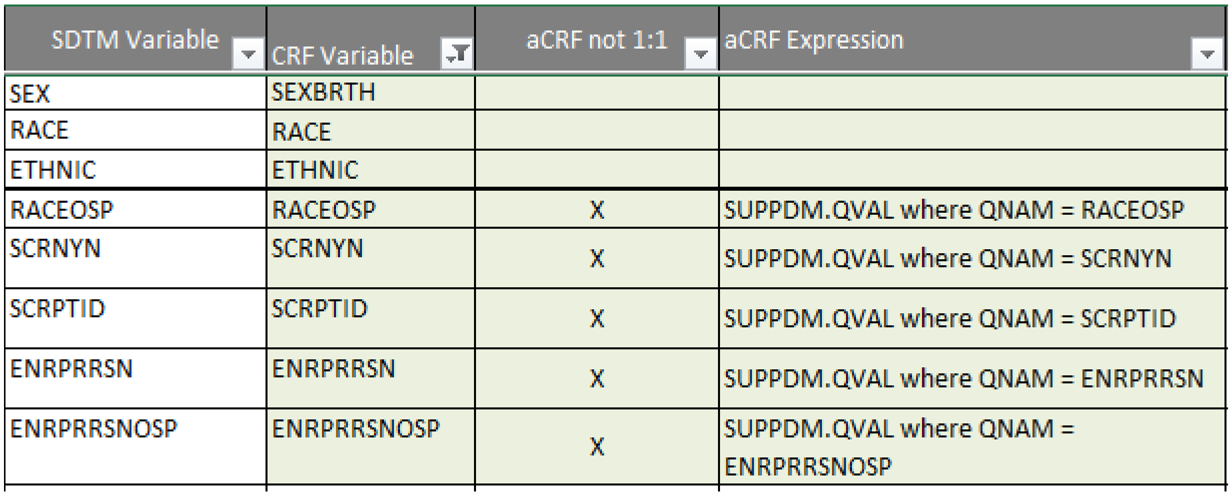


FormOID	FieldOID	ControlType	PreText
DM	BRTHDAT	DateTime	Date of Birth
DM	AGE	Text	Age
DM	AGEU	Text	Age unit (years)
DM	SEXBRTH	RadioButton	What was the participant's sex at birth?
DM	SCGENDER	DropDownList	What is the participant's self-identified gender?
DM	SCGENDEROSP	LongText	If "self-identify, other", specify:
DM	SCMARISTAT	DropDownList	What is the participant's current marital status?
DM	SCMARISTATOTH	LongText	If other, specify:
DM	SCJOBCLAS	DropDownList	What is the participant's current employment status?
DM	SCTEDULEVEL	DropDownList	What is the participant's highest level of education?
DM	ETHNIC	RadioButton	Ethnicity
DM	RACE	Dynamic SearchList	Race
DM	RACEOSP	Text	If other race, specify:

Figure 2. Study Design Specification for the Demographics Domain

SDTM MAPPING SPECIFICATION

The SDTM Mapping Specification is an excel file which has the information for converting the raw input data to the CDISC SDTM standard. The columns used for the automation are “Source dataset” which is as same as FormOID from SDS, “SDTM Variable” has the SDTM variable name used for annotation, “CRF Variable” which is same as FieldOID from SDS, “aCRF not 1:1” which indicates if the CRF should be annotated with the “aCRF Expression”.



SDTM Variable	CRF Variable	aCRF not 1:1	aCRF Expression
SEX	SEXBRTH		
RACE	RACE		
ETHNIC	ETHNIC		
RACEOSP	RACEOSP	X	SUPPDM.QVAL where QNAM = RACEOSP
SCRNYN	SCRNYN	X	SUPPDM.QVAL where QNAM = SCRNYN
SCRPTID	SCRPTID	X	SUPPDM.QVAL where QNAM = SCRPTID
ENRPRRSN	ENRPRRSN	X	SUPPDM.QVAL where QNAM = ENRPRRSN
ENRPRRSNOSP	ENRPRRSNOSP	X	SUPPDM.QVAL where QNAM = ENRPRRSNOSP

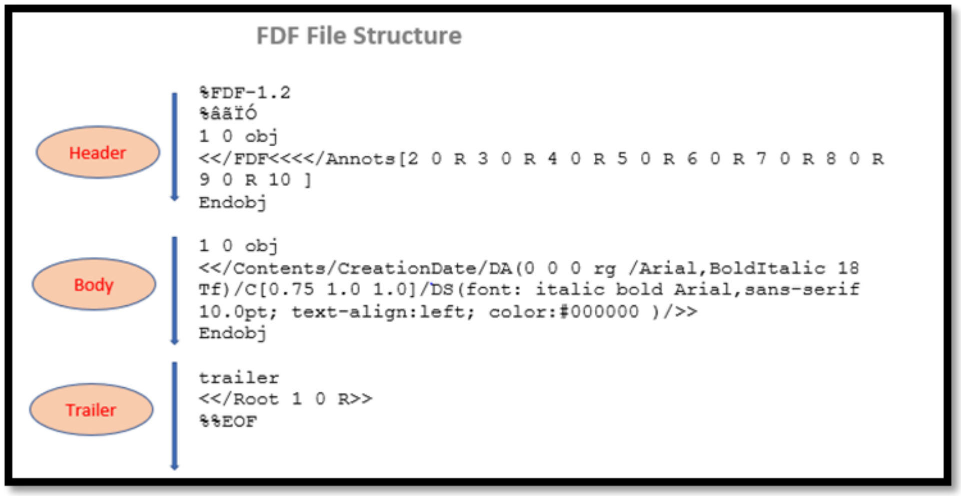
Figure 3. SDTM Mapping Specification showing how SDTM variables are mapped to CRF variables, either with 1:1 mapping or with an expression.

ANNOTATION OUTPUT

- FDF file (Forms Data Format)
- Case Report Form (PDF)

UNDERSTANDING THE FDF FILE

FDF is a file format representing data and annotations in PDFs. The FDF has 3 parts: header, body and trailer. The header describes the number of annotations and the annotation source file name. This is followed by one or more body sections which describe each annotation’s attributes (placement on the CRF page, size of the box surrounding an annotation, font and size of the annotation text, background color of the annotation box and PDF page to display the annotation on). The trailer signals the end-of-file.



```
%FDF-1.2
%âãÏÓ
1 0 obj
<</FDF<<<</Annots[2 0 R 3 0 R 4 0 R 5 0 R 6 0 R 7 0 R 8 0 R
9 0 R 10 ]
Endobj

1 0 obj
<</Contents/CreationDate/DA(0 0 0 rg /Arial,BoldItalic 18
Tf)/C[0.75 1.0 1.0]/DS(font: italic bold Arial,sans-serif
10.0pt; text-align:left; color:#000000 )/>>
Endobj

trailer
<</Root 1 0 R>>
%%EOF
```

Figure 4. Sample FDF file to understand its structure

COLOR CODING

Annotations should respect specific format (color, size) according to CDISC guidelines. Adding background color to the annotation box is one of the important aspects in the CRF annotation process. The color of the annotation box for the domain in the header of the CRF page should match the color of the annotation box for the variables that belongs to that domain. In the code we have considered that there will be maximum of five domains which might use the information from a single CRF form. Background colors are defined as latex colors (http://latexcolor.com/) . All the "NOT SUBMITTED" annotations are in grey color. The code uses Enum to define the colors for each successive domain

```
class DOMAIN_COLOR(Enum):
     FIRST_COLOR_LIGHT_BLUE = "[0.75 1.0 1.0]"
     SECOND_COLOR_LIGHT_YELLOW = "[1.0 1.0 0.66]"
     THIRD_COLOR_LIGHT_GREEN = "[0.75 1.0 0.75]"
     FOURTH_COLOR_PURPLE = "[0.66 0.75 1.0]"
     FIFTH_COLOR_LIGHT_ORANGE = "[1.0 0.75 0.66]"
```

PYTHON PACKAGES

This process relies heavily on pdfquery (https://pypi.org/project/pdfquery/). We are using the most recently available version, 0.4.3, but this has a bug that throws a "cant' concat str to bytes" error on `load()`. You must correct line 273 of pdfquery.py:

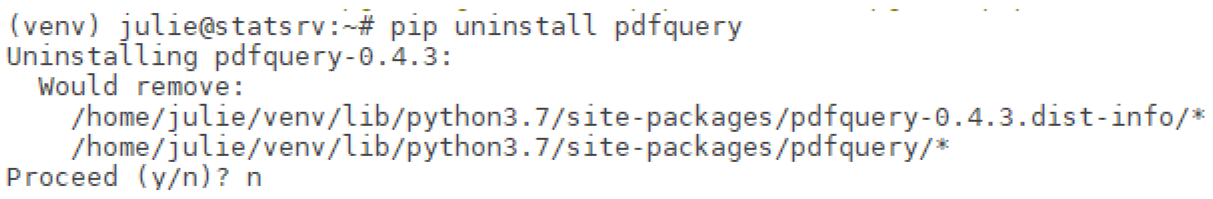
CURRENT:

```
    if 'P' in label_format:
        page_label = label_format['P']+page_label
```

UPDATE TO:

```
    if 'P' in label_format:
        page_label = label_format['P']+page_label.encode()
```

Note to python newbies: if you don't know where pdfquery.py is installed in your system, you can get the full path by asking to uninstall it (but don't actually uninstall it!):



```
(venv) julie@statsrv:~# pip uninstall pdfquery
Uninstalling pdfquery-0.4.3:
  Would remove:
    /home/julie/venv/lib/python3.7/site-packages/pdfquery-0.4.3.dist-info/*
    /home/julie/venv/lib/python3.7/site-packages/pdfquery/*
Proceed (y/n)? n
```

PYTHON SCRIPTS

We use two Python scripts for the annotation:

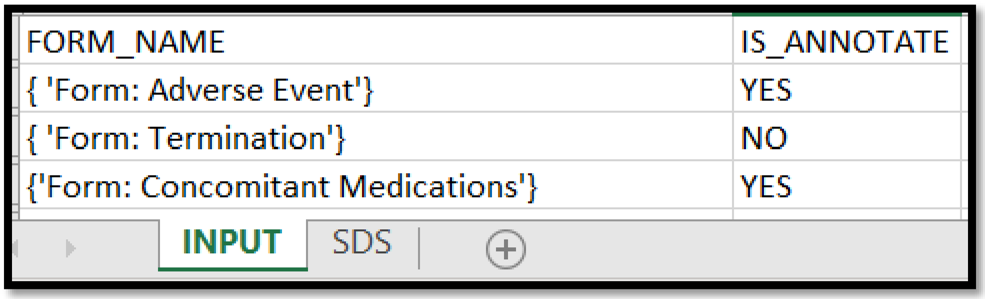
- generate_crf_input_form.py: creates excel sheet listing CRF pages and SDS forms
- create_crf_mapping_from_input_form.py: reads the excel sheet created by the first script and creates the annotation.

PROCESS

STEP1: CREATING INPUT FORM

First, a CRF Input Form Excel file with two tabs is created by accessing the SDS, the SDTM mapping specification and the CRF PDF.

A sample illustration of the CRF Input file with its tabs is shown in Fig. 7 below:



FORM_NAME	IS_ANNOTATE
{ 'Form: Adverse Event'}	YES
{ 'Form: Termination'}	NO
{'Form: Concomitant Medications'}	YES

Figure 5. Input form with sheet names and columns

The INPUT tab is a row listing of the CRF form names and their page numbers. Another field, sourced from the SDTM specification, specifies whether each form is to be annotated or not. Forms marked as ‘YES’ for annotation will have their form elements (e.g., textboxes or dropdowns) individually annotated. Forms marked as ‘NO’ will have a single ‘NOT SUBMITTED’ annotation on the form’s top left corner. Corresponding SDS form names are included. The last column, a concatenation of the SDS form name and page numbers, will become tab names in the Mapped CRF Excel file in Fig. 1 above. Annotations will be done by iterating over each form.

The SDS tab contains more details on the form elements. It is a listing by field OID, of the expected form element control types (e.g. radio button or check boxes) and labels. Generation of this CRF Input Form is handled by the first python script.

STEP2: READING INPUT FORM TO DETERMINE IF FORM WILL BE ANNOTATED

The second python script create_crf_mapping_from_input_form.py reads in the CRF file. From the INPUT tab, if the “IS_ANNOTATE” column is “YES”, it will proceed to do individual form element annotation; if “IS_ANNOTATE” is “NO”, it will annotate the form as “NOT SUBMITTED” as described previously.

Below is a function that handles forms to receive no annotation. It opens the CRF Input Form created in Step 1 and from the INPUT tab, will append to a dictionary consisting of annotation items and their text positions the ‘NOT SUBMITTED’ label as the key, and a values list consisting of its CRF PDF page number, coordinates of where to place it on the page as well as the color of that label in RGB format.

```python
def get_not_submitted_forms(retval, inputFormPath):
    """read CRF_Input_Annotations xlsx file==inputFormPath
       Returns a dict. Appends to dict of text positions created by
       get_text_position_in_dict those items on  forms for which annotation is not required
    """
    try:
        theFile = openpyxl.load_workbook(inputFormPath)
        inputFrmSheetNames = theFile.sheetnames                          # will be ['INPUT','SDS']
        for sheet in inputFrmSheetNames:                                 # Really pertains to 'INPUT' tab
            currentSheet = theFile[sheet]
            for row in range(2, currentSheet.max_row + 1):
                for column in "C":                                       # IS_ANNOTATE col
                        cell_name = "{}{}".format(column, row)           # from 'C2' on...
                        page_no_column = "{}{}".format("A", row)         # col A is Page_No
                        if currentSheet[cell_name].value.upper() == "NO":   # if IS_ANNOTATE is No...Append:
                            myarrval = []
                            myarrval.append('[89.5365 763.677 201.7595 777.922]')  # -Position of Not Submitted lbl
                            myarrval.append(currentSheet[page_no_column].value)    # -Page No,
                            myarrval.append('[0.55 0.57 0.67]')                    # -Not submitted rgb color.
                            tmpstr = CONST_NOT_SUBMITTED + str(row)                # e.g. NOT_SUBMITTED1 etc.
                            retval[tmpstr]= myarrval
    except Exception as e:
        logger.error('Exception occurred at get_not_submitted_forms : %s',e)
        print(e)
```

STEP3: EXTRACTING TEXT POSITION FROM CRF

This is the most important step in the whole process, where we will read in the actual case report form and extract the text position of the required fields. For each form in the CRF PDF Input file, if it is marked to annotate, the field label in Fig 8 below is compared against the CRF PDF text for that particular form. The code below returns the lower left (X_1,Y_1) and upper right (X_2,Y_2) coordinates of the exact text field from the case report form, and this is mapped with the values in the Mapped CRF excel sheet, and the SDTM annotations are placed depending on the control type. The text position is written to the Mapped CRF sheet, including the color for the label.

```
strval ="LTTextLineHorizontal:contains(" + "\""+

         returnSearchableQuestion(currentSheet[fieldcolumnval].value) +
          "\""+")
 arrval = pdf.extract ([
         ('with_parent', 'LTPage[pageid="{}"]'.format(pgnum)),
         (currentSheet[finColumn].value + '_' + format(pgnum),strval)
               ])
```

To place the annotation box on the CRF, we use the “control type” of the field as key factor in deciding the position for annotation. If the control type is a text, then the annotation box will be placed next to the question. If the control type is Radio button or Check box, then the annotation box will be placed below the question.

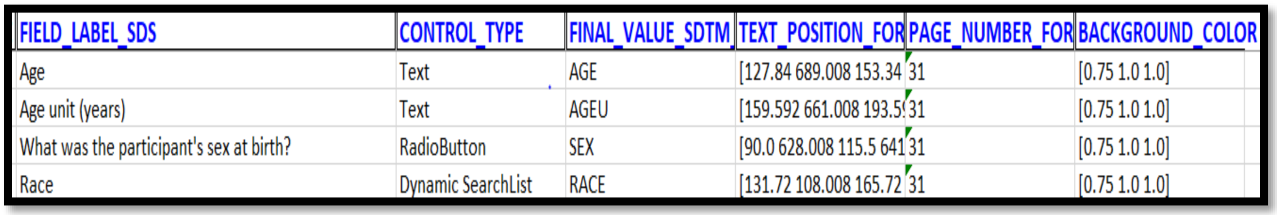
```
def buildrectpositionfromobject(label,finalValue,controlType):
    try:
        finalval = ""
        strval = finalValue
        lenval = len(strval.replace(" ",""))
        if(label != "" and label != null_variable ):
            left_corner = float(label.attr("x0"))
            bottom_corner = float(label.attr("y0"))
            left_top_corner = float(label.attr('x1'))
            bottom_top_corner = float(label.attr('y1'))
            strtempval = "[x0 y0 x1 y1]"
            if(controlType.upper() != "RADIOBUTTON" and controlType.upper()
                                   != "DROPDOWNLIST"):
                new_left_corner = left_top_corner - (bottom_top_corner –
                                   bottom_corner) + 20.0
                new_left_corner = round(new_left_corner,3)
                new_top_left_corner = new_left_corner + (lenval*8.5)
                new_top_left_corner = round(new_top_left_corner,3)
                finalval=strtempval.replace("x0",str(new_left_corner))
                                   .replace("y0",str(bottom_corner 3 )) \
                                   .replace("x1",str(new_top_left_corner))
                                   .replace("y1",str(bottom_top_corner-3))
            else:
                new_bottom_corner = bottom_corner - 20.0
                new_bottom_corner = round(new_bottom_corner,3)
                new_top_left_corner = left_corner + (lenval*8.5)
                new_top_left_corner = round(new_top_left_corner,3)
                new_bottom_top_corner = bottom_top_corner - 20.0
                new_bottom_top_corner = round(new_bottom_top_corner,3)
                finalval= strtempval.replace("x0",str(left_corner))
                            .replace("y0",str(new_bottom_corner-3)) \
                            .replace("x1",str(new_top_left_corner))
                             .replace("y1",str(new_bottom_top_corner-3))
        return finalval
```

STEP4: CREATING MAPPED CRF EXCEL FOR FORMS TO BE ANNOTATED

The second python script creates a new excel workbook called “Mapped CRF” which will eventually hold all the information required for the annotation, including the text position. The inputs for the python script are the CRF Input file created by the first python script, the SDTM mapping specification and the CRF

PDF file. The generated “Mapped CRF” excel workbook, consisting of a tab for each form as previously discussed, will also be output and can be used for cross verification.

Prior to text position extraction, the Mapped CRF will hold elements from the SDTM conversion specification and the SDS control type and label form element attributes. For each form, annotation placement will be done by searching for the label text in each CRF PDF form then updating the columns as shown in Fig. 8 below with text position, page number and annotation background color.



FIELD_LABEL_SDS	CONTROL_TYPE	FINAL_VALUE_SDTM	TEXT_POSITION_FOR	PAGE_NUMBER_FOR	BACKGROUND_COLOR
Age	Text	AGE	[127.84 689.008 153.34	31	[0.75 1.0 1.0]
Age unit (years)	Text	AGEU	[159.592 661.008 193.5	31	[0.75 1.0 1.0]
What was the participant's sex at birth?	RadioButton	SEX	[90.0 628.008 115.5 641	31	[0.75 1.0 1.0]
Race	Dynamic SearchList	RACE	[131.72 108.008 165.72	31	[0.75 1.0 1.0]

Figure 6. Mapped CRF excel with required columns

STEP5: CREATING FDF FILE

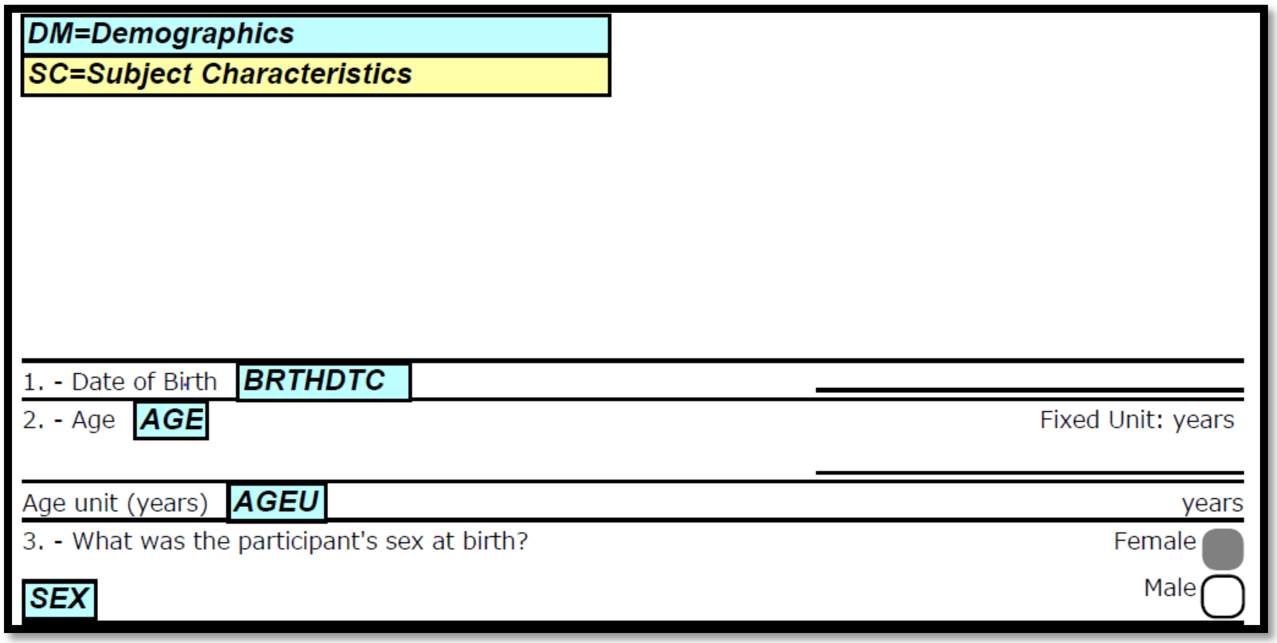
The FDF is populated with the following function:

```
def createfdf(crfpdfPath,dataval,fdffilePath):
    try:
        fdf = createfieldstring(crfpdfPath,dataval)
        fdf_file = open(fdffilePath, "wb")
        fdf_file.write(fdf)
        fdf_file.close()
    except Exception as e:
        print(e)

def createfieldstring(crfpdfPath,dataval):
    try:
        fdf = [b'%FDF-1.2\x0a%\xe2\xe3\xcf\xd3\x0d\x0a']
        fdf.append(b'1 0 obj\x0a<</FDF<</Annots[')
        fdf.append(return_header_string(dataval))
        fdf.append(b']/F(/')
        fdf.append(returnPDFFilenameforFDF(crfpd fPath))
        fdf.append(b')/ID[<001FEA6C53E75228A7C5F8E1EA52F3E1>
               <0E1BC1355C2B2A4A8DC60A7F0D2B29F6>]/UF(/')
        fdf.append(returnPDFFilenameforFDF(crfpdfPath))
        fdf.append(b')>>/Type/Catalog>>\x0aendobj\x0a)')
        fdf.append(b''.join(handle_data_strings(dataval)))
        fdf.append(b'trailer\x0a<</Root 1 0 R>>\x0a')
        fdf.append(b'%%EOF')
        return b''.join(fdf)])

subprocess.call("explorer "+fdffilePath, shell=True)
```

FINAL RESULT


DM=Demographics

SC=Subject Characteristics

1. - Date of Birth BRTHDTC

2. - Age AGE Fixed Unit: years

Age unit (years) AGEU years

3. - What was the participant's sex at birth? SEX

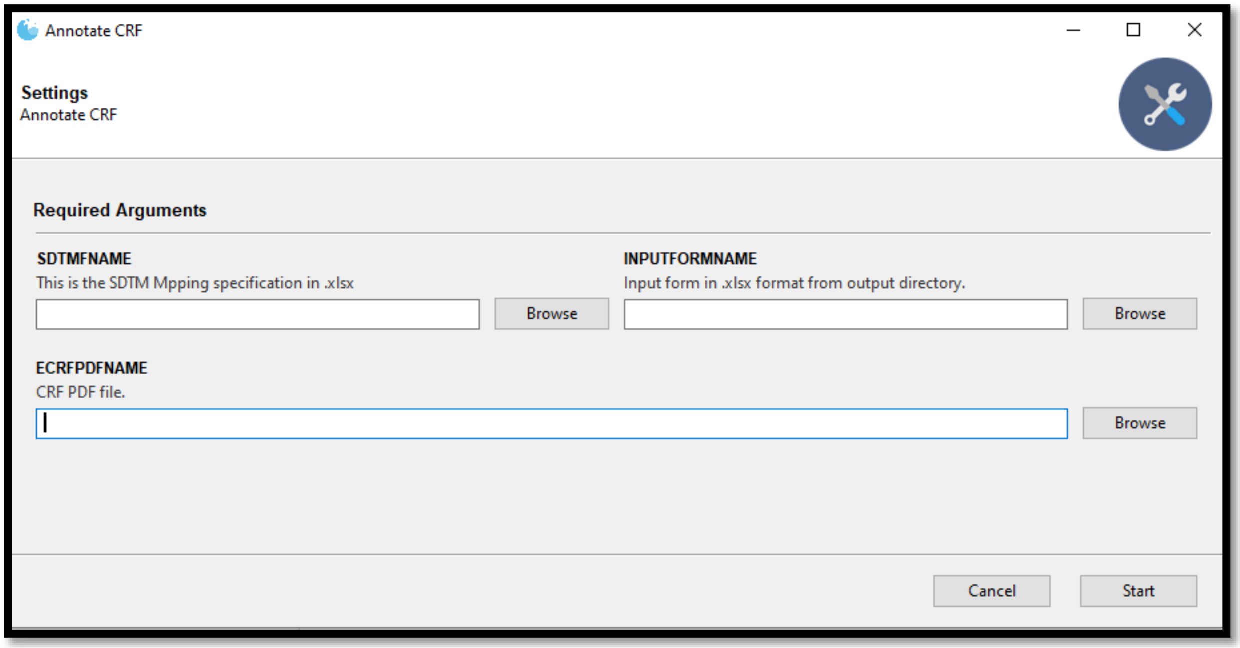
Female ●

Male ○

Figure 7. Final Annotated CRF

BONUS ADDITION: GOOEY INTERFACE

To make the application more usable and more accessible for non-programmers, we added the gooey module (https://pypi.org/project/Gooey/). We used the Gooey module to create the Graphical User Interface (GUI) for the python script. The required input files as mentioned above can then be accessed by simply browsing to them on the file system, making it more flexible, less error-prone and easier to run. See below for a sample setup.


Annotate CRF

Settings

Annotate CRF

Required Arguments

SDTMFNAME

This is the SDTM Mpping specification in .xlsx

INPUTFORMNAME

Input form in .xlsx format from output directory.

ECRFPDFNAME

CRF PDF file.

Figure 8. Gooey GUI application for python script

```
@Gooey(program_name="Annotate CRF")
def parse_args():
    stored_args = {}
    script_name = os.path.splitext(os.path.basename(__file__))[0]
    args_file = f"{script_name}-args.json"
    if os.path.isfile(args_file):
        with open(args_file) as data_file:
            stored_args = json.load(data_file)
    parser = GooeyParser(description = 'Annotate CRF')
    parser.add_argument('CRF_PDF_NAME',
                        action = 'store',
                        default = stored_args.get('CRF_PDF_NAME'),
                        widget='FileChooser',
                        help = 'CRF PDF file.')
    parser.add_argument('SDS_EXCEL_NAME',
                        action='store',
                        default=stored_args.get('SDS_EXCEL_NAME'),
                        widget='FileChooser',
                        help='SDS Excel file. Must be .xlsx file')
    parser.add_argument('SDTMFNAME',
                        action='store',
                        default=stored_args.get('SDTMFNAME'),
                        widget='FileChooser',
                        help='SDTM File. Must be .xlsx file')

    args = parser.parse_args()
    with open(args_file,'w') as data_file:
        json.dump(vars(args), data_file)
    return args
```

CONCLUSION

This method significantly reduces the time and effort required to create an annotated CRF while eliminating inconsistent annotations. It is very flexible and can be implemented to annotate CRFs for different types of trials and organizations. This method does not require a previously annotated CRF or a global dictionary with text position, which is the biggest advantage of using this annotation method.

REFERENCES

SDTM Annotations: Automation by implementing a standard process, Geo Joy, Novartis, Cambridge, MA Andre Couturier, Novartis, East Hanover, NJ

Have SAS Annotate your Blank CRF for you! Plus dynamically add color and style to your annotations. Steven Black, Agility-Clinical Inc., Carlsbad, CA

CDISC SDTM implementation guide: https://www.cdisc.org/standards/foundational/sdtmig

Cushman, Jack. 2020. PDFQuery v0.4.3 https://github.com/jcushman/pdfquery

Kiehl, Chris. 2020. Gooey v1.0.3 https://github.com/chriskiehl/Gooey

ACKNOWLEDGMENTS

The authors would like to thank Lawrence Madziwa for his review and guidance.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the authors at:

Author Name: Julie Stofel
Company: Statistical Center for HIV/AIDS Research & Prevention (SCHARP) at Fred Hutchinson Cancer Research Center, Seattle
Email: julie@scharp.org