PharmaSUG China 2021 - Paper AD-040

Automation for CRF Annotation and Bookmark Generation via Python

Peng Wang and Yiqun Wang, Cstone Pharmaceuticals

ABSTRACT

Annotated case report form (aCRF) is a mandatory PDF document for study data submissions to health authorities. CRF annotations embody the mapping relationship between captured data and their corresponding variables in Study Data Tabulation Model (SDTM). Besides annotation, CRF also should have particularly organized bookmark as illustrated in the published guidelines for aCRF. However, either manual annotation or bookmark creation for CRF is tedious, time consuming and error prone. In this paper, we proposed an integrated automation stream to obtain desired annotation and bookmark at the same time via python programming, which can dramatically boost efficiency in daily work. It is applicable for either English or Chinese aCRF preparation. Finally, source code of this annotation tool is bundled into one folder with all its dependencies and an executable file, which is clickable for triggering tool user interface.

INTRODUCTION

Much of our work here is inspired by a previous PharmSUG conference paper (Muthukumar Hema et al, 2020), including graphical user interface (GUI) and employment of Study Design Specification (SDS). SDS is an Excel document from electronic data capture (EDC) system, and the order of questions appearing in CRF is the same as they are listed in SDS. We insert some new columns of annotation rule into original SDS to get a modified one. Overall, we strive for eXtensible Markup Language (XML) based annotation file and automatic bookmark generation to better fit our reality. XML is a widely used markup language, which is convenient not only for human reading, but also for programmatical manipulation. The whole automation workflow is classified into three major steps (Figure 1). Firstly, we use modified SDS and blank CRF as input files. Blank CRF file is converted as a representative XML format internal data structure. Through parsing this xml and combination with data in modified SDS, we can obtain desired annotation dataframe and bookmark dataframe. Dataframe is a tabular data structure, as introduced by Pandas (https://pandas.pydata.org/docs/user_guide/index.html, a popular Python data science package), which is equivalent to the data set concept in SAS® language and dataframe concept in R language. Secondly, we use annotation dataframe to output an XML format annotation file, with XFDF (XML Forms Data Format) as extension name. At the same time, we also use bookmark dataframe and blank CRF file to output a bookmarked CRF. Lastly, annotation XFDF file is imported into bookmarked CRF to get final aCRF with bookmark.

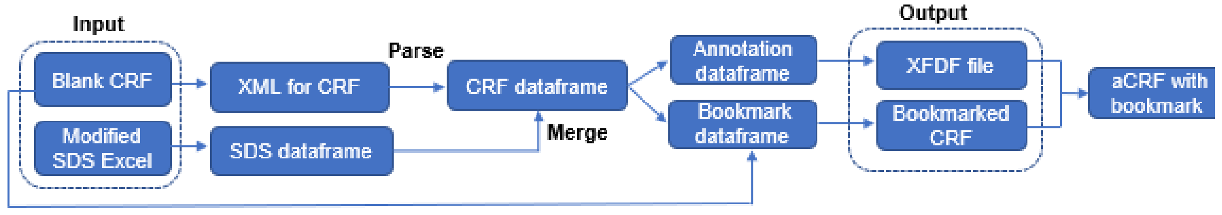


Figure 1. Overall Workflow of CRF Annotation and Bookmark Generation

GRAPHICAL USER INTERFACE DESIGN

To make this application more user friendly, we develop a GUI with Gooey (a Python GUI build up package, https://github.com/chriskiehl/Gooey). Users can select input blank CRF file, modified SDS file and directory for outputs through interface (Figure 2):

```
import gooey
@gooey.Gooey(program_name="CRF Annotation",program_description="Version\
1.0",header_show_title=False)
def parse_args():
    parser = gooey.GooeyParser()
    parser.add_argument("CRF_PDF",widget="FileChooser",help="Get blank CRF\
PDF file",gooey_options={"full_width":True})
    parser.add_argument("SDS_Excel",widget="FileChooser",help="Get SDS \
Excel file",gooey_options={"full_width":True})
    parser.add_argument("Output_Directory", \
widget="DirChooser",help="Directory for output xfdf file")
    args=parser.parse_args()
    return [args.SDS_Excel,args.CRF_PDF,args.Output_Directory]
lstPath=parse_args()
pathSourceExcel=lstPath[0]
pathPDF=lstPath[1]
pathOutput=lstPath[2]
```

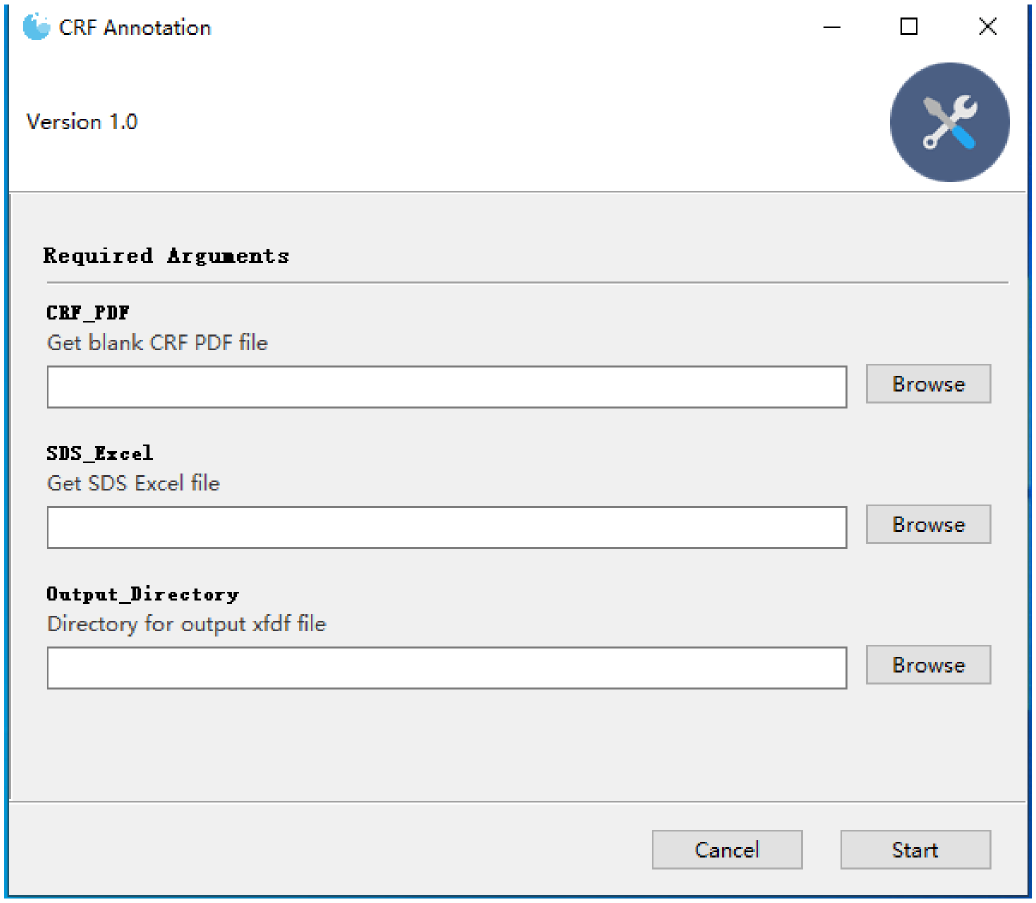
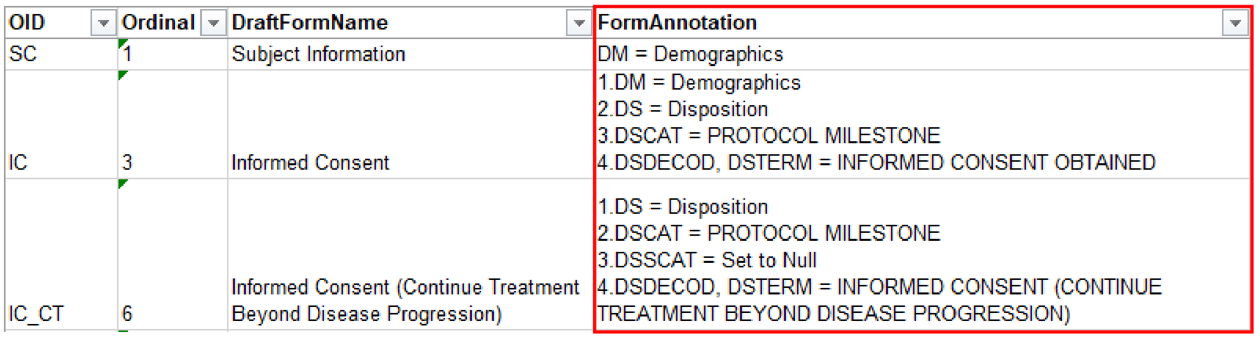


Figure 2. GUI of CRF Annotation Tool

GENERATION OF ANNOTATION AND BOOKMARK DATAFRAMES

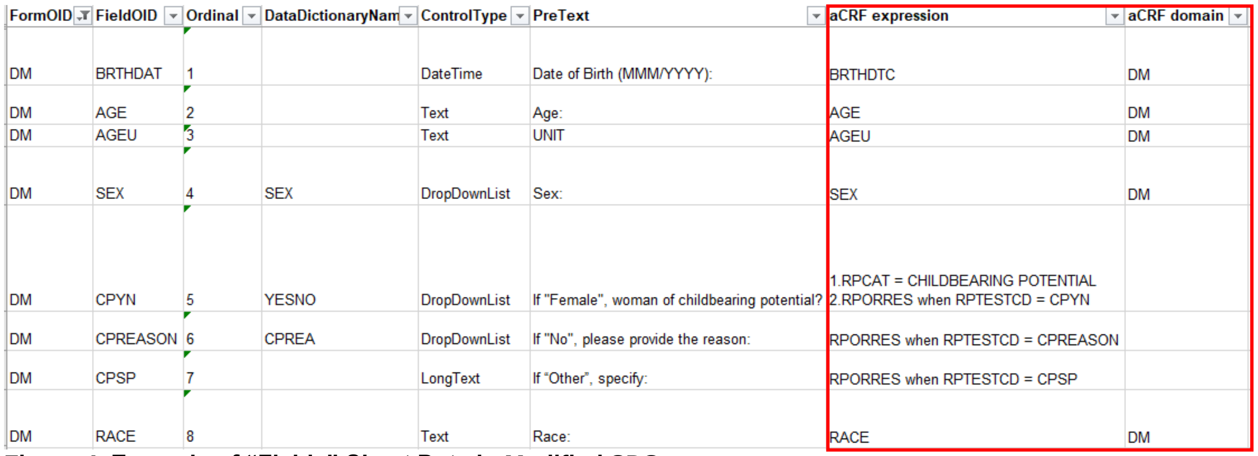
READING DATA FROM MODIFIED SDS

Modified SDS Excel file comes with several sheets. Some sheets, such as “Forms”, “Fields”, “DataDictionaries” and “DataDictionaryEntries”, are used for CRF annotation (Figure 3 and Figure 4). While sheets “Folders” and “MatrixPDF” are reserved for bookmark generation (Figure 5 and Figure 6). Data from these sheets is read with pandas as separate dataframes. In case of necessary merge for different dataframes, common key between them is used, e.g: “OID” column in “Forms” sheet is equal to “FormOID” column in “Fields” sheet.



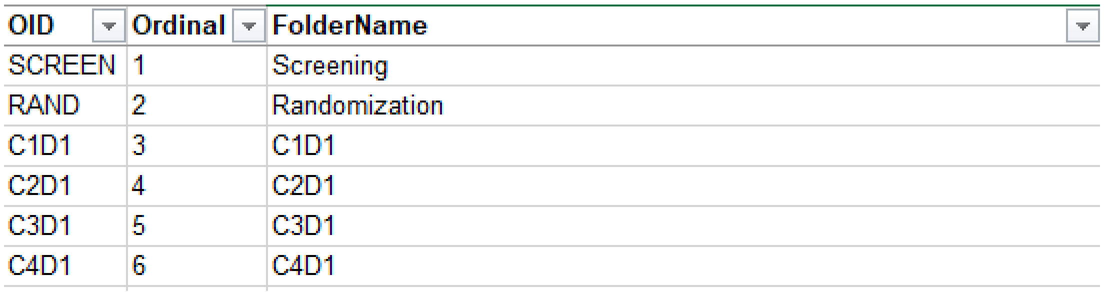
OID	Ordinal	DraftFormName	FormAnnotation
SC	1	Subject Information	DM = Demographics
IC	3	Informed Consent	1.DM = Demographics 2.DS = Disposition 3.DSCAT = PROTOCOL MILESTONE 4.DSDECOD, DSTERM = INFORMED CONSENT OBTAINED
IC_CT	6	Informed Consent (Continue Treatment Beyond Disease Progression)	1.DS = Disposition 2.DSCAT = PROTOCOL MILESTONE 3.DSSCAT = Set to Null 4.DSDECOD, DSTERM = INFORMED CONSENT (CONTINUE TREATMENT BEYOND DISEASE PROGRESSION)

Figure 3. Example of “Forms” Sheet Data in Modified SDS
Note that red color encircled column “FormAnnotation” is not from original SDS but inserted here for CRF domain annotation purpose. This sheet indicates the relationship between form name and its SDTM domain.



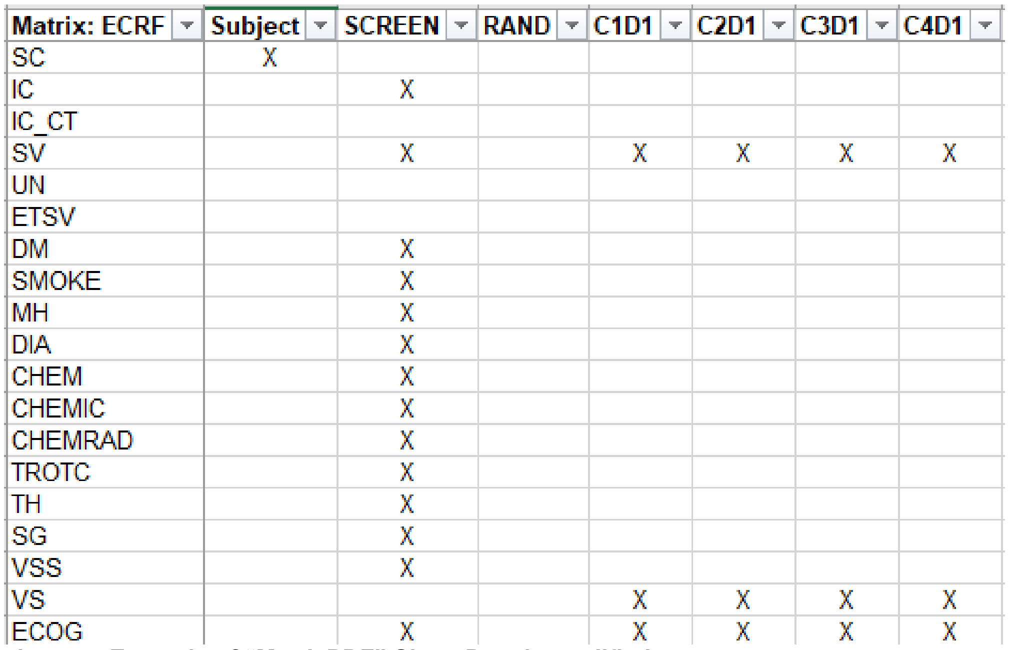
FormOID	FieldOID	Ordinal	DataDictionaryNam	ControlType	PreText	aCRF expression	aCRF domain
DM	BRTHDAT	1		DateTime	Date of Birth (MMM/YYYY):	BRTHDTC	DM
DM	AGE	2		Text	Age:	AGE	DM
DM	AGEU	3		Text	UNIT	AGEU	DM
DM	SEX	4	SEX	DropDownList	Sex:	SEX	DM
DM	CPYN	5	YESNO	DropDownList	If "Female", woman of childbearing potential?	1.RPCAT = CHILDBEARING POTENTIAL 2.RPORRES when RPTESTCD = CPYN	
DM	CPREASON	6	CPREA	DropDownList	If "No", please provide the reason:	RPORRES when RPTESTCD = CPREASON	
DM	CPSP	7		LongText	If "Other", specify:	RPORRES when RPTESTCD = CPSP	
DM	RACE	8		Text	Race:	RACE	DM

Figure 4. Example of “Fields” Sheet Data in Modified SDS
Note that red color encircled columns “aCRF expression” and “aCRF domain” are not from original SDS but inserted here for CRF annotation purpose. Some fields in column “aCRF domain” are left as blank. As we can get its SDTM domain with a logic: initial two capital letters of aCRF expression or two capital letters after “SUPP”. For the exception annotation “BRTHDTC”, which is a variable in SDTM DM domain but not begins with “DM”, then its SDTM domain name must be filled in “aCRF domain” column (the first row of this example).



OID	Ordinal	FolderName
SCREEN	1	Screening
RAND	2	Randomization
C1D1	3	C1D1
C2D1	4	C2D1
C3D1	5	C3D1
C4D1	6	C4D1

Figure 5. Example of “Folders” Sheet Data in Modified SDS
Column “FolderName” provides the names of folders under visits bookmark, each contains one or more CRF forms. “Ordinal” indicates sequence of folders.



Matrix: ECRF	Subject	SCREEN	RAND	C1D1	C2D1	C3D1	C4D1
SC	X						
IC		X					
IC_CT							
SV		X		X	X	X	X
UN							
ETSV							
DM		X					
SMOKE		X					
MH		X					
DIA		X					
CHEM		X					
CHEMIC		X					
CHEMRAD		X					
TROTC		X					
TH		X					
SG		X					
VSS		X					
VS				X	X	X	X
ECOG		X		X	X	X	X

Figure 6. Example of "MatrixPDF" Sheet Data in Modified SDS.
This sheet shows which CRF forms are under each folder/visit, which is used for visits bookmark creation. Column names (e.g: Subject, SCREEN) are folder OID, which correspond to column "OID" data in "Folders" sheet. Row names (e.g: SC, IC) are CRF form OID, which correspond to "OID" column in "Forms" sheet. Symbol "X" denotes presence of specific CRF form in that folder.

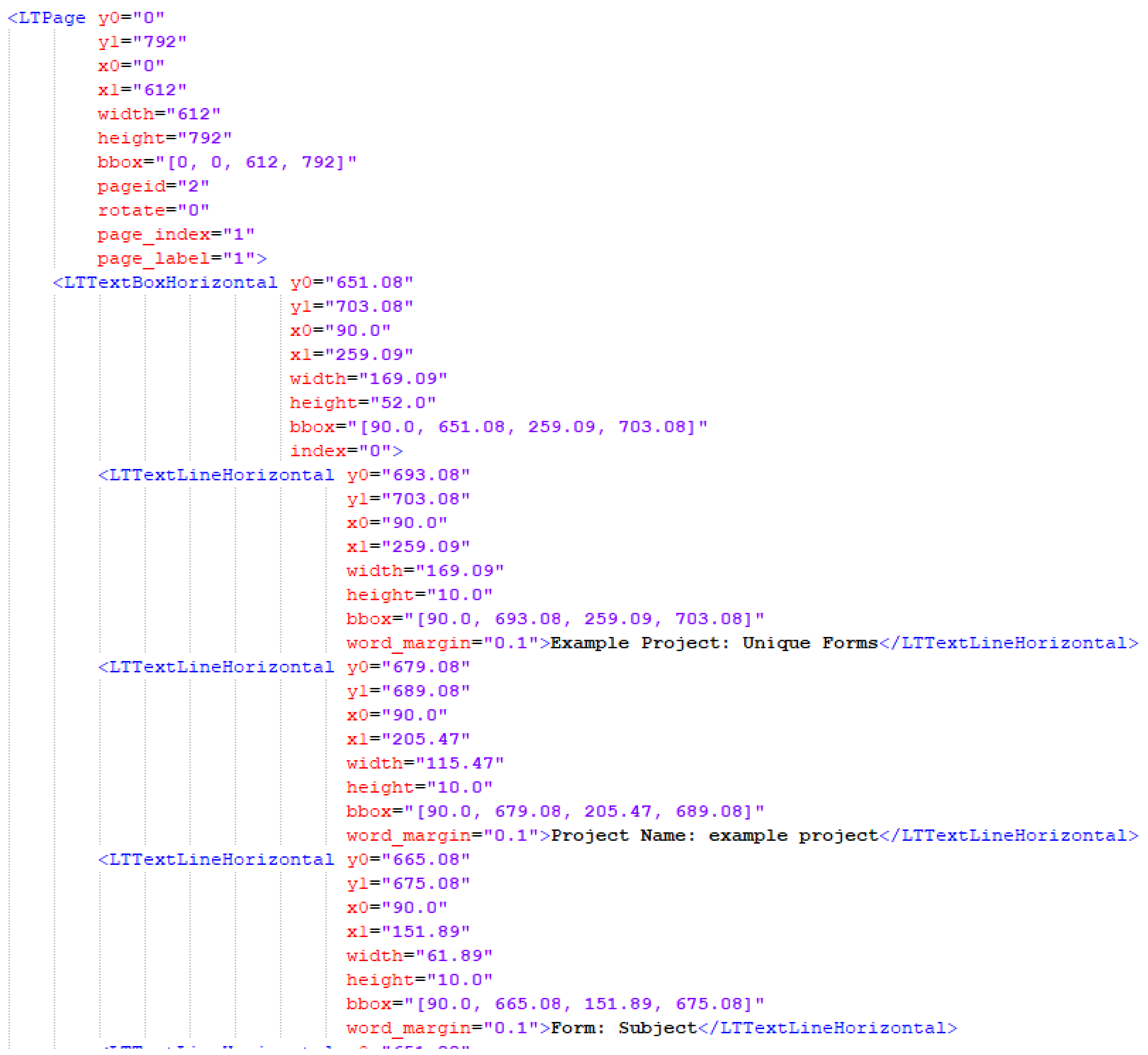
In summary, through reading data in "Forms", "Fields", "DataDictionaries" and "DataDictionaryEntries" sheets, we generate SDS dataframe `dfS`, which includes SDTM annotation rule, through reading data in "Folders" and "MatrixPDF" sheets, we obtain dataframe `dfMatrix`, which includes visits bookmark creation rule.

PARSING FOR XML CONVERTED FROM BLANK CRF

We employ pdfquery (a Python PDF parsing package, https://github.com/jcushman/pdfquery) to convert blank CRF to an XML format internal data structure:

```
import pdfquery
pdf=pdfquery.PDFquery(file=pathPDF)
pdf.load()
root=pdf.tree.getroot()
```

Variable `pathPDF` represents blank CRF file absolute path, which is assigned during GUI operation. Resulted xml is parsed with a Python package called lxml (https://lxml.de/tutorial.html, a Python package for XML processing). Variable `root` here represents "pdfxml" element, which is indeed the root of this tree data structure. It contains several "LTPage" sub elements ("branches of tree"). And each "LTPage" element represents a page in blank CRF PDF file (Figure 7). Element "LTPage" has different attributes. Values from "width" and "height" indicates dimension of this PDF page, which we can also calculate from formulas: width=x1-x0, height=y1-y0. Certainly, we need to convert string into integers before calculation. For another attribute "page_index", the integer format of its value is equal to actual CRF page number minus one, as "page_index" starts from zero and CRF PDF file page number starts from one. This is a very important information to locate each form during annotation or bookmark generation. Additionally, we find that "LTPage" also contains many children elements ("LTTextBoxHorizontal" or "LTTextLineHorizontal"). Through further parsing these elements, we can acquire not only project name (string after "Project Name:") and form name (string after "Form:"), but also coordinates of questions in CRF via element attribute value of "bbox" (a string of square bracket with coordinates x0,y0,x1 and y1, Figure 8).



```xml
<LTPage y0="0"
        y1="792"
        x0="0"
        x1="612"
        width="612"
        height="792"
        bbox="[0, 0, 612, 792]"
        pageid="2"
        rotate="0"
        page_index="1"
        page_label="1">
    <LTTextBoxHorizontal y0="651.08"
                         y1="703.08"
                         x0="90.0"
                         x1="259.09"
                         width="169.09"
                         height="52.0"
                         bbox="[90.0, 651.08, 259.09, 703.08]"
                         index="0">
        <LTTextLineHorizontal y0="693.08"
                              y1="703.08"
                              x0="90.0"
                              x1="259.09"
                              width="169.09"
                              height="10.0"
                              bbox="[90.0, 693.08, 259.09, 703.08]"
                              word_margin="0.1">Example Project: Unique Forms</LTTextLineHorizontal>
        <LTTextLineHorizontal y0="679.08"
                              y1="689.08"
                              x0="90.0"
                              x1="205.47"
                              width="115.47"
                              height="10.0"
                              bbox="[90.0, 679.08, 205.47, 689.08]"
                              word_margin="0.1">Project Name: example project</LTTextLineHorizontal>
        <LTTextLineHorizontal y0="665.08"
                              y1="675.08"
                              x0="90.0"
                              x1="151.89"
                              width="61.89"
                              height="10.0"
                              bbox="[90.0, 665.08, 151.89, 675.08]"
                              word_margin="0.1">Form: Subject</LTTextLineHorizontal>
```

Figure 7. Partial XML Format Data Structure Converted from Blank CRF PDF File

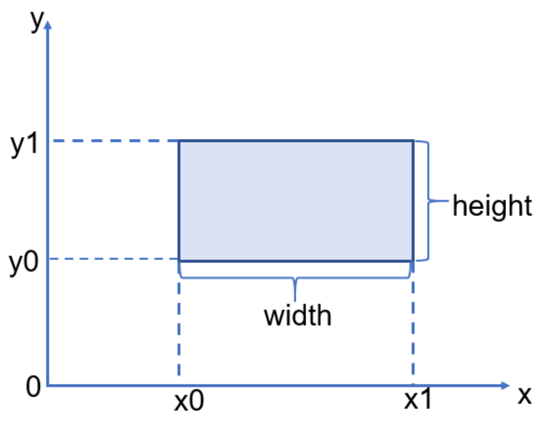
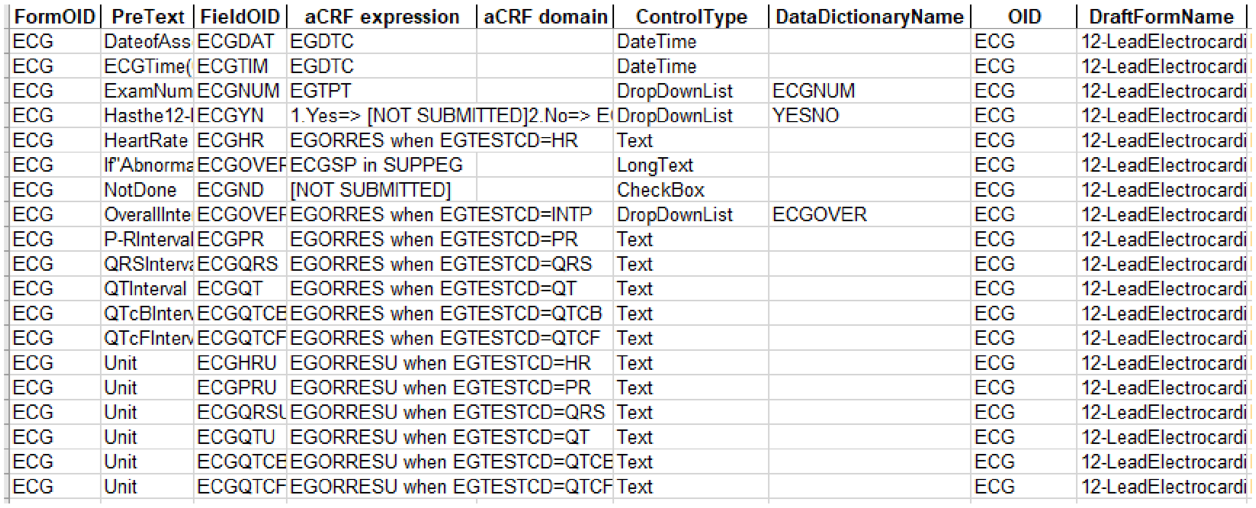


Figure 8. Coordinates x0,y0,x1,y1 of Element LTTextBoxHorizontal

To store parsing result for CRF converted xml, we create a CRF dataframe named as `dfMap` with four columns ("FormName","PageIndex","PreText" and "PreTextPos"). Specifically, we keep CRF form names in "FormName", "page_index" attribute value in "PageIndex", CRF form questions and their coordinates in "PreText" and "PreTextPos" separately. This position information is crucial for annotation purpose. Since we also have built a dataframe `dfS` from modified SDS data reading step. Naturally, we want to merge them to get a new dataframe `df` (Figure 9 and Figure 10), which can correlate modified SDS data (e.g: "FormAnnotation", "aCRF expression", "aCRF domain", "ControlType" and "DraftFormName") with `dfMap` data (e.g: "PageIndex" and "PreTextPos"). The key for this merge is a defined string:

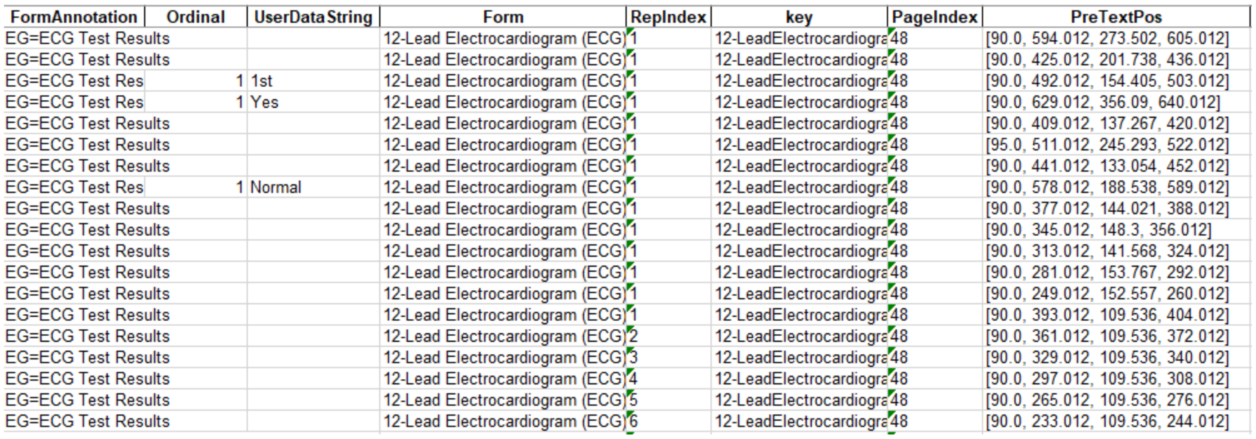
```
dfMap['key']=dfMap['FormName']+'_'+dfMap['PreText']+'_'+dfMap['RepIndex']
dfS['key']=dfS['DraftFormName']+'_'+dfS['PreText']+'_'+dfS['RepIndex']
```

"RepIndex" is the number for occurrences of "PreText" (the question in CRF) in that specific form. The reason we choose the key as above not the one as `['FormName']+'_'+['PreText']` and `['DraftFormName']+'_'+['PreText']` is same "PreText" may raise more than once in the same form.



FormOID	PreText	FieldOID	aCRF expression	aCRF domain	ControlType	DataDictionaryName	OID	DraftFormName
ECG	DateofAss	ECGDAT	EGDTC		DateTime		ECG	12-LeadElectrocardi
ECG	ECGTime(	ECGTIM	EGDTC		DateTime		ECG	12-LeadElectrocardi
ECG	ExamNum	ECGNUM	EGTPT		DropDownList	ECGNUM	ECG	12-LeadElectrocardi
ECG	Hasthe12-l	ECGYN	1.Yes=> [NOT SUBMITTED]2.No=> E		DropDownList	YESNO	ECG	12-LeadElectrocardi
ECG	HeartRate	ECGHR	EGORRES when EGTESTCD=HR		Text		ECG	12-LeadElectrocardi
ECG	If"Abnorma	ECGOVEF	ECGSP in SUPPEG		LongText		ECG	12-LeadElectrocardi
ECG	NotDone	ECGND	[NOT SUBMITTED]		CheckBox		ECG	12-LeadElectrocardi
ECG	OverallInte	ECGOVEF	EGORRES when EGTESTCD=INTP		DropDownList	ECGOVER	ECG	12-LeadElectrocardi
ECG	P-RInterval	ECGPR	EGORRES when EGTESTCD=PR		Text		ECG	12-LeadElectrocardi
ECG	QRSInterva	ECGQRS	EGORRES when EGTESTCD=QRS		Text		ECG	12-LeadElectrocardi
ECG	QTInterval	ECGQT	EGORRES when EGTESTCD=QT		Text		ECG	12-LeadElectrocardi
ECG	QTcBInter	ECGQTCE	EGORRES when EGTESTCD=QTCB		Text		ECG	12-LeadElectrocardi
ECG	QTcFInter	ECGQTCF	EGORRES when EGTESTCD=QTCF		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGHRU	EGORRESU when EGTESTCD=HR		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGPRU	EGORRESU when EGTESTCD=PR		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGQRSU	EGORRESU when EGTESTCD=QRS		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGQTU	EGORRESU when EGTESTCD=QT		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGQTCE	EGORRESU when EGTESTCD=QTCE		Text		ECG	12-LeadElectrocardi
ECG	Unit	ECGQTCF	EGORRESU when EGTESTCD=QTCF		Text		ECG	12-LeadElectrocardi

Figure 9. Columns 1-9 of Merged Dataframe `df` for 12-Lead Electrocardiogram (ECG) Form



FormAnnotation	Ordinal	UserDataString	Form	RepIndex	key	PageIndex	PreTextPos
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 594.012, 273.502, 605.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 425.012, 201.738, 436.012]
EG=ECG Test Res	1	1st	12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 492.012, 154.405, 503.012]
EG=ECG Test Res	1	Yes	12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 629.012, 356.09, 640.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 409.012, 137.267, 420.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[95.0, 511.012, 245.293, 522.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 441.012, 133.054, 452.012]
EG=ECG Test Res	1	Normal	12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 578.012, 188.538, 589.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 377.012, 144.021, 388.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 345.012, 148.3, 356.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 313.012, 141.568, 324.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 281.012, 153.767, 292.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 249.012, 152.557, 260.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	1	12-LeadElectrocardiogra	48	[90.0, 393.012, 109.536, 404.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	2	12-LeadElectrocardiogra	48	[90.0, 361.012, 109.536, 372.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	3	12-LeadElectrocardiogra	48	[90.0, 329.012, 109.536, 340.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	4	12-LeadElectrocardiogra	48	[90.0, 297.012, 109.536, 308.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	5	12-LeadElectrocardiogra	48	[90.0, 265.012, 109.536, 276.012]
EG=ECG Test Results			12-Lead Electrocardiogram (ECG)	6	12-LeadElectrocardiogra	48	[90.0, 233.012, 109.536, 244.012]

Figure 10. Columns 10-17 of Merged Dataframe `df` for 12-Lead Electrocardiogram (ECG) Form

GENERATION OF ANNOTATION DATAFRAME

Now we have CRF questions (named as “PreText” in modified SDS) coordinates in specific CRF page (“PreTextPos” in dataframe `df`). Next steps are getting CRF questions annotation coordinates and textbox background color, CRF form domain annotation textbox background color, text color, text size and coordinates.

For “PreText” annotation coordinates, as introduced by the previous paper (Muthukumar Hema et al, 2020), we use “ControlType” and “aCRF” data together to determine its location. If control type belongs to a list of “DropDownList”, “CheckBox”, “RadioButton” and “SearchList”, then we place annotation text under “PreText”. If not, we choose right side.

For CRF form domain annotation textbox background color (a different color means a different SDTM domain in a form), we prepare four colors for differentiation, including blue, green, yellow, and purple:

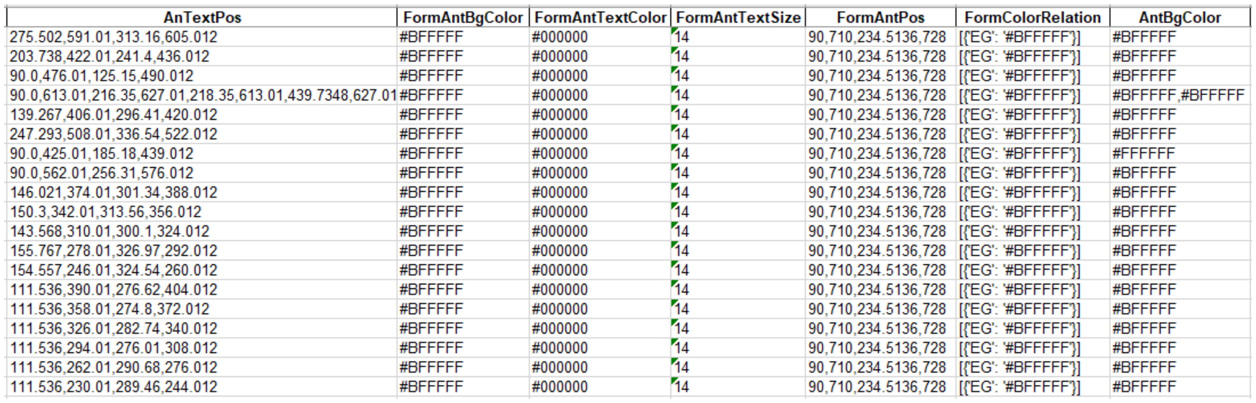
```
def getFormAntBgColor(formAnt):
    '''determine form annotation text box background color'''
    lst = formAnt.split("\n")
    if len(lst)==1 and lst[0].strip()=="[NOT SUBMITTED]":
        return "#FFFFFF" #white background color
    lst = list(filter(lambda x:(x!=None) and (x!=""),lst))
    if len(lst) >= 2:
        lstDomain = [x[2:4] for x in lst]
    elif len(lst) == 1:
        lstDomain = [x[0:2] for x in lst]
    lstDomainUnique = []
    for item in lstDomain:
        if item not in lstDomainUnique:
            lstDomainUnique.append(item)
    d = {} #dictionary for mapping domain with color
    if len(lstDomainUnique) == 1:
        d[lstDomainUnique[0]] = "#BFFFFF" #blue background
    elif len(lstDomainUnique) == 2:
        d[lstDomainUnique[0]] = "#BFFFFF" #blue background
        d[lstDomainUnique[1]] = "#BFFFBF" #green background
    elif len(lstDomainUnique) == 3:
        d[lstDomainUnique[0]] = "#BFFFFF" #blue background
        d[lstDomainUnique[1]] = "#BFFFBF" #green background
        d[lstDomainUnique[2]] = "#FFFF66" #yellow background
    elif len(lstDomainUnique) == 4:
        d[lstDomainUnique[0]] = "#BFFFFF" #blue background
        d[lstDomainUnique[1]] = "#BFFFBF" #green background
        d[lstDomainUnique[2]] = "#FFFF66" #yellow background
        d[lstDomainUnique[3]] = "#CC99CC" #purple background
    lstColor = [d[x] for x in lstDomain]
    strColor = ",".join(lstColor)
    return strColor
```

For CRF form domain annotation text color and size, if there are two capital letters before equation symbol, then its color is black and font is fourteen, otherwise, color is red, and font is ten.

For CRF form domain annotation coordinates, number of annotations for the form and text size are both taken into considerations.

For CRF “PreText” annotation text background color, to be consistent with SDTM domain color in each CRF form, we first set up a mapping relationship between color and SDTM domain for each CRF form, then get desired color based on SDTM domain of “aCRF expression” (either initial two capital letters or two capital letters after “SUPP” in aCRF expression). However, there are some exceptions, e.g: annotation “BRTHDTC” is a variable in SDTM DM domain but neither begins with “DM” or nor with “DM” after “SUPP”. Hence, we immediately get corrected color just for these exceptions with “aCRF domain” column data, which is originally from “aCRF domain” in “Fields” sheet of modified SDS (Figure 4 and Figure 9).

With all above steps, we add some extra columns (Figure 11) into dataframe `df` and achieve the goal of annotation dataframe generation.



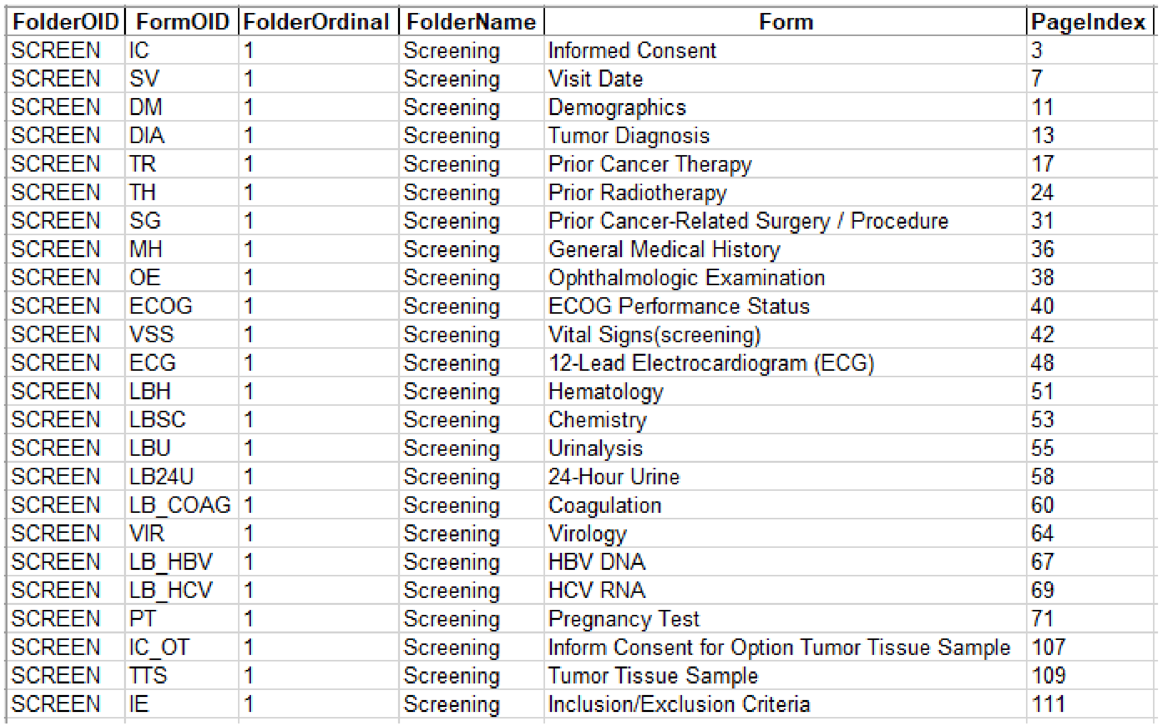
AnTextPos	FormAntBgColor	FormAntTextColor	FormAntTextSize	FormAntPos	FormColorRelation	AntBgColor
275.502,591.01,313.16,605.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
203.738,422.01,241.4,436.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
90.0,476.01,125.15,490.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
90.0,613.01,216.35,627.01,218.35,613.01,439.7348,627.01	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF,#BFFFFF
139.267,406.01,296.41,420.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
247.293,508.01,336.54,522.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
90.0,425.01,185.18,439.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#FFFFFF
90.0,562.01,256.31,576.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
146.021,374.01,301.34,388.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
150.3,342.01,313.56,356.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
143.568,310.01,300.1,324.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
155.767,278.01,326.97,292.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
154.557,246.01,324.54,260.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,390.01,276.62,404.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,358.01,274.8,372.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,326.01,282.74,340.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,294.01,276.01,308.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,262.01,290.68,276.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF
111.536,230.01,289.46,244.012	#BFFFFF	#000000	14	90,710,234.5136,728	[{'EG': '#BFFFFF'}]	#BFFFFF

Figure 11. Extra Columns Added in Dataframe `df` for 12-Lead Electrocardiogram (ECG) Form

GENERATION OF BOOKMARK DATAFRAME

Bookmark of aCRF has two parts, “visits” and “domain”, visits bookmark is timeline based, domain bookmark is firstly arranged alphabetically on SDTM domain and then on the order of forms as they appear in CRF.

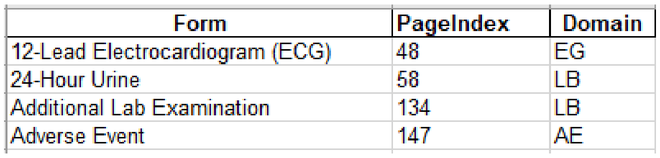
For visits bookmark creation, it requires names of visits, involved CRF form names and CRF form page index numbers. The first two information comes from “Folders” and “MatrixPDF” sheets in modified SDS (Figure 5 and Figure 6) and stored in dataframe `dfMatrix`. The third one comes from our dataframe `df` obtained in previous section. Thus, we can get visits bookmark dataframe by merging them together (Figure 12). Here, “FolderName” is the name of each visit under visits bookmark.



FolderOID	FormOID	FolderOrdinal	FolderName	Form	PageIndex
SCREEN	IC	1	Screening	Informed Consent	3
SCREEN	SV	1	Screening	Visit Date	7
SCREEN	DM	1	Screening	Demographics	11
SCREEN	DIA	1	Screening	Tumor Diagnosis	13
SCREEN	TR	1	Screening	Prior Cancer Therapy	17
SCREEN	TH	1	Screening	Prior Radiotherapy	24
SCREEN	SG	1	Screening	Prior Cancer-Related Surgery / Procedure	31
SCREEN	MH	1	Screening	General Medical History	36
SCREEN	OE	1	Screening	Ophthalmologic Examination	38
SCREEN	ECOG	1	Screening	ECOG Performance Status	40
SCREEN	VSS	1	Screening	Vital Signs(screening)	42
SCREEN	ECG	1	Screening	12-Lead Electrocardiogram (ECG)	48
SCREEN	LBH	1	Screening	Hematology	51
SCREEN	LBSC	1	Screening	Chemistry	53
SCREEN	LBU	1	Screening	Urinalysis	55
SCREEN	LB24U	1	Screening	24-Hour Urine	58
SCREEN	LB_COAG	1	Screening	Coagulation	60
SCREEN	VIR	1	Screening	Virology	64
SCREEN	LB_HBV	1	Screening	HBV DNA	67
SCREEN	LB_HCV	1	Screening	HCV RNA	69
SCREEN	PT	1	Screening	Pregnancy Test	71
SCREEN	IC_OT	1	Screening	Inform Consent for Option Tumor Tissue Sample	107
SCREEN	TTS	1	Screening	Tumor Tissue Sample	109
SCREEN	IE	1	Screening	Inclusion/Exclusion Criteria	111

Figure 12. Example of Visits Bookmark Dataframe for “Screening” Visit

For domain bookmark, it requires SDTM domain, involved CRF form names and their page index numbers (Figure 13). All of them are already present in our annotation dataframe in previous section.



Form	PageIndex	Domain
12-Lead Electrocardiogram (ECG)	48	EG
24-Hour Urine	58	LB
Additional Lab Examination	134	LB
Adverse Event	147	AE

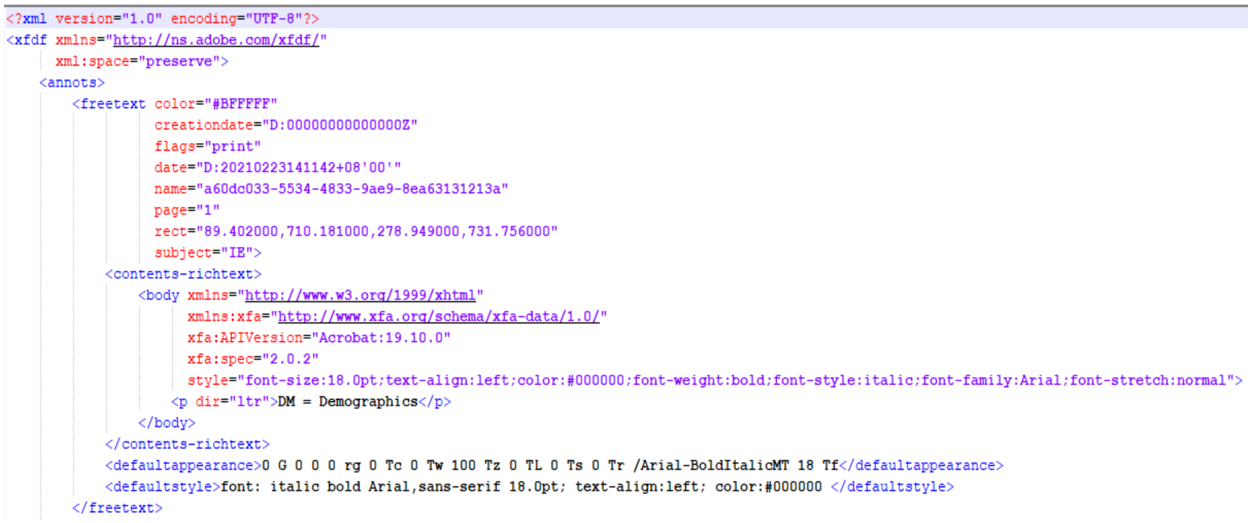
Figure 13. Example of Partial Domain Bookmark Dataframe

OUTPUT ANNOTATION XFDF FILE AND BOOKMARKED CRF

OUTPUT ANNOTATION XFDF FILE

Annotations of CRF are comments of a PDF file. We can export annotations from aCRF to a single file or import this annotation file into blank CRF to get aCRF. Involved steps are listed in Adobe® official website (https://helpx.adobe.com/ca/acrobat/using/importing-exporting-comments.html). This annotation file extension can be either FDF (Forms Data Format) or XFDF (XML Forms Data Format). As XFDF is XML based and easier for us to read and manipulate, here we aim to output an annotation XFDF file.

In an example of annotation XFDF file (Figure 14), we can see it has a typical XML structure. XML version and encoding are declared in the first row. Element "xfdf" has a sub element "annots", which literally means annotations. Element "annots" comes with many children elements. And each child element "freetext" corresponds to exactly an annotation of CRF. We also observe that content of element "p" inside "freetext" is annotation content itself, value of "freetext" attribute "page" is "PageIndex" in annotation dataframe `df` we get previously, value of "freetext" attribute "rect" is annotation coordinates (x0,y0,x1,y1), value of "freetext" attribute "color" is annotation text box background color (we use this background color to mark different SDTM domains in the same CRF form), value of element "body" attribute "style" is a group of key and value pairs ("font-size" value indicates annotation font size, "color" indicates annotation text color, "font-weight", "font-style" and "font-family" also tell us weight, style and family of annotation text font).



```
<?xml version="1.0" encoding="UTF-8"?>
<xfdf xmlns="http://ns.adobe.com/xfdf/"
      xml:space="preserve">
    <annots>
        <freetext color="#BFFFFF"
                  creationdate="D:00000000000000Z"
                  flags="print"
                  date="D:20210223141142+08'00'"
                  name="a60dc033-5534-4833-9ae9-8ea63131213a"
                  page="1"
                  rect="89.402000,710.181000,278.949000,731.756000"
                  subject="IE">
            <contents-richtext>
                <body xmlns="http://www.w3.org/1999/xhtml"
                      xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
                      xfa:APIVersion="Acrobat:19.10.0"
                      xfa:spec="2.0.2"
                      style="font-size:18.0pt;text-align:left;color:#000000;font-weight:bold;font-style:italic;font-family:Arial;font-stretch:normal">
                    <p dir="ltr">DM = Demographics</p>
                </body>
            </contents-richtext>
            <defaultappearance>0 G 0 0 0 rg 0 Tc 0 Tw 100 Tz 0 TL 0 Ts 0 Tr /Arial-BoldItalicMT 18 Tf</defaultappearance>
            <defaultstyle>font: italic bold Arial,sans-serif 18.0pt; text-align:left; color:#000000 </defaultstyle>
        </freetext>
```

Figure 14. Example of Partial Annotation XFDF File

To output an annotation XFDF file, we just need to utilize our annotation dataframe to generate similar XML structure (Figure 14) and output it as a file. Package lxml (https://lxml.de/tutorial.html) can also help us fulfill this goal. As it is designed not only for XML parsing (we have used it in parsing xml converted from CRF), but also for its generation. A defined Python function to create "freetext" element is shown as below:

```
def FreeTextTag(annotation,position,pageIndex,textFont,textBoxBgColor=\
"#BFFFFF",textColor="#FF0000"):
    freeText=lxml.etree.Element("freetext")
    freeText.set("color",textBoxBgColor)
    freeText.set("flags","print")
    freeText.set("page",pageIndex)
    freeText.set("rect",position)
    richText=lxml.etree.SubElement(freeText,"contents-richtext")
    body=lxml.etree.SubElement(richText,"body")
    body.set("xmlns","http://www.w3.org/1999/xhtml")
    body.set("{xmlns}xfa","http://www.xfa.org/schema/xfa-data/1.0/")
    body.set("{xfa}APIVersion","Acrobat:11.0.0")
    body.set("{xfa}spec","2.0.2")
    body.set("style","font-size:"+textFont+" pt;"+\
    "text-align:left;color:"+textColor+\
    ";font-weight:bold;font-style:italic;"+\
    "font-family:Times-Roman;font-stretch:normal")
    p=lxml.etree.SubElement(body,"p")
    p.text=annotation
    p.set("dir","ltr")
    defaultStyle=lxml.etree.SubElement(freeText,"defaultstyle")
    defaultStyle.text="font:Times-Roman 10.0pt;"+\
    "text-align:left;color:#000000"
    return freeText
```

OUTPUT BOOKMARKED CRF FILE

We resort to PyPDF2 (https://pythonhosted.org/PyPDF2/, a Python package designed as PDF toolkit) for bookmark creation. From its documentation, we find class PyPDF2.PdfFileWriter has a method called "addBookmark"(https://pythonhosted.org/PyPDF2/PdfFileWriter.html). This method requires several parameters, the important ones include title (bookmark name), pagenum (equivalent to integer format of "PageIndex" in our visits or domain bookmark dataframes) and parent (parent bookmark for sub level bookmark creation). The returned value of this method is a reference for the bookmark it has created. Thus, it can be assigned to parent parameter when we need to create its sub level bookmarks.

We create a project name bookmark first (project name obtained in parsing xml converted from blank CRF step), which points to CRF first page (parameter pagenum=0). Then, we create sub level visits and domain bookmarks with their dataframes (Figure 12 and Figure 13). Example of generated bookmark is shown as below (Figure 15).

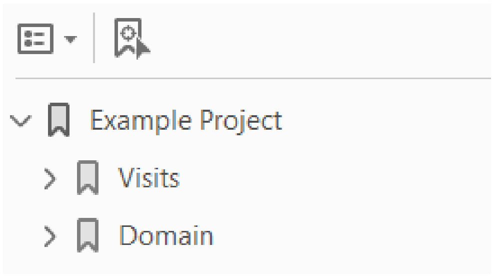
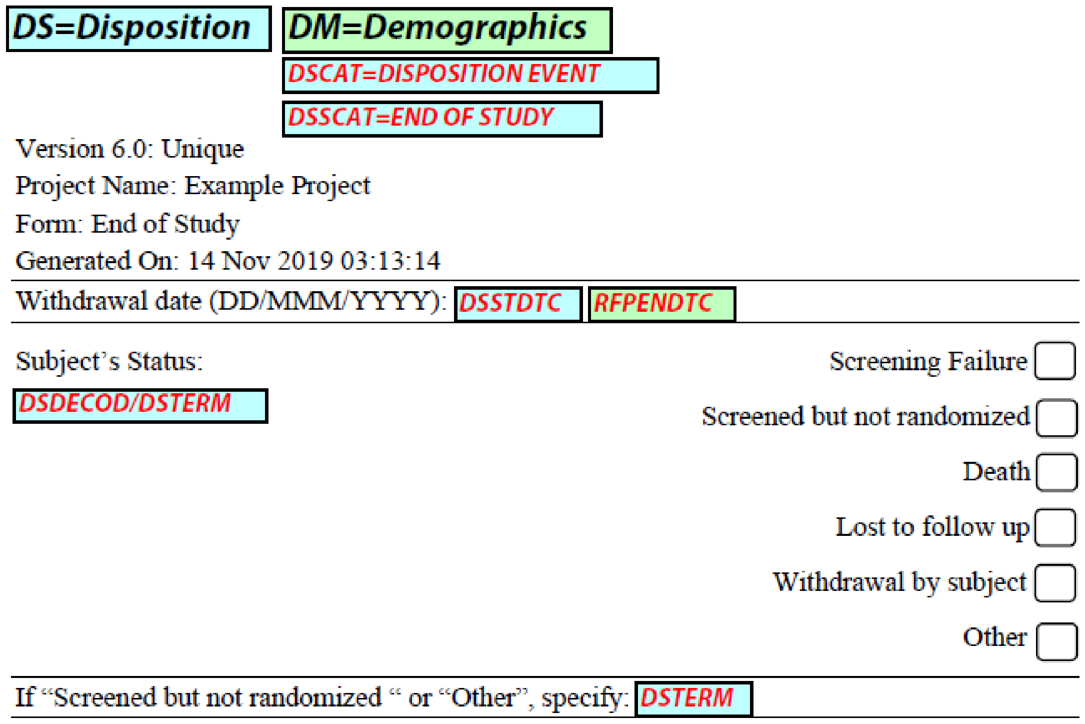


Figure 15. Example of Generated Bookmark for English CRF.

COMBINATION OF ANNOTATION FILE WITH BOOKMARKED CRF

Right now, we have an annotation XFDF file and a bookmarked CRF. We just need to import this annotation file into CRF (https://helpx.adobe.com/ca/acrobat/using/importing-exporting-comments.html) to get aCRF (Figure 16). However, we still need to go over this aCRF manually to get it tuned (e.g: some minor adjustments for the annotation text box).



DS=Disposition DM=Demographics

DSCAT=DISPOSITION EVENT

DSSCAT=END OF STUDY

Version 6.0: Unique

Project Name: Example Project

Form: End of Study

Generated On: 14 Nov 2019 03:13:14

Withdrawal date (DD/MMM/YYYY): DSSTDTC RFPENDTC

Subject's Status: DSDECOD/DSTERM

Screening Failure ☐

Screened but not randomized ☐

Death ☐

Lost to follow up ☐

Withdrawal by subject ☐

Other ☐

If "Screened but not randomized " or "Other", specify: DSTERM

Figure 16. Example of Annotated English CRF for Form End of Study

CHINESE VERSION CRF ANNOTATION

According to the latest regulation requirements in China, we also need to submit a Chinese version aCRF for review. Due to language character difference, Chinese and English CRF usually do not have exact same PDF page number and "PreText" coordinates for each form. Thus, we cannot simply import English CRF annotation XFDF file into Chinese CRF to get a Chinese aCRF. Instead, we should annotate directly on Chinese CRF. This sounds another tremendous work, but reality is not. All we need is to get a translated SDS from data management colleagues, which has following columns in Chinese: "DraftFormName" column in "Forms" sheet, "PreText" column in "Fields" sheet and "FolderName" column in "Folders" sheet. To determine whether displaying English words "Visits" and "Domain" or their Chinese translations in bookmark, we can detect if there is any Chinese character present in form name during parsing CRF converted xml. If the answer is yes, then we know input is a Chinese CRF and use Chinese translations (Figure 17). Otherwise, input should be an English CRF, and we should have English words "Visits" and "Domain" under project name bookmark (Figure 15). As English CRF form names should not have any Chinese character.

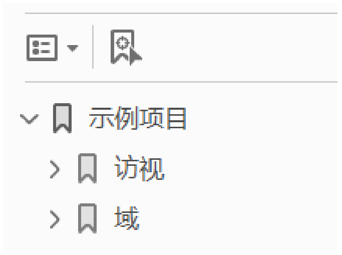
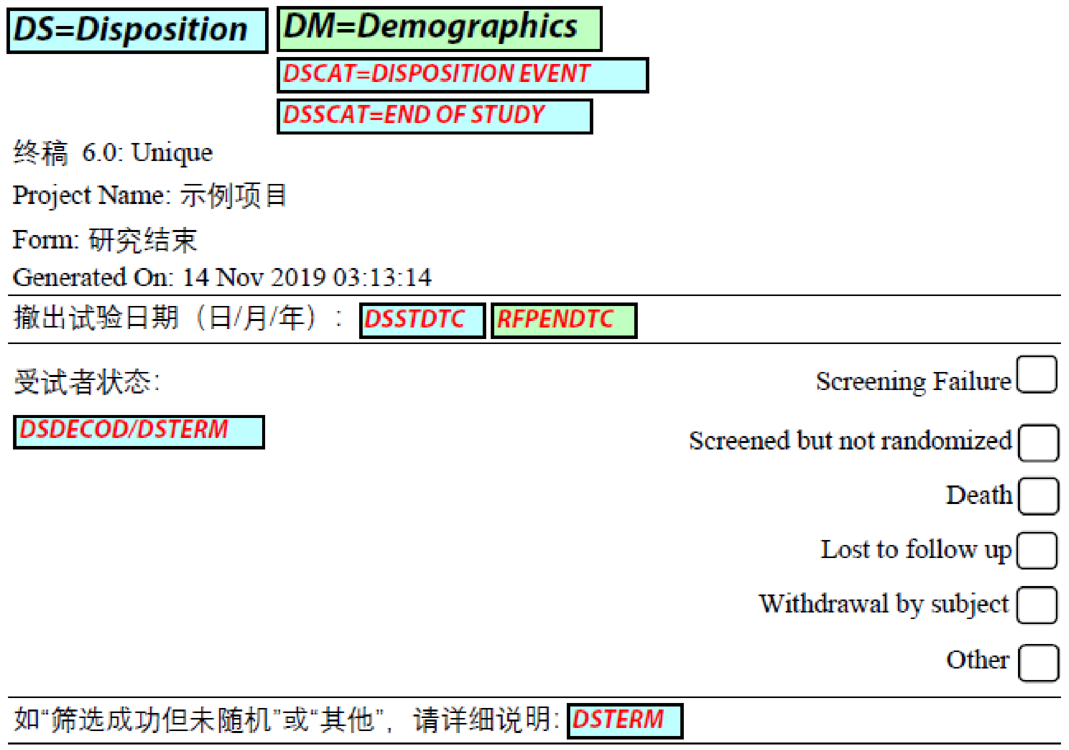


Figure 17. Example of Generated Bookmark for Chinese CRF

In a word, we can also use Chinese versions of CRF and modified SDS as inputs to get Chinese aCRF with the same automation workflow (Figure 18).



DS=Disposition

DM=Demographics

DSCAT=DISPOSITION EVENT

DSSCAT=END OF STUDY

终稿 6.0: Unique

Project Name: 示例项目

Form: 研究结束

Generated On: 14 Nov 2019 03:13:14

撤出试验日期（日/月/年）： DSSTDTC RFPENDTC

受试者状态:

DSDECOD/DSTERM

Screening Failure ☐

Screened but not randomized ☐

Death ☐

Lost to follow up ☐

Withdrawal by subject ☐

Other ☐

如“筛选成功但未随机”或“其他”，请详细说明: DSTERM

Figure 18. Example of Annotated Chinese CRF for Form End of Study

BUNDLE OF SCRIPT

Usually, programmers want to run this annotation script by themselves to get aCRF. However, this requires not only the installation of Python interpreter itself, but also all employed third party packages. To simplify this working stream, we bundle the script into one folder with all its dependencies and an executable file via Pyinstaller (https://www.pyinstaller.org/, a Python script bundle package). Then, users can access GUI of this annotation tool by just clicking this executable file.

As a best practice, during this bundle, we create an isolated virtual Python environment with virtualenv (a Python virtual environment builder package, https://virtualenv.pypa.io/en/latest/). In this designated Python environment, we only have all required packages installed.

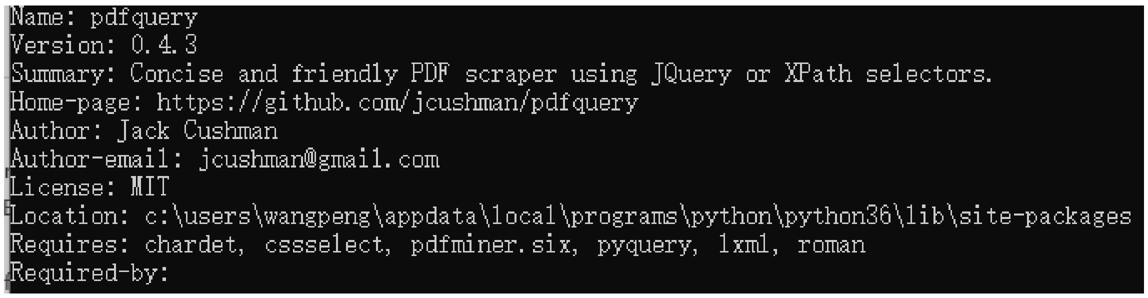
CONCLUSION

In this paper, we automate not only CRF annotation, but also its bookmark creation in the same process, which are time-consuming work if carried out in a manual way. What is more, this automation logic is flexible and applicable for both of English and Chinese aCRF preparations. Through this automation, we significantly reduce the manual burden and make it more consistent.

PYTHON AND ITS PACKAGES

Python version 3.6.8 is used for development in this paper.

To locate installed packages, command “pip show package_name” can be used, e.g: pip show pdfquery (Figure 19), package installation path is shown in “Location”.



```
Name: pdfquery
Version: 0.4.3
Summary: Concise and friendly PDF scraper using JQuery or XPath selectors.
Home-page: https://github.com/jcushman/pdfquery
Author: Jack Cushman
Author-email: jcushman@gmail.com
License: MIT
Location: c:\users\wangpeng\appdata\local\programs\python\python36\lib\site-packages
Requires: chardet, cssselect, pdfminer.six, pyquery, lxml, roman
Required-by:
```

Figure 19. Information for the Installed Python Package "pdfquery"

For package pdfquery version 0.4.3, as explained in previous conference paper (Muthukumar Hema et al, 2020), there is a bug in the source code in pdfquery.py. Original source code at line 271 is:

```
page_label = label_format['P']
```

The corrected version should be:

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

For package PyPDF2 version 1.26.0, to create a collapsed bookmark, per a discussion at Stack Overflow website (https://stackoverflow.com/questions/35878369/collapse-bookmarks-using-pypdf2), we need to adjust source codes in pdf.py and generic.py as below.

In pdf.py under PyPDF2, original source code at line 690 is:

```
   def addBookmark(self,title,pagenum,parent=None,color=None,\
bold=False,italic=False,fit='/Fit',*args):
```

Parameter collapse=False can be added:

```
   def addBookmark(self,title,pagenum,parent=None,color=None,\

bold=False,italic=False,fit='/Fit',collapse=False,*args):
```

In pdf.py under PyPDF2, original source code at line 747 is:

```
   parent.addChild(bookmarkRef,self)
```

Parameter collapse can be added:

```
   parent.addChild(bookmarkRef,self,collapse)
```

In generic.py under PyPDF2, original source code at line 665 is:

```
   def addChild(self,child,pdf):
```

Parameter collapse=False can be added:

```
   def addChild(self,child,pdf,collapse=False):
```

In generic.py under PyPDF2, original source code at line 677 is:

```
   self[NameObject('/Count')]=NumberObject(self[NameObject('/Count')]+1)
```

It can be changed as following:

```
   if collapse:
       self[NameObject('/Count')]=NumberObject(self[NameObject('/Count')]-1)
   else:
       self[NameObject('/Count')]=NumberObject(self[NameObject('/Count')]+1)
```

After these changes, we can call method "addBookmark()" with parameter "collapse=True" and generate collapsed bookmarks.

REFERENCES

Muthukumar, Hema. O'Brian, Kobie and Stofel, Julie. 2020. "Automating CRF Annotation using Python". PharmSUG 2020. Available at https://www.pharmasug.org/proceedings/2020/SS/PharmaSUG-2020-SS-159.pdf.

Pandas Dev Team. 2020. pandas version 1.1.5 https://github.com/pandas-dev/pandas/tree/v1.1.5

Cushman, Jack. 2020. pdfquery version 0.4.3 https://github.com/jcushman/pdfquery.

LXML Dev Team. 2021. lxml version 4.6.3 https://github.com/lxml/lxml/tree/lxml-4.6.3.

Fenniak, Mathieu. 2016. PyPDF2 version 1.26.0 https://github.com/mstamy2/PyPDF2

Kiehl, Chris. 2020. Gooey version 1.0.8 https://github.com/chriskiehl/Gooey/tree/1.0.8

Goebel, Hartmut. Bajo, Giovanni. Vierra, David. Cortesi, David and Zibricky, Martin. 2021. pyinstaller version 4.3 https://github.com/pyinstaller/pyinstaller/tree/v4.3

Gabor, Bernat. 2021. virtualenv version 20.4.6 https://github.com/pypa/virtualenv/tree/20.4.6

ACKNOWLEDGMENTS

I would like to thank review and comments from Hangjun Dai at Cstone Pharmaceuticals.

CONTACT INFORMATION

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

Peng Wang
Company: Cstone Pharmaceuticals
E-mail: wangpeng@cstonepharma.com