PharmaSUG 2021 - Paper QT-129

Easing the Process of Creating CRF Annotations

Samadi Karunasundera, University of California, Berkeley;
Jun Wang, Edwards Lifesciences;

ABSTRACT

According to FDA and CDISC guidance, annotated case report form (aCRF) is a mandatory document that need to be submitted to authorities. This paper addresses two ways of creating aCRFs. The first method involves the manual consolidation of distributed work on distinct CRFs into a single collaborative pdf file. The second method involves replicating the annotations programmatically by creating a dictionary of annotated standard forms which the program will use to match up the annotations to the correct CRFs.

INTRODUCTION

For a successful clinical trial submission, it's crucial to follow the guidance of FDA and CDISC, and submit all required documents, including annotated case report forms (aCRFs). However, the annotations of SDTM domains and their variables corresponding to data collected through CRFs could be a tedious process. This paper will elucidate on two ways of easing the process while also ensuring that annotation standards are followed.

INSTRUCTIONS FOR THE FIRST METHOD <MANUAL METHOD>

For this manual method multiple copies of the CRF book must be created. There can be a group which splits the CRFs copies and then assigns different individuals the task to annotate select CRFs. Then each individual would export his/her created annotations as fdf files; and then these fdf file copies must be manually imported to the matching CRF on the original master CRF book. The fdf files are text files that describe the annotations. They provide information regarding the annotations' content, locations, page number, etc.

Figure 1 Flow Chart of Manual Method

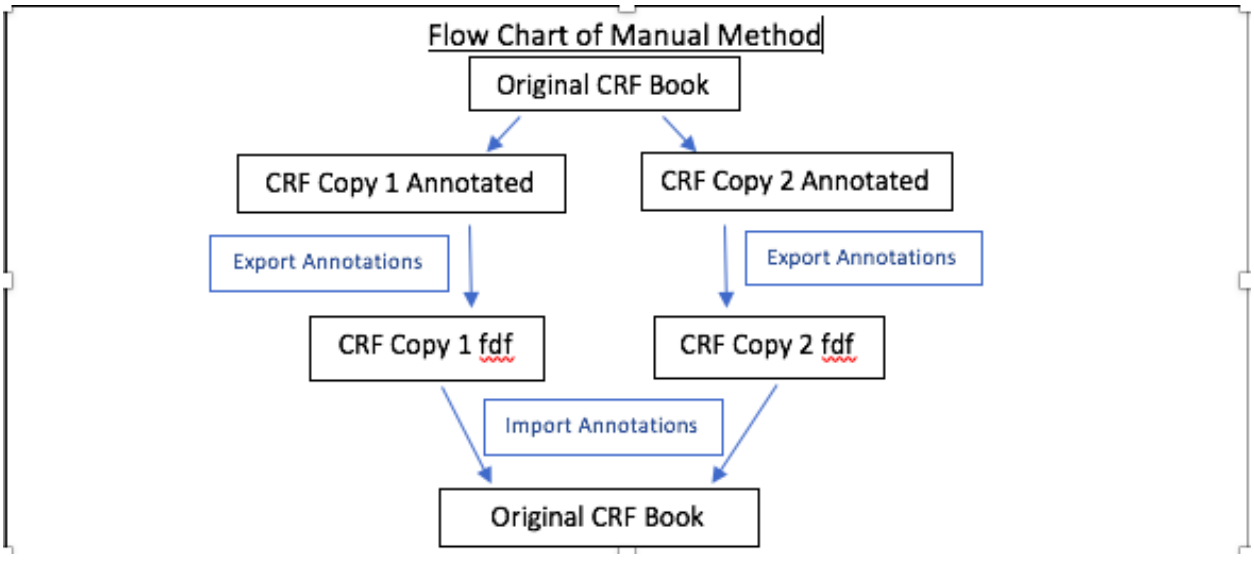

Figure 2. Caption for Flow Chart

Figure 1 serve as a visual aid on the process of importing and exporting annotated CRFs.

Figure 2 Manual CRF Exportation

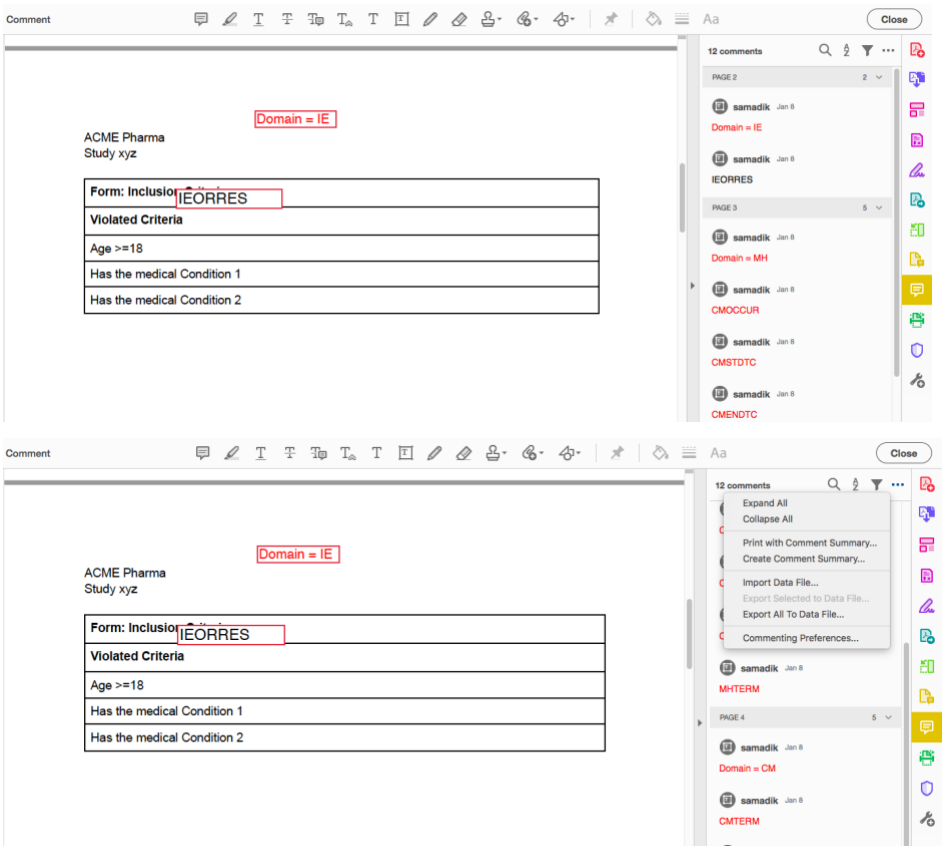

Figure 2. Caption for Manual CRF Exportation

Figure 2 serves as a visual aid for the exact locations of icons of icons required in the import/export process of fdf files specified under Direct Instructions.

DIRECT INSTRUCTIONS

Create an fdf file which contains the annotation descriptions in pdf mark.

It could be completed by steps below

1. click on the CRF's comment icon,
2. click options icon (...),
3. click export data file which will create the fdf file.
4. click on the unannotated CRF's comment icon,
5. click options icon (...) and select import data file and choose the newly created fdf file.

The annotations on this CRF will then match up exactly with the annotations from copies. If the CRFs are updated and new pages are added after the annotations, the annotations of the original version can still be imported into the updated version. First insert the new pages into the previously annotated version and follow the same approach to import the annotations from the original CRF into the updated CRF.

INSTRUCTIONS FOR THE SECOND METHOD <PROGRAMMATICAL METHOD>

Before Running the Program:

The programmatical method requires individual annotated standard forms. These multiple individual annotated standard forms may be collected from collaboration and/or the splitting of existing standard CRFs with annotations. An individual must then export the annotations of the standard forms as fdf files. Then the user should create a dictionary matching the fdf file name and CRF name. This program uses a dictionary object which is a collection of keyword and value pairs. The user must manually fill in a dictionary object which will be used to search for the existence of a keyword and its associated value. The creation of the fdf files and the dictionary would be a one-time occurrence, exemplified by the sample code's myMap object.

```
//create a dictionary containing the fdf files of the forms
const myMap = new Map([
["Demographic Information", "PSUGDM.fdf"],
["Concomitant Medication", "PSUGCM.fdf"],
["Inclusion Criteria", "PSUGIE.fdf"],
["Medical History", "PSUGMH.fdf"] ]);

//finds the matching CRF name to its fdf file
function createFormName() {
    var wloc = -1;
    Cpage = this.pageNum;
    haswordForm=0;
    numWords = this.getPageNumWords(Cpage);
    for (n=0; n < numWords; n++) {
        nextWord = this.getPageNthWord(Cpage, n);
        if (nextWord == "Form") {
            haswordForm=1;
            wloc = n;
            break;}
     }
     newWord = this.getPageNthWord(Cpage, wloc+1);
     Finder = newWord;
     for (i=0; i<5; i++) {
               console.println(Finder);
          if (myMap.has(Finder)) { break; }
              else { newWord = this.getPageNthWord(Cpage, wloc+i+2);
                        Finder=Finder + " "+ newWord; }
    }
    if (haswordForm==0) {Finder="Not a Form";}
    return Finder;
}
```

```
function makeAnnote(Cpage)

{
    finder = createFormName();

    //find and import the fdf file for the associated form
    if (myMap.has(finder)) {
          fdfname = myMap.get(finder);
          console.println("Fdf file name:" + fdfname);
          this.importAnFDF({cPath:fdfname});
          //transferring annotations on page 0 to current page
          annote = this.getAnnots( {nPage : 0} );
          for (var a=0; a<annote.length; a++)
                annote[a].page = Cpage;
    }
    else {console.println("no fdf found for this page" + Cpage);}

}

//ignores the first page
for (p=1; p< this.numPages; p++) {
        this.pageNum = p;
        makeAnnote(p);
}
```

RUNNING THE PROGRAM

The programmatical method requires opening the aCRF in Adobe Acrobat, not Acrobat Reader. Copy the program above into the Javascript console; remember to modify the dictionary to suit your fdf file names. The Javascript console can be accessed via “Control + J”. Select all and press “Control + Enter” to run the entire program on the unannotated blankCRF book. The program will import the corresponding fdf files to the matching CRF onto the blankCRF book resulting in a fully annotated aCRF book, provided that the annotations are at most 1 page long in length.

Program Explanation <Modification Possibilities>:

The program scans for the word “Form” in a PDF and then look for the next consecutive word(s) being the form’s name which would be utilized as a keyword for the matching fdf file name. You may alter the createFormName() function to find the correct CRF’s name i.e. changing the initial search term from “Form” to “CRF”. The program then imports the matching fdf file name initially onto the cover page and then moves the annotation onto its corresponding standard form’s name on the unannotated Study CRF Book. To modify the program to handle annotations which exceed one page in length, the user must add an extra cover page to the blank CRF book. This modification will only work for CRFs with a maximum of 2 pages of annotations.

```
//Below specifies alteration to the existing program function if a CRF has at
most 2 pages of annotations
function makeAnnot(Cpage)
{
    finder = createFormName();
    //find and import the fdf file for the associated form
    if (myMap.has(finder)) {
          fdfname = myMap.get(finder);
          console.println("Fdf file name:" + fdfname);
          this.importAnFDF({cPath:fdfname});
          //transfer annotations on page 0 to current page
          annot = this.getAnnots( {nPage : 0} );
          if (annot!=null) {
          for (var a=0; a<annot.length; a++)
                annot[a].page = Cpage;   }

          // check if the second cover page is annotated
          annot=this.getAnnots({nPage:1});
          if (annot!=null) {
          this.pageNum=Cpage+1;
          nextpage= createFormName();
                if (nextpage==finder){
                    for (var a=0; a<annot.length; a++) {
                       annot[a].page = Cpage+1;}
                 }
                  else {
                    for (x=0; x< annot.length; x++) {
                         annot[x].destroy(); }
             }
       }
          // the second page is not annotated
          else {this.pageNum=Cpage;}
    }
    else {console.println("no fdf found for this page" + Cpage);}
}
// the previous form name
prevForm="Last Form";
//ignore the cover pages
for (p=2; p< this.numPages; p++) {
        this.pageNum = p;
```

```
        makeAnnot(p);
        // there may have been a CRF with two pages
        p=this.pageNum;
          // delete the Annotations in page 0 and page 1 (if any)
         for (d=0;d<2;d++){
            var annots = this.getAnnots({nPage: d});
            if (annots!=null && annots.length!=0) {
               for (x=0; x< annots.length; x++)
                   annots[x].destroy();
               }


    }
}
```

Figure 3 Flow Chart of Programmatical Method

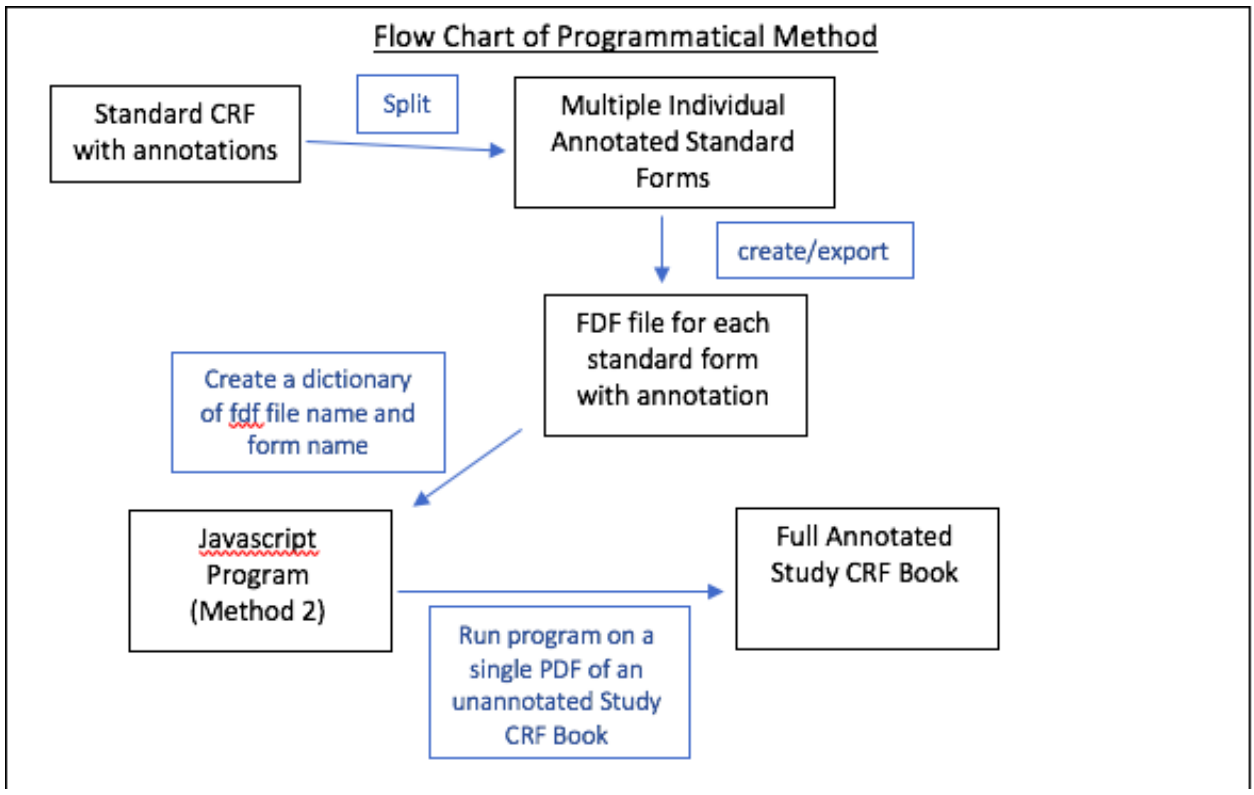

Figure 3. Caption for Flow Chart

Figure 3 serves as a visual aid on the process of importing and exporting annotations using the programmatical method.

ADDITIONAL FEATURES <PROGRAM EXTENSION>

The function destroyAnnot() specified below may be useful when utilized in conjunction with the previous programmatical methods encoded above.

```
Bonus Features of Javascript Program


//destroys all the annotations on a single pdf
function destroyAnnot() {
    this.syncAnnotScan();
    for (d=0; d< this.numPages; d++) {
//change d to current page if you want to just delete the annotations of current page
//& add break to exit the loop
        this.pageNum = d;
        var annots = this.getAnnots();
        if (annots!=null && annots.length!=0)
        for (x=0; x< annots.length; x++) {
annots[x].destroy();
        }
    }
}
```

CONCLUSION

This paper addressed 2 ways to create an annotated CRF. The first method involves the manual consolidation of distributed work, while the second method involves programmatically annotate to the correct CRFs. Although some sort of manually work is needed, both ways could serve as efficient ways to create aCRF. The programs featured in this paper are based upon the sample methods outlined in JavaScript for Acrobat API Reference**.**

RECOMMENDED READING

- ***JavaScript for Acrobat® API Reference***

CONTACT INFORMATION <AUTHORS >

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

Samadi Karunasundera
University of California, Berkeley
sambak17@berkeley.edu

Jun Wang
Edwards Lifesciences
Jun_wang@edwards.com