Paper DH12

Semi-automating Case Report Form Annotations using Python and a Metadata Repository

Alvin Mathew, Alnylam, Cambridge, MA, USA
Ella Chee, Alnylam, Cambridge, MA, USA

ABSTRACT

Accurate annotated case report forms (aCRF) are critical for SDTM development and submissions, yet the existing time-consuming and error-prone manual process is at odds with the needs of the fast-paced study start-up phase. The usual practice requires manually creating individual text boxes, changing fonts and colors, and checking adherence to guidelines, including Metadata Submission Guidelines (MSG), SDTM Implementation Guide, and applicable company standards, across potentially hundreds of pages. To streamline this process, we developed a tool to semi-automate CRF annotation. By leveraging Python, the study's architect loader specification, and our company's internal metadata repository, our tool automatically applies the MSG requirements to annotations filled in a control spreadsheet. This spreadsheet is prepopulated with our standard annotations and allows multiple users to fill in the remaining study-specific annotations concurrently. Altogether, this tool allows for faster turnarounds, better adherence to standards, and enhances multi-user collaboration.

INTRODUCTION

CRFs show how and what data are collected throughout the course of a study and serve as the primary reference for understanding the raw data received. Annotating these forms is important for both SDTM programming and meeting regulatory submission requirements. Despite their importance, aCRFs are usually produced through a manual process that demands significant time, meticulous attention to detail, and a deep understanding of CDISC standards. Teams must work page by page across numerous forms and potentially thousands of annotations – creating text boxes, ensuring correct mapping to SDTM variables, and verifying adherence to multiple guidance documents, including the MSG, SDTMIG, and internal company standards.

This manual effort can become a bottleneck during the study start-up phase or a frenzy at database lock, periods already characterized by tight timelines and competing deliverables. The labor-intensive nature of the work increases the risk of inconsistencies and errors, especially in large studies with complex forms. To address these challenges, we sought to semi-automate the most tedious and repetitive aspects of annotation process, where agreed upon standards are consistently applied, allowing programmers to focus on reviewing and resolving study-specific scenarios.

CRF ANNOTATION TOOL

OVERVIEW

In developing the tool, we first consolidated all relevant sources of information, including the study CRF and architect loader specification (ALS), MSG, SDTMIG, and internal company standards stored in our metadata repository (MDR). We then leveraged Python to extract structured content from the CRF PDF and convert it into a spreadsheet that supported quick updates and multi-user collaboration. We parsed form names, page numbers, question and answer text, and positional coordinates for each item on every page. In addition, we matched standard annotations from our MDR based on text and ALS IDs. With these elements organized, we implemented an algorithm that could identify available whitespace and automatically place the annotations in appropriate locations.

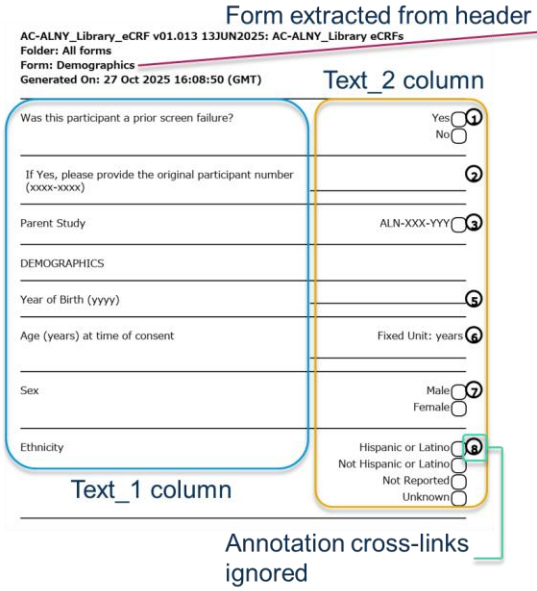
EXTRACTING CRF INFORMATION

In order to work with the CRF, usually provided as PDFs, we extract text using the Python package pdfminer's *extract_pages* function to iterate over each page and collect text, bounding box coordinates, and page numbers, while excluding technical metadata such as EDC IDs (**Figure 1**). This provides a single list of values per page. Text boxes that share a common vertical coordinate are then grouped together to reconstruct text lines across columns. The coordinates of both question-and-answer columns are consolidated and raw information not used for annotations are filtered out. The form names are also extracted per page from the header, ensuring each row is associated with the correct form.

Occasionally, the scraped text can put more than one question or choice into a single cell, so we used rule-based checks to ensure that multiline text blocks are split or merged correctly, preserving semantic groupings such as continued lines, specifications, or parenthetical text. Header and footer rows are explicitly detected and excluded from splitting. Because this splitting occurs post-scrape, the y-coordinates for the bounding box are re-calculated for newly created rows to ensure accurate spatial positioning.

All steps are handled by a single execution function, which logs timing data and progress labels throughout. The result is a Python pandas dataframe representative of the CRF PDF, where each row contains the corresponding page number, form name, text fields, and coordinates for text boxes, as well as placeholder columns for annotations.

Figure 1. Extracting CRF information.



Page	Form	Text_1	Text_2	Coord_1	Coord_2
8	Form: Demographics	AC-ALNY_Library_eCRF v01.013 13JUN2025: AC-ALNY_Library eCRFs Form: Demographics Generated On: 27 Oct 2025 16:08:50 (GMT)		90.0, 650.0, 461.3, 702.0	
8	Form: Demographics	Was this participant a prior screen failure?	Yes	90.0, 613.0, 296.4, 623.0	489.0, 613.0, 506.0, 623.0
8	Form: Demographics		No		492.7, 597.0, 506.0, 607.0
8	Form: Demographics	If Yes, please provide the original participant number (xxxx-xxxx)		95.0, 546.0, 354.9, 568.0	
8	Form: Demographics	Parent Study	ALN-XXX-YYY	90.0, 511.0, 153.0, 521.0	438.5, 511.0, 506.0, 521.0
8	Form: Demographics	DEMOGRAPHICS		90.0, 472.0, 170.9, 482.0	
8	Form: Demographics	Year of Birth (yyyy)		90.0, 437.0, 186.2, 447.0	
8	Form: Demographics	Age (years) at time of consent	Fixed Unit: years	90.0, 400.0, 239.1, 412.0	436.2, 400.0, 533.2, 412.0
8	Form: Demographics	Sex	Male	90.0, 349.0, 107.7, 359.0	483.6, 349.0, 506.0, 359.0
8	Form: Demographics		Female		470.8, 333.0, 506.0, 343.0
8	Form: Demographics	Ethnicity	Hispanic or Latino	90.0, 294.0, 131.9, 304.0	418.8, 294.0, 506.0, 304.0
8	Form: Demographics		Not Hispanic or Latino		398.3, 278.0, 506.0, 288.0
8	Form: Demographics		Not Reported		441.1, 262.0, 506.0, 272.0
8	Form: Demographics		Unknown		460.4, 246.0, 506.0, 256.0

Text and coordinates are parsed from the PDF using pdfminer's extract_pages while ignoring raw variable annotation cross-reference links. Form names are pulled from header text and CRF content is logically arranged between two columns in Python.

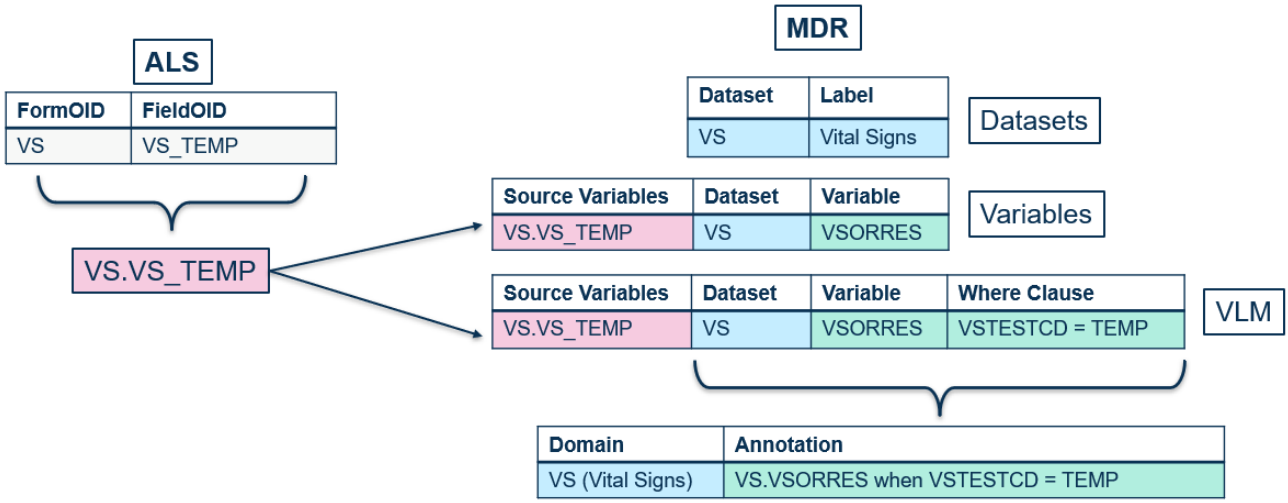
PRE-POPULATING ANNOTATIONS

The MDR system contains our internal SDTM specifications and mappings and is maintained by our data standards team. The Variables and ValueLevel sections contain the usual information like SDTM variables, type, and codelists, but also have a column for source raw variable from the EDC.

The study ALS (Architect Loader Specification) is used to build the EDC. It contains a Forms sheet, which holds information on form names (FormOIDs or "Form Object Identifier"), a Fields sheet holding field variables (FieldOID or "Field Object Identifier"), and CRF text (PreText and SASLabel columns).

Combining these two sources allows us to automatically map specific CRF text to our SDTM standards (**Figure 2**). Processing is done to normalize both text (e.g., set to lower case, replace newline characters, remove leading/trailing blanks) and Form/FieldOIDs (e.g., remove vendor-specific prefixes), and pre-population occurs as a text-based and ID-matching procedure. Source variables are pulled from the MDR raw variables list to create unique dataset, variable, and source variable rows. Then, Form/FieldOIDs are extracted from the source variable value, which is structured as "<FormOID>.<FieldOID>" (e.g. "DM.AGE" is expanded to FormOID: DM, FieldOID: AGE). A series of pandas merges is performed on the Form/FieldOID columns to map possible dataset/variable annotations from the MDR to the ALS. Value-level annotations are further processed based on certain conditions within the "where clause" column. As vendors may have their own internal standards for EDC development, text-based pre-population is also implemented, merging the ALS and MDR dataframes on form names and CRF text instead of the Form/FieldOIDs.

Figure 2. The general process for mapping annotations from the MDR to the study ALS.



FormOID/FieldOIDs are matched between the MDR and ALS, which allows the CRF text in the ALS to be matched to the SDTM variable in the MDR. Similar matching occurs between CRF text and global MDR text.

At this point, the pre-populated annotation dataframe contains formatted SDTM variable annotations, Form/FieldOIDs, and CRF text. This information is merged on text and form columns back to the recreated CRF dataframe. Post-processing clean-up, such as adding extra instruction text, is carried out, and any duplicate annotations are also removed. Finally, the dataframe is exported as a control spreadsheet.

ANNOTATION CONTROL SHEET

The annotation control sheet holds the recreated CRF and the pre-populated annotations (**Figure 3**). The user can then modify these annotations as needed, as well as add any new annotations manually. Domain codes can be placed in a domain column and cascaded to all annotations in the row, or in a dot-notation style if multiple domains are needed for a single row. The tool cross-checks the sheet is filled correctly, flagging if domains are missing, multiple domains are used for one variable, or if the domain does not exist in the MDR standards. If errors are detected, a domain status column will be appended to the control sheet, indicating the problem-row and offering guidance on how to fix the issues.

Users can also follow a set of rules for specially formatted annotations:

1. Italicized cells will be formatted with a dashed border style. The script internally creates a flag for each annotation column within the row for downstream use in applying annotations.
2. Domains left blank with annotation text as “[NOT SUBMITTED]” will be formatted with a gray box.
3. Rows with “xxORRES” and “xxTESTCD = …” are formatted as (“xxORRES when xxTESTCD = …”).

Figure 3. Example of annotation control sheet.

page	form	text_1	text_2	coord1	coord2	domain	anno1	anno2
13	Form: Demographics	Was this participant a prior screen failure?	Yes	90.0, 627.01, 296.43, 637.01	488.95, 627.01, 506.0, 637.01		DM.RESCREEN in SUPPDM	
13	Form: Demographics		No		492.66, 611.01, 506.00000000000006, 621.01			
13	Form: Demographics	If Yes, please provide the original participant number (ssxxxx-xxxx)		95.0, 560.01, 354.8500000000001, 582.01			DM.PSUBJID in SUPPDM	
13	Form: Demographics	DEMOGRAPHICS		90.0, 525.01, 170.91, 535.01				
13	Form: Demographics	Year of Birth (yyyy)		90.0, 490.01, 186.17000000000002, 500.01			DM.BRTHDTC	
13	Form: Demographics	Age (years) at time of consent	Fixed Unit: years	90.0, 453.01, 239.07, 465.01	436.18, 453.01, 533.21, 465.01		DM.AGE	
13	Form: Demographics	Age unit	Years	95.0, 402.01, 135.66, 412.01	479.02, 402.01, 506.0, 412.01		DM.AGEU	
13	Form: Demographics	Sex	Male	90.0, 363.01, 107.66, 373.01	483.6, 363.01, 506.0, 373.01		DM.SEX	
13	Form: Demographics		Female		470.83, 347.01, 505.99999999999994, 357.01			
13	Form: Demographics	Is the participant of childbearing potential?	Yes	95.0, 308.01, 304.3300000000001, 318.01	488.95, 308.01, 506.0, 318.01		RP.RPORRES when RPTESTCD = CHILDPOT	
13	Form: Demographics		No		492.66, 292.01, 506.00000000000006, 302.01			

The coordinate information is hidden in the final file and users can fill out the domain and annotations for specific rows as necessary. CRF fields that match the MDR standards will have their annotations prefilled in the Excel sheet.

CALCULATING COORDINATES

A major feature of the program is the ability to programmatically place annotations in whitespace on the page. This process is split into two steps: handling domain annotations placed at the top of the page (within a domain dataframe) and managing in-text annotations throughout the page (annotation dataframe).

The Python Imaging Library (PIL) provides an *ImageFont* module to load a specified TrueType font with a certain size (e.g., Arial, size 12). The *getlength* function returns the width of a given string in "points" (=1/72 inch, an 8.5x11 inch page is 612x792 points) and is used to calculate the width for domain text (bold style, larger size font), and annotation text (regular style, size 9-12). Each annotation gets a point width and vertical bounds are based on font height.

The domains are laid across the header margin, wrapping as needed. First, default horizontal bounds are established (10-600 points), a vertical anchor at the top of the page is fixed, and the row height for wrapping is established. Then, for each row, domain labels are placed in order from left to right and the width is totaled one at a time. If the next domain label exceeds the rightmost boundary, a new row is needed, and placement continues below by shifting y-coordinates down by the domain annotation height and starting from the left bound again. These final coordinates are stored for each domain label within the domain dataframe.

The process for calculating annotation text box coordinates is slightly more complicated (**Figure 4**). An initial step is required to establish constraints for coordinates. First, existing CRF text boxes are parsed to get left and right boundaries. For each row, the number of annotations and their total width are computed, along with available space between left and right text content. Each row is also classified into two layout types: TYPE_L where annotations flow from a left anchor (question only text) or TYPE_R, where annotations must be packed from the right side (answer text).

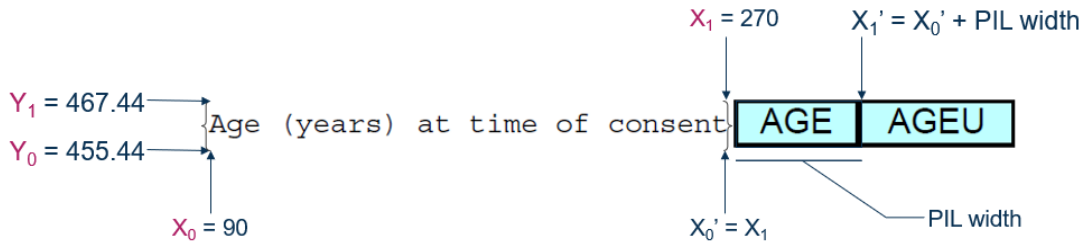
Next, collision detection is carried out to determine whether annotations fit on a single line or must wrap to the next. Annotation widths are added sequentially with spacing, and if the running total exceeds the allowed horizontal range, the index of the first overflowing annotation is recorded, marking where the layout must shift to a new row.

With these components, final coordinates can be assigned to each annotation box using the following logical process:

```
For each row in the dataframe:
   1. Initialize prev_box_right = NULL
   2. Set y-coordinates (y0, y1) above or below source text
   3. Determine collision_index (first annotation that does not fit in row)
   4. Determine layout_type (left-anchored or right-anchored)
   5. If layout_type == "TYPE_R":
          a. Compute starting at the leftmost position to align annotations relative
             to the right boundary
   6. For annotation_index in range(i, j):
          a. If annotation width is 0 or missing:
                i. Clear all coordinates for this annotation
               ii. Continue to next
          b. If annotation_index == collision_index and vertical shift not applied:
                i. Apply vertical shift (move y0 and y1 down to next row)
               ii. Mark as adjusted
          c. Assign vertical coordinates (y0, y1)
          d. If annotation_index is the first annotation:
                i. Place box at leftmost boundary
          e. Else if a previous annotation exists:
                i. Place box immediately after the previous box with spacing
          f. Update prev_box_right = this box's right edge
   7. If a collision occurred:
          a. Collect all annotations from collision_index onward
          b. Evenly divide available page width across remaining annotations.
          c. For each remaining annotation:
                i. Center the annotation box within its given segment
               ii. Assign new left and right coordinates
```

Figure 4. Example calculation for annotation placement.

page	text_1	coord1	domain	anno1	anno2
15	Age (years) at time of consent	90.0, 455.44, 270.0, 467.44	DM	AGE	AGEU



The Y-coordinates for an annotation would remain the same as the row's CRF text (adjusted for font height) and the X-coordinates would be the end of the CRF text box plus the width of the text as determined by PIL. These coordinates are saved and applied in the final step.

APPLYING ANNOTATIONS

For this step, the tool requires both the CRF PDF and the annotated control sheet (**Figure 5**). The control sheet is read into a pandas dataframe and domains/annotations are cleanly parsed into values based on the supplied domains. Once these values are collected, domains must be assigned a color based on the CDISC SDTM-MSG. The MSG outlines four initial colors that are colorblind-accessible (blue, yellow, green, and orange) and six more were added to handle complex forms with several domains.

An ordered unique list of domains for each form is created to preserve the appearance order throughout the form. Then, domains within a form are mapped to colors from the MSG sequence, ensuring that "SUPPxx" domains inherit the parent domain color, and empty domains are gray. Form-level color dictionaries are then attached to each row in a new column. To create page-based domain color maps, rows are grouped by form and page, and colors are assigned. Then, only domains containing annotations on that page are mapped to the corresponding color within the form color order. This form-level color map is used to assign colors to each annotation. Finally, an execution function carries out these processes in sequence, creating one dataframe of annotations with colors assigned, and another of unique form, page, and domain combinations with color and form order. "XX" domains that existed as placeholders are removed from the dataframe.

Figure 5. Assigning colors.

Input dataframe

Page	Form	Domain	Anno1	Anno2
1	Form: Participant		XX.[NOT SUBMITTED]	DM.SUBJID
2	Form: Demographics		DM.BRTHDTC	DC.BRTHDTC
2	Form: Demographics	DS	DSCAT = PROTOCOL MILESTONE	
2	Form: Demographics	RP	RPORRES when RPTESTCD = CHILDPOT	
3	Form: Demographics	DS	PROTVERS in SUPPDS	

Create form-based domain order

Assign MSG colors to domain order

Form	Order	Order
Form: Participant	DM	DM
Form: Demographics	DM, DC, DS, RP	DM, DC, DS, RP

Distribute to page color maps

Page	Form	Color
1	Form: Participant	DM
2	Form: Demographics	DM, DC, DS, RP
3	Form: Demographics	DS

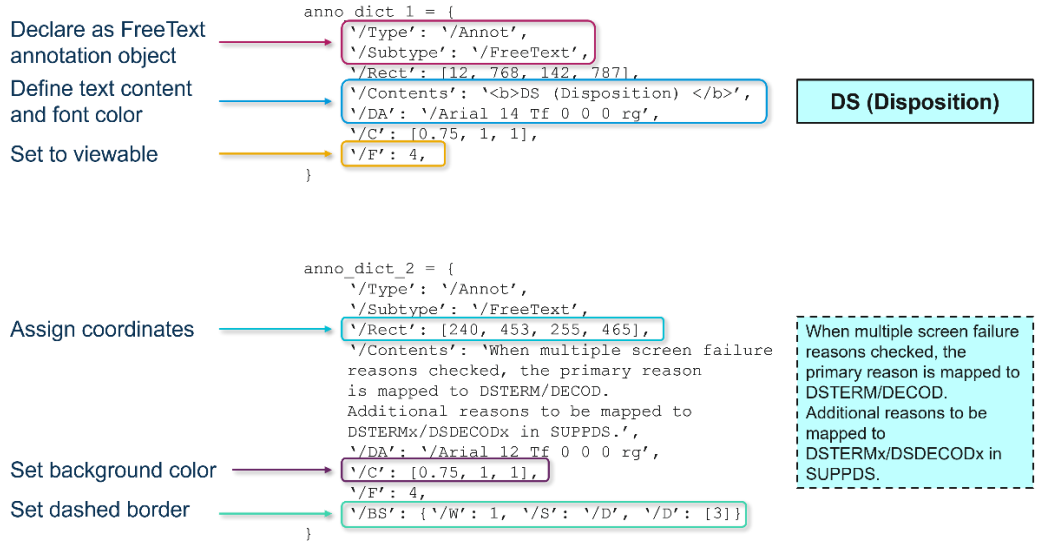
Page	Form	Anno1	Anno2	Color1	Color2
1	Form: Participant	[NOT SUBMITTED]	SUBJID	XX	DM
2	Form: Demographics	BRTHDTC	BRTHDTC	DM	DC
2	Form: Demographics	DSCAT = PROTOCOL MILESTONE		DS	
2	Form: Demographics	RPORRES when RPTESTCD = CHILDPOT		RP	
3	Form: Demographics	PROTVERS in SUPPDS		DS	

Apply colors to each annotation

To ensure that domain colors are consistent across different pages of the same form, domain-color mappings are created on a form, then page-basis.

Now, annotations can be exported directly to the CRF (**Figure 6**). For each annotation, a dictionary is created containing text content, text box coordinates, color assignments, font, and border styling. These annotation dictionaries are compiled into lists for each page. CRF pages are first read from the blank CRF using the pypdf *PdfReader* function and added into a new PDF with the *PdfWriter add_page* functionality. Bookmarks are preserved by recursively copying parent and children outline items from the CRF, using the *PdfWriter add_outline_item* function. Finally, annotations are added by page, using the *PdfWriter add_annotation* function. At this point, a new annotated CRF copy is created, containing SDTM-MSG-compliant annotations.

Figure 6. Breakdown of annotation dictionary used to create and format PDF annotations.




```
anno_dict_1 = {
    '/Type': '/Annot',
    '/Subtype': '/FreeText',
    '/Rect': [12, 768, 142, 787],
    '/Contents': '<b>DS (Disposition) </b>',
    '/DA': '/Arial 14 Tf 0 0 0 rg',
    '/C': [0.75, 1, 1],
    '/F': 4,
}
```

```
anno_dict_2 = {
    '/Type': '/Annot',
    '/Subtype': '/FreeText',
    '/Rect': [240, 453, 255, 465],
    '/Contents': 'When multiple screen failure
    reasons checked, the primary reason
    is mapped to DSTERM/DECOD.
    Additional reasons to be mapped to
    DSTERMx/DSDECODx in SUPPDS.',
    '/DA': '/Arial 12 Tf 0 0 0 rg',
    '/C': [0.75, 1, 1],
    '/F': 4,
    '/BS': {'/W': 1, '/S': '/D', '/D': [3]}
}
```

TRANSFERRING ANNOTATIONS

Another major feature of the annotation tool is the ability to transfer annotations from an annotated source CRF to a blank destination CRF (**Figure 7**). This is valuable when mid-study updates occur, providing an alternative to starting from scratch or copying and moving old annotations when text fields or pages are added/removed. Some annotations are moved after the initial automatic placement, and this new location is preferred to be kept in future CRF versions.

There are three key steps in this process:

1. Extract CRF information from the blank CRF into a dataframe.
2. Extract CRF information and annotation information from the aCRF into a dataframe.
3. Match the blank CRF dataframe to the aCRF dataframe to populate annotations.

Step 1. As described in the **Extracting CRF Information** section, the extract CRF class handles extracting page, form, text, and coordinates into a *pandas* dataframe from a given PDF. The blank CRF is brought cleanly into a dataframe.

Step 2. A second class, called AnnoCRF, deals with pulling annotations from the aCRF and matching them to aCRF text. aCRF text is extracted using the CRF class, then annotations are pulled from the aCRF using pypdf to parse annotation details (i.e., coordinates, contents, font and text box formatting) from each page. To reverse engineer annotations to match the style of control sheet contents, domain abbreviations must be extracted and assigned. To achieve this goal, annotations of the same color are grouped, domain annotations are identified (by different font styling), and domain abbreviations pre-pended to annotations (e.g., domain annotation = “DM (Demographics),” variable annotation = “AGE,” formatted annotation = “DM.AGE”). Occasionally, annotations can be manually added to the PDF and colors may not be an exact match. In this case, annotations are instead scanned for the domain abbreviation (e.g., domain abbreviation = “RP,” variable annotation = “RPORRES,” formatted annotation = “RP.RPORRES”).

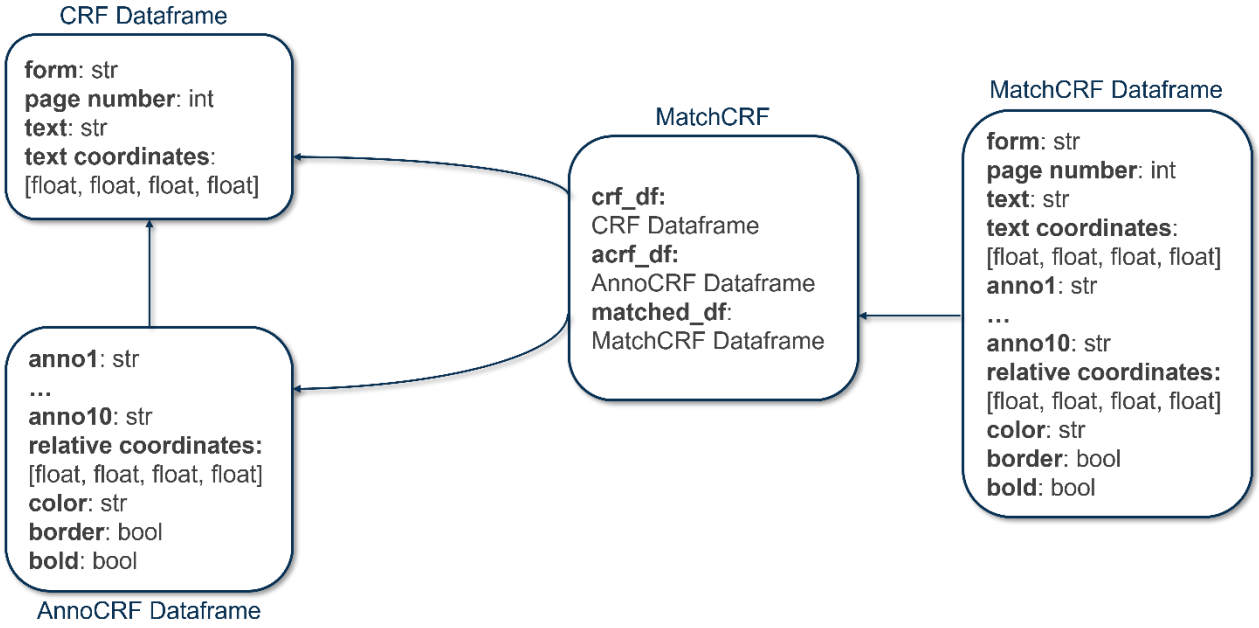
Step 3. Once these annotations are formatted, they must be matched to text based on coordinates. Given text and annotation coordinates, a match is determined if alignment exists directly below, inline, or above the text (based on the SDTM-MSG assumption that annotations are closely located to text fields). To ensure that only annotations present on a specific page of a form are matched to text, rows from the CRF dataframe and annotation dataframe are grouped by page number and form name. Each group is iterated over, matches are attempted, and annotations assigned accordingly. Any annotations that fail to match are grouped by form, compiled into leftover lists, and placed

in an “unmatched” column in the header space of the form. Multiple annotations in a row are distributed across the control sheet columns. The result is a populated dataframe containing CRF page number, form name, text fields, text coordinates, annotations, annotation formatting information, and lists of unmatched annotations for each form.

Next, this populated dataframe must be matched to the destination blank CRF. This functionality is encapsulated within a third class, called MatchCRF, which takes the source aCRF populated dataframe, and the destination blank CRF dataframe. To ensure that annotations placed in header/footer rows can be matched, processing is carried out to identify headers in the aCRF dataframe (containing keywords, like “Form:” and “Generated On:”) and reformat to match the destination. Since duplicated text can appear repeatedly throughout the same form, a variable is created to track ordering and assigned to each dataframe. A simple *pandas* merge is used to join the dataframes on form name, text, and ordering variable columns. Relative coordinates are calculated by taking the difference of annotation and text coordinates for each annotation. Unmatched annotations from lists within the “unmatched” column are distributed down a row to allow users to easily copy/paste into annotation columns or delete as needed. Lastly, the dataframe is exported to Excel as a populated control sheet.

If the control sheet containing relative annotation coordinates from a Transfer step is passed to the script and these newly adjusted coordinates properly fit the bounds of the page as well as the length of the annotation text, they will be preferred over calculated coordinates.

Figure 7. General overview of the class organization.



The AnnoCRF dataframe builds from the CRF dataframe, supplying annotation and annotation formatting information. The MatchCRF class merges the destination CRF dataframe with the source AnnoCRF dataframe.

RESULTS

The CRF Annotation Tool improves efficiency, standardization, and reproducibility in a few key ways:

1. By combining text-based and ID-based pre-population of annotations from MDR, the tool streamlines the annotation process. The MDR allows the Standards team to centrally manage updates to SDTM datasets and variable annotations, ensuring alignment with CDISC standards. Pre-populating standard forms (e.g., Demographics, Medical History) significantly reduces the programmer’s workload, leaving only study-specific annotations for manual entry.
2. CRFAT applies annotations programmatically using algorithmic placement, color coding, font, and border styling. This automation saves substantial time compared to manual formatting – thousands of annotations can be processed in minutes rather than hours. As a result, programmers can focus on ensuring standards compliance rather than performing repetitive, tedious formatting. This approach also guarantees consistency and reproducibility across studies.
3. The ability to transfer annotations from an aCRF to a blank CRF is particularly valuable for mid-study updates or initiating subsequent study phases. Instead of re-annotating from scratch, this functionality enables rapid reuse of approved annotations, as most forms remain unchanged. This reduces review cycles with the Standards team and accelerates study startup.

FUTURE IMPROVEMENTS

While the current CRFAT implementation significantly improves efficiency compared to manual annotation, several enhancements could further optimize functionality:

The current MatchCRF class relies on exact text matching when transferring annotations from aCRF to a blank CRF. This approach can fail during mid-study updates where text fields or pages are added, removed, or rephrased. Incorporating NLP-based techniques (e.g., cosine similarity) or AI-driven smart matching would enable more robust alignment of aCRF text with CRF text. This enhancement would also improve form name matching, which currently depends on rule-based reductions (e.g., stripping “Vital Signs – Timepoints” to “Vital Signs”).

The CDISC SDTM-MSG recommends a default font size of 12pt, but CRF pages can become overcrowded due to numerous or lengthy annotations. Allowing users to specify font size (9-12pt) for individual pages via the control sheet would improve readability. Additionally, automating font adjustments for forms with known layout constraints could further streamline the process.

Consistent formatting reduces annotator workload and improves clarity. While some automatic formats exist (e.g., “when” conditions for original result annotations, “Datepart”/ “Timepart” prefixes), expanding these standards would be beneficial. Similarly, automating border styles and other formatting for known annotation types would save time and ensure compliance with style guidelines.

CONCLUSION

The CRF Annotation Tool addresses critical inefficiencies in the traditional manual annotation process by introducing automation, standardization, and collaborative capabilities. By leveraging metadata-driven pre-population, algorithmic formatting, and annotation transfer functionality, the tool significantly reduces turnaround time, minimizes human error, and ensures compliance with CDISC and internal standards. These improvements not only accelerate study start-up but also enhance reproducibility and consistency across studies, positioning the tool as a scalable solution for modern clinical data workflows.

CONTACT INFORMATION

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

Author Name: Alvin Mathew
Company: Alnylam
Email: amathew@alnylam.com