PharmaSUG 2021 - Paper SS-016

Design ADaM Specification Template that Simplifies ADaM Programming and Creation of Define XML in CDISC Era

Xiang Wang, Daniel Huang, Bristol-Myers Squibb

ABSTRACT

Regulatory agencies such as FDA require sponsors to implement CDISC standards for any clinical studies started after December 17, 2016, thus it is essential if companies can automate the process from ADaM specifications to Define.xml and ensure they are fully CDISC compliant. To our knowledge, many companies still conduct heavily manual work on this process. In this paper we exhibited a well-designed ADaM specification template that is user-friendly for ADaM datasets programming, and then using R tool to convert the specification in the format that is ready to be imported into Pinnacle 21 Enterprise (P21E) to create ADaM define.xml. This new approach helps statistical programmer for creating ADaM datasets more smoothly and generating Define xml more efficiently.

INTRODUCTION

It is known that we are already in CDISC era and most pharmaceutical companies/organizations have their own **ADaM specification** templates that can be utilized to create **ADaM datasets** and downstreatm **Define.xml**. To our knowledge, most of these templates are more user-friendly for ADaM programming instead of creating Define.xml, .i.e users need to call very complicated SAS macros or do heavy manual copy/paste work to create Define.xml; some sponsors design the ADaM specification templates with the same structure as the define specification from Pinnacle 21 Enterprise (**P21E**) which is easy for creating Define.xml but it is not user-friendly for ADaM programming. Inspired by the recent tremendous advancement from P21E that P21E can automatically populate the four time-consuming tabs (Variables, Codelists, ValueLevel and WhereClauses) of define specification in quality after importing ADaM transport (.XPT) datasets, in this paper we proposed a well-designed ADaM specification template that is both easy for ADaM programming and downstream Define.xml creation with the help of P21E and R which is relatively easy to maintain for both experienced and junior statistical programmers.

FOUR STEPS OF CREATING DEFINE.XML UNDER P21E

There are many published papers about how to create Define.xml [1-3] under P21E recently. Basically, the generalized four-step process of creating Define.xml under Pinnacle 21 is shown below in **Figure 1**. Note that the standards used in this paper are ADaMIG v1.1 [4] and Define.xml 2.0 [5]. The brief details for each step are described below.

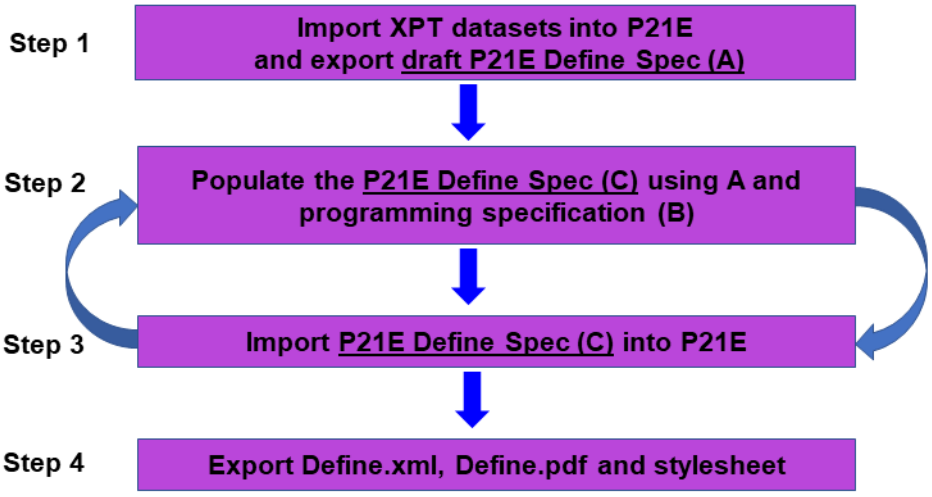
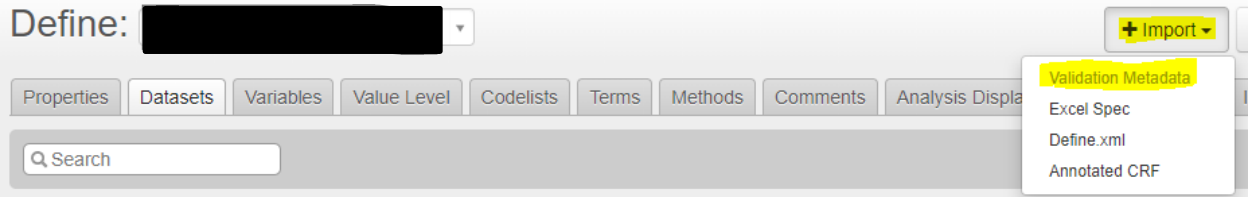


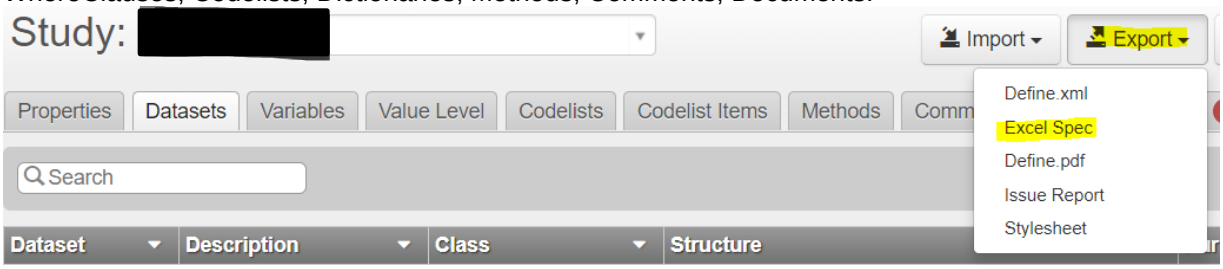
Figure 1. Steps Creating Define.xml Under Pinnacle 21 Enterprise (P21E)

STEP 1: IMPORT ADAM XPT DATASETS INTO P21E AND EXPORT DRAFT P21E DEFINE SPEC (A)

It is assumed that users should already have a P21E account **AND** the ADaM XPT datasets should have been validated under P21E (**Note** that *in this step the zipped validation datasets should **not** have Define.xml*). Then users just need to import the validated datasets as shown below, i.e. click **Import** and then **Validation Metadata**.



After importing, immediately export the **draft P21E Define Spec (A)** shown below, i.e. click **Export** and then **Excel Spec**. Keep in mind that this is one of the two excel files to be used to generate Define.xml in the last section of this paper. It includes the following **10 tabs**: Study, Datasets, Variables, ValueLevel, WhereClauses, Codelists, Dictionaries, Methods, Comments, Documents.

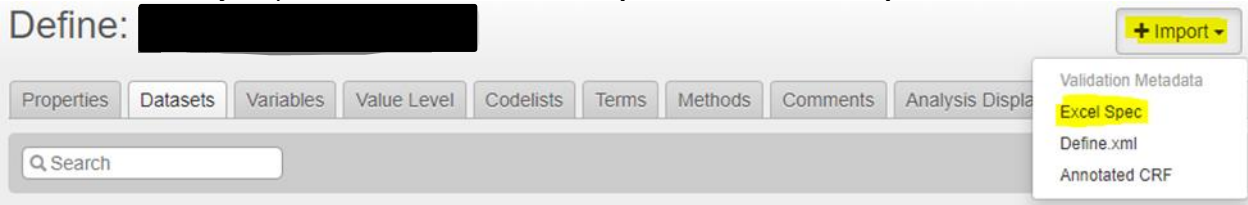


STEP 2: POPULATE THE P21 DEFINE SPEC (C) USING A AND PROGRAMMING SPECIFICATION (B)

P21 Define Spec (**C**) has the same structure as **A**. *The key challenge of this step and Define.xml creation is to populate correct information from **A** and **B** into C*. It is now known that P21E can populate these **four** most time-consuming tabs (**Variables, Codelists, ValueLevel** and **WhereClauses**) in **A** quite well with minor modifications needed in **Variables** tab which will be described in the next sections. Tabs **Methods** and **Comments** can be populated using R from **B**. The rest 4 tabs can be directly copied from **B**. For detailed instruction about what to enter for each column and each of these 10 tabs, users can refer to these references [1-3, 6].

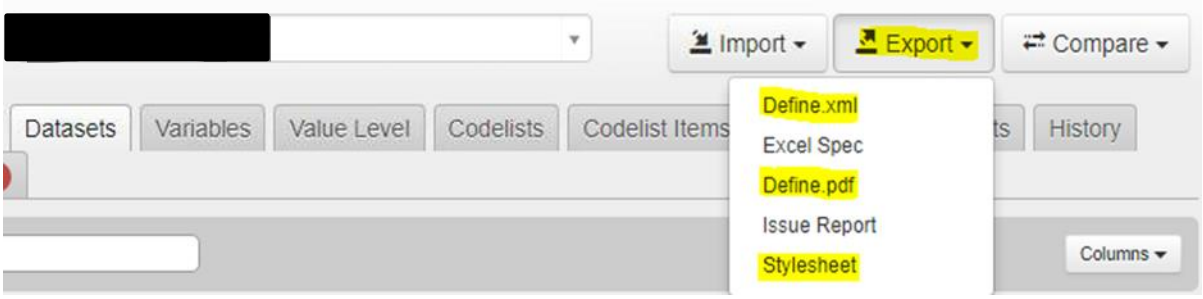
STEP 3: IMPORT P21E DEFINE SPEC (C) INTO P21E

Now that **C** is ready, import **C** into P21E, i.e. click **Import** and then **Excel Spec** shown below.



STEP 4: EXPORT DEFINE.XML, DEFINE.PDF AND STYLESHEET

It might take a few cycles of step 2 and step 3 until no unresolved issues are pending. The last step is to export the Define.xml related files, i.e. click **Export** and download the **three** highlighted files (Define.xml, Define.pdf and Stylesheet) shown below.



CREATING ADAM SPECIFICATION TEMPLATE THAT SIMPLIFIES BOTH ADAM PROGRAMMING AND CREATION OF DEFINE XML

Having the general steps of creating Define.xml in mind, let us see how ADaM specification, e.g. programming specification (B) in step 2 above, can be designed. This is the key to efficient ADaM programming and Define.xml creation. Each P21E sponsor/customer can obtain a free excel file - sample ADaM P21E Define Specification from Pinnacle 21 company which has 10 tabs shown below in **Figure 2**.

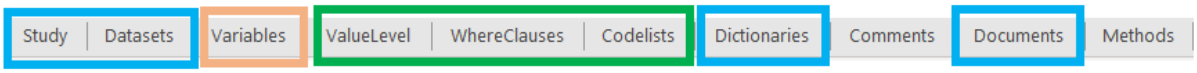


Figure 2. Ten tabs of ADaM P21E Define Specification (A and C)

While it is possible to use the above specification as the ADaM programming specification template, the **Variables** tab which includes all ADaM datasets/variables will have overwhelming rows, resulting in tedious and difficult review experience for programmers and statisticians, **AND** it is absolutely not user-friendly to check through the tabs **Methods** and **Comments** to find the definition/derivation/comments rules for variables which are derived or assigned. To make it more user-friendly for programming and reviewing, the information of three tabs **Variables, Methods,** and **Comments** can be put into individual tabs (ADSL – ADVS; each data per tab) shown below in **Figure 3**. This proposed ADaM specification template has the same 9 tabs (except **Variables** tab) as P21E Define Specification in **Figure 2** *plus* individual dataset tabs. The first two tabs (**ChangeLog** and **Overview&Instructions**) are optional and sponsors are free to enter whatever information appropriate. The following five tabs can be left **blank** as they can be obtained from **A** and **B** later: **Codelists**, **ValueLevel**, **WhereClauses**, **Comments**, and **Methods**. The four tabs (**Study**, **Datasets**, **Dictionaries**, and **Documents**) are quite straightforward and can be entered the information according to the instruction in the reference [6]. These 9 tabs can be completed just before creating Define.xml so that programmers only need to focus on ADaM programming using individual dataset tabs until the agency submission stage.

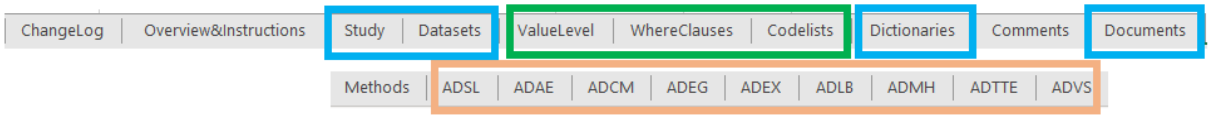


Figure 3. Tabs of Designed ADaM Specification for Statistical Programming

Now let's describe the details of individual dataset tabs. For **ADSL** tab shown below, it includes 19 columns in which the orange columns are the same as those in P21E Define Specification and green columns are user-defined columns. Again, users/sponsors can check the details about what to enter in each of these orange columns from the references [1-3, 6]. However, programmers can start programming after completing those eight highlighted columns shown **below**, i.e. **Variable**, **Label**, **Data Type**, **Length**, **Origin**, **Source/Derivation**, **Comment Description**, and **Common Variable**. The other columns can be completed later when Define.xml creation is needed, and they can be obtained from P21E Define Specification after we import ADaM XPT datasets into P21E.

For columns **C Variable** and **D Label** above, programmers need to make sure they are compliant to CDISC standards. For column **E Data Type**, we only need to enter either '**Char**' or '**Num**' for simplicity instead of 'float', 'integer', 'text' and so on because P21E will automatically obtain the detailed data types after we import ADaM XPT datasets into P21E. For column **F Length**, we do not need to enter the actual length of each variable as this will take us much time, instead, we can enter the length making sure no truncation will appear during programming. For examples, it can be 8 for all numeric variables; it can be 1 for flag variables; it can be 200 for all other character variables. We only need to re-size the variables' length to the maximum length of actual data during agency submission stage which P21E will automatically obtain the length after importing ADaM XPT datasets into P21E. For column **K Origin**, it can be only entered as 'Predecessor' (if copied directly from SDTM or other ADaM variables), 'Derived' (if derived based on specific rules/definitions), or 'Assigned' (if values are assigned or a simple one-to-one map to its primary variable). If the **Origin** of a variable is either **Predecessor** or **Derived**, enter information in column **Q Source/Derivation**; otherwise, enter information in column **R Comment Description**. For column **S Common Variable**, enter 'Y' for those common variables to be copied into all other ADaM datasets. Additional columns can be added into this specification for any other purposes such as statistician review, notes and so on. It is worth noting that programmers only need a relatively simple SAS macro to read these **five** columns (C, D, E, F, S) of the specification above preferring with no dependency on other tabs.

For all other ADaM datasets, they have the same specification structure except no column **S Common Variable** shown below.

AUTOMATION OF ADAM P21E DEFINE SPECIFICATION USING R

TWO PREREQUISITES

Before using R to do the automation, users should be familiar with the general steps creating Define.xml under P21E described in the first section of this paper. Users should also have the ADaM programming specification with the same structure described in the previous section. The following tabs should be correctly populated according to the references [6]: Study, Datasets, Dictionaries, Documents.

INPUTS AND OUTPUT

As shown in **Figure 1**, the two inputs are:

- **A** (draft P21E Define Spec) which is the excel specification exported from P21E immediately after importing '**Validation Metadata**' *without any modifications*.
- **B** (ADaM programming specification) which has the same structure as the ADaM specification template described in the previous section.

The output is **C** (P21E Define Spec) which is the specification to be imported into P21E again after populating correct information from **A** and **B**.

AUTOMATION RULES

The following table describes the detailed rules about how to obtain **C** from **A** and **B**. The first column shows the tab names (**bold**) and/or column names (non-bold); the second column **Rules** describes the detailed rules about how each tab and/or column can be populated from A and/or B; column **Order** is the creation order shown in the R code.

Tab and **Column**	**Rules**	**Order**
Study	copy the whole tab from B into C	1
Datasets	copy the whole tab from B into C	4
Variables (get information from ADSL --- ADVS in **B**)	See the detailed rules below for each column	2
Order	copy this column from A into C	
Dataset Name	copy this column from A into C	
Variable	copy this column from A into C	
Label	copy this column from A into C	
Data Type	copy this column from A into C	
Length	copy this column from A into C	
Significant Digits	copy this column from A into C	
Format	copy this column from A into C	
Mandatory	copy this column from A into C	
Codelist	copy this column from A into C	
Origin	copy this column from individual dataset tab of B into C; for common variables, they should be set to "Predecessor" except for ADSL; column S in ADSL tab from B shows the common variables	
Pages	set this column blank	
Method	copy this column from individual dataset tab of B into C	
Predecessor	copy this column from individual dataset tab of B into C	
Role	set this column blank	
Comment	copy this column from individual dataset tab of B into C	
ValueLevel	copy whole tab from A into C	5
Codelists	copy whole tab from A into C	5
Methods	See the detailed rules below for each column	6
ID	copy from column M (Method) of individual dataset tab from B	
Name	concatenation of “Algorithm to derive ” + ID	

Type	enter as “Computation”	
Description	copy column Q (Source/Derivation) of individual dataset tab from B	
Expression Context	This is left blank	
Expression Code	This is left blank	
Document	This is left blank	
Pages	This is left blank	
WhereClauses	copy whole tab from A into C	5
Comments	See the detailed rules below for each column	3
ID	copy from column P (Comment) of individual dataset tab from B	
Description	copy column R (Comment Description) of individual dataset tab from B	
Document	This is left blank	
Pages	This is left blank	
Dictionaries	copy whole tab from B into C	4
Documents	copy whole tab from B into C	4

The complete R code that follows the above rules is included in the **appendix** with enough comments/explanations. Sponsors/users can also develop SAS macro, Visual Basic or Python tool to implement the above rules as the rules are so much simpler than typical ones across the industries.

CONCLUSION

The proposed ADaM specification template and automating Define.xml using R in this paper greatly simplifies two major works for statistical programmers: ADaM programming and Define.xml creation, from which sponsors and users can save tremendous resource and time to create and maintain ADaM datasets and Define.xml. We also believe it is easy for junior programmers to master these without steep learning curve. It can be well adopted and utilized by both small and big pharmaceutical/biotech companies which seek simplicity and efficiency without complicated and difficult implementation. With this approach, we are confident that users can generate draft ADaM Define.xml within dedicated hours.

DISCLAIMER: The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of Bristol-Myers Squibb Company.

REFERENCES

1. Haloui M, Qi H. 2020. Supplementary Steps to Create a More Precise ADaM define.xml in Pinnacle 21 Enterprise. PharmaSUG.
2. Lei Y., Xu Y., and Pupek M. 2019. “Creation of ADaM Define.xml v2.0 Using SAS Program and Pinnacle 21.” *PhUSE US Connect*.

3. Dutta, PA. 2018. “Generating Define.xml from Pinnacle 21 Community.” *PharmaSUG*.
4. CDISC. 2016. “Analysis Data Model (ADaM) Implementation Guide” v1.1. https://www.cdisc.org/standards/foundational/adam
5. CDISC Define-XML Team, “CDISC Define-XML Specification.” CDISC, Version 2.0, 24 April 2014. https://www.cdisc.org/standards/data-exchange/define-xml
6. Pinnacle 21 Enterprise 4.1 User’s Guide

CONTACT INFORMATION

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

Xiang Wang
Principal Statistical Programmer
Bristol-Myers Squibb
400 Connell Drive, Berkeley Heights, NJ 07922
Email: Xiang.Wang@bms.com

Daniel Huang
Director, Statistical Programming
Bristol-Myers Squibb
400 Connell Drive, Berkeley Heights, NJ 07922
Email: Daniel.Huang1@bms.com

APPENDIX

```
### Input: A (draft P21E Define Spec) - this is the specification exported from P21E right after importing 'Validation Metadata' without any modifications
### Input: B (programming specification) - this is the programming spec used for ADaM datasets creation
### Output: C (P21E Define Spec) - this is the spec to be imported into P21E after populating correct information from both A and B

## Install the following packages if not installed previously
if (!require('openxlsx')) install.packages('openxlsx')
if (!require('dplyr')) install.packages('dplyr')
if (!require('stringr')) install.packages('stringr')

## Load libraries to be used
library('dplyr')
library('stringr')

## Set the working directory where the input and output files are located
## Two input/excel files: draft P21E Define Spec (A) and programming specification (B)
## One output/excel file: P21E Define Spec (C)
setwd("C:\projects\xxx\xxx")

library('openxlsx')
wb1 <- loadWorkbook("A.xlsx")
# Column numbers/names for each of the ten tabs in Define Specification
max_col <- c(2, 9, 16, 16, 5, 8, 5, 8, 4, 3)
names(max_col) <-
c("Study","Datasets","Variables","ValueLevel","WhereClauses","Codelists","Dictionaries","Methods","Comments","Documents")
tabs <- list()
wb <- createWorkbook()
wb2 <- loadWorkbook("B.xlsx")

for (i in 1:length(names(wb1))) {
  name <- names(wb1)[i]

  # tab "Study" – direct copy from B into C
  if (name=="Study") {
    df=read.xlsx(wb2, name,check.names=FALSE)
    tabs[[name]] <- df
    colnames(df) <- gsub("\\.", " ", colnames(df))
    wb$addWorksheet(name)
    df[df$Attribute=="StandardName", "Value"] <- 'ADaM-IG'
    df[df$Attribute=="StandardVersion", "Value"] <- '1.1'
    df[df$Attribute=="Language", "Value"] <- 'en'
    writeData(wb=wb, x=df[, 1:max_col[name]], sheet=name)
  }

  # tab "Variables"
  if (name=="Variables") {
    df = read.xlsx(wb1, name, check.names=FALSE)
    colnames(df) <- gsub("\\.", " ", colnames(df))
    # column "Origin" - get this column from individual dataset tabs of B into C
    # column "Origin" - for common variables, set to "Predecessor" except ADSL
    # column "Method" - get this column from individual dataset tab of B into C
```

```
    # column "Predecessor" - get this column from individual dataset tab of B into C
    # column "Comment" - get this column from individual dataset tab of B into C
    ds <- unique(df$Dataset)
    df_common <- read.xlsx(wb2, "ADSL", check.names=FALSE) %>%
      rename(Description="Source/Derivation") %>%
      select(Variable, Origin, Pages, Method, Predecessor, Role, Comment, Common.Variable,
Description, Comment.Description) %>%
      filter(Common.Variable=='Y') %>%
      select(Variable, Origin, Pages, Method, Predecessor, Role, Comment, Description,
Comment.Description)
    df_B <- data.frame()
    for (d in ds) {
      df2 <- read.xlsx(wb2, d, check.names=FALSE) %>%
        rename(Description="Source/Derivation") %>%
        select(Dataset, Variable, Origin, Pages, Method, Predecessor, Role, Comment, Description,
Comment.Description)
      if (d!="ADSL") {
        df2 <- df_common %>%
          mutate(Dataset=d, Origin="Predecessor") %>%
          bind_rows(df2)
      }
      df_B <- df2 %>%
        mutate(Dataset=str_trim(Dataset, side="both"), Variable=str_trim(Variable, side="both")) %>%
        bind_rows(df_B)
    }

    df <- df %>%
      select(-c(Origin, Pages, Method, Predecessor, Role, Comment)) %>%
      mutate(Dataset=str_trim(Dataset, side="both"), Variable=str_trim(Variable, side="both")) %>%
      left_join(df_B, by=c("Dataset", "Variable")) %>%
      mutate(Predecessor=ifelse(Origin=="Predecessor", paste0("ADSL.", Variable), ""),
             Method=ifelse(Origin=="Derived", Method, ""),
             Comment=ifelse(Origin=="Assigned", Comment, ""),
      )
    # column "Pages" - set this column blank
    df$Pages <- ""
    # column "Role" -   set this column blank
    df$Role <- ""
    tabs[[name]] <- df
    wb$addWorksheet(name)
    df <- df %>%
      select(-c(Description, Comment.Description))
    writeData(wb=wb, x=df[, 1:max_col[name]], sheet=name)
  }

  # tab "Comments"
  if (name=="Comments") {
    #column 'ID' - copy from column P (Comment) of individual dataset tabs from B
    #column 'Description' - copy from column 'Comment Description' of individual dataset tabs from B
    #column 'Document' - if there is any value in column C (Document) of Comments tab in B, copy it into
this tab; otherwise, leave it blank
    #column 'Pages' - if there is any value in column D (Pages) of Comments tab in B, copy it into this tab;
otherwise, leave it blank
    df <- tabs$Variables %>%
      filter(Origin=="Assigned" & Comment >" " & Comment.Description >" " &
!is.na(Comment.Description)) %>%
```

```
    select(-Description) %>%
    mutate(ID=str_trim(Comment, side="both"))  %>%
    rename(Description="Comment.Description") %>%
    select(ID, Description) %>%
    left_join(read.xlsx(wb2, name, check.names=FALSE) %>%
                select(-Description) %>%
                mutate(ID=str_trim(ID, side="both")), by=c("ID"))
  tabs[[name]] <- df
  colnames(df) <- gsub("\\.", " ", colnames(df))
  wb$addWorksheet(name)
  writeData(wb=wb, x=df[, 1:max_col[name]], sheet=name)
 }

 # tabs - "Datasets", "Dictionaries", "Documents" - direct copy from B into C
 if (name %in% c("Datasets","Dictionaries","Documents")) {
  df = read.xlsx(wb2, name, check.names=FALSE)
  tabs[[name]] <- df
  colnames(df) <- gsub("\\.", " ", colnames(df))

  wb$addWorksheet(name)
  writeData(wb=wb, x=df[, 1:max_col[name]], sheet=name)
 }

 # tabs - "ValueLevel", "WhereClauses", "Codelists" - direct copy from A into C
 if (name %in% c("ValueLevel", "WhereClauses", "Codelists")) {
  df = read.xlsx(wb1, name,check.names=FALSE)
  tabs[[name]] <- df
  wb$addWorksheet(name)
  writeData(wb=wb, x=df[, 1:max_col[name]], sheet=name)
 }

 # tab 'Methods'
 if (name =="Methods") {
  # column 'ID' - copy from column M (Method) of individual dataset tabs from B
  df_Methods <- subset(tabs$Variables, Origin=="Derived")
  df <- data.frame(ID=df_Methods$Method)
  # column 'Name' -   concatenation of "Algorithm to derive " + column 'ID'
  df$Name <- paste0("Algorithm to derive ", df$ID)
  # column 'Type' - set to "Computation"
  df$Type <- "Computation"
  # column 'Description' - copy from column 'Source/Derivation' of individual dataset tabs from B
  df$Description <- df_Methods$Description
  # column 'Expression Context' - set to NULL
  # column 'Expression Code' -    set to NULL
  df$'Expression Context' <- ""
  df$'Expression Code' <- ""
  # column 'Document' - if there is any value in column G (Document) of Methods tab in B, copy it into
this tab; otherwise, leave it blank
  # column 'Pages' - if there is any value in column H (Pages) of Methods tab in B, copy it into this tab;
otherwise, leave it blank
  df <- df %>%
    filter(!is.na(ID)) %>%
    mutate(ID=str_trim(ID, side="both"))  %>%
    left_join(read.xlsx(wb2, name, check.names=FALSE) %>%
                select(ID,Document, Pages) %>%
                mutate(ID=str_trim(ID, side="both")), by=c("ID"))
```

```
    tabs[[name]] <- df
    colnames(df) <- gsub("\\.", " ", colnames(df))
    wb$addWorksheet(name)
    writeData(wb=wb, x=df[, 1:max_col[[name]]], sheet=name)
  }

}

## Save/output C (P21E Define Spec to be imported into P21E) to working directory
saveWorkbook(wb, file="P21E_Define_Spec.xlsx", overwrite=TRUE)
```