PharmaSUG 2025 - Paper DS-075

Controlling attributes of .xpt files generated by R

Yachen Wang, Chen Ling, AbbVie Inc.

ABSTRACT

In the process of electronic data submission to Food & Drug Administration (FDA), the most common way is using SAS to generate Simplified Transport (.xpt) files. Currently, more and more companies are exploring using R to generate statistical outputs for regulatory submission, and programmers are trying to generate .xpt file in R. However, due to the software setup difference between SAS and R, the attributes of dataset and variables are not always handled very well. As we know, labels are crucial to SDTM and ADaM datasets, also labels, length and format are indispensable components of variables, which should follow CDISC standard. In this paper, we provide comprehensive guides about checking and assigning attributes to datasets (labels) and variables (length, labels and formats) while developing. xpt files in R. Moreover, we also introduce efficient ways to assign these attributes in R generated .xpt files based on specifications documents. Through practical examples, we compare those different attributes controlling methods, thereby demonstrating how to ensure the exported datasets from R meet the requirement.

INTRODUCTION

R language is increasingly popular in the Life Science industry and has been seen as a potential alternative software for analysis and reporting in clinical trials and real evidence study. It is known for its flexibility in data manipulation, analysis and visualization, and the growing number of programming extensions can really enhance programming efficiency (Ling & Wang, 2025). Nowadays, there is an increasing trend from SAS® to R (Wang & Ling, 2025)and programmers are looking for a method in R that can generate well formatted datasets and output ideal .xpt files. In a previous paper (Case & Tian, 2021), the authors introduced write_xpt() function from {haven} package to generate .xpt files in R. However, using write_xpt() alone is not sufficient to handle the attributes of the output .xpt files and their variables.

Attributes are crucial for ensuring transparency and reliability in data usage and analysis. For SDTM and ADaM datasets, their labels and variables attributes are especially essential as CDISC standard and SDTM/ADaM IG have a strict requirement for them. Programmers may find that defining datasets and variables attributes in R is not as easy as SAS, as base R doesn't have a function similar with LABEL and LENGTH in SAS, moreover datetime-related syntax in R is different from SAS. Fortunately, advanced R packages can assist in assigning attributes.

In this paper, the first four main sections introduce different methods assigning labels, lengths, and formats to variables, as well as labeling datasets during the dataset development and output process in R. We use a sample dummy ADLB dataset to walk through every detail in assigning attributes step by step. The advantages and disadvantages of each method are compared, and the challenges are also discussed. At the end of the paper, we provide a real-world example of controlling all attributes with a single specifications document and make a comparison with the .xpt file generated by SAS.

SOURCE DATA

The source data is a dummy ADLB dataset, including USUBJID, PARAMCD, PARAM, ADT, ADTM, ATM, shown in Figure 1, and the sample specification as well as Metadata are in Figure 2 and Figure 3.

SUBJID	ADT	ATM	ADTM	PARAM	PARAMCD
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Calcium (mmol/L)	CCA
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Calcium (mmol/L)	CCA
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Chloride (mmol/L)	CCL
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Chloride (mmol/L)	CCL
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Potassium (mmol/L)	CK
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Potassium (mmol/L)	CK
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Potassium - Hyperkalemia (mmol/L)	CKH
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Potassium - Hyperkalemia (mmol/L)	CKH
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Potassium - Hypokalemia (mmol/L)	CKL
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Potassium - Hypokalemia (mmol/L)	CKL
1001	2024-04-03	08:18:00	2024-04-03 08:18:00	Magnesium (mmol/L)	CMG
1001	2024-04-07	08:01:00	2024-04-07 08:01:00	Magnesium (mmol/L)	CMG
1001	2024-04-03	08:24:00	2024-04-03 08:24:00	pH	UPH
1001	2024-04-07	06:43:00	2024-04-07 06:43:00	pH	UPH
1002	2024-04-03	08:07:00	2024-04-03 08:07:00	Calcium (mmol/L)	CCA

Figure 1. Sample dummy ADLB

dataset	variable	label	length	format
adlb	SUBJID	Subject Identifier	10	*NA*
adlb	ADT	Analysis Date	8	DATE9.
adlb	ATM	Analysis Time	8	TIME5.
adlb	ADTM	Analysis Datetime	8	DATETIME20.
adlb	PARAM	Parameter	20	*NA*
adlb	PARAMCD	Parameter Code	8	*NA*

Figure 2. Specification for dummy ADLB

dataset	label	class
adsl	Subject-Level Analysis Dataset	SUBJECT LEVEL ANALYSIS DATASET
adlb	Laboratory Test Results Analysis Dataset	BASIC DATA STRUCTURE

Figure 3. Sample Metadata

LABEL

Functions in {haven} package, {Hmisc} package, {labelled} package and {xportr} package can be utilized to check and to edit label for variables.

{HAVEN} PACKAGE

In {haven} package, `attr()` function is effective for handling attribute tasks. `attr(variable_name,"label")` function can be used to check or to edit a label:

```
> attr(adlb$SUBJID, "label")<-"Subject Identifier for the Study"
> attr(adlb$SUBJID, "label")
[1] "Subject Identifier for the Study"
```

SUBJID Subject Identifier for the Study	ADT	ATM	ADTM	PARAM	PARAMCD

Figure 4. Output for attr(variable_name,"label")

Generally, the specification for a dataset will be provided before programming, where having 1-1 matching of variables and their labels. Utilizing `attr()` function within a for-loop can effectively assign labels accordingly. The code below is intended to assign variable labels to the dummy ADLB (Figure 1)

by utilizing the specification (Figure 2).

```
for (varnames in specadlb$variable) {
  attr(adlb[[varnames]],"label")<-specadlb$label[speclabel$variable == varnames]
}
```

R Program 1. Assigning label using attr() based on specification

SUBJID Subject Identifier	ADT Analysis Date	ATM Analysis Time	ADTM Analysis Datetime	PARAM Parameter	PARAMCD Parameter Code

Figure 5

{HMISC} PACKAGE

The **{Hmisc}** package is a package in R to handle various data management and analysis tasks, the `label()` function within this package is especially useful for variable labeling tasks. One thing we need to pay attention is `label()` function will change the variable class.

```
> label(adlb$PARAM)<-"Parameter"
> label(adlb$PARAM)
[1] "Parameter"
> class(adlb$PARAM)
[1] "labelled"  "character"
```

SUBJID	ADT	ATM	ADTM	PARAM Parameter	PARAMCD

Figure 6. Output for label(variable_name)

In addition, we can use `label()` to check the labels of all variables in a dataset via `label(dataset_name)`.

```
> label(adlb)
                     SUBJID                         ADT                         ATM                        ADTM
"Subject Identifier for the Study"                  ""                          ""                          ""
                      PARAM                     PARAMCD
                "Parameter"                          ""
```

Figure 7. Output for label(dataset_name)

The `label()` function can also be used in a for-loop to read the specification and to assign labels. The code below is to assign labels to variables (Figure 1) by using the specification (Figure 2), the output is Figure 5.

```
for (varnames in specadlb$variable) {
  label(adlb[[varnames]])<-specadlb$label[speclabel$variable == varnames]
}
```

R Program 2. Using label() to read the specification

{LABELLED} PACKAGE

The **{labelled}** package, designed for handling labels, offers a powerful `var_label()` function which is particularly useful for assigning labels to variables. Unlike `label()` function, `var_label()` does not change the class of variables. This flexibility makes it easier to use in R scripts and R Shiny applications.

```
> var_label(adlb$PARAM) <-"Parameter"
> var_label(adlb$PARAM)
[1] "Parameter"
> class(adlb$PARAM)
[1] "character"
```

SUBJID	ADT	ATM	ADTM	PARAM Parameter	PARAMCD

Figure 8. Output for var_label()

While handing multiple variables together, the {labelled} package can not only check all labels in the dataset but also edit these labels with a single command. By using `var_label()` function with the dataset name, all variable labels will be displayed in the output.

```
> var_label(adlb)
$SUBJID
NULL

$ADT
NULL

$ATM
NULL

$ADTM
NULL

$PARAM
[1] "Parameter"

$PARAMCD
NULL
```

Figure 9. Output for var_label(dataset)

While managing multiple labels, we can store them in a vector and directly pass this vector to the `var_label()` function along with dataset. It's important to ensure that the order of the labels must be the same as that of the variables in the dataset.

```
var_label(adlb) <-
    c("Subject Identifier for the Study", "Ana Date", "Ana Datetime","Ana Relative Day",
"Param Name","Param Code")
```

R Program 3. Using var_label() to assign ordered label

SUBJID	ADT	ATM	ADTM	PARAM	PARAMCD
Subject Identifier for the Study	Ana Date	Ana Time	Ana Datetime	Param Name	Param Code

Figure 10. Output for var_label in multiple variable (ordered label)

If the labels are not ordered, we need to specify which variable those labels are associated with.

```
var_label(adlb) <- list(
  SUBJID="Subject Identifier for the Study",
  ATM="Ana Time",
  ADT="Ana Date",
  ADTM="Ana Datetime",
  PARAM="Param Name",
  PARAMCD="Param Code"
  )
```

R Program 4. Using var_label() to assign unordered label

SUBJID	ADT	ATM	ADTM	PARAM	PARAMCD
Subject Identifier for the Study	Ana Date	Ana Time	Ana Datetime	Param Name	Param Code

Figure 11. Output for var_label in multiple variable (unordered label)

While reading and assigning labels from a specification, `var_label()` function also needs to be included within a for-loop. The code below demonstrates how to use `var_label()` function to assign labels to variables in the dummy ADLB dataset (Figure 1) based on the specification (Figure 2).The results are as same as `attr()` and `label()` function in Figure 5.

```
for (varnames in specadlb$variable) {
  var_label(adlb[[varnames]])<-specadlb$label[speclabel$variable == varnames]
}
```

R Program 5. Using var_label() to read specifications

{XPORTR} PACKAGE

The {xportr} package, a powerful developed package, is used for outputting CDISC compliant SDTM/ADaM XPT files. The `xportr_label()` function within the package is used to assign labels to variables. Unlike the functions introduced earlier, `xportr_label()` cannot assign label to a certain variable during the development of the dataset; however, it can read and assign labels based on the specifications. Program 6 contains example code, with the output shown in Figure 5. One thing needs to

pay attention is the value of label must be saved in “label” column, if not, R will give error message (Figure 12).

```
adlb <- xportr_label(adlb, specadlb, domain = "adlb")
```

R Program 6. Using xportr_label () to read specifications

```
Error in names(label) <- metadata[[variable_name]] :
  attempt to set an attribute on NULL
```

Figure 12.

Table 1 makes a comparison of methods handing labels above in R from different perspectives.

	Single variable	Multiple variable	Read specification
attr()	Check and edit	---	Need to use for-loop
label()	Check and edit	Check	
var_label()	Check and edit	Check and edit	
xportr_label()	---	---	Can read the specification directly. Values should be saved in “label” column

Table 1.

LENGTH

The methods for processing the number of characters in a variable differ between R and SAS. In SAS, users can explicitly define the number of characters for a variable using “length” statement. In contrast, R does not have a built-in function to set a variable's character length explicitly. When R outputs the .xpt file, the length of it will be automatically defined as the largest number of characters presented in the data.

The default number of characters for variables in R is often not ideal. Instead, it needs to be defined by programmers to make sure its compliances with CDISC standards. `attr(variable_name, "width")` function, {stringr} and {xpoter} package facilitate this process.

`nchar()` function can check the number of characters of variable generated by the R system, and `max(nchar())` can be used to get length for a variable.

```
> nchar(adlb$PARAMCD)
 [1] 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3
> max(nchar(adlb$PARAMCD))
[1] 3
```

```
> nchar(adlb$PARAM)
 [1] 16 16 17 17 18 18 33 33 32 32 18 18  2  2 16
> max(nchar(adlb$PARAM))
[1] 33
```

Figure 13. nchar() example

Let’s take the sample ADLB as an example, if each value we have in PARAMCD is less than 8, we will get less than 8 in `max(nchar())`, which means the length of PARAMCD in the .xpt file is also less than 8. Also, the length of PARAM is 33 because the max number of characters of PARAM is 33 in R dataset.

	Variable	Type	Length
1	SUBJID	Character	4
2	ADT	Numeric	8
3	ATM	Numeric	8
4	ADTM	Numeric	8
5	PARAM	Character	33
6	PARAMCD	Character	3

Figure 14. Length of PARAM and PARAMCD in original sample ADLB

{HAVEN} PACKAGE

`attr (variable_name, "width")` have capability to define the length in the output .xpt file and change nothing in R internal dataset. For instance, `attr(adlb1$PARAMCD, "width")<-8` defined the length of PARAMCD as 8 in the sample dummy ADLB (Figure 15) .

	Variable	Type	Length
1	SUBJID	Character	4
2	ADT	Numeric	8
3	ATM	Numeric	8
4	ADTM	Numeric	8
5	PARAM	Character	33
6	PARAMCD	Character	8

Figure 15

However, if the max number of characters for a variable exceeds the `width` we assign in `attr(variable_name, "width")`, R will give a warning message. Consequently, the variable length in the .xpt file will be determined by the maximum number of characters for the variable, according to the R system. Figure 16 is the example for `attr(adlb1$PARAM, "width")<-30`. Therefore, we need to do data wrangling before using `attr(variable_name, "width")`.

```
Warning message:
Column `PARAM` contains string values longer than user width 30. Width set to 33 to accommodate.
```

	Variable	Type	Length
1	SUBJID	Character	4
2	ADT	Numeric	8
3	ATM	Numeric	8
4	ADTM	Numeric	8
5	PARAM	Character	33
6	PARAMCD	Character	3

Figure 16

Similar to labels, the specification also provides a one-to-one match between variables and their lengths. Using `attr(variable_name, "width")` can assign those variables whose maximum number of characters is less than or equal to what is defined in the specification. (Figure 17).

```
for (varnames in specadlb$variable) {
  attr(adlb[[varnames]],"width")<-specadlb$length[specadlb$variable == varnames]
}
```

R Program 6. Using attr(,length) to read specifications

	Variable	Type	Length
1	SUBJID	Character	10
2	ADT	Numeric	8
3	ATM	Numeric	8
4	ADTM	Numeric	8
5	PARAM	Character	33
6	PARAMCD	Character	8

Figure 17

{XPORTR} PACKAGE

The `xportr_length` function in **{xportr}** package is used for handing variable length. The results are shown in Figure 17. Similar to `attr(varible_name, "width")`, the maximum number of characters for a variable should not exceed the specified length in the specifications. The value of length should be saved in a specific column (“length”) in the specification, if not R will give error message (Figure 18).

```
# assigned length is from the metadata length
adlb <- xportr_length(adlb, specadlb, domain = "adlb",length_source = "metadata")
```

R Program 8. Using xportr_length () to read specification

```
Error in names(length_metadata) <- metadata[[variable_name]] :
  attempt to set an attribute on NULL
```

Figure 18

{STRINGR} PACKAGE

As we can see, `attr()` and `xportr_length()` only modify the length in output .xpt file, and cannot handle those variables whose number of characters exceeds the desired limit. If you want to truncate strings like SAS, R can also achieve this. However, this approach is not recommended, as it may lead to a loss of information, which is not desirable in both SDTM and ADaM datasets. In this section, we will introduce two simple functions in **{stringr}** package, which provides a set of consistent, simple, and easy-to-use functions for string manipulation, to handle these two issues.

`str_pad()` in **{stringr}** package can adjust the number of characters displayed in R if the desirable number of characters exceeds the maximum number of characters in a variable. `adlb$PARAMCD<- str_pad(adlb$PARAMCD, 8,"right")` is what can be used to define length for PARAMCD in the dummy ADLB.

```
> nchar(adlb$PARAMCD)
 [1] 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
> max(nchar(adlb$PARAMCD))
[1] 8
```

Figure 19. example for str_pad()

If the max number of characters for a variable is greater than what we desire, `str_sub()` function in the **{stringr}** package can be used to substring the value and get the correct number of characters. The max number of characters for PARAM is 33 in dummy ADLB (Figure 1). By applying `adlb$PARAM<- str_sub (adlb$PARAM, 1,30)`, the number of characters for PARAM will be reduced to 30 (Figure 20).

```
> nchar(adlb$PARAM)
 [1] 16 16 17 17 18 18 30 30 30 30 18 18  2  2 16
> max(nchar(adlb$PARAM))
[1] 30
```

1	SUBJID	Character	4	
2	ADT	Numeric	8	D
3	ATM	Numeric	8	T
4	ADTM	Numeric	8	D
5	PARAM	Character	30	
6	PARAMCD	Character	3	

Figure 20. example for str_sub()

Using `str_pad()` and `str_sub()` within a for-loop allows both R and the xpt file to display the correct length according to the specification. The sample code and output are shown below.

```
for (varnames in specadlb$variable) {
  if (is.character(adlb[[varnames]])){
  max_length <- specadlb$length[specadlb$var == varnames]
  if (max(nchar(adlb[[varnames]]))<=max_length ) {
    adlb[[varnames]]<- str_pad(adlb[[varnames]],max_length,”left” )
    }
  if (max(nchar(adlb[[varnames]]))>max_length ) {
    warning(paste(varnames,"is truncated due to length is smaller than the maximum number
of characters"), call. = FALSE)
    adlb[[varnames]]<- str_sub(adlb[[varnames]], 1,max_length )
     }
  }
}
```

R Program 9. Standardized length

	Variable	Type	Length	
1	SUBJID	Character	10	
2	ADT	Numeric	8	[
3	ATM	Numeric	8	T
4	ADTM	Numeric	8	[
5	PARAM	Character	20	
6	PARAMCD	Character	8	

Figure 21. Output dataset for length

Table 2 makes a comparison of methods handing lengths above in R from different perspectives.

	Single variable	Requirement	Results	Read spec
attr()	Check and edit	Clean data	Only .xpt file	Need to use for-loop
str_pad() and str_sub()	Edit	---	R dataset and .xpt file	
xportr_length()	----	Clean data	Only .xpt file	Can read the specification directly. Values should be saved in “length” column

Table 2

FORMAT

Format is also an essential part in attributes. For variables in SDTM and ADaM dataset, one thing we need to care about is the date and time format. Usually YYYY-MM-DD or DD-MON-YYYY is used in date, HH:MM or HH:MM:SS in time and DD:MON:YYYY: HH:MM:SS in datetime variable in traditional SAS programming.

In R, date, time and datetime variables are saved in different types, they are in Date, hms, POSIXct type respectively and their default format in R are YYYY-MM-DD, HH:MM:SS and YYYY-MM-DD HH:MM:SS. Also, the output .xpt file is not well formatted without data processing.

SUBJID Subject Identifier for the Study	ADT Analysis Date	ATM Analysis Time	ADTM Analysis Datetime
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:18:00	2024-04-03 08:18:00
1001	2024-04-07	08:01:00	2024-04-07 08:01:00
1001	2024-04-03	08:24:00	2024-04-03 08:24:00
1001	2024-04-07	06:43:00	2024-04-07 06:43:00
1002	2024-04-03	08:07:00	2024-04-03 08:07:00

```
> class(adlb$ATM)
[1] "hms"      "difftime"
> class(adlb$ADTM)
[1] "POSIXct" "POSIXt"
> class(adlb$ADT)
[1] "Date"
>
```

Figure 22. Default format in R

	Variable	Type	Length	Format	
1	SUBJID	Character	4		
2	ADT	Numeric	8	DATE.	D
3	ATM	Numeric	8	TIME.	T
4	ADTM	Numeric	8	DATETIME.	D
5	PARAM	Character	33		
6	PARAMCD	Character	3		

Figure 23. Default format in .xpt from R

{HAVEN} PACKAGE

`attr(variable_name, "format.sas")` is useful for assigning formats, the usage is similar with

`attr(variable_name, "label")` and `attr(variable_name, "width")` mentioned above. `attr(adlb$ADT,"format.sas")<-"DATE9"` is used to assign SAS "DATE9" format to the ADT variable in exported .xpt file. One important point to note is the desired format can only be viewed in the output .xpt file. If we want to customize what displays in R, you need to use Appendix A (UC Berkeley Statistics Department, 2006) to figure out the R format.

Based on the specification, using `attr(variable_name, "format.sas")` within a for-loop also assign the correct format to the variables (Figure 24).

```
for (varnames in specadlb$variable) {
  attr(adlb[[varnames]],"format.sas")<-specadlb$format[specadlb$var == varnames]
}
```

R Program 10. Standardized format

	Variable	Type	Length	Format
1	SUBJID	Character	4	
2	ADT	Numeric	8	DATE9.
3	ATM	Numeric	8	TIME5.
4	ADTM	Numeric	8	DATETIME20.
5	ADY	Numeric	8	
6	ADTF	Character	1	
7	PARAM	Character	33	
8	PARAMCD	Character	3	

Figure 24. Output dataset for format

{XPORTR} PACKAGE

`xportr_format` function in the **{xportr}** package is also a useful tool for this purpose. To avoid error message, the format needs to be saved in "format" column in the specification. Using `adlb <- xportr_format(adlb, metadata, domain = "adlb")` can get the same results (Figure 24), also error message will occur if values are not saved in "format" (Figure 25).

```
Error in `select()`:
! Can't select columns that don't exist.
✖ Column `format` doesn't exist.
Run `rlang::last_trace()` to see where the error occurred.
Warning message:
There was 1 warning in `filter()`.
i In argument: `dataset == .env$domain & !is.na(format)`.
Caused by warning in `is.na()`:
! is.na() applied to non-(list or vector) of type 'closure'
```

Figure 25

Table 3 makes a comparison of methods handing formats above in R from different perspectives.

	Single variable	Read specification
attr()	Check and edit	Need to use for-loop
xportr_format ()	---	Can read the specification directly. Values should be saved in "format" column

TABLE 3

DATASET LABEL

{HAVEN} PACKAGE

Dataset label is a requirement for SDTM and ADaM dataset, it is the text description for the entire SDTM or ADaM dataset, summarizing its purpose and content. Usually, this information is saved in metadata.

`attr(dataset_name,"label")` function provides capability to check and edit the dataset label. R Program 11 is an example of assigning labels for ADLB dataset by using metadata (Figure 3). This resulting label will be passed to `write_xpt()` unless `label=NULL` is specified. It's important to note that dataset labels in

SAS transport files must be 40 characters or fewer, as per the function setup. A longer dataset label will result in an error in R (Figure 27).

```
#Assign dataset label
attr(adlb,"label")<-metadata$label[metadata$dataset ==”adlb”]
#Using the assigned label when writing xpt
#Methods 1:
write_xpt(adlb,“Y:/adlb.xpt”)
#Methods 2:
write_xpt(adlb,“Y:/adlb.xpt”, label = attr(data, “label”))
#Not using the assigned label when writing xpt
write_xpt(adlb,“Y:/adlb.xpt”,label=NULL)
```

R Program 11. attr(dataset_name,”label”) to update dataset label

```
> attr(adlb,"label")<-metadata$label[metadata$dataset =='adlb']
> attr(adlb,"label")
[1] "Laboratory Test Results Analysis Dataset"
```

Figure 26. Output for attr(dataset_name,"label")

```
Error in `write_xpt()`:
! `label` must be 40 characters or fewer.
Run `rlang::last_trace()` to see where the error occurred.
```

Figure 27. Error when dataset label greater than 40

{XPORTR} PACKAGE

`xportr_df_label()` function in {xportr} package assigns dataset label from a dataset level metadata to a given data frame. `adlb <- xportr_df_label(adlb, metadata, domain = "adlb")` can be used to assign label for the dummy ADLB. The dataset and label value need to be saved in the “dataset” and “label” column respectively in the metadata. Failing to do so, similar with other functions in {xportr} package, R will pop up an error (Figure 28).

```
Error in `filter()`:
i In argument: `dataset == .env$domain`.
Caused by error:
! object 'dataset' not found
Run `rlang::last_trace()` to see where the error occurred.
```

Figure 28

Table 4 makes a comparison of methods handing dataset label above in R from different perspectives.

	Single variable	Read metadata
attr()	Check and edit	Can read metadata
xportr_df_label()	---	Can read the metadata Dataset names should be in ‘dataset’ column, dataset label value should be in ‘label’ column in metadata

TABLE 4

CONTROLLING ALL ATTRIBUTES

Specification served as a guide and reference while completing the analysis dataset and Metadata provides descriptive information about dataset. This section provides two methods for assigning attributes based on the specifications and metadata.

{XPORTR} PACKAGE

The `xportr` function in the {xportr} package is a useful tool, it can apply all core xportr functions (e.g `xportr_df_label` , `xportr_label`, `xportr_length` and `xportr_format`) and write xpt., R program 12 below is an example of using `xportr` function and the output is Figure 28. One thing that needs to be noticed is the value of these attributes should be saved in specific columns, which follows the standard in this package. Failure to do so, as discussed in the first four sections, will result in errors.

```
xportr(
  adlb,                       #Dataset Name
  var_metadata = spec,        #Spec Name
  df_metadata=metadata,       #Metadata Name
  domain = "adlb",            #CDISC Dataset Name
  path ="Y:/final1.xpt"       #Output .xpt Path and Name
)
```

R Program 12. {xportr} package to control all attributes

Variable	Type	Length	Format	Informat	Label
SUBJID	Character	10			Subject Identifier
ADT	Numeric	8	DATE9.	DATE9.	Analysis Date
ATM	Numeric	8	TIME5.	TIME5.	Analysis Time
ADTM	Numeric	8	DATETIME20.	DATETIME20.	Analysis Datetime
PARAM	Character	20			Parameter
PARAMCD	Character	8			Parameter Code

Figure 28. Final output xpt by using specification

FOR-LOOP

If you prefer to use your customized specification and metadata, it's suggested to develop a personalized R function (Ling & Wang, 2025). In R program 13, a demo function is developed by combining the functions above in a for-loop to assign attributes based on the customized specification (Figure 2) and metadata (Figure 3), the output is Figure 28. This function can control attributes of dataset and variables in one simple step. We hope this demo can provide R programmers with valuable hints when creating their own .xpt files.

```
readspec<- function(inpath,inname, df_metadata,outpath,outname, df) {
  #Get specification
  spec<-read.csv(paste0(inpath,inname))
    #Assign dataset label
  attr(df,"label")<-df_metadata$label[df_metadata$dataset == deparse(substitute(df))]
    #For loop start
  for (varnames in spec$Variable) {
    #Assign variables label
    var_label(df[[varnames]]) <- spec$label[spec$Variable == varnames]
    #Assign format
    attr(df[[varnames]],"format.sas")<-spec$sasformat[spec$Variable == varnames]
    #Assign length using attr() - Recommended
    attr(df[[varnames]],"width")<-spec$length[spec$Variable == varnames]
    # Assign length using {stringr} package - Not recommended
    if (is.character(df[[varnames]])){
      max_length <- spec$length[spec$Variable == varnames]
      if (max(nchar(df[[varnames]]))<=max_length ) {
        attr(df[[varnames]],"width")<-spec$length[spec$Variable == varnames]
      }
      if (max(nchar(df[[varnames]]))>max_length ) {
        warning(paste(varnames,"is truncated due to length is smaller than the maximum
number of characters"), call. = FALSE)
        df[[varnames]]<- str_sub(df[[varnames]], 1,max_length )
        attr(df[[varnames]],"width")<-spec$length[spec$Variable == varnames]
      }
    }
  }
  #Output xpt
  write_xpt({{df}},paste0(outpath,outname,".xpt"), version=5)
}

readspec(inpath="Y:/",                                  #Spec path
         inname="specification.csv",                    #Spec Name
         df_metadata=metadata,                          #Metadata Name
         outpath="Y:/",                                 #Output xpt Path
         outname="final1",                              #Output xpt Name
         df=adlb                                        #Dataset Name
```

```
)
```

R Program 13. R function to control all attributes

COMPARISION

To guarantee the xpt files generated by R are correct and ready for submission, we compare it with xpt file generated by SAS. Figure 29 is the results from **PROC COMPARE** in SAS. The results indicate that, except for the “Informat”, all other attributes are identical.

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Format	Informat	Label
ADT	WORK.ADLB_SAS	Num	8	DATE9.		Analysis Date
	WORK.ADLB_R	Num	8	DATE9.	DATE9.	Analysis Date
ATM	WORK.ADLB_SAS	Num	8	TIME5.		Analysis Time
	WORK.ADLB_R	Num	8	TIME5.	TIME5.	Analysis Time
ADTM	WORK.ADLB_SAS	Num	8	DATETIME20.		Analysis Datetime
	WORK.ADLB_R	Num	8	DATETIME20.	DATETIME20.	Analysis Datetime

Figure 29. Comparison between R generated. xpt and SAS generated .xpt

CONCLUSION

Although handling attributes in R is not as straightforward as in SAS when exporting .xpt files, R packages offer excellent tools to facilitate this process. By exploring and comparing various R tools, we have recommended strategies to streamline the process of exporting .xpt files from R. This empowers users to choose aligned with their specific needs and preference. This paper aims to provide readers with valuable approaches when creating and customizing R generated .xpt files.

REFERENCES

Case, T., & Tian, Y. (2021). *PharmaSUG 2021.* Retrieved from PharmaSUG 2021: https://pharmasug.org/proceedings/2021/EP/PharmaSUG-2021-EP-057.pdf

Ling, C., & Wang, Y. (2025). TLFQC: A High-compatible R Shiny based Platform for Automated and Codeless TLFs Generation and Validation. *PharmaSUG 2025 conference proceedings.* San Diego.

Ling, C., & Wang, Y. (2025). Writing SAS MACROs in R? R functions can help! *PharmaSUG 2025 conference proceedings.* San Diego, CA.

UC Berkeley Statistics Department. (2006, 2 3). *Dates and Times in R*. Retrieved from Dates and Times in R: https://www.stat.berkeley.edu/~s133/dates.html

Wang, Y., & Ling, C. (2025). Comparing SAS® and R Approaches in Reshaping data. *PharmaSUG 2025 Conference Proceedings.* San Diego, CA.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to Rina Loke, Christelle Raynaud and the R team, Xiangdong Zhou, Holly Peterson and Ujjwala Powers for their support, trust, and encouragement all along.

CONTACT INFORMATION

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

Yachen Wang
AbbVie Inc.
Yachen.wang@abbvie.com

Chen Ling
AbbVie Inc.
Chen.ling@abbvie.com

APPENDIX

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (abbreviated)
%B	Month (full name)
%y	Year (2 digit)
%Y	Year (4 digit)
%H	Decimal hours (24 hour)
%M	Decimal Minute
%S	Decimal Second