From R to Regulatory Submission: Lessons Learned from Creating, Validating, and Submitting R-Generated XPT Files to PMDA

Jun Hasegawa

PharmaSUG Japan 2026

Table of contents

INTRODUCTION

P.3 From R Programming to Regulatory Submission
P.4 Main agenda & disclaimer

CREATING XPT FILES IN R

P.5 Introduction to {xportr}
P.6 Why not haven::write_xpt()?
P.7 Creating XPT Files: SAS vs. R
P.8 Metadata preparation: standard SDTM/ADaM specs are sufficient
P.9 Metadata preparation: error prevention tips
P.10 Getting started with {xportr}
P.11 Simplify your code with xportr()
P.12 Recommended setting: verbose and strict_checks

VALIDATION & PMDA SUBMISSION

P.13 Validation approach to ensure quality and submission readiness
P.14 Step-by-step comparison across the SAS and R workflows
P.15 What was submitted along with the R-generated XPT files?
P.16 PMDA submission experience with R-generated XPT files

WRAP-UP & APPENDIX

P.17 Summary
P.18 Acknowledgements
P.19 Appendix 1: Reading Excel specifications as {xportr} metadata
P.20 Appendix 2: Known differences in XPT files created with {xportr}

NOVARTIS

From R programming to regulatory submission

1. Years of R discussion: many practical approaches have been shared for creating SDTM/ADaM in R
2. Yet less has been shared on creating submission-ready XPT files
3. We applied {xportr} to create XPT files in R and submitted them to PMDA
4. Result: Gateway transfer succeeded, with no queries from PMDA on the electronic data
5. By sharing this experience, I aim to bridge the gap between R programming and real submissions

Main agenda & disclaimer

MAIN AGENDA

1. How XPT files were created
2. How XPT files were validated
3. How XPT files were accepted by PMDA (Gateway transmission, PMDA inquiries, and inspection)

DISCLAIMER

- This session focuses on XPT file creation, validation, and submission.
  Creation of SDTM/ADaM data frames in R is **out of scope** (e.g., usage of {admiral}).
- R package management is also **out of scope**. For our package management approach,
  please refer to PharmaSUG-Japan-2025-06.pdf.
- This presentation shares past submission experience but **does not guarantee a similar outcome for future submissions**.
- **The views expressed are the presenter's own** and do not necessarily reflect those of the presenter's organization.
- The submission shared in this presentation used **R version 4.3.1 and {xportr} 0.4.0.**

Introduction to {xportr}

xportr

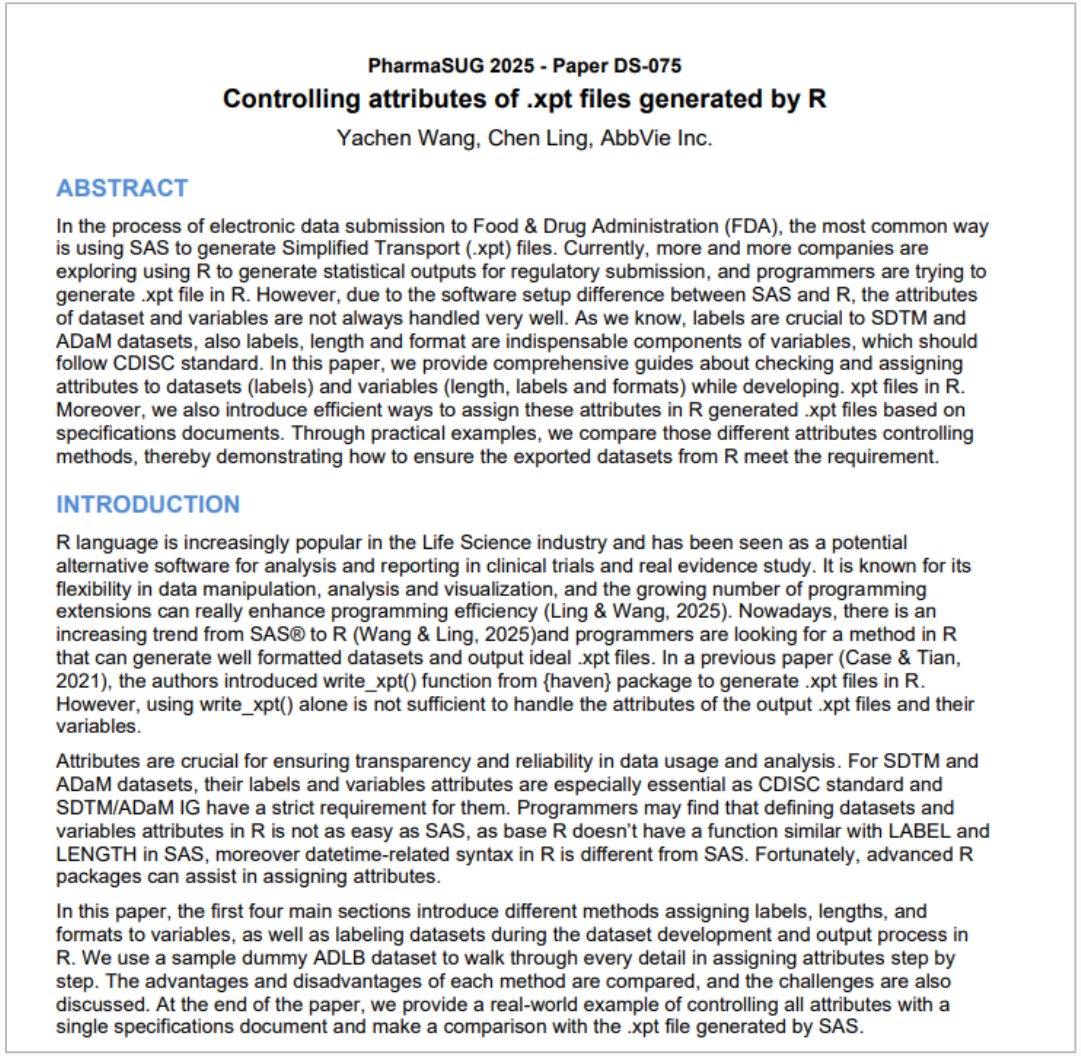


Welcome to `xportr`! We have designed `xportr` to help get your xpt files ready for transport either to a clinical data set validator application or to a regulatory agency. This package has the functionality to associate metadata information to a local R data frame, perform data set level validation checks and convert into a transport v5 file(xpt).

- R package in the pharmaverse
- Provides functions to create submission-ready XPT files (SDTM/ADaM) by applying dataset/variable metadata to R data frames. For example,
  - xportr_label()
  - xportr_type()
  - xportr_length()
  - xportr_format()
  - xportr_write()
- You can understand further through
  - Getting Started • xportr
  - Deep Dive into xportr • xportr

Why not haven::write_xpt()?

- haven::write_xpt() can output raw xpt files but they are not submission-ready. The function does not support effective assignment of attributes such as variable labels, lengths, or formats which are essential components of SDTM/ADaM.
- Hence, using haven::write_xpt() alone is not sufficient, and an alternative such as {xportr} is needed. Details can be found in Controlling attributes of .xpt files generated by R.



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.

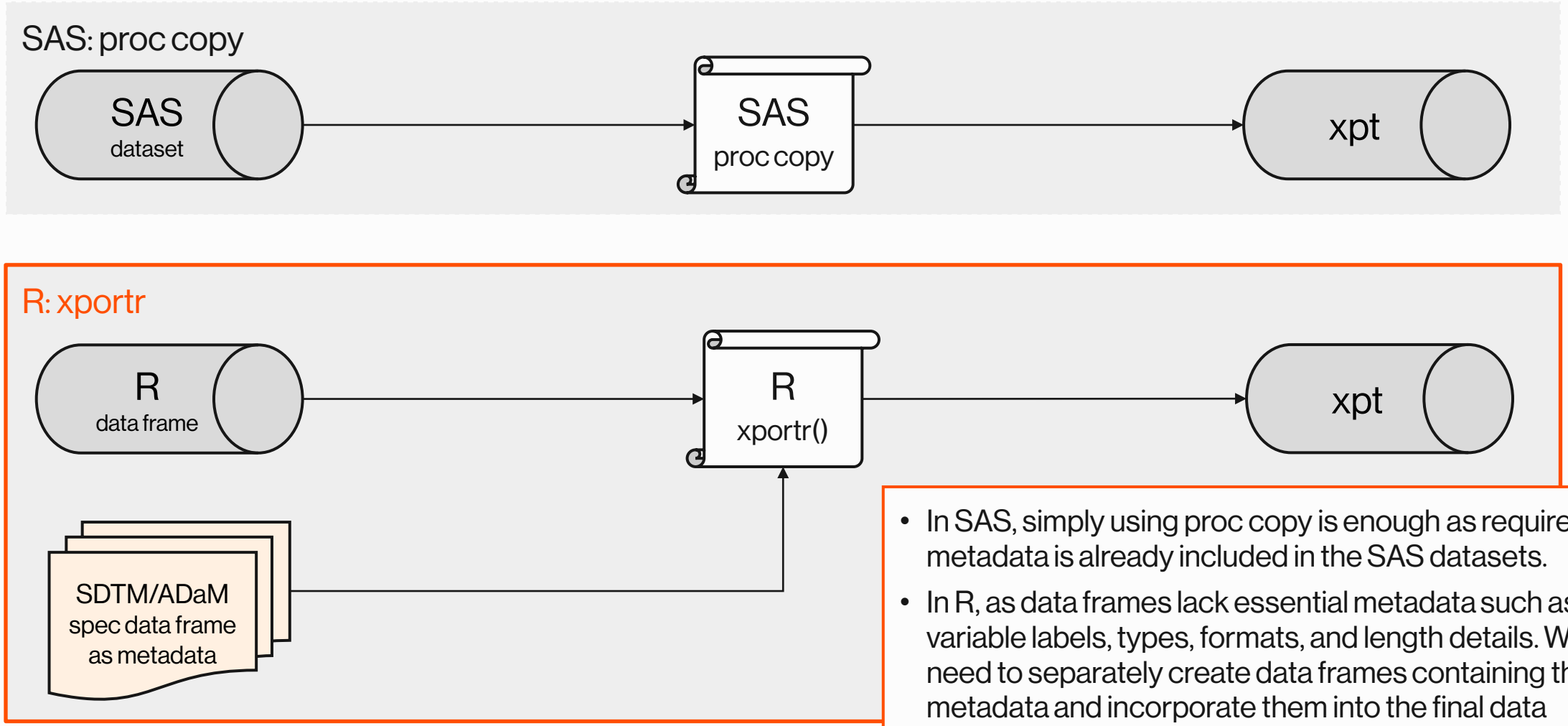
INTRODUCTION

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.

Creating XPT Files: SAS vs. R



- In SAS, simply using proc copy is enough as required metadata is already included in the SAS datasets.
- In R, as data frames lack essential metadata such as variable labels, types, formats, and length details. We need to separately create data frames containing the metadata and incorporate them into the final data frame accordingly.

Metadata preparation: standard SDTM/ADaM specs are sufficient

Order	Dataset	Variable	Label	Data Type	Length	Format
1	ADAE	STUDYID	Study Identifier	text	14	
2	ADAE	USUBJID	Unique Subject Identifier	text	21	
3	ADAE	SUBJID	Subject Identifier for the Study	text	6	
4	ADAE	AGE	Age	integer	8	
5	ADAE	SEX	Sex	text	1	
6	ADAE	RACE	Race	text	5	
7	ADAE	TRT01A	Actual Treatment for Period 01	text	11	
8	ADAE	SAFFL	Safety Population Flag	text	1	
9	ADAE	AESEQ	Sequence Number	integer	8	
10	ADAE	AETERM	Reported Term for the Adverse Event	text	32	
11	ADAE	AEDECOD	Dictionary-Derived Term	text	21	
12	ADAE	AEBODSYS	Body System or Organ Class	text	52	
13	ADAE	AESTDTC	Start Date/Time of Adverse Event	date	10	
14	ADAE	AEENDTC	End Date/Time of Adverse Event	date	10	
15	ADAE	ASTDT	Start Time of Adverse Event	integer	8	DATE9.
16	ADAE	AENDT	End Time of Adverse Event	integer	8	DATE9.

7 parameters below are used in {xportr} functions as variable metadata:

1. Order (VARNUM: variable order)
2. Dataset
3. Variable
4. Label
5. Data Type
6. Length
7. Format

Dataset	Label
ADAE	Adverse Events Analysis Dataset
ADEFF	Efficacy Dataset

2 parameters below are used in {xportr} functions as dataset metadata:

1. Dataset
2. Label

Metadata preparation: error prevention tips

Order	Dataset	Variable	Label	Data Type	Length	Format
1	ADAE	STUDYID	Study Identifier	text	14	
2	ADAE	USUBJID	Unique Subject Identifier	text	21	
3	ADAE	SUBJID	Subject Identifier for the Study	text	6	
4	ADAE	AGE	Age	integer	8	
5	ADAE	SEX	Sex	text	1	
6	ADAE	RACE	Race	text	5	
7	ADAE	TRT01A	Actual Treatment for Period 01	text	11	
8	ADAE	SAFFL	Safety Population Flag	text	1	
9	ADAE	AESEQ	Sequence Number	integer	8	
10	ADAE	AETERM	Reported Term for the Adverse Event	text	32	
11	ADAE	AEDECOD	Dictionary-Derived Term	text	21	
12	ADAE	AEBODSYS	Body System or Organ Class	text	52	
13	ADAE	AESTDTC	Start Date/Time of Adverse Event	date	10	
14	ADAE	AEENDTC	End Date/Time of Adverse Event	date	10	
15	ADAE	ASTDT	Start Time of Adverse Event	integer	8	DATE9.
16	ADAE	AENDT	End Time of Adverse Event	integer	8	DATE9.

IMPORTANT: Follow these rules to avoid errors

1. When the Data Type is “integer” or “float”, Length should be set to 8
2. For --DTC variables, set "date" for Data Type, and Length should be set as 10 (date) or 20 (datetime)
3. Be sure to include the trailing period in numeric date formats (e.g., "DATE9." instead of "DATE9")

Getting started with {xportr}

The basic syntax of {xportr} functions are shown below.

Three data frames must be prepared before using {xportr} functions.

Required data frame	Example
A final data frame to be converted to xpt	ADSL
Variable-level metadata created by reading SDTM/ADaM spec document in the previous slide	var_spec
Dataset-level metadata created by reading SDTM/ADaM spec document in the previous slide	dataset_spec

```
ADSL %>%
  xportr_metadata(var_spec, "ADSL") %>%          # load var_spec and identify "ADSL" as the target domain
  xportr_type() %>%                              # based on var_spec: assign variable type (char or num)
  xportr_length(length_source = "metadata") %>%  # based on var_spec: assign variable length
  xportr_label() %>%                             # based on var_spec: assign variable label
  xportr_order() %>%                             # based on var_spec: assign variable order
  xportr_format() %>%                            # based on var_spec: assign variable format
  xportr_df_label(dataset_spec) %>%              # load dataset_spec and assign dataset label
  xportr_write("adsl.xpt")                       # export the final "adsl.xpt" to the current directory
```

Simplify your code with xportr()

The two approaches below are equivalent. In practice, xportr() provides a simpler and more concise solution. A recommended code example is shown at the bottom.

```
ADSL %>%
  xportr_metadata(var_spec, "ADSL") %>%
  xportr_type() %>%
  xportr_length(length_source = "metadata") %>%
  xportr_label() %>%
  xportr_order() %>%
  xportr_format() %>%
  xportr_df_label(dataset_spec) %>%
  xportr_write(path = "adsl.xpt", strict_checks = FALSE)
```

=

```
xportr(
  ADSL,
  var_metadata = var_spec,
  df_metadata = dataset_spec,
  domain = "ADSL",
  verbose = "none",
  path = "adsl.xpt"
)
```

```
# Recommended setting
xportr(
  ADSL,
  var_metadata = var_spec,
  df_metadata = dataset_spec,
  domain = "ADSL",
  verbose = "stop",
  path = "adsl.xpt",
  strict_checks = TRUE
)
```

Recommended setting: verbose and strict_checks

- **verbose** applies to xportr_type(), xportr_length(), xportr_label(), xportr_order(), and xportr_format(). It mainly controls how discrepancies between the data and metadata are handled. I recommend "stop" as the most conservative setting.

none	Disables any messaging, keeping the console output clean
message	Sends a standard message to the console
warn	Sends a warning message to the console
stop	Stops execution and sends an error message to the console

- **strict_checks** applies to xportr_write() and performs data format checks before writing the XPT file, such as verifying that variable names do not exceed 8 characters. "TRUE" is my recommendation.

FALSE	Issues are reported as warnings, but XPT file will still be created.
TRUE	Issues are treated as errors, and XPT file creation is aborted.

```
# Recommended setting
xportr(
  ADSL,
  var_metadata = var_spec,
  df_metadata = dataset_spec,
  domain = "ADSL",
  verbose = "stop",
  path = "adsl.xpt",
  strict_checks = TRUE
)
```

Validation approach to ensure quality and submission readiness

As this was our first experience creating XPT files in R, we performed two conservative checks:

1. a step-by-step comparison with SAS-generated datasets (see next slide)
2. confirming that no errors occurred when opening the XPT files using the SAS XPORT engine with the code below

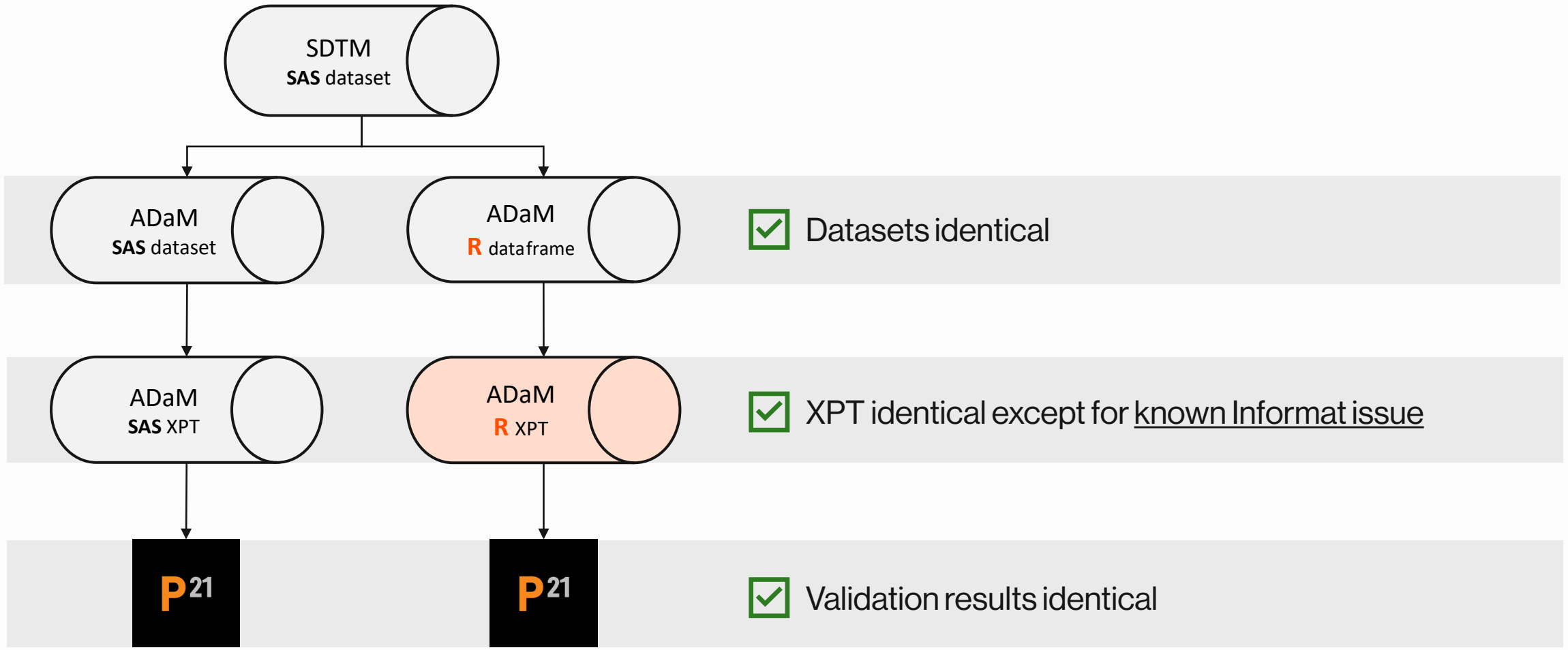
```
%let path = %str(folderpath);
%macro readxpt(dsname=);
   libname xpt xport "&path./&dsname..xpt" access=readonly;
   proc copy in = xpt out = work memtype = data;
     select &dsname.;
   run;
%mend;

%readxpt(dsname=adsl);
%readxpt(dsname=adae);
```

Why the XPORT engine check?

- We had experienced PMDA Rejects due to XPT format issues that were not detected during internal P21 pre-submission validation.
- Those Reject-ed datasets were later found to produce SAS errors when read with the SAS XPORT engine. Therefore, this check was considered effective in detecting XPT format issues.

Step-by-step comparison across the SAS and R workflows



What was submitted along with the R-generated XPT files?

1. ADaM programs (e.g., adsl.R)
2. Analysis data reviewer's guide containing the information required by the PMDA TCG as below.
   a. R version (e.g., 4.3.1) and environment information (e.g., operating system name and version)
   b. Program descriptions, similar to those provided for SAS program submissions
   c. No R package information (package name, version, and description) was included.

PMDA Technical Conformance Guide on Electronic Study Data Submissions 4.1.2.3 regarding what to include in analysis data reviewer's guide

- *Information on the program*
  - *Analysis environment and software used*
  - *Explanation of programs that were used to create the ADaM datasets to be submitted and programs for analyses (if they cannot be submitted, specifications that show the analysis algorithm) to be submitted*

Presenter's view

Based on our experience from several projects that submitted R programs, no queries were raised by PMDA solely because R packages were not described in the ADRG. It might be more appropriate to describe package information selectively, depending on whether a package is widely used or provides advanced functionality not available in SAS or other software. In this way, we can facilitate efficient PMDA review of the submitted programs while streamlining sponsors' workload of ADRG preparation.

PMDA submission experience with R-generated XPT files

- ☑ Data delivery via the PMDA Gateway system was successful (no error)
- ☑ Confirmed that PMDA P21 validation results matched our internal validation results
- ☑ No PMDA inquiries regarding R
- ☑ No questions regarding R during the PMDA inspection

Summary

1. {xportr} enables the creation of submission-ready XPT files in R.

2. No critical differences were observed between SAS- and R-generated XPT files.

3. As with SAS programs, at least the information required by the PMDA Technical Conformance Guide should be documented in the ADRG when submitting R programs.

4. Overall, submitting R-generated XPT files along with R programs to PMDA was no different from our previous experiences submitting SAS-generated XPT files with SAS programs.

Thank you

ACKNOWLEDGEMENTS

Yoshiumi Ouchi（大内 喜海）

Takashi Kitahara（北原 孝志）

Manabu Uehara（上原 学）

Mari Osaka（大坂 真里）

Shunsuke Goto（後藤 駿輔）

for their insightful feedback and generous support for this presentation.

Jun Hasegawa（長谷川 準）

jun-1.hasegawa@novartis.com

NOVARTIS

Appendix 1: Reading Excel specifications as {xportr} metadata

- The Excel tables shown on this slide can be used as {xportr} metadata by simply reading them with the code below.
- {xportr} expects default column names for each variable/dataset attribute (see here for default setting). In this example, we converted column names to lowercase and renamed "Data Type" to "type" to match the defaults.
- If your specification file uses different column names, you can either align them with the {xportr} defaults like this example, or customize the defaults. Read "You've got options() or xportr_options()" in this article for how to customize the defaults.

```
# "var_spec" is used in xportr() to apply variable order, label, type, length, and format.
var_spec <- readxl::read_xlsx(path = paste(folderpath, "adam_spec.xlsx", sep='/'), sheet = "variable") %>%
  rename(type = "Data Type") %>%
  rename_with(tolower) %>%
  select(order,dataset,variable,label,type,length,format)

# "dataset_spec" is used in xportr() to apply dataset label.
dataset_spec <- readxl::read_xlsx(path = paste(folderpath, "adam_spec.xlsx", sep='/'), sheet = "dataset") %>%
  rename_with(tolower) %>%
  select(dataset, label)
```

Appendix 2: Known differences in XPT files created with {xportr}

- An XPT file created in SAS and an XPT file created with {xportr} were both converted to SAS datasets and compared using PROC COMPARE.
- The result illustrates XPT files created with {xportr} contained unnecessary Informat attributes. The same values specified for Format were also assigned to Informat. All other attributes and data values themselves were identical.
- However, this discrepancy was deemed acceptable as the Informat attributes did not affect dataset display or data processing.

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 in Controlling attributes of .xpt files generated by R