PharmaSUG China 2025 - Paper AP-108

Automating Document and Report Generation: A Framework Using SAS and Typst

Tao Zhang, Phastar

ABSTRACT

Generating documents and reports is a common task in statistical programming. Manual formatting, editing, and fine-tuning consume significant time and introduce opportunities for error. To address this, we propose a framework that combines the power of SAS and Typst for automated document and report generation. We provide a detailed overview of the framework's structure and implementation, along with two example applications: automated Study Data Reviewer's Guide (SDRG) and Narratives generation. These examples demonstrate how the framework can automate complex document generation processes and enhance efficiency, accuracy and consistency.

Keywords: Automation, Document, Report, SAS, Typst, SDRG, Narratives

INTRODUCTION

In the daily work of statistical programmers, we generate a variety of documents and reports. Some, like the Study Data Reviewer's Guide (SDRG) and Analysis Data Reviewer's Guide (ADRG), are manually created by editing Word documents. This often involves copying information from multiple sources, formatting tables, and adjusting styles in Word. This is an error-prone and time-consuming process that must be repeated when source data changes. Other outputs, such as TLFs and patient narratives, are produced through SAS programs using tools like PROC REPORT and ODS RTF/PDF. These require extensive fine-tuning to achieve the desired formatting and presentation. Narratives are particularly complex due to the volume of free text, often necessitating large SAS macros to generate well-structured outputs.

To improve efficiency and reduce manual effort in document/report generation, we developed a small framework named SASTYP that combines the strengths of SAS and Typst. It enables direct generation of simple documents and also serves as a foundation for building more complex applications. In this paper, we provide a detailed overview of the framework's structure and implementation and introduce two applications developed based on it. One is an SDRG automation tool and the other is a narratives automation tool.

What is Typst?

Typst is a modern, open-source typesetting system designed for creating high-quality documents with ease and flexibility. Unlike traditional tools such as LaTeX, Typst features a user-friendly syntax, fast compilation, and built-in support for advanced layouts. It allows users to define document structure and styling in a clear and concise manner, making it ideal for automating report and document generation. Typst is especially well-suited for technical and scientific documents, offering powerful features for tables, figures, mathematical notation, and custom templates.

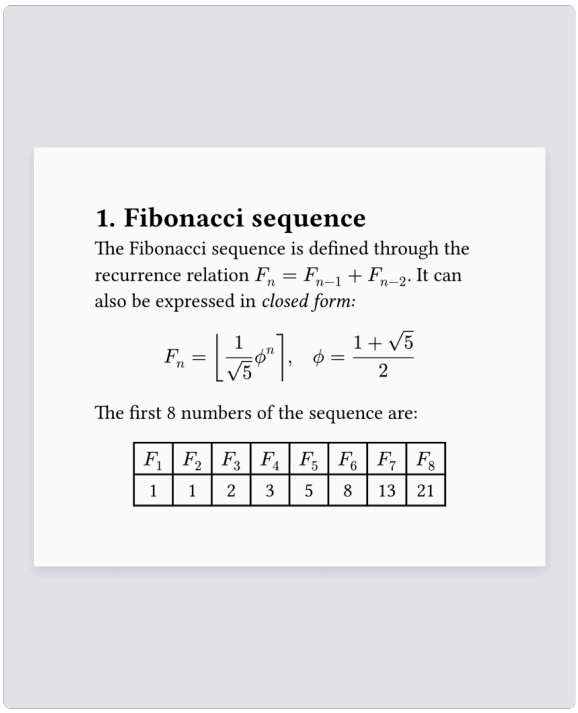
The Typst open-source project is hosted at https://github.com/typst/typst, and the documentation is available at https://typst.app/docs/. To use Typst locally, simply download the Typst compiler. On Windows, it is a standalone executable file with a size of approximately 33 MB. Once you have the compiler, we can use it like this:

```
# Creates `file.pdf` in working directory.
typst compile file.typ
```

```
# Creates PDF file at the desired path.
typst compile path/to/source.typ path/to/output.pdf
```

The Typst compiler takes a source file written in Typst and produces a PDF file as output. Below is an example of Typst source code that showcases the power of Typst encapsulated in a single image.

```
#set page(width: 10cm, height: auto)
#set heading(numbering: "1.")

= Fibonacci sequence
The Fibonacci sequence is defined through the
recurrence relation $F_n = F_(n-1) + F_(n-2)$.
It can also be expressed in _closed form:_

$ F_n = round(1 / sqrt(5) phi.alt^n), quad
  phi.alt = (1 + sqrt(5)) / 2 $

#let count = 8
#let nums = range(1, count + 1)
#let fib(n) = (
  if n <= 2 { 1 }
  else { fib(n - 1) + fib(n - 2) }
)

The first #count numbers of the sequence are:

#align(center, table(
  columns: count,
  ..nums.map(n => $F_#n$),
  ..nums.map(n => str(fib(n))),
))
```



1. Fibonacci sequence

The Fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$. It can also be expressed in *closed form:*

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rceil, \quad \phi = \frac{1+\sqrt{5}}{2}$$

The first 8 numbers of the sequence are:

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
1	1	2	3	5	8	13	21

Source: https://github.com/typst/typst?tab=readme-ov-file

STRUCTURE AND IMPLEMENTATION

The structure of the framework is quite simple. It consists of a collection of SAS macros that provide essential functionality for working with Typst in SAS. We refer to it as a framework to emphasize its role as a foundation for building higher-level applications. The macros are described in below table and the code of the macros can be found at Appendix I.

Macro	Dscription
%st_create_folder(path)	Creates a folder at the specified path if it does not already exist. This folder can then be used to save CSV files and keep them organized in a clear and structured manner.
%st_delete_folder(path)	Deletes a folder at the specified path if it exists. This helps in cleaning up temporary files and directories generated during the document creation process.
%st_csv(dataset, folder)	Converts a SAS dataset into a CSV file.
%st_typ(datasets, typfile)	Takes a list of datasets, sets them together, generates a Typst file using the values of the variable `typst`.
%st_compile(typfile=, pdffile=, compiler=&g_typst_compiler.)	Invokes the Typst compiler to compile a Typst source file into a PDF file. If compilation fails, an error log is generated for troubleshooting. For this macro to work, we need to define a global macro variable **`&g_typst_compiler`** that points to the Typst compiler or provide the path in the macro call.

Macro	Dscription
%st_simple_doc(contents, pdffile)	A macro that supports the creation of simple documents. It takes a list of SAS datasets, reads in the contents, generates a temporary Typst source file, and compiles it into a PDF file. This macro also serves as an example for building other document-generation applications. The internal process is shown in Figure 1.

The diagram below illustrates the simplest use case of the framework: generating a Typst file and compiling it to produce a PDF. We can insert any valid Typst code into a SAS dataset. In the example shown, we include a title, some sample text, and a square shape rendered with a gradient fill.

SAS

```
data test;
  length typst $500;
  typst = "= Lorem ipsum"; output;
  typst = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua."; output;
  typst = "#square(fill: gradient.conic(..color.map.rainbow))"; output;
run;
```

↓ `%st_typ(test, test.typ)`

Typst

```
= Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
#square(fill: gradient.conic(..color.map.rainbow))
```

↓ `%st_compile(typfile=test.typ, pdffile=test.pdf)`

PDF

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

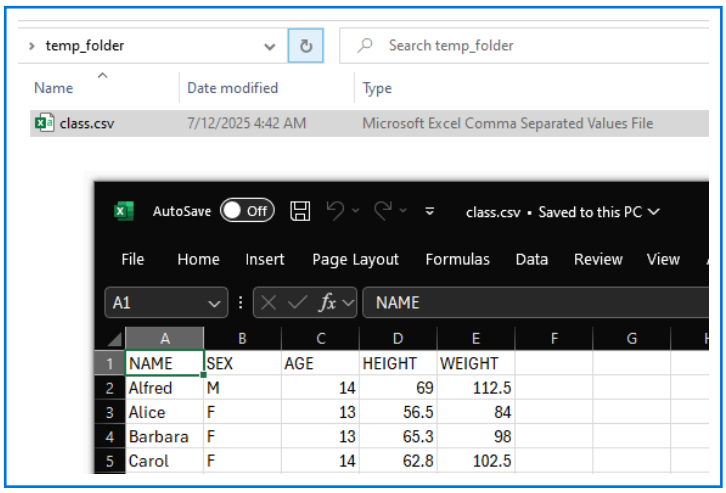
What if we want to pass a SAS dataset and render it as a table? In this case, we definitely need a mechanism to share data between SAS and Typst. Fortunately, Typst can easily load data from external files such as CSV, JSON, and XML. Since CSV files are straightforward to generate from SAS, we chose CSV as the medium for data sharing. Below is a diagram that illustrates the process.

SAS

```sas
%st_create_folder(.\temp_folder\);
data class;
  set sashelp.class;
  if _n_ <= 4;
run;
%st_csv(class, .\temp_folder\);
```

↓

CSV



↓

SAS

```sas
data csv_example;
  length typst $500;
  typst = '#let csvdata = csv("./temp_folder/class.csv")'; output;
  typst = '= An Example of Table from CSV'; output;
  typst = '#table(columns: 5, ..csvdata.flatten())'; output;
run;
```

↓ `%st_typ(csv_example, test.typ)`

Typst

```typst
#let csvdata = csv("./temp_folder/class.csv")
= An Example of Table from CSV
#table(columns: 5, ..csvdata.flatten())
```

↓ `%st_compile(typfile=test.typ, pdffile=test.pdf)`

PDF

An Example of Table from CSV

NAME	SEX	AGE	HEIGHT	WEIGHT
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5

The Macro %st_simple_doc

The macro `%st_simple_doc` is a built-in application within the framework to support the generation of simple documents. It also serves as an example for building other document-generation applications. The internal process is shown below.

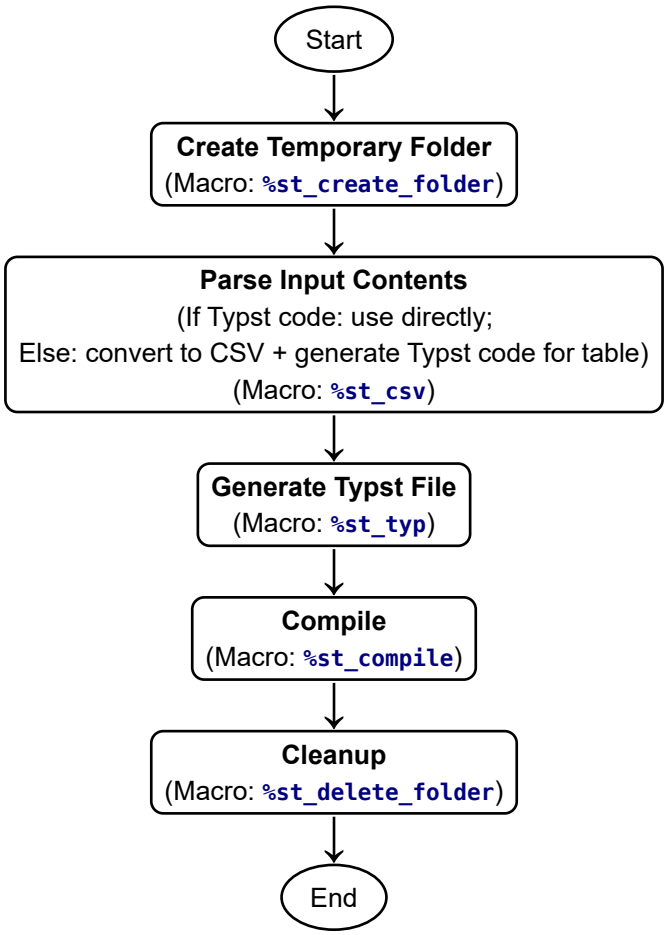


Figure 1: Internal Process of `%st_simple_doc`

Below is an example of using `%st_simple_doc` to create a simple document with a title, some text, a figure and a data table. The output PDF is shown in below block with dotted borders.

```
%include "sastyp.sas";

%global g_typst_compiler;
%let g_typst_compiler=\path\to\typst.exe;

data title;
  typst = "= A Simple Document Generated by SASTYP";
run;
```

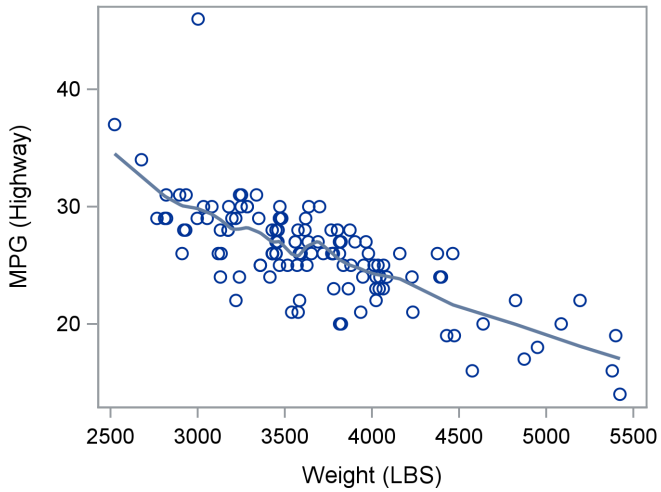
```
data text;
  typst = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec
 odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem
 at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec
 tellus sed augue semper porta.';
run;

ods html gpath="." image_dpi=400;
ods graphics / outputfmt=png width=3.5in imagename="img" reset=index border=off;
proc sgplot data=sashelp.cars noautolegend;
  where (origin = "Europe");
  loess x=weight y=mpg_highway;
run;

data figure;
  typst = '#figure(image("img.png"))
Mauris massa. Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu
 ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales
 ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque
 nibh. Aenean quam.';
run;


data table;
  set sashelp.cars;
  if _n_ <= 3;
  keep model msrp invoice enginesize weight mpg_highway;
run;

%st_simple_doc(
  contents = title text figure table,
  pdffile = simple_doc.pdf,
);
```

Let's break down what's happening:

- We set the global macro variable **`&g_typst_compiler`**.
- We create a dataset named **`title`**, which contains a single variable called **`typst`**. The values of this variable are treated as Typst source code. In this case, the value "= A Simple Document Generated by SASTYP" represents a top-level heading in Typst markup.
- We create another dataset named **`text`**, where the special variable **`typst`** is assigned plain text content.
- Using **`PROC SGPLOT`**, we create a figure and save it as "img.png". Later, in a data step, the figure is referenced using the Typst function call `#figure(image("img.png"))` followed by some text.
- Next, we create a normal dataset. This dataset will be rendered as a table in the output.
- Finally, we invoke the macro **`%st_simple_doc`**, passing in the list of datasets and the output PDF file path.

A Simple Document Generated by SASTYP

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta.



Mauris massa. Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam.

MODEL	MSRP	INVOICE	Engine Size (L)	MPG (Highway)	Weight (LBS)
MDX	$36,945	$33,337	3.5	23	4451
RSX Type S 2dr	$23,820	$21,761	2	31	2778
TSX 4dr	$26,990	$24,647	2.4	29	3230

simple_doc.pdf

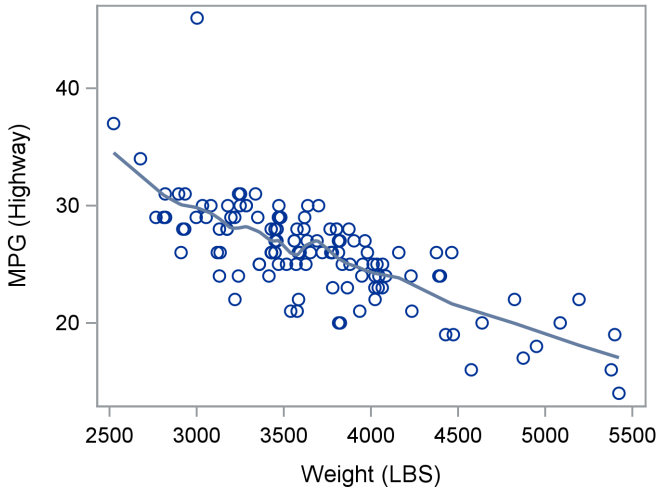
This framework makes it straightforward to generate simple documents. Since we can use Typst code directly, it also provides powerful flexibility for styling and layout customization. Let's make some simple updates to demonstrate this. We add a **`style`** dataset which contains Typst code to emphasize the table header and apply three-line border style. The Typst styling can even be embedded directly within table cells. While preparing the dataset **`table`**, we emphasize the text if the value contains `'TSX'`. The new output PDF with additional styles is shown in below block.

```
*Updated to add styles;
data style;
  typst = '#set table(stroke:
  (x, y) => (left: 0pt, right: 0pt, bottom: 1pt, top: if y <= 1 {1pt} else {0pt})
  )';
  output;
  typst = '#show table.cell.where(y: 0): strong';
  output;
run;
```

```sas
data table;
  set sashelp.cars;
  if _n_ <= 3;
  if index(model, 'TSX') then model = "[*"||strip(model)||"*]";
  keep model msrp invoice enginesize weight mpg_highway;
run;

%st_simple_doc(
  contents = title text figure style table,
  pdffile = simple_doc_3line.pdf,
);
```

A Simple Document Generated by SASTYP

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta.



Mauris massa. Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam.

MODEL	MSRP	INVOICE	Engine Size (L)	MPG (Highway)	Weight (LBS)
MDX	$36,945	$33,337	3.5	23	4451
RSX Type S 2dr	$23,820	$21,761	2	31	2778
TSX 4dr	$26,990	$24,647	2.4	29	3230

simple_doc_3line.pdf

EXAMPLE APPLICATIONS

Study Data Reviewer's Guide (SDRG) Automation

The SDRG is a key deliverable in eCRT package. Traditionally, creating the SDRG has been a manual, time-consuming, and error-prone process. Based on the SASTYP framework, we developed an SDRG automation tool. The overall process flow is illustrated in Figure 2. The internal process of the maccro `%sdrg_create` powered by SASTYP is shown in Figure 3.

Figure 2: SDRG Automation Process Flow

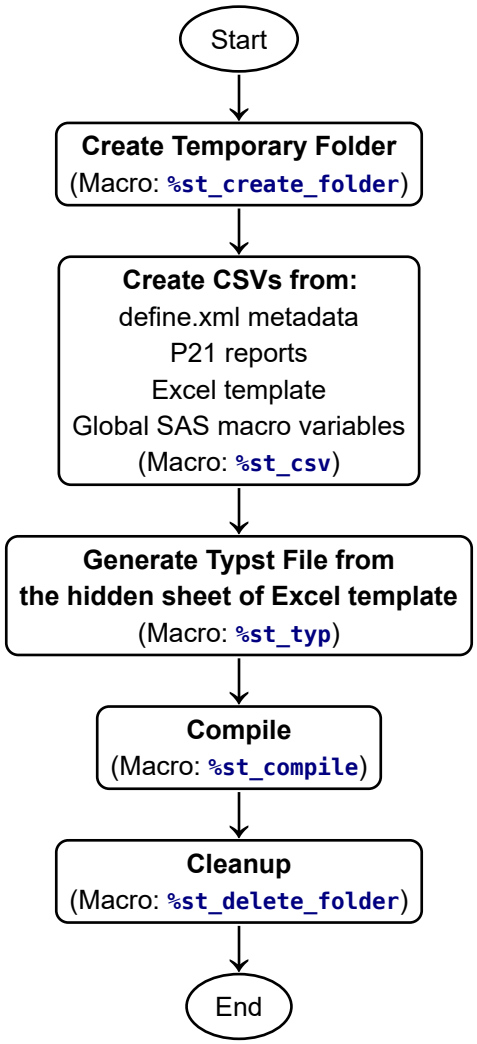


Figure 3: Internal Process of `%sdrg_create`

In this process, we achieved maximum automation by programmatically extracting and presenting the SDTM Define metadata. This ensures that the content of the SDRG remains tightly aligned with the underlying clinical study data. The entire workflow becomes data-driven—any updates to the metadata are automatically reflected in the SDRG through a simple rerun of the tool. This not only eliminates manual copy-paste errors but also ensures a high level of accuracy and consistency across study documents, reducing the risk of discrepancies between the SDRG and Define.xml.

The Excel template serves as the main user interface for the tool. Users can provide free-text inputs and conveniently insert tables by adding new sheets to the Excel file. Within the template, it is also possible to reference any SAS macro variables already defined in the SAS environment. This makes the generation of the SDRG highly dynamic and adaptable.

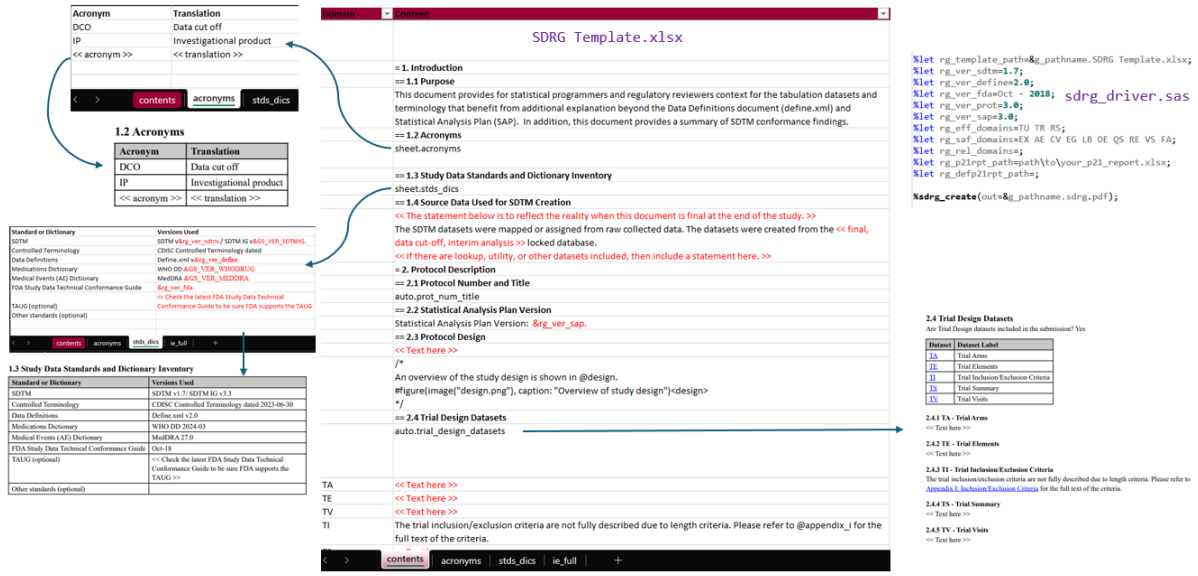


Figure 4: SDRG Automation User Interface

Narratives Automation

In clinical trials, narratives are concise, detailed descriptions of specific events or experiences related to individual participants, most commonly focusing on serious adverse events (SAEs), deaths, or significant adverse events of interest. Each narrative typically outlines the participant's demographics, medical history, treatment regimen, event chronology, clinical outcome, and investigator's assessment of causality.

Generating narratives is often technically complex due to the need to construct large volumes of free-text sentences from structured clinical data. If done purely through SAS programming, it would involve extensive string concatenation, resulting in code that is difficult to work with. In contrast, Typst is well-suited for this task. We developed a narratives automation tool based on SASTYP. The process flow is shown in Figure 5.

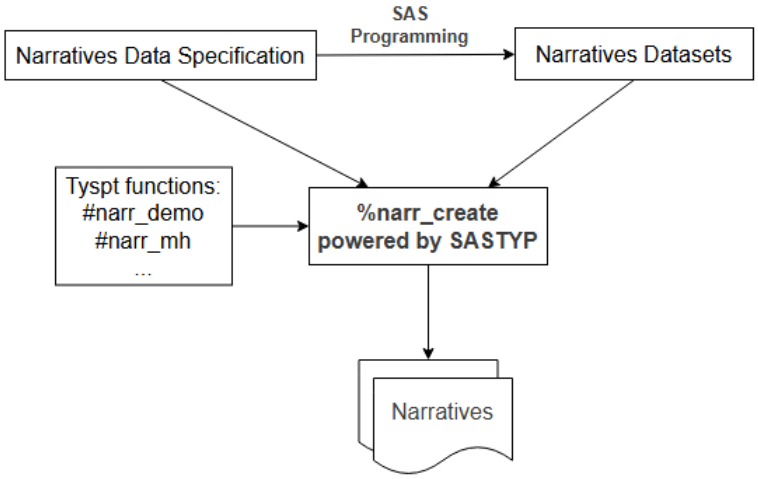


Figure 5: Narratives Automation Process Flow

Below is a minimal example of this tool, demonstrating functionality with only one sentence. The narrative follows a structured template:

```
<Sentence 1>
Subject <Subject ID DM.SUBJID>, a <Age DM.AGE>-<Age units DM.AGEU - If DM.AGEU in
("YEAR" "YEARS") then display "year"; if DM.AGEU in ("MONTH" "MONTHS") then display
"month">-old <race DM.RACE - If DM.RACE="Other"display the value in SUPPDM.RACEOTH in
propcase> <gender If DM.SEX="F" display "female", if DM.SEX="M" display "male"> from
<Country DM.COUNTRY>, with <Disease Under Study (DUS) lowcase(TS.TSVAL) where
TS.TSPARMCD="INDIC" and TS.TSVAL= MH.MHTERM> diagnosed on <Date DUS First Diagnosed
MH.MHSTDTC where (MH.MHSTDTC ne missing) and (MH.MHTERM=TS.TSVAL where
PARAMCD="INDIC" and MH.MHEVTYPE="FIRST DIAGNOSIS" and MHOCCUR ne "N")>
Paragraph 2 includes sentence 2
<For sentence 2, display the code for either 2a or 2b depending on the data for each
subject>
<Sentence 2>
The subject
<if randomised to blinded treatment where TS.TSPARMCD="TBLIND" and TS.TSVAL ne "OPEN
LABEL", display the following: "was randomized to ">
<if randomised to open label treatment where TS.TSPARMCD="TBLIND" and TS.TSVAL="OPEN
LABEL", display the following: "initiated treatment with ">

<Sentence 2a><If the randomisation date and the first IP dose date are the same,
display the following (similar to event-based narrative):> <Investigational product
dose and dosing regimen if blinded treatment display "<Dummy Treatment>", otherwise,
display DM.ACTARM without any dosing information> and received the first dose on
<dosing start date DM.RFXSTDTC>.

<Sentence 2b><If the randomisation date and the first IP dose date are different,
display the following:> <Investigational product dose and dosing regimen if blinded
treatment display "<Dummy Treatment>", otherwise, display DM.ACTARM without any
dosing information> on <date randomised DS.DSSTDTC where DS.DSDECOD="RANDOMIZATION
CODE ALLOCATED"> and received the first dose on <dosing start date display
DM.RFXSTDTC>.
```

Based on this template, we create a specification as shown in Figure 6. The corresponding dataset is generated through SAS programming. In this example, we treat the two sentences described in the template as a single unit and assign it the ID `narr_demo`.

ID	Type	Dataset	Variable	Description	Derivaion
					ADSL.SUBJID
narr_demo	Sentence	ADSL	SUBJID	Subject ID	Save output dataset to NARRADAM library.
narr_demo	Sentence	ADSL	AGE	Age	ADSL.AGE
narr_demo	Sentence	ADSL	AGEU	Age unit	ADSL.AGEU
narr_demo	Sentence	ADSL	RACE	Race	ADSL.RACE
narr_demo	Sentence	ADSL	SEX	Sex	ADSL.SEX
narr_demo	Sentence	ADSL	COUNTRY	Country	ADSL.COUNTRY
narr_demo	Sentence	ADSL	INDIC	Disease Under Study (DUS)	lowcase(TS.TSVAL) where TS.TSPARMCD="INDIC"
narr_demo	Sentence	ADSL	INDICDT	Date DUS First Diagnosed	MH.MHSTDTC where MH.MHEVTYPE="FIRST DIAGNOSIS"
narr_demo	Sentence	ADSL	TBIND	Trial Blinding Schema	TS.TSVAL where TS.TSPARMCD="TBLIND"
narr_demo	Sentence	ADSL	ACTARM	Actual Arm	ADSL.ACTARM
narr_demo	Sentence	ADSL	RANDDT	Date Randomised	ADSL.RANDDT
narr_demo	Sentence	ADSL	TRTSDT	First IP Dose Date	ADSL.TRTSDT

Figure 6: Narratives Data Specification

Suppose our data appears as shown in Figure 7.

Obs	SUBJID	AGE	AGEU	RACE	SEX	COUNTRY	INDIC	INDICDT	TBIND	ACTARM	RANDDT	TRTSDT
1	1001-001	62	year	Asian	female	China	disease under study	2022-Apr-08	OPEN LABEL	Drug A	2025-Jan-05	2025-Jan-06
2	1001-002	80	year	Asian	male	China	disease under study	2021-Oct-06	OPEN LABEL	Drug A	2024-Jul-06	2024-Jul-06

Figure 7: Narratives Data

After running the tool, we obtain two outputs, shown in Figure 8 and Figure 9.

Protocal ABC-XYZ

1001-001 Narrative

Subject 1001-001, a 62-year-old Asian female from China with disease under study diagnosed on 2022-Apr-08. The subject initiated treatment with Drug A on 2025-Jan-05 and received the first dose on 2025-Jan-06.

Figure 8: Narratives output 1

Protocal ABC-XYZ

1001-002 Narrative

Subject 1001-002, an 80-year-old Asian male from China with disease under study diagnosed on 2021-Oct-06. The subject initiated treatment with Drug A and received the first dose on 2024-Jul-06.

Figure 9: Narratives output 2

Within the tool, each sentence is generated using a corresponding Typst function. Below is the function used to produce the demo sentence. The syntax is straightforward and easy to extend. It supports conditional content generation, allowing different narrative paths depending on the data—for example, switching between “was randomized to” and “initiated treatment with” based on the blinding schema of the study. Additionally, helper functions like `#preage` handles proper article selection (“a” or “an”) based on age values, accommodating edge cases such as ages that begin with a vowel sound (e.g., “an 80-year-old”).

```
#import "narr_utils.typ": *

#let narr_demo(subjid, csv_path: "./Narratives_Data/adsl.csv")  = {

  let adsl = csv(csv_path, row-type: dictionary).filter(row=>row.subjid ==
subjid).first()

  [Subject #adsl.subjid, #preage(adsl.age)-#adsl.ageu\-old #adsl.race #adsl.sex from
#adsl.country with #adsl.indic diagnosed on #adsl.indicdt.
  The subject #if adsl.tbind == "OPEN LABEL" {[initiated treatment with]} else {[was
randomized to]} #adsl.actarm #if adsl.randdt!=adsl.trtsdt {[on #adsl.randdt ]} and
received the first dose on #adsl.trtsdt.]
}
```

CONCLUSION

The proposed framework offers a practical solution to the inefficiencies in document generation by combining SAS's data handling with Typst's flexible layout capabilities. By using CSV as the bridge, we maintain a clear separation between data and presentation while allowing seamless integration. The SDRG example demonstrates how automation can reduce manual effort, improve consistency, and ensure data alignment. Similarly, the narrative automation workflow showcases how even highly textual and data-dependent content—often difficult to manage in SAS alone—can be elegantly and efficiently generated using Typst functions. This approach can potentially be extended to other documents and reports, such as ADRGs, patient profiles, or even Clinical Study Reports (CSRs) in the future.

REFERENCES

1. Typst Documentation. https://typst.app/docs/

ACKNOWLEDGEMENTS

The author would like to thank the developers of Typst for creating such an excellent tool. The author also thanks the Phastar programming team for their support and feedback during development of this framework.

Appendix I

```
/*****************************************************************************
*
* Program Name      : sastyp.sas
* Type              : Utility
* Description       : This program defines SAS utility macros for Typst
*
* Author            : Tao Zhang
*
*****************************************************************************/

*****************************************************************************
Macro %st_create_folder(path)
Creates a folder with the given path.
Nothing happens if the folder already exists.
*****************************************************************************;
%macro st_create_folder(path);
```

```
  options dlcreatedir;
  libname _tmp "&path.";
  libname _tmp clear;
  options nodlcreatedir;
%mend;

*******************************************************************************
Macro %st_delete_folder(path)
Seletes a folder if it exists.
*******************************************************************************;
%macro st_delete_folder(path);
    %if %sysfunc(fileexist(&path)) %then %do;
        x "rmdir /s /q ""&path""";
        %put NOTE: Folder "&path" has been deleted.;
    %end;
    %else %do;
        %put NOTE: Folder "&path" does not exist.;
    %end;
%mend;


*******************************************************************************
Macro %st_typ(datasets, typfile)
Takes a list of datasets, sets them together, generates a Typst file using the
values of the variable typst.

Parameters:
1) datasets: A list of datasets.
2) typfile: Path of the output Typst file.
*******************************************************************************;
%macro st_typ(datasets, typfile);
  %*Set length of code to max to avoid truncation;
  data _empty;
    array var[*] $32767 typst;
  run;

  %*Generate typst source file;
  filename _typst "&typfile.";
  data _null_;
    file _typst flowover lrecl=32767;
    set _empty &datasets.;
    put typst;
  run;
%mend;

*******************************************************************************
Macro %st_csv(dataset, folder)
Converts a SAS dataset into a CSV file.

Parameters:
1) dataset: SAS dataset name, e.g. test, sashelp.class.
   The filename of the CSV file will be the dataset_name.csv.
2) folder: The path of the folder to save the CSV file.
*******************************************************************************;
%macro st_csv(dataset, folder);
  %local lib mem;
  data _null_;
```

```
    dataset = symget('dataset');
    if index(dataset, '.') then do;
      lib = lowcase(scan(dataset, 1, '.'));
      mem = lowcase(scan(dataset, 2, '.'));
    end;
    else do;
      lib = "work";
      mem = lowcase(dataset);
    end;
    call symput('lib', strip(lib));
    call symput('mem', strip(mem));
  run;

  data _csv_nolinebrk;
    set &dataset.;
    dummy_char = '';
    array charvars[*] _character_;
    do i = 1 to dim(charvars);
      if index(charvars[i], "0d0a"x) then do;
        *Add this in your Typst code: #show "\u{0000}\u{0020}": "\n";
          *to correctly show linebreaks in CSV data;
        charvars[i] = tranwrd(charvars[i], "0d0a"x, "0020"x);
      end;
    end;
    drop dummy_char i;
  run;

  proc export data=_csv_nolinebrk
      outfile="&folder.&mem..csv"
      dbms=csv
      label
      replace;
  run;
%mend;


*******************************************************************************
Macro %st_compile(typfile=, pdffile=, compiler=&g_typst_compiler.)

The macro calls the Typst compiler to compile a Typst file and generate a PDF file.
If there is any compilation error, an error.txt file will be genereated within the
same folder of the Typst file.

Parameters:
1) typfile: Path of the Typst file.
2) pdffile: Path of the output PDF file.
3) compiler: Path of the Typst compiler.
*******************************************************************************;
%macro st_compile(
  typfile=,
  pdffile=,
  compiler=&g_typst_compiler.
);

  %local typst_working_folder;
  data _null_;
    length fullpath folder $1000;
```

```
    fullpath = symget('typfile');
    folder = prxchange('s/^(.*\\)?([^\\]+)$/$1/', -1, fullpath);
    call symput('typst_working_folder', strip(folder));
  run;

  %local batfile errfile;
  %let batfile = &typst_working_folder.__tmp.bat;
  %let errfile = &typst_working_folder.error.txt;
  filename batfile "&batfile.";
  filename errfile "&errfile";

  data _null_;
    length comp_cmd1 comp_cmd2 $2000;
    comp_cmd1 = "22"x||"&compiler."||"22"x|| " compile " ||"22"x||
strip(symget('typfile'))||"222022"x||strip(symget('pdffile'))||"22"x;
    comp_cmd2 = "22"x||"&compiler."||"22"x|| " compile " ||"22"x||
strip(symget('typfile'))||"222022"x||strip(symget('pdffile'))||"22"x||" 2>"||"22"x||
strip(symget('errfile'))||"22"x;
    compile_status = system(strip(comp_cmd1));
    call execute('data _null_; if fexist("errfile") then rc = fdelete("errfile");
run;');
    if compile_status ^=0 then do;
      put "ERR" "OR: Something went wrong when generating the PDF.";
      put "ERR" "OR: Please contact support network and send this file:";
      put "ERR" "OR: &errfile.";
      call execute('
      data _null_; file batfile; comp_cmd2 = '''||strip(comp_cmd2)||''';put
comp_cmd2;run;
      data _null_; rc = system("""&batfile."""); rc1 = fdelete("batfile"); run;
      '
      );
    end;
  run;
%mend;


*******************************************************************************
Macro %st_simple_doc(contents, pdffile, debug=N)
Generates a simple document.

Parameters:
1) contents: A list of datasets
   If only one variable named `typst`, the values will be inserted as Typst source
code
   If multiple variables in a dataset, a table will be inserted.
2) pdffile: Path of the output PDF file.
3) debug: Y/N, control whether to keep the data folder and Typst file.
*******************************************************************************;
%macro st_simple_doc(
  contents,
  pdffile,
  debug=N
);
  %*g_pathname is the global macro variable for currently folder;
  %*If not defined, set to null;
  %if not %symexist(g_pathname) %then %do;
    %let g_pathname =;
```

```
  %end;
  %local table_data_folder typfile;
  data _null_;
    table_data_folder = "&g_pathname._temp_simple_doc\";
    typfile = "&g_pathname._temp_simple_doc.typ";
    call symput('table_data_folder', strip(table_data_folder));
    call symput('typfile', strip(typfile));
  run;

  %st_create_folder(&table_data_folder.);

  data _content_list;
    length contents $2000;
    contents = compbl(symget('contents'));
    do i = 1 to count(strip(contents), ' ')+1;
      memname = lowcase(scan(strip(contents), i, ' '));
      idx = i;
      output;
    end;
    drop i;
  run;

  proc sql noprint;
  create table _content_vars as
    select a.idx, a.memname, lowcase(b.name) as name, count(b.name) as nvars
    from _content_list a inner join dictionary.columns b
    on lowcase(b.libname) = "work" and lowcase(a.memname) = lowcase(b.memname)
    group by a.memname
    order by a.idx
    ;
  quit;

  %*Generate Typst code for each content;
  data _content_list_typst;
    set _content_vars;
    by idx;
    if first.idx;
    if nvars > 1 or (nvars = 1 and name ^= "typst") then type = 'table';
    length typst_dsname $50 _sas_code $200 _sas_code_callcsv $2000;
    if type = 'table' then do;
      typst_dsname = "_t_"||strip(memname);
      _sas_code = 'data {typst_dsname}; typst = ''#csvtable("{memname}")''; run;';
      _sas_code = tranwrd(_sas_code, '{typst_dsname}', strip(typst_dsname));
      _sas_code = tranwrd(_sas_code, '{memname}', strip(memname));
      call execute(_sas_code);

      _sas_code_callcsv = '%nrstr(%st_csv({memname}, {folder}))';
      _sas_code_callcsv = tranwrd(_sas_code_callcsv, '{memname}', strip(memname));
      _sas_code_callcsv = tranwrd(_sas_code_callcsv, '{folder}',
symget('table_data_folder'));
      call execute(_sas_code_callcsv);
    end;
    else typst_dsname = memname;
    drop nvars name _sas_code _sas_code_callcsv;
  run;

  %local typst_dsname;
```

```
  proc sql noprint;
    select typst_dsname into :typst_dsname separated by ' '
    from _content_list_typst;
  quit;

  %* Define a Typst function that display CSV as a table;
  data _typst_csvtable;
    typst = '
#let csvtable(memname) = {~nl~
  show "\u{0000}\u{0020}": "\n"~nl~
    let datapath = ".\\_temp_simple_doc\\"+memname+".csv"~nl~
    let data = csv(datapath).map(~nl~
        row=>row.map(~nl~
            cell=>if cell.starts-with("[") and cell.ends-with("]") {eval(cell)} else
{cell})~nl~
    )~nl~
    let colhdr = data.first()~nl~
    let body = if data.len() >=2 {data.slice(1)} else {()}~nl~
    table(columns: colhdr.len(), table.header(..colhdr), ..body.flatten())~nl~
}
';
  typst = tranwrd(typst, "~nl~", "0d0a"x);
  run;

  %st_typ(_typst_csvtable &typst_dsname., &typfile.);

  %st_compile(typfile=&typfile., pdffile=&pdffile.);

  %if &debug. ^= Y  %then %do;
  data _null_;
    rc1 = fdelete("typfile");
  run;

  %st_delete_folder(&table_data_folder.);
  %end;
%mend;
```