SEASUG 2026 - Paper 86

Le petit SAS®package - a workshop about the "Hello World" SAS package

Bartosz Jabłoński - yabwon / Warsaw University of Technology

ABSTRACT

Paraphrasing Antoine de Saint-Exupéry's "*The Little Prince*"[a] [**de Saint-Exupéry 1943**]:

It was then that the SAS package appeared.
- "Good morning," said the SAS package.
- "Good morning," the little SAS programmer responded politely, although when he turned around he saw nothing.
- "I am right here," the voice said, "under the apple tree."
- "Who are you?" asked the little SAS programmer, and added, "The way you help solving SAS problems is very pretty to look at."
- "I am a SAS package," said the SAS package.
- "Come and play with me," proposed the little SAS programmer. "I am so unhappy."
- "I cannot play with you," the SAS package said. "I am not tamed."
- "Ah! Please excuse me," said the little SAS programmer. But, after some thought, he added [...] "I am looking for friends. What does that mean - 'tame'?"
- "It is an act too often neglected," said the SAS package. "It means to establish ties."
- "'To establish ties'?"
- "Just that," said the SAS package. "To me, you are still nothing more than a SAS programmer who is just like a hundred thousand other SAS programmers. And I have no need of you. And you, on your part, have no need of me. To you, I am nothing more than a code snippet like a hundred thousand other snippets. But if you tame me, then we shall need each other. To me, you will be unique in all the world. To you, I shall be unique in all the world..."

The SAS package gazed at the little SAS programmer, for a long time.
- "Please - tame me!" he said.
- "I want to, very much," the little SAS programmer replied. "But I have not much time. I have friends to discover, and a great many things to understand." - "One only understands the things that one tames," said the SAS package. "Men have no more time to understand anything. They buy things all ready made at the shops. But there is no shop anywhere where one can buy friendship, and so men have no friends any more. If you want a friend, tame me..."
- "What must I do, to tame you?" asked the little SAS programmer.

This article is a background for a demonstration of how easy it is to tame the SAS packages building tool - the SAS Packages Framework - and become friends with SAS packages.

[a] see `https://en.wikipedia.org/wiki/The_Little_Prince`

Le Petit SAS package

a workshop about the "Hello World" SAS package.



Table of contents

IF YOU PLEASE - DRAW ME A SHEEP! . . . 3
WHAT IS A SAS PACKAGE? . . . 3
LINKS AND LOCATIONS . . . 3
IF YOU ARE LAZY OR FORGETFUL . . . 4
CODE TYPES . . . 5
A MACRO . . . 5
A FORMAT . . . 6
A FUNCTION . . . 6
IF YOU PLEASE - BUILD ME A SAS PACKAGE! . . . 7
THE DIRECTORY . . . 7
THE DESCRIPTION . . . 8
CODE PREPARATION . . . 9
HELP NOTES . . . 9
ONE FILE - ONE OBJECT, AND OTHER RULES . . . 10
TESTS . . . 11
GENERATE PACKAGE . . . 11
CONCLUSIONS, A NEW SAS SESSION . . . 12
REFERENCES . . . 12
Appendix A - code coloring guide . . . 14
INDEX . . . 14

IF YOU PLEASE - DRAW ME A SHEEP!

The first thing The Little Prince asked Antoine de Saint-Exupéry, when they met in the Sahara Desert, was to draw him a sheep. The sole purpose of the sheep was to protect The Little Prince's planet from destruction by eating the little baobabs before trees became too large and crush the little globe with they roots. The drawings Exupéry provided were criticized by The Little Prince. The first sheep was "already very sickly." The second was to big because: "This is a ram. It has horns." The third one "is too old."

Finally, exhausted Exupéry, tossed off a drawing of a box and threw out an explanation with it:

"*This is only his box. The sheep you asked for is inside.*"

The Little Prince, to Exupéry's great surprise, answered: "*That is exactly the way I wanted it!*"

The animal that seemed perfectly suited for baobabs was the one that was not even in the picture! Well, it was, but in the box...

Looking at SAS programming tasks that can be presented, let's say by a customer[1], to a SAS programmer, one can easily see the analogy: "*this code that I need to do A, B, C you wrote... is to hard for me, you are using hash tables there...*", "*this code that I need to do E, F, G... is to complex, I can't handle 17 macros...*", or "*this code that I need to do X, Y, Z... is too convoluted, I never heard about FCMPs...*", and so on. But the very same code encapsulated inside a nicely documented and tested SAS package (a little box) with simple user interface, e.g., through a single macro or singe function, can suddenly become an acceptable approach that everyone says "OK" to.

In this article, I want to take you for an adventure where you learn how to tame SAS packages and become friends with them!

WHAT IS A SAS PACKAGE?

Packages are code containers created for users by users (called *developers* in this case) to share SAS code in a convenient and practical way. A semi-formal definition goes as follows:

Definition: SAS PACKAGE

A **SAS package** is an automatically generated, single, stand-alone `zip` file containing organized and ordered code structures, created by the developer and extended with additional automatically generated "driving" files (i.e., description, metadata, load, unload, and help files).

The *purpose* of a package is to be a simple, and easy to access, code-sharing medium, which allows: on the one hand, to separate the code complex dependencies created by the developer from the user experience with the final product and, on the other hand, reduce developer's and user's unnecessary frustration related to the deployment (installation) process.

The essence of a package is to combine all code the developer wants to share, contained and organized into a single file, with additional driving code to facilitate all those features that a tool like a package would be expected to have or to provide (e.g., loading, getting help, etc.). All this is done according to the "K.I.S.S." principle[2] to minimize possible confusion.

LINKS AND LOCATIONS

If you felt like you had read that previous paragraph somewhere already it is because there is a fair chance you did. There are many publicly available articles and resources that can give you a pretty solid background and knowledge boost about SAS packages.

[1] By a customer here I mean not only a "regular" customer; I mean also a fellow programmer, a teammate who asks for help, or just another/unknown SAS user.

[2] K.I.S.S. - Keep It Simple Stupid

Articles presenting, discussing, or explaining the idea of code sharing through SAS packages, with the help of SAS Packages Framework are the following: [**Jablonski 2020**] and its extended version, [**Jablonski 2021**], [**Jablonski 2023**], [**Jablonski 2024**], [**Jablonski 2025**], or [**Jablonski 2026**].

The tool that allows you to build and use SAS packages, the SAS Packages Framework , with all the documentation and examples, is hosted at the following GitHub repository (the Framework):

https://github.com/yabwon/SAS_PACKAGES

The Hands-on Workshop Materials that will teach you how to work with SAS packages and that will take you "from 0 to hero" in SAS packages world can be located at:

https://github.com/yabwon/HoW-SASPackages

These are very extensive materials, 4+ hours long with instructor or 8+ hours long without one. I recommend using them after you finish reading this article.

If you have not had the chance to see talk presenting this article at the SEASUG conference, you can watch a video recording from the NJSUG Webinar (May 8th 2026), where this subject was discussed for the first time. The link is:

https://drive.google.com/file/d/1y2_HO0tTnuiv3pk5swwETsd5Mi7EeFBP/view

and all programs used during the recording can be found in:

https://github.com/yabwon/SAS_PACKAGES/tree/main/SPF/Documentation/

in the **LePetitSASpackage** directory.

Other SAS packages related recordings can be found in the "Recordings and Presentations" section of the SAS Packages Framework's GitHub repository:

https://github.com/yabwon/SAS_PACKAGES#recordings-and-presentations

This **YouTube introductory video series explaining the basics** may also be a thing worth watching:

https://www.youtube.com/playlist?list=PLeMzGEImIT5eV13IGXQIgWmTFCJt_cLZG

Additionally, there are two officially supported public SAS packages repositories: SAS Packages Archive (SASPAC for short) and PharmaForest. SASPAC, is the default target location for searching packages and is set up under the following link:

https://github.com/SASPAC

SASPAC is a dedicated place for all sorts of publicly available packages (as long as they satisfy certain quality standards) created for solving various tasks and needs. You can say it is an "industry-agnostic" repository, dedicated to all SAS packages users.

The second one, PharmaForest, organized by the *PHUSE Data Visualization & Open Source Technology* working group (started in PHUSE Japan OST), is located under:

https://github.com/PharmaForest

It is focused on pharmaceutical industry SAS programmers.

In case you have any troubles finding any of above resources, do not hesitate to contact me directly (see CONTACT INFORMATION). I will gladly provide you with further explanations or directions.

IF YOU ARE LAZY OR FORGETFUL

Now, I will quietly assume that you have looked at some of the articles or links from previous session. I assume that at least, you have watched that short YouTube series showing you how to start and do the "baby steps", like installation of the SAS Packages Framework and publicly accessible packages (see SASPAC), running your SAS session with SPF enabled, and loading packages.

If you are lazy or forgetful, like I am, my recommendation is to use the `autoexec.sas` file to automatically start you SAS session with the Framework on and packages ready. For example, an `autoexec.sas` file similar to this one (a snippet from my own) can do the job for you:

code: autoexec.sas

```
/* set "current working path" to WORK location */
%put %sysfunc(dlgcdir(%sysfunc(pathname(work))));
filename _ ".";
filename _ list;
filename _ clear;

/* enable the SAS Packages Framework */
filename packages "C:/SAS_WORK/SAS_PACKAGES"; /* or e.g. "/sas/PACKAGES"; */
%include packages(SPFinit.sas);

/* Load SAS packages */
%loadPackageS(SQLinDS BasePlus)

/* List loaded packages */
%put NOTE- SAS session with SAS Packages: &SYSLoadedPackages.;

%put NOTE- %workpath(); /* macro from basePlus package */
```

can be used for the job. First, the code snippet sets "current working path" to WORK location (using `%SYSFUNC`). Next it enables the SPF (the `FILENAME` and the `%INCLUDE` statements), loads two packages (the `%loadPackageS()` macro), lists loaded packages (the `&SYSLOADEDPACKAGES.` macro variable), and uses `%workpath()` macro to display WORK's location.

CODE TYPES

SAS Packages can contain various code types, including, but not limited to: macros, FCMP functions, formats and informats, IML modules, DS2 packages and threads, CAS-L functions, data sets, libraries, etc. Full list is provided in the Hands on Workshop Materials (see the LINKS AND LOCATIONS section). In this article we will focus on three basic and probably the most popular cases: a macro, a function, and a format. Let's assume their code is as follows.

A MACRO

The `%fox()` macro prints one of three most famous quotes of the fox to the LOG.

code: a macro

```
%macro fox(quote);
  %local n e w;
  %let n = NOTE;
  %let e = ERROR;
  %let w = WARNING;
  %if 1=%superq(quote) %then
    %do;
      %put &n.- And now here is my secret, a very simple secret:;
      %put &n.- It is only with the heart that one can see rightly%str(;);
      %put &n.- what is essential is invisible to the eye.;
    %end;
  %else %if 2=%superq(quote) %then
    %do;
      %put &w.- It is the time you have wasted for your rose;
      %put &w.- that makes your rose so important.;
      %put &w.- Men have forgotten this truth. But you must not forget it.;
      %put &w.- You become responsible, forever, for what you have tamed.;
      %put &w.- You are responsible for your rose...;
    %end;
```

```
  %else
    %do;
      %put &e.- One only understands the things that one tames.;
      %put &e.- Men have no more time to understand anything.;
      %put &e.- They buy things all ready made at the shops.;
      %put &e.- But there is no shop anywhere where one can buy friendship,;
      %put &e.- and so men have no friends any more.;
      %put &e.- If you want a friend, tame me...;
    %end;
%mend fox;
```

You would call this macro by providing a value 1, 2, and anything else goes to the default quote, like this:

code: macro call

```
%fox(1)
%fox(2)
%fox()
```

and quotes printed are "color coded" according to LOG's message type.

A FORMAT

The ROSE. format displays values from 1 to 4 as rose's quotes.

code: a format

```
PROC FORMAT;
  value rose
  1="Ah! I am scarcely awake. I beg that you will excuse me.
     My petals are still all disarranged..."
  2="Of course I love you. It is my fault that you have not known
     it all the while. [...] Try to be happy..."
  3="My cold is not so bad as all that...
     The cool night air will do me good. I am a flower."
  4="Well, I must endure the presence of two or three caterpillars
     if I wish to become acquainted with the butterflies."
  other="ERROR: QUOTE OUT OF RANGE!"
  ;
RUN;
```

The ROSE. format can be used to decorate the PUT statement.

code: format use case

```
data _null_;
  do i = 1 to 5;
    put "NOTE- " i rose. /;
  end;
run;
```

A FUNCTION

This FCMP function PRINCE returns the prince's quote: "*If you please - draw me a sheep!*", and as an extra thing, it generates a random rose's quote in the LOG.

code: a function

```
PROC FCMP outlib=work.little.prince;
  function prince() $ 42;
    file log;

    length i $ 256;
    r=rand('integer',1,4);
```

```
    i = put(r, rose.);
    put @1 "RANDOM NOTE:" i /;

    return("If you please--draw me a sheep!");
  endfunc;
QUIT;
```

For an FCMP function to work the CMPLIB option has to be updated.

code: cmplib update

```
options append=(cmplib=work.little);

proc options option=cmplib;
run;
```

code: function use case

```
data _null_;
  do i = 1 to 5;
    prince=prince();
    rc=sleep(1,0.2);
  end;
  put prince=;
run;
```

Those three "object" are very simplified versions of what you as a programmer could develop or would like to share with other SAS users, but they are "complex" enough to work as teaching examples for the upcoming sections.

IF YOU PLEASE - BUILD ME A SAS PACKAGE!

If the code (like those three snippets from previous section) you would like to share through a package is written, believe it or not, *you are 80 to 90 percent done* with building a package. The remaining 10 to 20 percent is the process of adding the organizational structure to the code and generating the package file. Creating the functionality and its code requires the most effort from you, generating a package is just a "formality". You can think of it as a birthday gift analogy. Imagine you want to give your friend a crocheted stuffed mascot. It is crocheting that is the hardest part, not wrapping the mascot in the gift box, isn't it?

[NOTE] The tutorial assumes you have your code split in threes separate files. If this is not the case, do not worry. Read more about the %splitCodeForPackage() macro, it will help you with the splitting job so you will not have to do it manually.

THE DIRECTORY

A package generation process starting point is creation of a directory where the source code and description file are located. The following code snippet uses the DLCREATEDIR option and the LIBNAME statement to help us create a storage location for package's content:

code: package content directory

```
options dlcreatedir;
%let dir = <path/of/your/choice>/LePetitSASpackageDirectory;

libname p "&dir.";
```

The LePetitSASpackageDirectory subdirectory is created in the <path/of/your/choice> directory.

Source code that establishes a package content is organized by type and ordering in subdirectories inside the `LePetitSASpackageDirectory`. The naming convention for those subdirectories is `NNN_TYPE`, where `NNN` indicates ordering number, e.g., `001`, `002`, `003`, etc. that resolves code dependencies and enforces the package's loading order. The `TYPE` part indicates a content type, i.e., the type/kind of programs that are located inside the directory. Content types, as discussed earlier, spreads across many different kinds of programs, but for the purpose of this article we are focusing on `macro`, `formats`, and `functions` (both last type names in plural form).

Since the programs we have are not independent (the function uses the format in its definition) the loading order of the package content will matter. We can choose from a few possible cases, but the format loading should be done before function loading, and therefore the `NNN` for `formats` has to be smaller than for `functions`. Having `DLCREATEDIR` option on the directories structure can be easily created with the following snippet:

code: ordering package content directories

```
libname p "&dir./01_macro";
libname p "&dir./02_formats";
libname p "&dir./03_functions";
```

If the option is not available, then it can be done by hand.

THE DESCRIPTION

Before distributing programs' files to their corresponding directories let's create the description file. Description file contains metadata for a package. Its structure is very simple and the best way to learn is by example. For the package discussed here the description.sas file can have the following form:

code: description.sas

```
/* **HEADER** */
Type: Package
Package: LePetitSASpackage
Title: Le petit SAS package - a workshop about the "*Hello World*" SAS package.
Version: 0.0.1
Author: Bartosz Jablonski
Maintainer: Bartosz Jablonski (yabwon@gmail.com)
License: MIT
Encoding: UTF8

Required: "Base SAS Software"
ReqPackages: "SQLinDS(2.3.3)"

/* **DESCRIPTION** */
/* All the text below will be used for help notes */
DESCRIPTION START:
The **LePetitSASpackage** package is an implementation
of a little "*Hello World*" SAS package presented during
**NJSUG** meetup (and **SEASUG** conference).

It is build for fun, but also (or foremost) to show us
how easy it is to build SAS packages.

*"I have serious reason to believe that the planet from which
the little SAS package came is the asteroid known as B612.
This asteroid has only once been seen through the telescope.
That was by a Turkish astronomer, in 1909."*
DESCRIPTION END:
```

The color coding helps to distinguish between obligatory and optional tags in the `description.sas` file. Orange components, i.e., Type:, Package:, or DESCRIPTION START: and DESCRIPTION END: are necessary, the SAS Packages Framework will not generate a package without them. The dark blue Required: and ReqPackages: are optional.

Tags are separated from their data portion by the colon (`:`) symbol. The Header part data should be one liners, i.e., whole tag data fit in one line. The Description part data, i.e., all text between DESCRIPTION START: and DESCRIPTION END: tags, is a free text that can span across multiple lines.

The meaning of the Header tags are the following:

- `Type:` - Indicates this description file is a package description file, it goes with "package" value;
- `Package:` - the name of a package, it goes with standard SAS naming convention except the length, name can be max 24 bytes;
- `Title:` - A single sentence, a slogan-like phrase, describing a package;
- `Version:` - Package version, it supports `major.minor.patch` versioning system;
- `Author:` - Package authors list;
- `Maintainer:` - Package maintainers list;
- `License:` - Type of license under which the package is distributed, the default is MIT (additionally MIT license file is generated automatically if license file is not provided), if you want different license then MIT you need to do two things: 1) provide license type name in the description file, and 2) provide a `license.sas` file with your license text.
- `Encoding:` - Encoding used to read files from the package, preferably UTF-8, but it is your decision which encoding to choose.
- `Required:` - [Optional] Quoted and comma separated list of licensed SAS components required by the package, e.g., `"Base SAS Software", "SAS/IML"`;
- `ReqPackages:` - [Optional] Quoted and comma separated list of SAS packages (with versions in parenthesis) on which this package is dependent and requires to run correctly, e.g., `"package1(1.2.3)", "package2(4.5.6)", "package3(7.8.9)"`;

The text between DESCRIPTION START: and DESCRIPTION END: tags serves as a general help information for the package that is displayed whenever the `%helpPackage()` macro is called for the package. Since the help notes and documentation popped up, let's discuss this subject.

[NOTE] The `SQLinDS(2.3.3)` package is added through `ReqPackages:` tag just for illustrative purpose, none of `fox`, `rose`, or `prince` need `SQLinDS` components to work so normally you would not add the dependency.

CODE PREPARATION

As Kurt Bremser's[3] Maxim 16 says: "Anything that is not properly documented, does not exist" (see [**Bremser 2019**]) The documentation is a crucial aspect of package creation and every solid SAS programmer should provide software that is adequately documented. Fortunately creating help notes and documentation for a package is very straightforward.

Help Notes SAS packages documentation is "in-code embedded documentation", i.e., you simply insert additional comment style tags and everything (text or code) between those tags became part of package's documentation. For example, documented code for the `PRINCE` function could look like this:

code: documentation tags

```
1 /*** HELP START ***//*
2
3 The 'prince()' function asks to draw a sheep...
4
```

[3] see: https://communities.sas.com/t5/SAS-Communities-Library/Maxims-of-Maximally-Efficient-SAS-Programmers/ta-p/352068

```
---

### Arguments: ############################

The function has no arguments.

### Dependencies: ########################

The 'prince()' function requires
the 'rose.' format to work.

### Example: #############################

Ask for a sheep:
~~~~~~~~~~sas
data _null_;
  s=prince();
  put s=;
run;
~~~~~~~~~~

---

*//*** HELP END ***/

function prince() $ 42;
  file log;

  length i $ 256;
  r=rand('integer',1,4);
  i = put(r, rose.);
  put @1 "RANDOM NOTE:" i /;

  return("If you please--draw me a sheep!");
endfunc;
```

Whenever the `%helpPackage()` macro is called as: `%helpPackage(LePetitSASpackage, prince)`, that lines between tagged portion are displayed in the LOG (notice how the comment markers "/*" and "*/" are excluded from the LOG printout by putting them in the same line as help tags). Additionally, when package generation is executed with `markdownDoc=` parameter set to 1 those parts of the program file marked by help tags are used to generate the markdown file documentation.

If you forget adding documentation markers, or you accidentally mess them up, the SAS Packages Framework will let you know about the situation issuing warnings and errors during package generation process.

One file - one object, and other rules When the documentation is added to all your program files, the next step is to arrange files into directories. There are some general rules you have to follow. The first rule is: one program (object) goes to one file, e.g., macro `%fox()` goes to `fox.sas` file, function `PRINCE` goes to `prince.sas` file. This is almost exactly the same way you would prepare your macros for `sasautos` There is one exception, for formats and informats. For formats and informats all four objects sharing the same name can be stored in a single file (have numeric and character format and numeric and character informat named ABC). The next rule is: all file names have to be in lowercase letters (e.g., use `fox.sas`, not `FoX.sas`). There is a difference when you prepare code for `functions` and `formats` (plural) versus `function` and `format` (singular). In the first situation, the file should contain only the definition of function or format/informat without `PROC FCMP` or `PROC FORMAT` header and the `RUN` statement, just like in the

code snipped in the Help Notes section above. The second case (singular form types) is described in details in [**Jablonski 2020**].

Tests The SAS Packages Framework offers a testing facility. Tests are optionally available, i.e., they are available if the XCMD option is on. When the option is on, one test, the package loading test, is always automatically executed. All other tests are the developer's job to do. The SPF automatically points to the package location for tests, the developer doesn't have to worry. To run tests you create a `test` type directory, e.g., `99_test`, and put all the tests you want to be executed during package generation there (e.g., `test1.sas` and `test2.sas`). One file indicates one test, it is your choice what granularity they will have. Every file is executed in a separate SAS session (that is why the XCMD is needed), the Framework automatically assigns SPF and the package you test through test dedicated autoexec.sas. At the end of the process the Framework returns an output window table and LOG notes summarizing the process.

At this point, taking into account tests, program files, and description, the structure of the `LePetitSASpackage` packages directory would look like this:

```
..
+-[LePetitSASpackageDirectory]
  |
  +-[01_macro]
  | |
  | +-<fox.sas>
  |
  +-[02_formats]
  | |
  | +-<rose.sas>
  |
  +-[03_functions]
  | |
  | +-<prince.sas>
  |
  +-[99_test]
  | |
  | +-<test1.sas>
  | |
  | +-<tets2.sas>
  |
  +-<description.sas>
```

Square brackets indicate directories, angle brackets indicate files.

GENERATE PACKAGE

The package content is ready! It is time to generate the `LePetitSASpackage` package.

In the SAS packages developer's SAS session you have to point the location for packages and enable the Framework `SPFinit.sas` file. A code snippet similar to the following one should be all you need:

code: enabling SPF

```
1 filename packages "/path/where/the/framework/and/packages/reside";
2 %include packages(SPFinit.sas);
```

The %generatePackage() macro is the one that takes source code and generates package out of it. How it works and options it takes can be easily reminded (in he LOG) by calling the macro with the HELP keyword.

code: help

```
%generatePackage(HELP)
```

The code that generate package and and additionally provide markdown documentation has just five lines:

code: Generate Package

```
%generatePackage(
 <path/of/your/choice>/LePetitSASpackageDirectory
,markdownDoc=1
,packages=/path/where/the/framework/and/packages/reside
)

/* REMEMBER! Always check the log.*/
```

The first parameter points to the source directory of the package, the second enables markdown documentation, and the third identifies the place where SPF and SQLinDS reside. After the execution, every step of generation process is fully traceable and documented in the LOG, summary tables are di/splayed in the output window too.

If you followed this tutorial step-by-step and did everything correctly your LOG should have no errors, and only one warning saying that the `licence.sas` file with the MIT license text was generated automatically by the SAS Packages Framework for you. The warning is the consequence of the fact that the `licence.sas` was not provided. The Framework recognizes this situation and alerts you to the fact.

If the LOG is clear and the summary output satisfies your expectations, and the provided test files passed, this is it! The package is ready! Congratulations! You can share your first package with other SAS programmers now.

CONCLUSIONS, A NEW SAS SESSION

In this article you have learned how to build SAS packages with the SAS Packages Framework. The learning example was pretty simple, but at the same time complex enough to point your attention to important parts of the process. The SEASUG conference presentation and the recording from NJSUG meetup were both prepared for 45 and 50 minutes talks respectively. It is sufficient enough time to share knowledge for a good start. I hope it was enough to pique your curiosity regarding SAS packages development. The next steps I recommend to keep the curiosity flame burning and level up your skills to "pro" are: start a new SAS session, go to the tutorial materials mentioned earlier, and do the full "from 0 to here" course.

The End

REFERENCES

[de Saint-Exupéry 1943] Antoine de Saint-Exupéry, "Le Petit Prince" ("The Little Prince"), Published in U.S. by Reynal & Hitchcock (1943)

[Bremser 2019] Kurt Bremser, *"Maxims of Maximally Efficient SAS Programmers"*, SAS Global Forum 2019 Proceedings, 3062-2019 https://support.sas.com/resources/papers/proceedings19/3062-2019.pdf

[Jablonski 2020] Bartosz Jabłoński, "SAS Packages: The Way to Share (a How To)", SAS Global Forum 2020 Proceedings, 4725-2020 https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/4725-2020.pdf

extended version is located at: https://github.com/yabwon/SAS_PACKAGES/blob/main/SPF/Documentation
and also: https://pages.mini.pw.edu.pl/~jablonskib/SASpublic/Paper_4725-2020_extended.pdf

[Jablonski 2021] Bartosz Jabłoński, "My First SAS Package - a How To",
SAS Global Forum 2021 Proceedings, 1079-2021
https://communities.sas.com/kntur85557/attachments/kntur85557/proceedings-2021/59/1/Paper_1079-2021.pdf
also available at: https://github.com/yabwon/SAS_PACKAGES/tree/main/SPF/Documentation/Paper_1079-2021

[Jablonski 2023] Bartosz Jabłoński, "Share your code with SAS Packages a Hands-on-Workshop",
WUSS 2023 Proceedings, 208-2023, https://www.wuss.org/proceedings/2023/WUSS-2023-Paper-208.pdf

[Jablonski 2024] Bartosz Jabłoński, "Use of SAS Packages in the Pharma Industry – Opportunities, Possibilities, and Benefits",
PHUSE 2024 Proceedings, SM01-2024,
https://phuse.s3.eu-central-1.amazonaws.com/Archive/2024/Connect/EU/Strasbourg/PAP_SM01.pdf

[Jablonski 2025] Bartosz Jabłoński, "SAS packages - a modern code sharing medium for SAS",
SESUG 2025 Proceedings, HOT-012, https://sesug.org/SESUG2025/index.php ,
select *SESUG 2025 Papers ZIP File* to download.

[Jablonski 2026] Bartosz Jabłoński, "SAS Packages - an Ask About Anything Game",
PharmaSUG 2026 Proceedings, AP-124, https://pharmasug.org/proceedings/2026/AP/PharmaSUG-2026-AP-124.pdf

ACKNOWLEDGMENTS

The author would like to acknowledge Louise Hadden for her help in proofreading and polishing linguistic aspects of the final article.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at one of the following e-mail addresses:

yabwon✉gmail.com or bartosz.jablonski✉pw.edu.pl

or via the following LinkedIn profile: www.linkedin.com/in/yabwon
or at the communities.sas.com by mentioning @yabwon.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

Appendix A - code coloring guide

The best experience for reading this article is in color and the following convention is used:

- The code snippets use the following coloring convention:

code: is surrounded by a black frame

```
In general, we use black ink for the code but:
- code of interest is in orange ink so that it can be highlighted,
- to distinguish code snippets for easier reading dark blue is used,
- and comments pertaining to code are in a bluish ink for easier reading.
```

- The LOG uses the following coloring convention:

the log - is surrounded by a blueish frame

```
The source code and general log text are blueish.
Log NOTEs are green.
Log WARNINGs are violet.
Log ERRORs are red.
Log text generated by the user is purple.
```

INDEX

concept
  AUTOEXEC.SAS, 5, 11
  FCMP FUNCTION, 6
  SAS PACKAGE, 3

format (numeric)
  ROSE., 6
function
  PRINCE, 6, 9, 10

macro
  %FOX(), 5, 10
  %GENERATEPACKAGE(), 12
  %HELPPACKAGE(), 9, 10
  %LOADPACKAGES(), 5
  %SPLITCODEFORPACKAGE(), 7
  %WORKPATH(), 5
macro-statement
  %SYSFUNC, 5
macro-variable
  &SYSLOADEDPACKAGES., 5

option
  CMPLIB, 7
  DLCREATEDIR, 7, 8
  XCMD, 11

procedure
  FCMP, 10
  FORMAT, 10

SPF code type
  formats, 8, 10
  format, 10
  functions, 8, 10
  function, 10
  macro, 8
  test, 11
SPF concept
  CONTENT TYPE, 8
  DESCRIPTION FILE, 8
  DESCRIPTION PART, 9
  DESCRIPTION.SAS, 8
  HEADER PART, 9
statement
  FILENAME, 5
  LIBNAME, 7
  RUN, 10
  %INCLUDE, 5