PHUSE EU 2024, Paper SM01 [Extended version, 2024.11.11]

Use of SAS®Packages in the ~~Pharma~~Any Industry – Opportunities, Possibilities, and Benefits

Bartosz Jabłoński, yabwon/Warsaw University of Technology, Warsaw, Poland

ABSTRACT

When working with SAS®code, especially with a complex one, there is a moment when you decide to break it into small pieces. You create separate files for macros, formats/informats, for functions or data too. Eventually the code is ready and tested, and sooner or later you will want to share it with another SAS programmer. You have developed a program using local PC SAS, but the deployment is on a server with a different OS. Your code is complex (with dependencies such as multiple macros, formats, datasets, etc.) and is difficult to share. Often when you try to share code, the receiver will quickly encounter an error because of a missing macro, missing format, or whatever...small challenge, isn't it? In this article we discuss a solution to the problem - the idea of SAS Packages, what they are, how to use and develop them, and how to make code-sharing a piece of cake. And of course what opportunities, possibilities, and benefits SAS Packages bring to the pharmaceutical (among others) industry are also discussed.

INTRODUCTION

The idea of SAS Packages and the SAS Packages Framework (SPF) project started late spring, early summer of 2019[1]. It was motivated and driven by two happenings. The first one was that I was seeking an interesting presentation idea to submit to the call-for-content for the (late) SAS Global Forum conference. The second was the end of a project happening in a bank where I was working at that time. The project was executed in a setup where the data (stored in the Hadoop's big data environment) were processed in analytic environments driven by SAS and R programming languages. The "crops, harvests, and fruits" of the project were, on "the SAS-side", a big pile of files with macros, formats and functions, that could not be managed without "30-minutes-to-one-hour-long" session explaining code dependencies. And on "the R-side" it was a nice single file - an R package - "enveloping" all the project's related R programs. This lead me to a reflection:

> *Modern data focused languages, like R or Python, have vast ecosystems for building packages. Those environments allow their users to share their ideas, inventions, and code in an easy, almost seamless way. Why doesn't SAS, with its profound and historically well-established impact on data analysis, embrace such a marvelous idea?*

This reflection led me to the SAS packages idea and development of the SAS Packages Framework, a tool which allows users to build, use, and share SAS Packages.

The initial development and programming, from an empty file to the first working tool, took about few weeks. The framework and packages form were influenced by the R programming language (see [**Wickham 2015**])

[1] According to the GIT repository's log, the first commit is dated June 18th, 2019.

and I dare to say that the idea of SAS Packages can be considered a rare case when a good thing came from jealousy.

A technical description of the SAS Packages Framework and tutorials on how to use and develop SAS Packages with help of the SPF can be found in articles [**Jablonski 2020**], [**Jablonski 2021**], and [**Jablonski 2023**]. All three together contains approximately 90+ pages of condensed technical knowledge and explanation about "how packages and SPF work". In this article we try to focus our attention about what opportunities, possibilities, and benefits SAS Packages bring to SAS developers and users, including (and not limited to) those working in the pharmaceutical industry.

THINGS WE ARE NOT TALKING ABOUT

To be clear, word "package" appears in the SAS ecosystem in multiple places and contexts but every time it is "different" from the one we are discussing in this article. To name at least few places where a curious reader struggling through the depths of SAS documentation may encounter packages or packages-like entities we have to mention the following:

- The `DS2` procedure [**SAS doc. 1**] packages are libraries storing PROC DS2 methods in data sets. The `DS2` procedure ofers both predefined and user-defined packages. See also [**Jordan 2018**] to learn more.
- The `FCMP` procedure [**SAS doc. 2**] packages separates functions groups in data sets storing functions. Unfortunately, these type of packages are practically useless, especially when user defined functions are used in DATA steps. See also [**Hughes 2024**] to learn more.
- The `IML` procedure [**SAS doc. 3**] package consists of SAS/IML source code (modules), documentation, data sets (matrices), and sample programs. In their idea IML packages are the closest to the one discussed in the article, but they are limited only to the IML language, and are not for the wholes SAS (understood as a "suite of languages"). See also [**Wicklin 2010**] to learn more.
- The SAS Publishing Framework [**SAS doc. 4**], according to the documentation, "*enables you to create packages that contain one or more information entities, including SAS data sets, SAS catalogs, SAS databases, and almost any other type of digital content.*"
- The SAS Output Delivery System (ODS) [**SAS doc. 5**] package is a zip file that, according to the documentation, *is a container for digital content that is generated or collected for delivery to a consumer. ODS packages allow ODS destinations to use the SAS Publishing Framework. An ODS package is an object that contains output files and data sets that are associated with any open ODS destinations.*
- The SAS Enterprise Guide project files [**SAS doc. 6**] with `.egp` is a zip file (like the ODS package above) containing EG project flows and "tiles" they are built of, and XMLs with meta-data.

SAS catalogs are sometimes used as "code containers" but since they are operating system dependent and do not ensure source code access, they are also not considered here.

OPPORTUNITIES, POSSIBILITIES, AND BENEFITS

Writing modular and reusable SAS programs seems to be an issue returning as a boomerang. Article [**Sun 2023**] highlight some of the "everlasting" issues related to the code submitted to the FDA. Examples, among others, are: missing macros, "*all analyses performed in one program*", insufficient comments or header information. Use of SAS packages as a code sharing medium could resolve at least some (if not all) of those unpleasant issues.

Before we describe in details what SAS Packages, SAS Packages Framework (SPF), and SAS Packages Archive (SASPAC) are, let's first analyze the list of reasons why considering SAS packages as a programming approach for code sharing is a good idea:

- SAS Packages allow you to extend SAS capabilities by referencing reusable, centrally maintained code. You write the code once and you can reuse it many times.
- Not only macros can be packaged! You can use different "types" like: user functions (in `FCMP` and `CASL`), IML modules, proc proto `C` language routines, formats, libraries, DS2 packages and threads, and even data generating code.
- When a package delivers formats or functions, automatic updates of the `cmplib=` and the `fmtsearch=` options is executed. Utility macros for IML modules and CAS-L user defined functions for fast loading (with dependency checks) are automatically generated.
- Source code is inside the package so it will never be lost.
- Loading order of the code is organized the way developer intends or needs it to be. This prevents the user from loading content in random order, which leads to dependency errors.
- It is all in one (1) file - you will not forget to share "all that is needed" with your peers.
- "Functionality over complexity" - share one file and say: "Here is my package with the `%ABC()` macro, you use the macro like this and that..." and you do not have to say that there are 173 other utility macros working in the "background" on behalf the `%ABC()`.
- A package contains additional metadata (e.g., version number or generation time-stamp) which supports DevOps approach and makes changes tracking easier. Also, the versioning makes it easier when working with historical projects. For example, a particular version can be saved with the project and there is no fear that "future improvements" will break the project execution.
- Help information provided by the developer can be printed automatically in the log. This makes packages "independent" from internet access. Furthermore, the package does not have to be loaded into the session so the help information can be accessed.
- A "safety vault" called *ICE-loading* allows a package to be loaded into the SAS session even if there is no access to the SPF framework.
- Packages can be shared between different operating systems (Windows, Linux and UNIX).
- The SPF supports dependencies handling between packages. If package X requires package Y to be loaded for running the framework verification of Y's presence in the packages directory will be confirmed.
- "Cherry picking" is allowed to selectively use the content of SAS packages. For example, the user can use only one selected function out of 123 provided by a package.
- It is not only BASE SAS! Packages work with Viya (dedicated macros for Viya are still macros; CAS-L user defined functions - is one of types served) and Workbench too.
- Code is 100% "inspectable" before even loading it to the SAS session. The `%previewPackage()` macro provides the functionality allowing to display the package source code in SAS session log.
- Package integrity/genuinity can be tested with the `%verifyPackage()` macro, so the user is assured that no one modified the package on the way between the user and the developer.
- The SPF facilitates the ability to store and use packages from multiple directories. For example, in the statistical computing environment the user can work with both centrally maintained packages and "private" ones, such as package located in user `home` directory).
- Packages can be provided with an "additional content", that is, not SAS code. Additional documents such as PDFs, figures in PNGs or JPGs, etc.

- "Lazy" data sets loading is supported. "Lazy" data set is a term that describes a data set which is not automatically loaded into the SAS session when a package is loaded. For it to load it has to be explicitly mentioned in the loading process.
- Cleaning functionality. When a package is no longer need in the SAS session its content can be unloaded automatically by the framework.
- Since structure of a package has to be organized in separate files and in sorted types of directories, the package development process "enforces" order in code.

In many of "*SAS vs. R*" or "*SAS vs. Python*" type discussions a mythical argument often returns. It is said that: "SAS stays behind or slowly adopts new statistical solutions because of its proprietary character, and the development process takes more time to adopt those new ideas". The SPF and SAS Packages, and the way they make code sharing easy, provides tools to "bust" that myth. Both the SPF and SAS packages are already used in pharmaceutical and health-care related industries. Here are some articles describing SPF use cases and packages with practical applications in the field: [**Ohuma** *et al.* **2024**], [**Tsutsugo 2024**], [**Jablonski (3) 2024**], [**Jablonski (1) 2024**], [**Jablonski (2) 2024**], [**Alias** *et al.* **2023**], or [**Pusarla** *et al.* **2022**].

SAS PACKAGES

In one way or another modern programming languages, like R or Python, allows you to build and share code libraries between users in the form of packages, which, in a nut shell, are code containers. And guess what? SAS packages work the same way! OK, so what is a SAS package? The description, as presented in [**Jablonski 2020**], would be more or less something like this:

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

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 developers and users unnecessary frustration related to the deployment (installation) process.

The only one "formal" requirement for packages to work is the location. A directory, known here as *the packages directory*, where the SAS Packages Framework (described in the upcoming sections) and SAS packages that are located on the user computer has to be referred with "`packages`" fileref. This means that a code snippet, similar to the one shown below, has to be executed at the beginning of the SAS session and the `packages` fileref has to stay unmodified for the session duration:

code: packages fileref

```
1 FILENAME packages "/path/to/the/framework/and/packages/"; /* packages directory */
```

In the next step the user enables the SPF and from then on the user can enjoy working with packages in the SAS session. Wonderful! But where can you find packages?

SAS PACKAGES ARCHIVE (SASPAC)

SAS packages can be accessed or shared between users in a few ways. One way is by "word of mouth". In this situation a developer creates a package and distributes it among other users by email or by saving it to the office internal drive, somehow like saying: "Hey, here is the package I developed. Use it if you want."

The second way differs from the first only by the distribution method. In this case, after creating a package, the developer locates the package in a network location and shares the `url`. So the package is "semi-publicly" available for those who has the access to the `url` location.

And finally, the third way, is when the developer creates a package, contacts SAS Packages Archive and publishes the package under an official repository available to everyone. The SAS Packages Archive[2] (aka SASPAC) is available under the following address:

https://github.com/SASPAC

SASPAC is a place where publicly available packages are hosted and it gives very convenient ("programmatically doable") way of installing packages. The SASPAC keeps each package in a separate repository. Both the latest version and the historical one are held there, with documentation notes and links, and (in some cases) additional content, such as articles, code snippets, etc.

If a user wants to use a package, that package has to be installed. The installation process boils down to copying the package `zip` file into the packages directory. You can simply "copy+paste" the package file and you are good to go, but... If the developer decides to distribute a package using the second (network location) or the third (SASPAC) "sharing approach", then the package can be installed programmatically from the SAS session using the `%installPackage()` macro (see next section). Of course the SASPAC, since it is publicly hosted, gives the developer the broadest front for sharing packages. And for the user, it is the most easy and convenient way of installing, because all the code needed narrows down to just the following one-liner:

code: installing a package

```
1 %installPackage(<packageName>) /* installation from the SASPAC */
```

Bottom line: the SAS Packages Archive is a place where the "world wide" SAS packages sharing takes place.

SAS PACKAGES FRAMEWORK

The SAS Packages Framework is a group of SAS macros that allows you to build and use SAS Packages. At the moment there is a dozen (12) macros in the framework[3]. Ten of those macros are dedicated for users:

- `%installPackage()`
- `%listPackages()`
- `%verifyPackage()`
- `%previewPackage()`
- `%helpPackage()`
- `%loadPackage()` and `%loadPackageS()`
- `%loadPackageAddcnt()`
- `%unloadPackage()`
- `%extendPackagesFileref()`

and the eleventh and the twelfth one are dedicated to developers:

- `%generatePackage()`
- `%splitCodeForPackage()`

[2] SAS Packages Archive is available publicly since September 30th, 2022.

[3] Initial public release, dated October 13th 2019, had five macros.

The above list is documented on the SPF documentation page, but for each macro similar documentation in text form can be displayed in the SAS log by running any of those macros with the "HELP" argument. For example:

code: framework help

```
%generatePackage(HELP) /* get the SPF help notes */
```

will print help notes for the %generatePackage() macro.

The framework is MIT licensed and fully open source. The SPF is written in pure SAS code and it does not require any additional software to work. If the SAS session is XCMD enabled, the framework allows running packages tests. Generation of automatic documentation file in the form of a markdown .md file is possible as long as the developer provides the help information in proper (i.e. markdown) format. Also, if there is a need, the SPF can be deployed in sasautos.

The file with the SAS Packages Framework is located in the SPF directory under the following GitHub repository:

https://github.com/yabwon/SAS_PACKAGES

As already mentioned, there is only one "formal" requirement for the SPF to work: the packages directory is referred by "packages" fileref. This basically means that a code snippet similar to the following one:

code: packages fileref

```
FILENAME packages "/path/to/the/framework/and/packages/"; /* we know this dir already */
```

has to be executed at the beginning of the SAS session and the packages fileref has to stay unmodified for the session duration.

When the "packages" fileref is set the user just has to enable the framework to run:

code: enabling the SPF

```
%INCLUDE packages(SPFinit.sas); /* get the framework working */
```

To learn how to use both the SPF and packages see "Appendix B - install the SAS Packages Framework and packages" (a brief lesson) or visit he the following location:

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

to find a comprehensive tutorial explaining all the "bells and whistles" for using SAS packages, and all the "nuts and bolts" of SAS packages development.

The SPF's macros allow you to (like their names listed above suggest):

- *install* publicly available packages from the SASPAC archive or from user defined network locations,
- *list* packages available for a SAS session,
- *verify* if a package is genuine (by comparing on-the-fly calculated check sum with developer provided hash digest) to ensure no one "messed up" with the package,
- *preview* source code of package components and print it in the SAS session log,
- print out *help information* provided by the developer in the SAS session log; what is important to note is that the help functionality, and also previewing and verifying, are executable without(!) the need for loading a package to the SAS session (this gives a full transparency of the package content before it will be used),

- *load* package contents (macros, functions, modules, data, formats, etc.) into the SAS session, also with check on packages dependencies or required SAS licenses; also possibility of loading multiple packages at once is facilitated; and if the user is interested in only particular component of a package the "cherry picking", i.e. content selection, is possible; in 99% cases all the code that is needed reduces to:

code: loadinging a package

```
%loadPackage(<packageName>) /* get that package to your SAS session */
```

- *load* and extract *additional content* of a package (if one exists); the functionality is also available when "regular" loading is executed or during installation process,
- *unload* package contents when it is not needed anymore,
- *extend* list of *directories* to which the `packages` fileref points to when access to multiple locations is required,
- *generate new packages* written by developers; testing facility and automatic documentation generation is also available,
- *split* "one-big-code" file into a package structure when the package content is prepared.

CONCLUSIONS

Other programming languages experience proves that packages are very practical tools for sharing your code with others. With the help of the SAS Packages Framework it is now very easily "accessible" also in SAS.

If you find the SAS Package Framework (or any of the official packages) a useful tool for your SAS programs remember to visit the project's GitHub repository and give it a "star". The more "stars", the better visibility. The the better visibility, the more SAS programmers can learn about SPF and have chance to experience and enjoy this code sharing functionality.

Finally, if you consider SPF to be a valuable extension to the SAS system itself, visit the following SASware Ballot Idea thread on `communities.sas.com` forum to support the idea of adding the SPF to SAS:

https://communities.sas.com/t5/SASware-Ballot-Ideas/Add-SAS-Packages-Framework-to-the-SAS-Base-Viya/idi-p/815508

If you know any talented SAS developers that struggle with "convenient" code sharing - share this article with them. If you have had a hard time sharing your code with other SAS users - consider building a SAS package and see how easy it is to work with it.

The End

REFERENCES

[Wicklin 2010] Rick Wicklin, "Statistical Programming with SAS/IML Software", SAS Institute Press, 2010

[Wickham 2015] Hadley Wickham, "R Packages: Organize, Test, Document, and Share Your Code", O'Reilly Media 2015, `http://r-pkgs.had.co.nz/description.html`

[Jordan 2018] Mark Jordan, "Mastering the SAS DS2 Procedure: Advanced Data Wrangling Techniques, Second Edition", SAS Institute Press, 2018

[Jablonski 2020] Bartosz Jabłoński, "SAS Packages: The Way to Share (a How To)", SGF Proceedings, 2020
`https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/4725-2020.pdf`
extended version available at: `https://github.com/yabwon/SAS_PACKAGES/blob/main/SPF/Documentation`

[Jablonski 2021] Bartosz Jabłoński, "My First SAS Package - a How To", SGF Proceedings, 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`

[Pusarla *et al.* 2022] Sunil Kumar Pusarla, Kevin Viel, Dante Di Tommaso, "An open-source project to map lab terminology and standardize units for analysis", PHUSE US 2022 Proceedings, 2022, `https://www.lexjansen.com/phuse-us/2022/ds/PAP_DS02.pdf`

[Alias *et al.* 2023] Eldho Alias, Prasanth Prabhakaran, Rahulraj V R, "SAS Packages for Clinical Programming", PHUSE EU 2023 Proceedings, 2023, https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PAP_CT04.pdf

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

[Sun 2023] Liping Sun, "Experiences of CDER Statistical Analysts with NDA/BLA Reviews: Some Helpful Tips for Sponsors", PharmaSUG 2023 Proceedings, 2023, https://www.lexjansen.com/pharmasug/2023/FDA/PharmaSUG-2023-FDA-G004.pdf

[Hughes 2024] Troy Martin Hughes, "PROC FCMP User-Defined Functions: An Introduction to the SAS Function Compiler", SAS Institute Press, 2024

[Jablonski (1) 2024] Bartosz Jabłoński, "Integration of SAS GRID environment and SF-36 Health Survey scoring API with SAS Packages", PharmaSUG 2024 Proceedings, 2024, https://www.lexjansen.com/pharmasug/2024/SD/PharmaSUG-2024-SD-262.pdf

[Jablonski (2) 2024] Bartosz Jabłoński, "Macro Variable Arrays Made Easy with macroArray SAS package", PharmaSUG 2024 Proceedings, 2024, https://www.lexjansen.com/pharmasug/2024/AP/PharmaSUG-2024-AP-108.pdf

[Jablonski (3) 2024] Bartosz Jabłoński, "Here Comes the Rain (Cloud Plot) Again", WUSS 2024 Proceedings, 2024, https://www.wuss.org/proceedings/2024/164/164_FINAL_paper_pdf.pdf

[Ohuma *et al.* 2024] Eric Ohuma, Linda Vesel, Simon Parker, Bartosz Jablonski, "Guidance for International Growth Standards (GIGS) project", GIGS SAS Package, 2024, https://github.com/SASPAC/gigs/

[Tsutsugo 2024] Kosuke Tsutsugo, "SAS Plotter", SASPlotter SAS Package, 2024, https://github.com/SASPAC/sasplotter

[SAS doc. 1] SAS 9.4 and SAS Viya 3.5 Programming Documentation, "DS2 Programmer's Guide - Introduction to DS2 Packages", September 2024, https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/ds2pg/n0dhudwlvki8d5n1e6itkwdsllrx.htm

[SAS doc. 2] SAS 9.4 and SAS Viya 3.5 Programming Documentation, "Base SAS Procedures Guide - PROC FCMP Statement", September 2024, https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p0urpv7yyzylqsn1g2fycva2bs3n.htm

[SAS doc. 3] SAS 9.4 and SAS Viya 3.5 Programming Documentation, "SAS/IML User's Guide - Packages - Overview of Packages", September 2024, https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/imlug/imlug_packages_sect001.htm

[SAS doc. 4] SAS 9.4 Integration Technologies, "Publishing Framework", September 2024, https://documentation.sas.com/doc/en/itechcdc/9.4/itechov/p0li3lwdzwkxezn172yujdp7wkya.htm

[SAS doc. 5] SAS 9.4 and SAS Viya 3.5 Programming Documentation, "ODS PACKAGE Statement", September 2024, https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/odsug/p19kppnlyy01n7n18jcfsy0eutvq.htm

[SAS doc. 6] SAS Enterprise Guide 8.3: User's Guide, "Working with Projects", September 2024, https://documentation.sas.com/doc/en/egug/8.3/n1v9z6rladl7jan1tsq9clpzrzo2.htm

ACKNOWLEDGMENTS

The author would like to acknowledge Filip Kulon and Krzysztof Socki. A discussion with them in summer of 2019 planted the seed for the idea of SAS Packages.

The author would like to acknowledge Lisa Mendez for help in polishing linguistic part of the paper.

GITHUB

To find SAS Packages Framework (SPF) visit:

https://github.com/yabwon/SAS_PACKAGES

To find SAS Packages Archive (SASPAC) visit:

https://github.com/SASPAC

To find out how to work with SPF and packages visit:

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

CONTACT INFORMATION

Your comments and questions are valued and encouraged! Contact me at the following e-mail addresses:

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

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

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,
- and comments pertaining to code are in a bluish ink for easier reading.
```

Appendix B - install the SAS Packages Framework and packages

To install the SAS Packages Framework and a SAS Package we execute the following steps:

- First we create a directory to install SPF and Packages, for example: `/home/user/packages` or `C:/packages`, we will use the first one in the upcoming code snippets.
- Next, depending if the SAS session has access to the internet:
  - if it does - we run the following code:

code: install from the internet

```
filename packages "/home/user/packages";

filename SPFinit url
 "https://raw.githubusercontent.com/yabwon/SAS_PACKAGES/main/SPF/SPFinit.sas";
%include SPFinit;

%installPackage(SPFinit)
%installPackage(packageNameYouWant)
```

  - If the SAS session does not have access to the internet we go to the framework repository:

    https://github.com/yabwon/SAS_PACKAGES

    Next, if not already done, we click the stargazer button [★] ;-) and then we navigate to the `SPF` directory and copy the `SPFinit.sas` file into the directory from step one (direct link: `https://raw.githubusercontent.com/yabwon/SAS_PACKAGES/main/SPF/SPFinit.sas`). And for packages - we just copy the package zip file into the directory from step one.
- From now on, in all subsequent SAS sessions, it is enough to just run:

code: enable framework and load packages

```
filename packages "/home/user/packages";
%include packages(SPFinit.sas);
%loadPackage(packageNameYouWant)
```

  to enable the framework and load packages.
- If need be in the future, to update the framework or a package to the latest version we simply run:

code: update from the internet

```
%installPackage(SPFinit packageYouWant1 packageYouWant2 packageYouWant3)
```

See [**Jablonski 2020**], [**Jablonski 2021**], and especially [**Jablonski 2023**] for details.

Appendix C - safety considerations

The SPF installation process, in a "nutshell", reduces to copying the `SPFinit.sas` file into the packages directory. It is the same for packages, too.

You may ask: *is it safe to install*?

Yes, it's safe! When you install the SAS Packages Framework, and later when you install packages, the files are simply copied into the packages directory that you configured above. There are no changes made to your SAS configuration files, or autoexec, or registry, or anything else that could somehow "break SAS." As you read, you can perform a manual installation simply by copying the files yourself. Furthermore, the SAS Packages Framework is:

- written in 100% SAS code, it does not require any additional external software to work,
- full open source (and MIT licensed), so every macro can be inspected.

When we work with a package, before we even start thinking about loading content of one into the SAS session, both the help information and the source code preview are available.

To *read help information* (printed in the log) you simply run:

code: get help info

```
%helpPackage(<packageName>, <*|componentName|license>)
```

To **preview source code of package components** (also printed in the log) you simply run:

code: get code preview

```
%previewPackage(<packageName>, <*|componentName>)
```

The asterisk means "print everything", the `componentName` is the name of a macro, a function, a format, etc. that you want see.