Paper SD-262-2024

Integration of SAS® GRID environment and SF-36 Health Survey scoring API with SAS Packages

Bartosz Jabłoński - yabwon / Warsaw University of Technology

ABSTRACT

The SF-36v2 Health Survey is a 36 questions survey which allows to measure functional health and well-being from the patient's point of view. As a health survey, SF-36v2 can be used across age, disease and treatment groups, in contrast to disease-specific health surveys, which focuses on a particular disease or condition (see [**QM SF36**]).

Raw SF-36v2 survey data are subject to scoring. Such scoring is provided by tools delivered, for instance, by QualityMetric Inc. The QM API provides all the functionality needed to score and save survey data in the PRO Insight Smart Measurement System. Once a survey is scored, the API returns scores and interpretations based upon processed data.

In Takeda Pharmaceuticals a SAS® Package named `QMsf36scoring` was developed to facilitate scoring of SF-36v2 data, stored in SAS data sets, by the QM API. The `QMsf36scoring` package is a generic solution which can be executed on any SAS environment which is able connect to the QM API tool. Thanks to a SAS Package approach interaction with the API is easy and straightforward, which allows to seamlessly embed the scoring process into the SDTM or ADaM development process.

The article presents details of the design and the implementation of the package.

PARTIES and ROLES

> *We exist to create better health for people and a brighter future for the world. While the science and technology we advance are constantly evolving, our ambition remains. We move science forward, so we can transform more lives.*[1]
>
> Takeda Pharmaceuticals
>
> *Put simply, we measure health and well-being from the patient's point of view.*[2]
>
> QualityMetric Inc.
>
> *"If it's somethin' weird; And codes don't look good; Who ya gonna call?..."*
>
> Bart

Three parties engaged in the project were: Takeda Pharmaceutical Company Ltd., QualityMetric Inc., and humble author of this article. Roles in the play were the following:

- Takeda Pharmaceutical Company Ltd. (Takeda), one of the oldest pharmaceutical companies in the world, established in 1781, required they patients data of the SF-36v2 Health Survey to be scored. Takeda provided working environment for statistical programmers on a SAS GRID server, where

[1] `https://www.takeda.com/`

[2] `https://www.linkedin.com/company/qualitymetric-incorporated/`

SDTM data were processed. The "appetite" was to execute scoring of SF-36v2 Health Survey data for patients in several clinical studies.

- The QualityMetric Inc. (QM), "an IQVIA business" - as their webpage says, provides REST API interface which allows to score data for several "quality of life" health surveys. Among offered scoring engines, one for scoring the SF-36v2 Health Survey data was available.
- Takeda and QM agreed the SF-36 scoring engine will be used to score Takeda's data.
- The humble author here was asked to write a program which "takes Takeda's survey data, pushes them into QM's scoring engine's API, retrieves results, and stores the results back on Takeda's GRID" and allows to execute all that process in the most comfortable way for statistical programmers working in Takeda.

REQUIREMENTS

The list of requirements for the project was rather short and contained the following items:

- Multiple surveys can be scored in single run.
- Process has to be easily integrated into process of SDTM development.
- Log from the scoring should be available.

SAS PACKAGES

In the world of programmers, software developers, or data analysts, the concept of a package, as a practical and natural medium to share your code with other users, is well known and common. To give evidence of this statement, let us consider a few very popular examples: Python, TEX, and R software. As an endorsement of this fact, it is enough to visit the following web pages:

```
https://pypi.org/
https://www.ctan.org/
https://cran.r-project.org/
```

to see the number of available packages. There are, literally, thousands of packages available for those languages! More than a dozen of TEX packages were used while writing this article.

With that said, the fundamental question of a new or a seasoned SAS user should be: "Why there is no such thing like packages in SAS?" To be clear, there are "packages" in the SAS ecosystem (see the footnote in the package definition) but they do not offer such functionality as those mentioned above. For example, the SAS/IML offers (limited) functionality similar to the concept of a package in the R language mentioned above (for comparison see [**Wickham 2015**]) but such functionality is not available in Base SAS, or even Viya.

The **SAS Packages Framework**, introduced in [**Jablonski 2020**], tries to fill that gap.

A **SAS package**[3] 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's complex dependencies created by the developer from the user experience with the final product and, on the other hand, to reduce developers' and users' unnecessary frustration related to a remote deployment process.

[3] The idea presented in this article should not be confused with other occurrences of "package" concept which could be found in the SAS ecosystem, e.g. Proc DS2 packages, SAS/IML packages, SAS ODS packages, SAS Integration Technologies Publishing Framework packages, or even SAS Enterprise Guide `*.egp` projects files.

The SAS Packages Framework is a "pack" of macros, which allows SAS users to *use* and to *develop* SAS packages.

To create a package, the developer executes a few simple steps which, in general, are:

- prepare the code (package content files) and a description file,
- "fit" them into a structured form,
- download the `SPFinit.sas` (the SAS Packages Framework) file,
- and execute the `%generatePackage()` macro.

To use a package, the user executes even fewer steps which, in general, are:

- download the `SPFinit.sas` (the SAS Packages Framework) file,
- execute the `%installPackage(packageName)` and the `%loadPackage(packageName)` macros.

The framework is open-source MIT licensed project available at GitHub:

`https://github.com/yabwon/SAS_PACKAGES`

In the [**Jablonski 2021**] article, a step-by-step tutorial, which allows to develop SAS packages, is presented. And the [**Jablonski 2023**] is a full "zero to hero" workshop on the subject. See also Appendix B - install the SAS Packages Framework and packages from SASPAC archive and Appendix C - safety considerations.

Note. Though using or building a package is a very straightforward process, the creation of a package content (i.e. macros, functions, formats, etc.) requires some experience. That is why the intended readers for the articles just mentioned are *at least* intermediate SAS programmers. A good knowledge of Base SAS and practice in macro programming will be an advantage. But "fear not" if you are an inexperienced programmer. Such background knowledge can be found, for example, in [**Carpenter 2012**].

THE SOLUTION

> *Equipped with the "*QualityMetric API Implementation Guide*" document and with a set of test scoring data, I started programming...*

The KISS (keep it simple, stupid) paradigm was adopted during the development and implementation process to make the interaction with the solution seamless and to minimize entry level for users.

After reading the Implementation Guide and after analyzing test data format it was decided that, from the statistical programmer point of view, the most convenient approach would be:

- to prepare an input data set with survey responses in proper format, and
- to call a macro which would execute the process and return results.

From the statistical programmer point of view.

The first step of the scoring process is to prepare an input data set. The input data set was designed to be possibly the minimal (but not empty). The list of required names and formats for variables in the input data set are the following:

- `STUDYID` of character type, containing Study Identifier,
- `USUBJID` of character type, containing Unique Subject Identifier,
- `QSSEQ` of numeric type, containing consecutive numbers identifying rows,
- `QSTESTCD` of character type, containing question short name in form `SFV201`, `SFV202`, ..., `SFV236`, where `01` suffix corresponds to the first question, `02` to the second, etc. up to `36` for thirty-sixth.
- `QSSTRESN` of numeric type, containing "numeric finding in standard units", i.e. values of response for survey's questions.

- `VISIT` of character type, containing visit name, e.g., `BASELINE`, `WEEK 1`, `WEEK 2`, etc.
- `QSDTC` of character type, containing Date/Time of Finding in ISO8601 format.

The logical key for data in the data set is composed of the following variables: `STUDYID`, `USUBJID`, `VISIT`, and `QSTESTCD`. This allows to provide data for multiple subjects and multiple visits in one file (for one scoring process). If the data set passed to the scoring macro contains other variables, from the listed above, "redundant" variables are ignored. An example of a survey-data-to-be-scored SAS data set can be found in Table 1.

Table 1: Example of data set with data for scoring.

STUDYID	USUBJID	QSSEQ	QSTESTCD	QSSTRESN	VISIT	QSDTC
ABC	SU-001	1	SFV201	3	BASELINE	2024-05-19T12:34:56
ABC	SU-001	2	SFV202	4	BASELINE	2024-05-19T12:34:56
		...				
ABC	SU-001	35	SFV235	5	BASELINE	2024-05-19T12:34:56
ABC	SU-001	36	SFV236	2	BASELINE	2024-05-19T12:34:56
ABC	SU-002	37	SFV201	5	VISIT 01	2024-05-20T12:34:56
ABC	SU-002	38	SFV202	3	VISIT 01	2024-05-20T12:34:56
		...				
ABC	SU-002	72	SFV236	5	VISIT 01	2024-05-20T12:34:56
...	...	...				
ABC	SU-042	1477	SFV201	3	VISIT 17	2024-05-21T12:34:56
ABC	SU-042	1478	SFV202	1	VISIT 17	2024-05-21T12:34:56
		...				
ABC	SU-042	1512	SFV236	1	VISIT 17	2024-05-21T12:34:56

See the Appendix D - test data to learn how to generate test data yourself.

With the input data set prepared, the next step is to execute scoring macro, the `%QualityMetricsRESTAPIsf36()` macro. Call to the macro has the following form (check Appendix A - code coloring guide for color convention):

code: call the QualityMetricsRESTAPIsf36 macro

```
%QualityMetricsRESTAPIsf36(
  /* inDs - input data set - positional parameter */
   <Library.Data-set-with-survey-data>

  /* connection parameters */
  ,globalKey=********-****-****-****-************
  ,groupID=******
  ,groupLogin=***-*****-******
  ,serverAddress=api-staging.qualitymetric.com
  ,siteLogin=***-***-****
  ,surveyID=52627 /* 52627 is ID for "sf36 acute" survey */

  /* output */
  ,scoredLib=<Library-to-store-results>
  ,scoredDS=<Results-data-set-name>
)
```

Parameters definitions for the macro are the following:

- `inDs`, the first and only positional parameter, *required*, indicate a list of data sets in the format described above containing data to be scored by QM engine.
- Key-value pair parameters for connection setup provided by QualityMetric:
  - `globalKey`, the Security Key assigned to the client by QualityMetric.
  - `groupID`, the Group Identifier assigned to the client by QualityMetric.
  - `groupLogin`, the Group Login assigned to the client by QualityMetric.
  - `serverAddress`, server address provided by QualityMetric.
  - `siteLogin`, the Site Login assigned to the client by QualityMetric.
  - `surveyID`, an identifier for the survey's version, for the `SF36v2` "acute" version used in the project it was `52627`.
- Key-value pair parameters for results:
  - `scoredLib` is a libnme where data set with scores is stored, default is `WORK`.
  - `scoredDS` is a data set name with scores, default is `SurveyRecords`.

The components required to execute the `%QualityMetricsRESTAPIsf36()` macro are provided as a SAS Package. Explanation what are SAS Packages and how to use them with SAS Packages Framework is given in the SAS PACKAGES section.

For the sake of this section, it is enough to know that, assuming that the package and the framework are located in the `/my/directory/for/packages` directory, all we need to run before calling the `%QualityMetricsRESTAPIsf36()` macro is:

code: enable the QMsf36scoring package

```
1 filename packages "/my/directory/for/packages";
2 %include packages(SPFinit.sas);
3 %loadPackage(QMsf36scoring)
```

Line one "points" the location, line two enables the SAS Packages Framework, and the third loads the package into the session.

The following interactions with the SAS Session take place during the `%QualityMetricsRESTAPIsf36()` macro execution:

- Filenames: `oAuth_B`, `inSurv`, and `outSurv` are created.
- Libnames: `J` (for reading content of `JSON` files received from QM API), `SrvScr`, and `_` are created.
- Data sets: `work.qss2_sorted0` to `work.qss2_sorted4`, and `work.outputFiles` are created.
- Data sets: `SrvScr.SurveyRecord1`, `SrvScr.SurveyRecord2`, ... `SrvScr.SurveyRecordN` are created, `N` is the number of surveys to be scored.
- Data set: `&scoredLib..&scoredDS.` is created.

[NOTE:] Unfortunately, since the `QMsf36scoring` package is Takeda's property its source cannot be presented. It feels like reading Lego instruction not having bricks to play... life... But to cheer you up, the code is very very generic (almost like `data _null_; run;`), so after reading this article you should not have any problem with writing one yourself. Of course you need to have SAS environment and QualitiMetric scoring API credentials.

Under the hood.

Though the user perspective was designed to be seamless, it does not mean the `QMsf36scoring` package is trivial "under the hood". Thanks to the SAS Package "envelope" all the complexity happening in the background is elegantly separated from the user experience, like a powerful V8 engine roaring covered under beautiful, red painted, hood of the Chevrolet Corvette.

The %QualityMetricsRESTAPIsf36() macro is the "main" macro of the QMsf36scoring package which facilitates primary user interface for scoring process execution. Except that macro, a few more objects are generated in the SAS session when the package is loaded. The remaining elements of the package are a bunch of utility formats:

- _1_3valuecheck,
- _1_5valuecheck,
- _1_6valuecheck,
- sf36decode,

and two macros:

- %getToken(),
- %getSurveyScore().

First three formats: _1_3valuecheck, _1_5valuecheck, and _1_6valuecheck, are there to assure that survey questions responses are kept in predefined values ranges (integers: between 1 and 3, between 1 and 5, and between 1 and 6, accordingly), in case question value is improper it is set to -1. Such situations are also reported in the process log.

The last format is to decode survey's question short names from SFV201, SFV202, ..., SFV236 to dedicated values recognized by QM scoring engine, which are: GH01, HT, PF01, PF02, PF03, PF04, PF05, PF06, PF07, PF08, PF09, PF10, RP01, RP02, RP03, RP04, RE01, RE02, RE03, SF01, BP01, BP02, VT01, MH01, MH02, MH03, VT02, MH04, VT03, MH05, VT04, SF02, GH02, GH03, GH04, GH05, accordingly, and the ERROR for all others.

The %getToken(globalKey=,groupID=,groupLogin=,serverAddress=) macro, as the name suggests, is dedicated to set up a connection token which is used to authenticate data transfer process between SAS server and QM API. QualityMetric provides values for globalKey, groupID, groupLogin, and serverAddress. Credentials provided in the call to the %QualityMetricsRESTAPIsf36() macro are used by Proc HTTP to access QualityMetric API, authenticate, and get the token. Token's value is kept in a macrovariable.

The %getSurveyScore(N,serverAddress=,inputFilePath=,lib=) macro takes a JSON file with a survey data (&inputFilePath.), sends it to the QM scoring engine, retrieves results in JSON format, and (using JSON libname engine) extracts result data into a SAS data set (with number &N.) in a library pointed by the lib= parameter. Data exchange is done by the Proc HTTP which uses the token to establish communication with the QualityMetric API.

Assuming that the user (statistical programmer) prepared the data set and provided valid credentials the process under the %QualityMetricsRESTAPIsf36() macro goes the following way:

(1) Data to be scored are sub-selected with a "QSTESTCD like 'SFV2%'" where condition, and only required variables are kept. If this step fails the %QualityMetricsRESTAPIsf36() macro ends execution.
(2) Temporary libraries (SrvSrc and _) are created, all are created as sub-directories of the WORK library location with help of the dlCreateDir option.
(3) Number of survey questions is verified for each survey, and mismatches, if exist, are printed out.
(4) Survey answers are formatted with help of _1_*valuecheck formats to align values with API requirements.
(5) A data step which reads in the input data and generates JSON files is executed. A list of JSON files is created in the SurveyDataFiles sub-directory of the WORK library. Each survey gets its own dedicated JSON file in a format required by the API (the "*QualityMetric API Implementation Guide*" document provides details). In the JSON file, according to the QM terminology, variable USUBJID is

translated to `loginName`, `VISIT` to `Timepoint`, and `QSDTC` to `AdministrationDate`. See Appendix E - data in JSON format

(6) The `%getToken()` macro is called to obtain authentication token. The `Proc HTTP` does the job by storing token value in an `oAuth_B` temporary file, and then, by reading the `oAuth_B` infile, with help of `call symputx()` macrovariable `token` is created. If the `%getToken()` fails (e.g., `Proc HTTP`'s status code is different than 200) the `%QualityMetricsRESTAPIsf36()` macro is terminated.

(7) Internal loop over the list of `JSON` files is run. In each iteration, with help of the `call execute()` subroutine, the `%getSurveyScore()` macro is called. A single call to the macro does the following:
   (a) The `Proc HTTP` calls to QM REST API and sends survey data for scoring. If scoring fails (status code is different than 200) a warning is issued in the log, data are not collected for a given survey, and macro terminates. The token is used for authentication (`OAUTH_BEARER="\&token."`). Additionally the `DEBUG` option is set to `level=1` and the `CLEAR_CACHE` is set.
   (b) When scoring is successful a temporary file with results in `JSON` format is created.
   (c) The `JSON libname` engine creates a temporary library, named `J`, which is pointing to results in just created `JSON` file.
   (d) Information about subject (`work.member` data set), survey fields (`work.surveyfields` data set), and scores (`work.scores` data set) are extracted from `J.ALLDATA` data set, transposed from long to wide format, and combined into one data set named `SrvScr.SurveyRecord*` (asterisk indicates sequence number of an iteration, taken from the `N` parameter of the macro).

(8) Since results from each `JSON` can can produce a data set with different variables lengths, the lengths are unified. The `Proc CONTENTS` on the `SrvScr._ALL_` collects metadata information, and `Proc SQL`'s `into` clause aggregates them in a macrovariable of the form: "`variable1 <$> length1 ... variableK <$> lengthK`" (the "`$`" for character variables only) to be used in the `length` statement.

(9) The "final" data set, combining results from `SrvScr.SurveyRecord*` data sets with unified variables lengths, is generated. The data set contains the following variables:
   - six character variables: `memberID`, `USUBJID` (renamed back from QM's `loginName`), `siteLogin`, `sessionID`, `QSDTC` (renamed back from QM's `AdministrationDate`), `VISIT` (renamed back from QM's `Timepoint`) which allows merging with input data, and
   - remaining numeric ones: `PFScore`, `RPScore`, `BPScore`, `GHScore`, `VTScore`, `SFScore`, `REScore`, `MHScore`, `PCSScore`, `MCSScore`, `PF_0_100`, `RP_0_100`, `BP_0_100`, `GH_0_100`, `VT_0_100`, `SF_0_100`, `RE_0_100`, `MH_0_100`, `SF6D_R2`, and `SF6DV2_UK`, with scores.

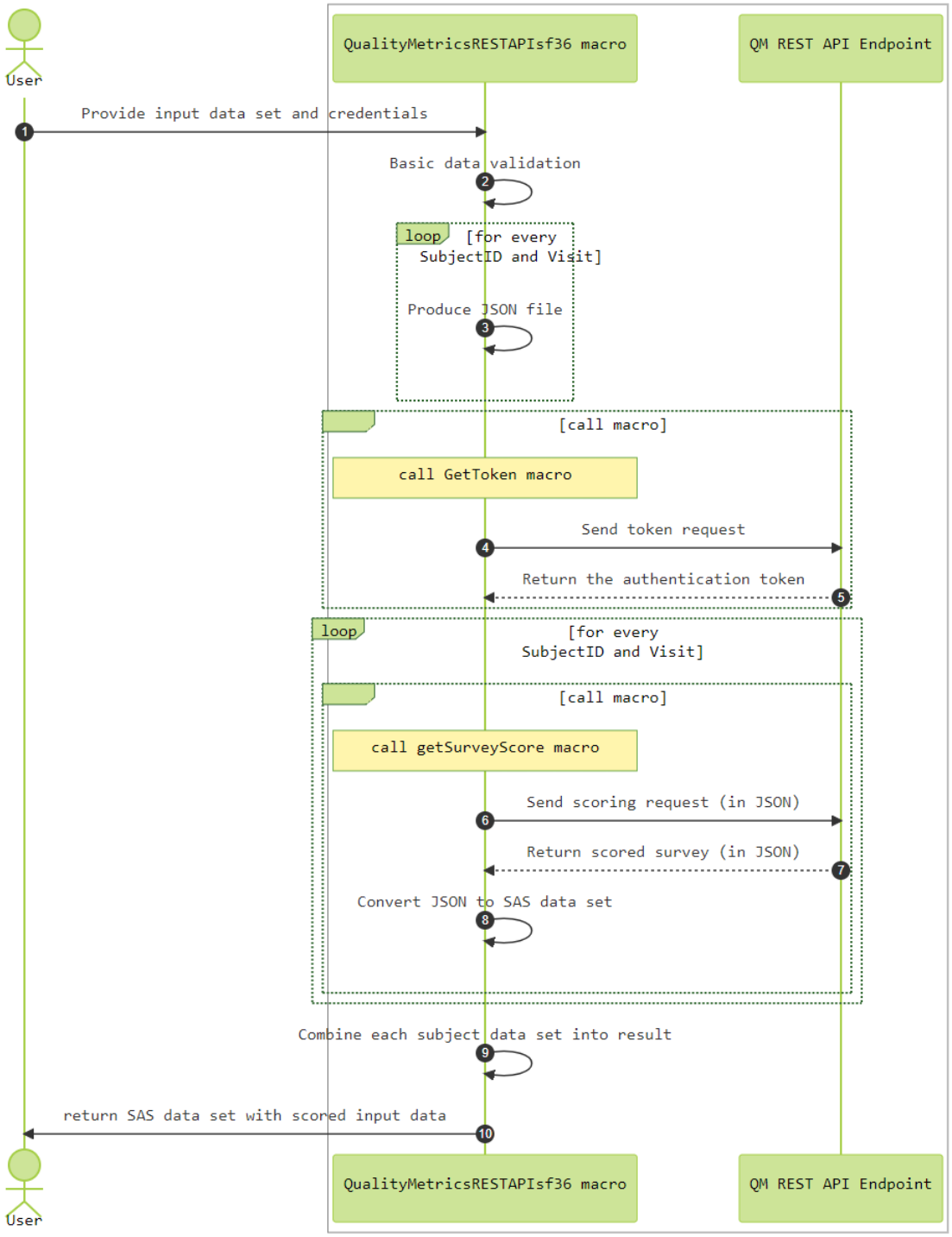
The process illustration can be found in Figure 1 (numbers on the diagram do *not* correspond with the above list numbers).

CONCLUSION

In the article, we learned how to solve the problem of creating connectivity between SAS GRID and external API for scoring the SF-36v2 Health Survey data. A SAS Package approach was used and the `QMsf36scoring` package, which allows to exchange and score data between Takeda Pharmaceuticals SAS GRID environment and the QualityMetric Inc. scoring engine, was described from both user and developer point of view. Finally, we observed that use of SAS Packages is practical and prospective way for designing and delivering project's code.

The End

Figure 1: Diagram of the scoring process.



REFERENCES

[Carpenter 2012] Arthur L. Carpenter, "Carpenter's Guide to Innovative SAS Techniques", SAS Institute Press, 2012

[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

[Carpenter 2016] Arthur L. Carpenter, "Carpenter's Complete Guide to the SAS Macro Language, Third Edition", SAS Institute Press, 2016

[Jablonski 2020] Bartosz Jabłoński, "SAS Packages: The Way to Share (a How To)", SGF Proceedings, 2020, 4725-2020 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 , 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.lexjansen.com/wuss/2023/WUSS-2023-Paper-208.pdf

[QM SF36] QualityMetric, The SF-36v2 Health Survey webpage, as of 2024-03-24, https://www.qualitymetric.com/health-surveys/the-sf-36v2-health-survey/

ACKNOWLEDGMENTS

The author would like to acknowledge Takeda's project team, especially Matt Wien (direct manager), who gave the author opportunity to work with this great project! Also, the support for "non programming" tasks from the project team cannot be underestimated. Thank you!
The author would like to acknowledge Filip Kulon, whose editorial contribution made this paper's linguistic side look and feel as it should!

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.

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 red ink so that it can be highlighted,
- 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 WARNINGSs are violet.
Log ERRORSs are red.
Log text generated by the user is purple.
```

Appendix B - install the SAS Packages Framework and packages from SASPAC archive

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`.
- 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)
/* if package is available in SASPAC archive */
%installPackage(packageName)
```

  - If the SAS session does not have access to the internet (or a package is not available in SASPAC archive) we go to the framework repository:

    https://github.com/yabwon/SAS_PACKAGES

    next (if not already) we click the stargazer button [★] ;-) and then we navigate to the `SPF` directory and we 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 created in the step one.
- From now on, in all subsequent SAS session, it is enough to just run:

code: enable framework and load packages

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

to enable the framework and load packages. To update the framework or a package to the latest version we simply run:

code: update from the internet

```
%installPackage(SPFinit packageName)
```

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.

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 saw, 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, or a function, or a format, etc. you want see.

Appendix D - test data

The following snippet generates test data for you:

code: test data

```
data work.QMsf36v2_test_data(
  label="Test data for SF36v2 Health Survey for the QMsf36scoring package"
 );
  length
    STUDYID $ 64 USUBJID $ 64 QSSEQ 8 QSTESTCD $ 6 QSSTRESN 8 VISIT $ 32 QSDTC $ 19;
  keep
    STUDYID USUBJID QSSEQ QSTESTCD QSSTRESN VISIT QSDTC;
  label
    STUDYID  = "STUDYID - variable of character type, containing Study Identifier"
    USUBJID  = "USUBJID - variable of character type, containing Unique Subject
    Identifier"
    QSSEQ    = "QSSEQ - variable of numeric type, containing sequential numbers"
    QSTESTCD = "QSTESTCD - variable of character type, containing question short
    name in form SFV201, SFV202, ..., SFV236, where 01 suffix corresponds to the
    first question, 02 to the second, etc. up to 36 for thirty-sixth"
    QSSTRESN = "QSSTRESN - variable of numeric type, containing ""numeric finding
    in standard units"", i.e. values of response for survey's questions"
    VISIT    = "VISIT - variable of character type, containing visit name,
    e.g., BASELINE, WEEK 1, WEEK 2, etc."
    QSDTC    = "QSDTC - variable of character type, containing Date/Time
    of Finding in ISO8601 format";

  call streaminit(007);
  STUDYID = "MI6-STUDY-007";

  array v[*] BASELINE WEEK_1 - WEEK_17;
  array q[*] SFV201 - SFV236;

  do i = 1 to 42;
    USUBJID = "SUBJECT-" !! put(i, z3.);
    VISIT = translate(vname(v[rand("integer", lbound(v), hbound(v))]), " ", "_");
    QSDTC = put(86400*(mdy(1, 1, 2017) + 100*i) + 34567 + 3456*i, E8601dt.);

    do j = lbound(q) to hbound(q);
      QSTESTCD = vname(q[j]);

      select;
        when (3<= j <=12) QSSTRESN = rand("integer", 1, 3);
        when (    j  =21) QSSTRESN = rand("integer", 1, 6);
        otherwise         QSSTRESN = rand("integer", 1, 5);
      end;
      QSSEQ + 1;
      output;
    end;
  end;
run;
```

Appendix E - data in JSON format

The survey data in JSON format looks like this:

JSON: example data of subject SU-001


```
{"surveyID":52627,
 "siteLogin":"XYZ123",
 "member":{"memberID":0,
           "loginName":"SU-001",
           "siteLogin":"XYZ123",
           "memberFields":[ ]},
 "surveyFields":[
  {"name":"AdministrationDate", "value":"05/19/2024"},
  {"name":"Timepoint", "value":"BASELINE"}],
 "surveyResponses":[
  {"name":"GH01", "value":"3"},
  {"name":"HT"  , "value":"4"},
  {"name":"PF01", "value":"1"},
  {"name":"PF02", "value":"1"},
  {"name":"PF03", "value":"3"},
  {"name":"PF04", "value":"3"},
  {"name":"PF05", "value":"2"},
  {"name":"PF06", "value":"1"},
  {"name":"PF07", "value":"2"},
  {"name":"PF08", "value":"1"},
  {"name":"PF09", "value":"3"},
  {"name":"PF10", "value":"3"},
  {"name":"RP01", "value":"1"},
  {"name":"RP02", "value":"5"},
  {"name":"RP03", "value":"4"},
  {"name":"RP04", "value":"5"},
  {"name":"RE01", "value":"5"},
  {"name":"RE02", "value":"2"},
  {"name":"RE03", "value":"5"},
  {"name":"SF01", "value":"3"},
  {"name":"BP01", "value":"6"},
  {"name":"BP02", "value":"2"},
  {"name":"VT01", "value":"5"},
  {"name":"MH01", "value":"4"},
  {"name":"MH02", "value":"3"},
  {"name":"MH03", "value":"5"},
  {"name":"VT02", "value":"3"},
  {"name":"MH04", "value":"3"},
  {"name":"VT03", "value":"4"},
  {"name":"MH05", "value":"1"},
  {"name":"VT04", "value":"4"},
  {"name":"SF02", "value":"1"},
  {"name":"GH02", "value":"1"},
  {"name":"GH03", "value":"2"},
  {"name":"GH04", "value":"5"},
  {"name":"GH05", "value":"2"}]}
```