Paper CT04

SAS Packages for Clinical Programming

Eldho Alias, Genpro, Now part of Catalyst Clinical Research, Trivandrum, India
Prasanth Prabhakaran, Genpro, Now part of Catalyst Clinical Research, Trivandrum, India
Rahulraj V R, Genpro, Now part of Catalyst Clinical Research, Trivandrum, India

ABSTRACT

A collection of SAS scripts can be reused by using a single, automatically generated stand-alone file, and this is what a SAS package is. This package can be used to load the SAS scripts into the environment. Without having to open the program / macro, or its documentation, we can also quickly get the documentation in the log window for any program / macro inside the package. Clinical research can benefit from SAS Packages. Methods for creating SAS packages are the primary topic of this paper. Here, we will go overloading the package, using it with the help file and creating helpful auxiliary programs for it. Additionally, we demonstrate how the programs included in the package can be used by anyone who has the package.

INTRODUCTION

In software development, a "package" refers to a collection or bundle of related code, files, and resources organized and distributed together. Packages are a fundamental concept in many programming languages and ecosystems and serve various purposes, including code organization, distribution, and reuse. The specific characteristics and usage of packages can vary between programming languages and development environments. Here's a general overview:

- Code Organization: Packages help developers organize their code into manageable units. Code within a package is typically related in some way, such as by functionality or purpose. This modular approach makes it easier to understand, maintain, and collaborate on codebases, especially in large software projects.
- Namespace Management: Packages often define namespaces, which provides a way to avoid naming conflicts. By placing code into packages, you can use names that make sense within the context of the package without worrying about clashes with names in other parts of your program.
- Code Reusability: Packages enable code reuse. Developers can create libraries or modules that can be used in multiple projects by simply importing or including the package. This promotes the "Don't Repeat Yourself" (DRY) principle.
- Dependency Management: In many programming languages, packages are used to manage dependencies. When a project relies on external libraries or modules, package managers can automatically download and install those dependencies.
- Distribution: Packages are often a convenient way to distribute software components.
- Versioning: Packages often have version numbers associated with them to indicate changes or updates. This allows developers to specify which version of a package their project depends on, ensuring compatibility.

SAS Package

A SAS package is an automatically generated, single, standalone 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 will allow on the one hand, to separate the complex code 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 a remote deployment process.

SAS PACKAGE FRAMEWORKS

The SAS Packages Framework is a set of macros which allow us to use and to develop SAS packages.

generatepackage	-	Macro to generate SAS packages

helppackage	-	Macro to get help about SAS packages
installpackage	-	Macro to installing a package
listpackages	-	Macro to list available SAS packages
loadpackage	-	Macro to load a package
loadpackages	-	Macro to load multiple packages
previewpackage	-	Macro to preview content of a SAS package
unloadpackage	-	Macro to unload a SAS package

Copy the usepackage.sas file into the C:/PCKG directory or directory you want to install the packages. The "usepackage.sas" program, which holds the above framework components.

ENABLE THE FRAMEWORK

To activate the SAS Packages Framework, we execute the following code:

```
filename packages "…..\Global Packages";
%include packages(usepackage.sas);
```

The first line associates the "packages" fileref with the specified directory, in this case, "…..\Global Packages" where the package is found. The second line incorporates the contents of the "usepackage.sas" file, which holds the framework components, into the current SAS session. It's crucial to maintain the association with the "packages" fileref throughout the entire SAS session since the framework's macros depend on it.

CREATING A PACKAGE

- Create a directory for your package and give it the identical name as the package itself.
- Create a description file inside the package folder, named description.sas. The file is mandatory, has simple structure, and it holds package metadata and a brief description. The simple structure of the description.sas file can be seen in Figure 1.

```
Type: Package
Package: gensdtm
Title: SAS PAckage for handling SDTM dataset creation.
Version: 1.0
Author: Genpro (Genpro@catalystcr.com)
Maintainer: Genpro
License: MIT
Encoding: UTF8

Required: "Base SAS Software"

DESCRIPTION START:
## The myPackage ##

The `gensdtm` is a SAS package and
it can be used for sdtm dataset creation with ease!
It is created during the 2020

It helps me to share my code with other SAS users!
DESCRIPTION END:
```

Figure 1

- Within the package directory, create subdirectories for the code files. These subdirectory names should follow these rules:

  a) They should consist only of lowercase letters, digits, and underscores (e.g., "my_folder_01").

  b) Each subdirectory name should have two parts separated by an underscore (e.g., "001_code" where "001" represents a sequence of digits with leading zeros, and "code" indicates the subdirectory's content type).

  c) For an example of how to structure subfolders for a package, refer to Figure 2. If the order of code execution is not important, you can omit the first part (digits and underscore).

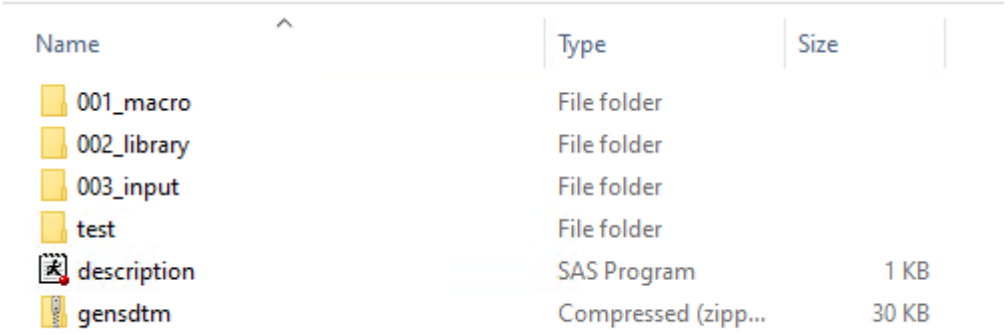


Figure 2

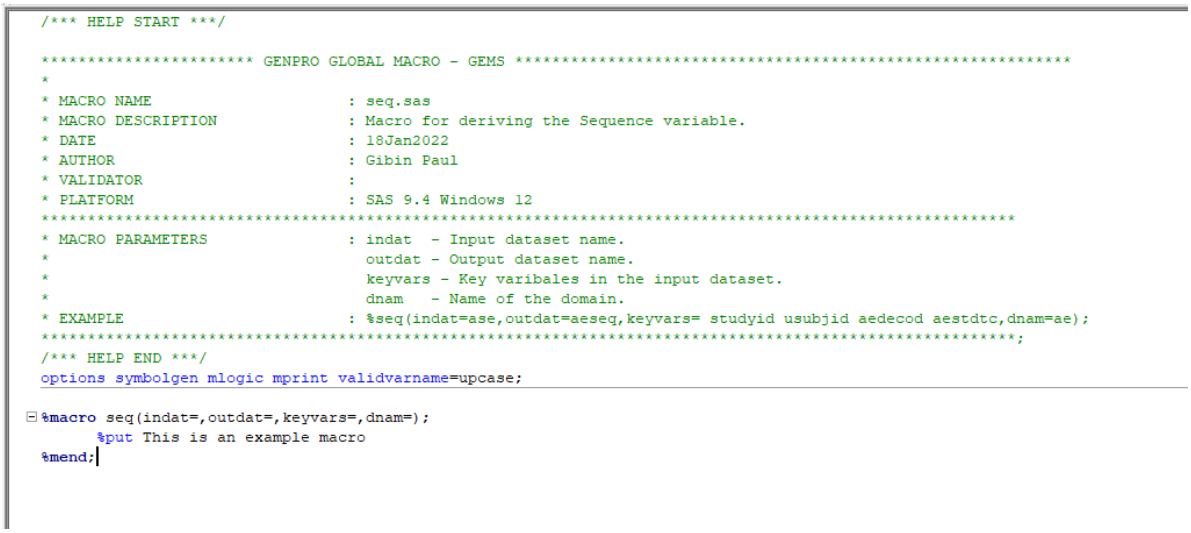
- Organize your code files by copying them into the appropriate subdirectories within the package, following these guidelines:

  a) Adhere to a one-file-one-object approach. For instance, a macro definition like %abc() should be placed in a single file without any other object definitions. The only exception to this rule is for formats/informats; in this case, a single file should contain all definitions of formats/informats that share the same name. For example, numeric format "abc.", character format "$abc.", numeric informat "abc.", and character informat "$abc." should all be included in one file.

  b) Assign file names based on the object name. For instance, a macro named %abc() should be stored in a file named "abc.sas." Ensure that all files use the ".sas" extension, as any other file types will be disregarded.

- To generate help information from sections of code files, you should enclose the relevant portions within specific text markers: "/*** HELP START ***/" to indicate the start and "/*** HELP END ***/" to denote the end. These markers are not obligatory, but if they are absent in a file, a warning will be logged during the package generation process.

  If help tags are included in a file, they must be arranged in the correct order (start followed by end) and should not be nested or overlap. In case of such a situation occurring, an error will be logged during the package generation process, and the process will be halted. For instance, consider a file that contains macro code, help text, and other comments as an example.



```
/*** HELP START ***/

********************* GENPRO GLOBAL MACRO - GEMS ***************************************************************
*
* MACRO NAME                    : seq.sas
* MACRO DESCRIPTION             : Macro for deriving the Sequence variable.
* DATE                          : 18Jan2022
* AUTHOR                        : Gibin Paul
* VALIDATOR                     :
* PLATFORM                      : SAS 9.4 Windows 12
*********************************************************************************************************
* MACRO PARAMETERS              : indat  - Input dataset name.
*                                 outdat - Output dataset name.
*                                 keyvars - Key varibales in the input dataset.
*                                 dnam   - Name of the domain.
* EXAMPLE                       : %seq(indat=ase,outdat=aeseq,keyvars= studyid usubjid aedecod aestdtc,dnam=ae);
*********************************************************************************************************;
/*** HELP END ***/
options symbolgen mlogic mprint validvarname=upcase;

%macro seq(indat=,outdat=,keyvars=,dnam=);
      %put This is an example macro
%mend;
```

Figure 3

- Generate a file named 'license.sas' that holds licensing information for the package. Put this file in the package folder along with 'description.sas' and the subfolders. If you don't supply a 'license.sas' file, a default MIT license will be automatically generated.

- Create a folder for packages,
  e.g., under Windows OS family C:/SAS_PACKAGES or
  under Linux/UNIX OS family /home//SAS_PACKAGES or any folder location you want to install the packages and copy the usepackage.sas file into this folder.

```
%generatePackage(filesLocation=…… \genadam\);
```

When the %generatePackage macro ends its execution the packagename.zip file, containing all package content inside it, is created inside the package folder.

INSTALLING THE PACKAGE

To install the package, the user can either manually download the package's zip file into the packages folder (refer to the "THE CODE" subsection for specific instructions), or they can run the following code (assuming that the framework is already enabled):

```
%installPackage(packageName);
%installPackage(packagesNames= genadam,sourcePath =..\genadam\);
```

LOAD PACKAGE

To load the package into current SAS session the user runs:

```
%loadPackage(packageName)
```

To load multiple packages:

```
%loadPackages(packageName1, packageName2)
```

To remove the contents of the package from the current SAS session, the user executes:

```
%unloadPackage(packageName)
```

HELP PACKAGE

To obtain information about the package and have it displayed in the log:

- for general information about the package the user runs:

```
%helpPackage(packageName)
```

- for all available information about the package the user runs:

```
%helpPackage(packageName,*)
```

- for a particular element of the package, e.g., a function or a macro, help the user runs:

```
%helpPackage(packageName, helpKeyword)
```

where helpKeyword is a single word which is used for context search." License" prints out license text.

PREVIEWING THE CODE OF A PACKAGE

If we want to preview the package content, e.g., code of a macro, an IML module, a format/informat, or a function then the:

```
%previewPackage(packageName, searchedElement)
```

will print out the code of searched element (including all comments). Behavior and searching rules are the same as for the %helpPackage() macro, except for the License keyword which is ignored.

Sample Package generation

The "Genadam" Package

This is a Package "Genadam", an example on how to build a package with help of the SAS Packages Framework.

We will generate a Genadam package. A package used to create ADAM datasets.

Follow the step-by-step instructions below.

1) Create a folder for the framework e.g., …..\Global Packages
2) Run the following code to install The SAS Packages Framework in the folder we created in Step 1.
   filename packages "…..\Global Packages";
   %include packages(usepackage.sas);
3) Check out the log to verify if the Step 2. was successful. If it was - then continue.
4) Create a folder for the content of the package we will generate e.g., …… \genadam\
5) Inside the directory from Step 4. We create three sub-folders: 01_macro ,02_library, and 03_input. Remember to use lower case letters only!
6) Inside the 01_macro folder we develop required macros or copy the existing ones. Remember to use lower case letters in the file name.
7) Inside the directory from Step 4 create a description.sas file. Remember to use lower case letters in the file name.
8) Execute the following code using the folder from Step 4. and check the log to see how the process of package generation went.
   %generatePackage(filesLocation=…… \genadam\);
9) See the information in the output window and in the log. The WARNING: [License] No license.sas file provided; default (MIT) license file will be generated. can be ignored.
10) Check the directory from Step 4. and look for the genadam.zip package file.
11) To install the genadam package use the following code:
    %installPackage(packagesNames= genadam,sourcePath =..\genadam\);
12) To load genadam package into SAS environment:
    %loadPackage(genadam);
    Now we can use the contents of the package genadam.

13) To obtain information about the package genadam
    %helpPackage(genadam);
14) To obtain information about the macro inside the genadam package:
    %helpPackage(genadam,dayder);
15) To unload genadam package:
    %unloadpackage(genadam)

We have successfully developed a package named 'genadam.' This package is now available for sharing with users who may find it useful for their specific needs.

CONCLUSION

The SAS package, as previously mentioned, is a valuable tool for code sharing. However, one challenge for programmers is understanding how to use shared macro effectively. The SAS package framework addresses this by providing instant access to help files, making it especially beneficial for programmers unfamiliar with the shared code. There's potential to popularize this framework as a repository for storing these packages, allowing enthusiasts worldwide to contribute to development and problem-solving collaboratively. SAS software's limited availability due to licensing makes this initiative even more attractive, expediting programmers' work. The showcased packages illustrate how this can streamline clinical trials, where many macros are employed. It simplifies communication for managers and lead programmers who can easily share a single zip file instead of copying numerous code scripts. Overall, implementation could result in significant cost and time savings, provided users are educated on how to utilize these packages effectively once they're created.

REFERENCES

- Paper 4725-2020, Extended Version, 2023.09.04, SAS◦R Packages - The Way To Share (a How To) Bartosz Jablonski, Warsaw University of Technology
- Read the My First SAS Package: A How-To - Paper 1079-2021 article, available at communities.sas.com , describing the process of a package creation with more advanced example and technical details.
- Go to “bare metal” and read the SAS Packages - The Way to Share (a How-To) - Paper 4725-2020 - extended version - an article which holds all technical details on how the SAS Packages Framework works and how to use it (both as a developer and as a user).

ACKNOWLEDGMENTS

The authors would like to thank management teams from Genpro, Now part of Catalyst Clinical Research, Trivandrum, India for their support on this paper/presentation and want to thank Bartosz Jablonski(yabwon@gmail.com) for his valuable contributions on the topic.

CONTACT INFORMATION

Your comments and questions are valued and encouraged!

Contact the authors at:

Author Name	: **Eldho Alias**
Company	: Genpro, Now part of Catalyst Clinical Research
Address	: Second Floor, Nila Building Technopark Campus
City / Postcode	: Thiruvananthapuram, Kerala – 695581
Work Phone	: +914717198600
Email	: eldho.alias@catalystcr.com
Web	: catalystcr.com

Co-Author Name : **Prasanth Prabhakaran**
Company : Genpro, Now part of Catalyst Clinical Research
Address : Second Floor, Nila Building Technopark Campus
City / Postcode : Thiruvananthapuram, Kerala – 695581
Work Phone : +914717198600
Email : prasanth.prabhakaran@catalystcr.com
Web : catalystcr.com

Co-Author Name : **Rahulraj V R**
Company : Genpro, Now part of Catalyst Clinical Research
Address : Second Floor, Nila Building Technopark Campus
City / Postcode : Thiruvananthapuram, Kerala – 695581
Work Phone : +914717198600
Email : rahulraj.raju@catalystcr.com
Web : catalystcr.com