SEASUG 2026 - Paper 85

Emulating libname ZIP engine for SAS®

Bartosz Jabłoński - yabwon / Warsaw University of Technology

ABSTRACT

SAS offers multiple library engines that allow reading (and writing) data from local single and multiple directories, external databases, even Excel spreadsheets or JSON files. Additionally SAS' files interface allows the use of different devices like DISKs, PIPEs, URLs, and even ZIP files to move file based data around. The libname statement unfortunately does not allow for storing and retrieving data sets from library assigned to a ZIP file (like Excel or JSON). The `LibnameZIP` package tries to overcome that limitation and introduces the functionality that mimics the library ZIP engine.

Table of contents

INTRODUCTION 1
THE LIBNAMEZIP PACKAGE CONTENT 1
The `%libnamezip()` macro 2
The `%libnamezipclear()` macro 2
The `%libnamezipdata()` macro 2
The `%datalibnamezip()` macro 3
The `%procdellibnamezip()` macro 3
The `%proccopylibnamezip()` macro 4
The `%libnamezipset()` macro 4
HOW DOES IT WORK? 4
INFORMATION FILE AND VIEWS 5
METADATA AND DATA STORAGE FORMAT 5
MACROS DETAILS 5
LIMITATIONS 5
CONCLUSION 6
REFERENCES 7
Appendix A - code coloring guide 7
Appendix B - getting started with SAS packages 8
INDEX 9

INTRODUCTION

To please programmers the SAS libname engine offers a plethora of connections to data sources, including (but not limited to) local directories, Excel files, JSON files, external databases, like Teradata, Oracle, Postgres (through SAS/ACCESS), or DubcDB (with JDBC), not to mention the XPORT "transport" files, or support for historical ones like V6. Library engines like SPDE allow to spread data across multiple disks and process them in parallel. Big data sources, e.g. Hadoop (through Hive or Impala) are supported too. Let's not forget about special-purpose engines like CVP or JMP. And of course, the latest in-memory CAS libnames in Viya. It seems like every possibility is covered, as you can access data from whatever "device" you want... Is it truly the case? Or maybe... is it not? As surprising as it may seem, there is no direct library engine that allows you to work with your data (read and write) inside a ZIP file! This paper you are reading introduces a SAS package named `LibnameZIP` that seeks to fill that gap, before SAS R&D delivers a native SAS libname ZIP engine!

The idea of creating such an engine came to me during discussion with Stu Sztukowski at SAS Innovate 2026 conference. We were talking about his comparison of SAS and DuckDB database local files. Stu said DuckDB files were on average a third of SAS data sets' sizes. We naturally started discussing compression options and then the idea of a ZIP libname engine that keeps data inside a ZIP file popped up. The rest is history... and some long (human done[1]) programming hours.

THE LIBNAMEZIP PACKAGE CONTENT

The `LibnameZIP` package is available for you to test, play, and use as an open source, MIT licensed, SAS package (see [**Jablonski 2020**] or [**Jablonski 2023**] to learn more about SAS packages).

[1] Though I find generative models quite practical, no "AI" was used for the job. Thus, all errors are sole consequences of the good old "natural stupidity."

You can find LibnameZIP in the SAS Packages Archive under the following address:

https://github.com/SASPAC/libnamezip

See "Appendix B - getting started with SAS packages" to learn how to install and load SAS packages to your SAS session.

The currently available version is the 0.2.0 and it is based on seven macros. The BasePlus package (version at least 3.1.4) is required for LibnameZIP to work. Subsequent sections introduce you to the "magnificent" seven macros constituting LibnameZIP package and provide you with an overview and examples. Upcoming subsections provide a "driver's seat" perspective, the "car mechanic" technical explanation of how it really works under the hood is done in next section.

The %libnamezip() macro

The %libnameZip(lib,path) macro is the "starting point". You use it to assign a libname lib to a zip file provided in the path parameter. A typical call to the macro would be similar to the following one:

code: assign libname

```
1 %libnameZip(A,/home/&sysuserid./A.zip)
```

After execution, the subsequent things happen:

- the A library is assigned to the /home/&sysuserid./A.zip file (the file is created if need be, but only after the first data are saved in it).
- A technical data set A.libnameZIP (used by other components of the package) shows up.
- A technical filename reference A pointing to the zip file provided by the path parameter is created.
- If the zip file contains data created earlier with the LibnameZIP, they will be visible in the A library (through SAS views, details later).

The %libnamezipclear() macro

The %libnameZipClear(lib) is a cleaning macro that un-assigns lib library from your session. the macro call to it looks like the following:

code: clear libname

```
1 %libnameZipClear(A)
```

As the result, both libname and filename assignments are cleared from the SAS session. The ZIP file is "considered" permanent so it is not deleted.

The %libnamezipdata() macro

There are two main macros populating ZIP file with data, the %libnameZipData(lib,ds,source) macro is the first, simpler, of the two. It allows you to save data from a source SAS data set as ds data set in the lib library. A canonical use case would be:

code: clear libname

```
1 %libnameZipData(A,cars,SASHELP.CARS)
```

After execution data are copied and the CARS entry is visible in the A library right away. The macro is used to move already existing SAS data sets to the ZIP-library. This is practical when you already have those data sets, if you don't, the macro presented in the next subsection can help.

The %datalibnamezip() macro

The name of the %dataLibnameZip() macro is designed so that the call to it could mimic a 4GL syntax of a DATA step, similar to the following one:

code: 4GL DATA step

```
DATA <libnameZIP>.datasetname;
  <valid 4GL DATA step code>
run;
```

The %dataLibnameZip() macro is designed to take your code as an argument and produce data directly inside the ZIP file. The syntax of a call to the macro is the following:

code: syntax

```
%dataLibnameZip(lib.ds;
 <valid 4GL DATA step code>
)
run; /* optional, just to have a "nice looking closure" */
```

The ds is the name of a data file created inside the lib ZIP-library. You do *not* add the DATA keyword before it. The semicolon (;) is *obligatory*, you have to put it after the data set name. *Do not add* the RUN statement at the bottom of the 4GL code (i.e., inside the macro call parenthesis). You could add it after the macro call code, but *purely* for aesthetic reasons. An example of a code snippet that uses the macro is:

code: syntax

```
%dataLibnameZip(A.cars2;
  SET sashelp.cars;
  BY origin notsorted;
  DROP invoice mpg:;
  IF last.origin THEN
    DO i = 1 to 5 by 2, 2 to 6 by 2;
      OUTPUT;
    END;
)
run; /* optional */
```

This particular example produces the following note in the LOG:

the log - execution note

```
NOTE: A total of 132 records were written to the file library A.
```

Since the macro interacts with your code syntax (the result, how the code was understood, is displayed in the LOG) there are some very simple rules you have to follow if you want the macro to work for you. Those rules are:

- only a single data set can be created at a time (sorry...),
- data set options on created data set are ignored (no point in adding them),
- only the PUTLOG statement is allowed when writing messages to the LOG, as a best practice, simply do not use the PUT statement at all when working with the macro,
- multiple OUTPUT statements are allowed, but none of them can contain a data set name, and
- if possible, try to avoid using word "output" in the code, unless it is the OUTPUT statement.

The %procdellibnamezip() macro

To permanently remove data from a ZIP-library use the %procDelLibnameZip(lib,ds) macro, it will (just like the DELETE procedure) delete the ds data set from the lib ZIP-library for you. As you may expect, the code is very simple:

code: delete

```
%procDelLibnameZip(A.cars2)
```

After the execution the `A.cars2` no longer exists in the ZIP file.

The `%proccopylibnamezip()` macro

The `%procCopyLibnameZip(in=,out=<,select=><,exclude=>)` macro serves as a utility that allows copying data between two ZIP-libraries. Let's say, you have ZIP-library `A` that contains 42 members (e.g. `cars1` to `cars42`) and you want only the first 17 of them to be copied to a ZIP-library `B`, The following code does that for you:

code: copy

```
%libnameZIP(B,/home/&sysuserid./B17.zip)

%procCopyLibnameZip(in=A,out=B,select=cars1-cars17)
```

The `%libnamezipset()` macro

The last one, the seventh, is the `%libnameZipSet(lib,ds)` macro. You may think that a `SET` statement "counterpart" macro should be an important one, but in the setup the `LibnameZIP` package provides this one is merely a utility macro. The `LibnameZIP` package setup serves your data through SAS views (details in the next section). In case you accidentally delete (from the `lib` library) the view `ds` (that provides an interface to your data enclosed in the ZIP file) or you simply want to be sure the view will be there (available for processing), the `%libnameZipSet()` macro can "do" the view for you. The macro executes as a macro-function and produces the "`&lib..&ds.`" string, whence you should call it in a statement that can "utilize" the string, for example:

code: set

```
%put %LibnameZipSet(A,cars2)

/* or */

data _null_;
  stop;
  set %LibnameZipSet(A,cars2);
run;
```

Of course, the `A` ZIP-library has to exist.

HOW DOES IT WORK?

Before a deep dive into the core (gory) details of the implementation, let me remind you that the `LibnameZIP` package is open source, and you can easily preview source code of its each component by simply calling the `%previewPackage()` macro, for example the following snippet:

code: source code

```
%previewPackage(LibnameZip,dataLibnameZip)
```

prints out the `%dataLibnameZip()` macro source code in your SAS session LOG. A short, dry, and general summary could just mention that: "*the package heavily relies on: the `DOSUBL` function, SAS views, and the ZIP `FILENAME` statement; see the source code.*" But please, allow me to be just a little bit more detailed.

INFORMATION FILE AND VIEWS

When the %libnameZip() macro assigns the library LIB to a /path/file.zip ZIP file the following things happen:

- A subdirectory named LIB (upcase letters) is created inside the SAS session's WORK location.
- A "regular" SAS library named lib is created and points to WORK's LIB subdirectory.
- A lib.libnameZip data set (literally named "libnameZip") is created. It stores the "/path/file.zip" string in the path variable so that other macros can use it as a navigation point.
- The ZIP file is checked for existence of files with the "meta" extension, if they exist, views corresponding to those files names are defined in the lib library.

METADATA AND DATA STORAGE FORMAT

Data stored inside the ZIP file are not "just compressed" SAS data sets. Every datum D inside the ZIP is organized through two separate files d.meta and d.data. The first file (*.meta), as the extension suggests, stores the metadata information, i.e., the order of variable names, information about every variable including: types, lengths, labels, formats, and informats, and lastly the record length. The later file (*.data) keeps the actual data. Every record is stored in two section blocks. Numeric variables are stored first in one block, with HEX16. format to keep numeric precision. Character variable are stored after numeric with fixed length $CHARn. format, so each variable is kept with its full byte length. Fixed record length is kept for every observation.

MACROS DETAILS

It is the %libnameZipData() that covers the job of both data and metadata writing to the ZIP file. Even the %dataLibnameZip() macro uses it internally in a portion.

Metadata extraction, views creation, data deletion, and data copying macros execute those processes inside the %sysfunc(dosubl(%nrstr(...))) block sandwiches that make the whole execution macro-level transparent to the user.

Among exceptions is the %libnameZipClear() that is a "direct" call to the DATASETS procedure and CLEAR option of the LIBNAME and FILENAME statements.

The %dataLibnameZip() macro uses the PARMBUFF macro option to capture the input string containing the user's code; next the macro takes advantage on the DATA step compilation phase to generate metadata and extracts them with the CONTENTS procedure from an "egg shell" temporary data set; lastly the code is parsed and searched for every occurrence of an "output<zero or more spaces>;" string and, when found, the string is replaced with specially formated LINK statement that redirect to a series of PUT statements that populate the "data" file (see the next section for more information).

LIMITATIONS

Knowing your limitations allows you to avoid painful "wall hits" while various activities, and since the LibnameZIP is only a "within language" extension and not a real language extension, it has some limitations too. Highlighting those limitations allows for the fuller and more comfortable use of the package. There are two kinds of limitations.

The first group of limitations is mostly related to the %dataLibnameZip() macro and it includes:

- due to the fact how the OUTPUT statement is handled, only one data set at a time can be generated with the macro call;

- the code provided by the user is searched for the "output<zero or more spaces>;" string, but it is not checked if the string is enclosed in quotes, so I recommend caution with the "output" word;
- the "output;" is replaced with the LINK statement that moves code execution to a series of PUT statements writing into the ZIP through the FILE statement, thus any use of PUT in the code will also write into the ZIP. Obvious solution is to use PUTLOG.
- Since PARMBUFF is used length of the code is limited by the macro variable length of 64k.
- When you read and write ZIP-libname through one DATA step (i.e., using the %dataLibnameZip() macro), you cannot use the same data set as input and as output.

The second group of limitations is related to the SAS Display Manager environment.

- The first is a "technical" one. The SAS DMS interface throws an error saying: "in use by you in resource environment DMSEXP" when you have ZIP-libname open in the DMS interface and at the same time you try to re-assign the library with the %libnameZip() macro. Easy to overcome by not selecting the library...
- The second limitation is a Windows DMS bug, registered as product defect (case CS0434008), unfortunately not scheduled for correction in an upcoming (M10?) release[2]. The limitation is the following. Imagine a code snippet that: 1) creates a data set in ZIP-libname A, and 2) executes %dataLibnameZip() macro that reads and writes to A ZIP-libname in on DATA step, something like this:

code: read & write in one step

```
/* 1) */
%libnameZIP(A,/home/&sysuserid./A.zip)
%LibnameZipData(A,class,SASHELP.CLASS)

/* 2) */
%DataLibnameZip(A.CLASS2;
  set A.CLASS;
  do i = 1 to 7;
    output;
  end;
)
run;
```

Under Linux/UNIX operating systems the code works like a charm: without any problems, data are read from ZIP and written to ZIP. Under Windows unfortunately, even though a note saying "NOTE: 133 records were written to the file A(class2.data)." is displayed in the LOG, the file is not created at all.

CONCLUSION

Trade-offs are a constant element of data processing. The unholy "CPU-RAM-DISK" trinity forces programmers toward continuous maneuvering between which component will be more utilized than other two. The LibnameZIP package allows you to save space at the cost of processing time (for compression and decompression). Space saving that sacrifice additional CPU cycles may be the choice you lean to, as one of test cases allowed to go from 6.3GB data set in standard SAS library to 123MB file in ZIP library at the cost of higher CPU utilization. Syntax changes needed to write code that works with ZIP library are almost imperceptible, and the possibility of having all data sets enclosed inside a single container is also an advantage. Standard data set metadata are preserved, even reading and writing to the same ZIP file in one DATA step is (partially) possible. Hopefully those arguments convince you to start using the LibnameZIP package.

The End

[2]But you never know. Remember it is the SAS team! They can surprise us just like they did with PROC PYTHON for M9!

REFERENCES

[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 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

[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.
```

Appendix B - getting started with SAS packages

Below text is an excerpt from the [**Jablonski 2026**] article.

Install SAS Packages Framework? Regardless of whether you will be installing SAS Packages Framework (SPF) on your local computer or on a company server (to the delight of the SAS folks) the step "zero" is to create a directory where the SPF and packages (in the future) will be stored. To use packages in a SAS session, there is, in fact, only one requirement: The packages directory *must be referred to* as `packages`, and that reference cannot be deleted during the session.

For example, if you decided the packages directory is `/sas/packages/`, in the SAS session run:

code: assign fileref

```
filename packages "/sas/packages/";
```

When: 1) the directory exists, 2) the `packages` fileref is assigned, and 3) the SAS session has the Internet access (to the GitHub repository of the SPF in particular), the following code snippet will install the SPF on the computer:

code: install the framework

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

%installPackage(SPFinit);

filename SPF clear;
```

Successful execution of the installation should produce a note like the following in the `LOG`:

the log - successful installation

```
INFO: Package SPFinit installed.
```

You need to execute this process only for the very first time. If the SPF file is on the computer and you, for example, want to install the latest version of the framework - do it exactly as you would install any other package.

If access to the Internet from the SAS session is not guaranteed, it is possible to execute the installation manually. All you need to do is: visit the `https://github.com/yabwon/SAS_PACKAGES` repository; navigate to subdirectory named `SPF`; and download file `SPFinit.sas` to the `/sas/packages/` directory. That is all; the installation is done.

How to enable the SPF?Similarly to how the libraries are assigned at the beginning of your SAS session, the SAS Packages Framework, too, must be enabled. This can be done by running the following snippet:

code: enable the framework

```
filename packages "/sas/packages/";
%include packages (SPFinit.sas);
```

After running those two lines, the SPF's macros are compiled, and the framework is ready for work. That code snippet can be executed at the start of a session, but personally, I prefer to save it in my `autoexec.sas` file, so I do not have to run it every time myself.

How to install a package from the SAS Packages Archive? The installation process for a package, in a nut shell, is all about downloading that package `zip` file into the packages directory on your computer (or the computer you want to use SAS packages on). For packages located in SAS Packages Archive (SASPAC) just run:

code: install from SASPAC

```
1 %installPackage(packageName)
```

The macro will search through SASPAC as the default location, and if it finds the package there, it will download it to the packages directory.

Loading a package To start using the SAS package you have just installed, first you must load it. Macro dedicated to that purpose is the `%loadPackage()` macro. In a brand new SAS session, *which has the SAS Packages Framework enabled*, run the following code:

code: load a package to SAS session

```
1 %loadPackage(packageName)
```

The `%loadPackage()` macro takes a package name as the first parameter and in 99 percent of cases, it is all you need to do.

INDEX

concept
  AUTOEXEC.SAS, 8
  LIBNAME ENGINE, 1
  SAS DISPLAY MANAGER, 6
  SAS VIEW, 4

format (character)
  $CHARn., 5
format (numeric)
  HEX16., 5
function
  DOSUBL, 4

macro
  %DATALIBNAMEZIP(), 3–6
  %LIBNAMEZIPCLEAR(), 2, 5
  %LIBNAMEZIPDATA(), 2, 5
  %LIBNAMEZIPSET(), 4
  %LIBNAMEZIP(), 2, 5, 6
  %LOADPACKAGE(), 9
  %PREVIEWPACKAGE(), 4
  %PROCCOPYLIBNAMEZIP(), 4
  %PROCDELLIBNAMEZIP(), 3

option
  CLEAR, 5
  PARMBUFF, 5, 6

package
  BasePlus, 2
  LibnameZIP, 1, 2, 4–6
procedure
  CONTENTS, 5
  DATASETS, 5
  DELETE, 3

statement
  DATA, 3
  FILENAME, 4, 5
  FILE, 6
  LIBNAME, 5
  LINK, 5, 6
  OUTPUT, 3, 5
  PUTLOG, 3, 6
  PUT, 3, 5, 6
  RUN, 3
  SET, 4