Paper SAS4201-2016

Writing Packages: A New Way to Distribute and Use SAS/IML® Programs

Rick Wicklin, SAS Institute Inc.

ABSTRACT

SAS/IML® 14.1 enables you to author, install, and call *packages*. A package consists of SAS/IML source code, documentation, data sets, and sample programs.

Packages provide a simple way to share SAS/IML functions. An expert who writes a statistical analysis in SAS/IML can create a package and upload it to the SAS/IML File Exchange. A nonexpert can download the package, install it, and immediately start using it. Packages provide a standard and uniform mechanism for sharing programs, which benefits both experts and nonexperts.

Packages are very popular with users of other statistical software, such as R. This paper describes how SAS/IML programmers can construct, upload, download, and install packages. They're not wrapped in brown paper or tied up with strings, but they'll soon be a few of your favorite things!

INTRODUCTION

After weeks of work, you finally have completed that complicated SAS/IML project. It was challenging to write all those functions, and finding that last bug was frustrating, but now your program works beautifully. You beam with pride when your boss compliments you in front of the whole team and suggests that you post your library of functions to the SAS/IML File Exchange so that other programmers can benefit from your hard work.

Now you face a whole new problem: what is the best way to disseminate your work? Your project has dozens of files. There are module definitions, sample data sets, documentation, programs that verify the numerical computations, and programs that demonstrate how to use the function library.

There is a solution to your dilemma. Beginning with SAS/IML 14.1, you can create a *package* to distribute your project. A SAS/IML package is a ZIP file that contains everything that relates to a project. Source files? Yes. Documentation? Yes. Sample programs and data sets? Yes and yes.

After you create a package, other programmers can easily download and install the package and use your work. As a bonus, authoring a package is a great to way to show the wider SAS® community what your boss already knows: you are a fantastic programmer!

INSTALL A PACKAGE

Packages enable SAS/IML programmers to use and build on the work of others. Probably the best way to see the power of packages is to download and install one yourself. Hopefully, that experience will inspire you to create your own package.

To demonstrate packages, I wrote the `polygon` package. This package is a collection of functions that compute properties of planar polygons. For example, the package contains a function that computes the perimeter of a polygon, another that computes the area, and another that computes the centroid. The package also contains a function that can solve the point-in-polygon problem (Hormann and Agathos 2001). In addition, the package includes documentation files, example programs, and data sets that contain example polygons.

The following steps show how to download and install the `polygon` package:

1. Download the **`polygon.zip`** file from the SAS/IML File Exchange: `https://communities.sas.com/sas-iml-exchange`. Save the ZIP file to your local network. This paper assumes that the ZIP file is saved to **`C:\Packages\polygon.zip`** on a Windows PC.
2. Use the PACKAGE INSTALL statement to install the package. Specify the location of the ZIP file, as follows:

```
proc iml;
package install "C:\Packages\polygon.zip";
quit;
```

The PACKAGE INSTALL statement unzips the package files to a system-specific location. In most situations, you only need to install the package one time.[1]

USE A PACKAGE

The standard SAS programming interfaces (the SAS windowing environment, SAS® Enterprise Guide®, and SAS® Studio) all load packages from the same location. Therefore, you can use any of these programming interfaces to call a package that is installed on a local SAS server.

Use the PACKAGE LOAD statement to load the function modules in the package, as follows:

```
proc iml;
package load polygon;
```

The SAS log displays a note for each module that is loaded. After the modules are loaded, you can call them. If you can't remember the calling syntax, you can use the PACKAGE HELP statement to see a brief overview of the package, as follows:

```
package help polygon;
```

In PROC IML, the PACKAGE HELP statement displays the help file in the SAS log. Part of the output is shown in Figure 1. The `polygon` package also contains documentation in PDF format. The PDF documentation is more extensive and provides a working example for each function in the package.

Figure 1 Partial Output for Help for Polygon Package

```
Polygon Package

Description: Computational geometry for polygons

A polygon is represented as an n x 2 matrix, where the rows represent
adjacent vertices for the polygon. The polygon should be "open,"
which means that the last row does not equal the first row.

Multiple polygons are represented by an n x 3 matrix. The third
column is an ID variable, and the unique values of the column identify
different polygons.

PolyArea(P);
   returns the areas of simple polygons.

PolyIsConvex(P);
   returns 1 if a simple polygon is convex. Otherwise, returns 0.

PolyPerimeter(P);
   returns the perimeter of a polygon.

PolyPtInside(P, pts);
   determines whether points are inside a polygon.
   A point is "inside" if the winding number at that point is odd.
```

[1] The SAS/IML® Studio application installs a package on the client PC, rather than on the SAS server. Therefore, you need to install a package from within SAS/IML Studio before you can call the package from an IMLPlus program.

With the help of the documentation, you can write a short program that computes the perimeter and area of a polygon and determines whether it is convex:

```
P = {0 0,  1 0,  1 2,  0 2};   /* vertices of rectangle */
Perimeter = PolyPerimeter(P);
Area = PolyArea(P);
IsConvex = PolyIsConvex(P);
print Perimeter Area IsConvex;
```

Figure 2 shows the results for a rectangle that has a width of 1 unit and a height of 2 units.

Figure 2 Properties of a Rectangle

Perimeter	Area	IsConvex
6	2	1

In a similar way, you can test whether a specified point is inside, outside, or on the boundary of a polygon. The following statements test whether each of five ordered pairs (each of which represents a point) is inside the rectangle. Figure 3 shows that the first and fifth points are outside, the second point is on the boundary, and the third and fourth points are inside.

```
pts = {-1 1, 0 1, 0.5 1, 0.8 1.5, 2 1}; /* five points */
inRect = PolyPtInside(P, pts);          /* test whether points are in rectangle */
print pts[colname={X Y}] inRect;
```

Figure 3 Five Points and Whether Each Is Inside the Polygon

pts		
X	Y	inRect
-1	1	0
0	1	.
0.5	1	1
0.8	1.5	1
2	1	0

The `polygon` package has other functions that are not shown in this paper. The package also includes sample data sets and programs that demonstrate features of the package.

THE PACKAGE STATEMENT

Packages are managed by using the PACKAGE statement. The PACKAGE statement has seven keywords that help you manage and use packages. Three of these were used in the previous section:

- The PACKAGE INSTALL statement installs a package.
- The PACKAGE HELP statement displays documentation for the package. Figure 1 shows part of the help file for the `polygon` package.
- The PACKAGE LOAD statement loads the modules that are defined by a package.

The following are the remaining PACKAGE statements:

- The PACKAGE LIST statement lists the installed packages.
- The PACKAGE UNINSTALL statement uninstalls a package.
- The PACKAGE INFO statement displays information about an installed package. For example, the following statement displays a table of information about the `polygon` package:

```
package info polygon;
```

Figure 4 displays information about the package. The table summarizes information that is contained in the *info.txt* file. It also displays the directory in which the package is installed.

Figure 4 Information about a Package

Package Information	
Name	polygon
Description	Computational geometry for polygons
Author	Rick Wicklin <Rick.Wicklin@sas.com>
Collection	Private
Version	1.0
Requires IML	14.1
Directory	C:\Users\frwick\Documents\My SAS Files\IML\Packages\polygon

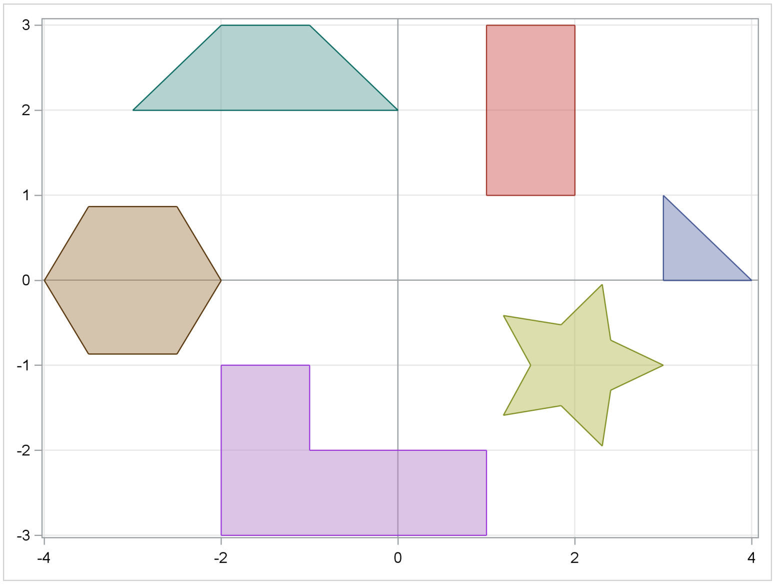
- The PACKAGE LIBNAME statement creates a SAS libref that points to the data directory. For example, the following statements assign the libref POLYDATA:

```
package libname PolyData polygon;
use PolyData.Simple;
read all var {u v ID} into P;   /* 3rd column is ID variable */
close PolyData.Simple;

run PolyDraw(P);
```

The USE statement opens the Simple data set, which is distributed with the package. The READ statement reads variables into the matrix **P**. The PolyDraw function is used to visualize four convex and two nonconvex polygons, as shown in Figure 5.

Figure 5 Plot of Polygons



HOW TO CREATE A PACKAGE

The purpose of this paper is not to describe the `polygon` package but to describe how you can create your own package. Hopefully the previous section has convinced you that packages are convenient and useful.

You create a package by creating a ZIP file that contains certain files in certain directories. This section assumes that you have already generated the package content, which is usually a collection of functions. (This is often the hardest part of creating a package.) It is also assumed that you have written documentation and example programs.

Before you create your package, choose its name. The name of a package should be concise but informative. The name must be a valid SAS name, which means it must contain 32 characters or less, begin with a letter or underscore, and contain only letters, underscores, and digits.

THE STRUCTURE OF A PACKAGE

The SAS/IML 14.1 documentation provides detailed instructions about how to create a package. This section summarizes the main steps.

To create a package, first create the root directory, which should have the same name as the package. For this example, create a root directory named *polygon*. Inside the root directory, create the following:

- A plain text (ASCII) file named *info.txt*. The contents of the *info.txt* file are described in the next section.
- A subdirectory named *source*. Copy the source files that define the SAS/IML modules into this directory.
- A subdirectory named *help*. Put the documentation into this directory. The documentation should contain a plain text file that has the same name as the package and has a *.txt* extension. The text file should contain the basic syntax for calling functions. The contents of this file are displayed in the SAS log when a user submits the PACKAGE HELP statement. The *help* directory can also contain a PDF or HTML file that provides more complete documentation.
- A subdirectory named *programs*. Put the example programs into this directory.
- A subdirectory named *data*. Copy SAS data sets into this directory. Example data are especially useful if your package analyzes data that must be in a certain format or have certain properties. You can also distribute data in other forms, such as in a CSV file or an Excel spreadsheet, but be sure to include a sample program that shows how to read data that are not in SAS data sets.

The *info.txt* file and *source* directory are required. The other directories are recommended, but be aware that some compression utilities do not allow you to include empty directories. A package can contain additional subdirectories, such as directories for macro code, templates, image files, and so on.

For this paper, the root directory was created on a Windows PC in the location **`C:%HOMEPATH%\My Documents\My SAS Files\polygon`**. (In Windows 7, **`%HOMEPATH%`** resolves to **`C:\Users\`*`username`***.) You can type the following statements into a command prompt to see the directories and files in a hierarchical structure:

```
C:> tree "C:%HOMEPATH%\My Documents\My SAS Files\polygon" /A /F
```

Figure 6 shows the output of this `tree` command:

Figure 6 Directory Structure for Polygon Package

```
C:\Users\frwick\My Documents\My SAS Files\polygon
|       info.txt
|
+---data
|       simple.sas7bdat
|       states48.sas7bdat
|
+---help
|       polygon.docx
|       polygon.pdf
|       polygon.txt
|
+---programs
|       Example.sas
|       TestNonSimplePoly.sas
|       TestPolyDraw.sas
|       TestPolyPtInside.sas
|
\---source
        PolyArea.iml
        PolyBoundingBox.iml
        PolyCentroid.iml
        PolyDraw.iml
        PolyDrawImpl.iml
        PolyInstallDir.iml
        PolyIsConvex.iml
        PolyPerimeter.iml
        PolyPtInside.iml
        PolyRandom.iml
        PolyRegular.iml
        PolyStack.iml
```

The source files in the `polygon` package have a *.iml* extension, but you can use a *.sas* extension if you prefer. The contents of these files are read into a SAS/IML session. These files should contain only valid SAS/IML statements, but should not contain the PROC IML or QUIT statements.

THE PACKAGE INFORMATION FILE

The *info.txt* file provides information to the PACKAGE INSTALL and PACKAGE LOAD statements. The file contains some header information, followed by keyword-value pairs.

The first line in the file specifies the format for the file. Future enhancements of the PACKAGE statement might support additional formats, but for SAS/IML 14.1 the first line must be as follows:

```
# SAS/IML Package Information File Format 1.0
```

Subsequent lines define keyword-value pairs. If a value requires more than one line of text (such as the SOURCEFILES keyword), subsequent lines must be indented by at least one space or tab character.

The following keyword-value pairs are recommended:

NAME:	Specify the name of the package. The case (uppercase, lowercase, or mixed case) of this keyword value is important. Even through SAS is a case-insensitive language, some operating systems (such as Linux) are case-sensitive. The package name and the filenames in the *help* directory should have the same case.
DESCRIPTION:	Specify a brief description of the package.
AUTHOR:	Specify the authors of the package and their contact information.

VERSION:	Specify the version of the package. The version can have up to four levels, separated by decimal points. For example, valid values are 1.0, 2.7.1, and 3.1.4.1.
REQUIRESIML:	Specify the version of SAS/IML software that is required to run the package.
NOTES:	Specify other information. You can acknowledge a collaborator or cite a journal article that provides additional information about the package.
SOURCEFILES:	Specify the source files for the package, one filename on each line. The case (uppercase, lowercase, or mixed case) of this keyword value is important because certain file systems (such as Linux) are case-sensitive. The source files usually have a *.iml* extension.

The *info.txt* file supports other keywords; see the *SAS/IML Studio User's Guide*.

The following statements show the *info.txt* file for the `polygon` package. For most packages, all files in the *source* directory are listed after the SourceFiles keyword.

Figure 7 Information File for a Package

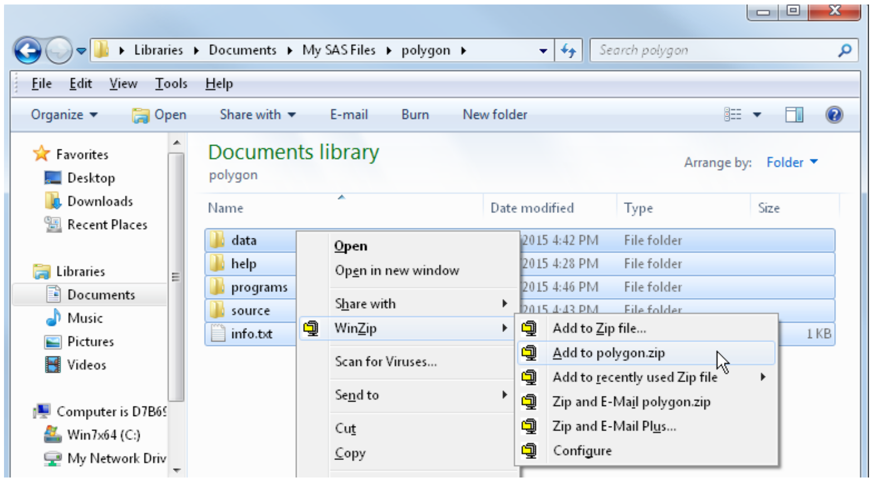
```
# SAS/IML Package Information File Format 1.0
Name:        polygon
Description: Computational geometry for polygons
Author:      Rick Wicklin <Rick.Wicklin@sas.com>
Version:     1.0
RequiresIML: 14.1
Notes: This package accompanies Wicklin (2016) "Writing Packages:
       A New Way to Distribute and Use SAS/IML Programs,"
       Proceedings of the SAS Global Forum 2016 Conference.
SourceFiles: PolyInstallDir.iml
             PolyArea.iml
             PolyBoundingBox.iml
             PolyCentroid.iml
             PolyDraw.iml
             PolyIsConvex.iml
             PolyPerimeter.iml
             PolyPtInside.iml
             PolyRandom.iml
             PolyRegular.iml
             PolyStack.iml
```

CREATE THE PACKAGE

A package is a ZIP file that contains certain files and a specified directory structure. The filename of the ZIP file must be the name of the package followed by the *.zip* extension.

You can use a standard compression utility program to create the ZIP file. This paper uses WinZip software. Figure 8 shows how to create a ZIP file by using WinZip. With WinZip and most other Windows compression utilities, you can open a Windows Explorer window and navigate to the root directory where you have stored all your package files. To create a ZIP file, select all the files and subdirectories, right-click on the selected files, and choose **WinZip ►Add to *`packagename`*`.zip`** from the pop-up menu.

Figure 8 Creating a ZIP File by Using WinZip



The program creates the file *packagename.zip*. So that the ZIP file can be correctly unzipped by the SAS/IML PACKAGE statement, make sure that the ZIP file uses relative paths and preserves the directory hierarchy when the files are unzipped. In WinZip 18.0, click the **Settings** tab and select **Use Folder Names** from the **Unzip Settings** menu.

Now you can send the ZIP file to a colleague, put it in a shared folder on your company's network, or upload it to the SAS/IML File Exchange for the whole world to see.

ADVANTAGES OF USING PACKAGES

Packages are a new way to distribute SAS/IML functionality. You might wonder how packages compare to older methods for sharing SAS programs. In particular, longtime SAS programmers are familiar with the %INCLUDE statement, and SAS/IML has provided the STORE and LOAD statements for modules for decades. In what ways are packages an improvement?

The short answer is that packages are more portable, more convenient, and more robust. Furthermore, if you upload your package to the SAS/IML File Exchange, there are additional advantages. The following list describes specific advantages that packages provide over previous techniques for sharing SAS/IML functions:

- Packages are self-contained. They contain functions, programs, documentation, data, and related materials.
- Package promote standards. Authors know how to structure the directories and the ZIP file. Consumers know how to install, load, and use the package.
- Packages are manageable. The PACKAGE statement provides a programmatic way to manage packages. For example, the PACKAGE LIST statement lists all installed packages. The PACKAGE HELP and PACKAGE INFO statements display information about a particular package. And of course the PACKAGE INSTALL and PACKAGE UNINSTALL statements control which packages are installed.
- Packages are transparent. After installing a package, you do not need to specify any directories when you use the package. The PACKAGE LOAD statement loads the modules. The PACKAGE LIBNAME statement defines a libref to the data directory. Neither statement requires a directory.
- Packages make it easy for a group of programmers to use a common library of functions. A SAS administrator can install the packages in the PUBLIC collection for all programmers to use.
- Packages persist across SAS releases. Assuming that your SAS administrator does not change the location of the PRIVATE and PUBLIC collections, any packages that you installed in a previous release of SAS are still available when you upgrade to a new release.
- Packages enable a programmer to build on previous work. You can write a package that calls functions in other packages. Your package merely has to use the PACKAGE LOAD statement to load the functions. (The DEPENDENCIES keyword in the *info.txt* file tells users of your package that they need to install those other packages.)

- Packages on the SAS/IML File Exchange are highly visible. The SAS/IML File Exchange is part of the popular SAS Support Communities, which gets a lot of traffic. Consequently, Internet searches are more likely to locate packages on the File Exchange than packages on less-visible websites.
- Packages promote feedback. When an author uploads a package to the SAS/IML File Exchange, other programmers can post comments about the package. They can ask questions, point out deficiencies, and suggest improvements. This results in better and more robust packages.
- Packages have a versioning feature. You might decide to update your package if you find a bug or if a future SAS/IML release contains a feature that you want to incorporate. When you update your package, you can update the value of the VERSION keyword in the *info.txt* file. You can also use the REQUIRESIML keyword to specify that the package requires a recent version of SAS/IML software.

CONCLUSION

This paper shows how to create packages in SAS/IML 14.1. A package is distributed as a ZIP file. After a programmer downloads and installs a package, he or she has immediate access to the functions, data, and documentation in the package. Programmers of all abilities can use packages to extend the functionality of SAS/IML software and to share their expertise with others.

REFERENCES

Hormann, K., and Agathos, A. (2001). “The Point in Polygon Problem for Arbitrary Polygons.” *Computational Geometry* 20:131–144.

ACKNOWLEDGMENTS

The author is grateful to Simon Smith for reading an early draft of this paper.

APPENDIX: FREQUENTLY ASKED QUESTIONS ABOUT SAS/IML PACKAGES

Q: I have not yet upgraded to SAS/IML 14.1. Can I install packages in earlier releases?

A: No. The PACKAGE statement was introduced in SAS/IML 14.1. You cannot install packages in a previous release.

Q: I wrote a paper for a journal that includes a long SAS/IML program. Can I use a package to distribute my program?

A: Yes, but more people are likely to use your program if you encapsulate the main functionality into a set of modules.

Q: Does someone at SAS verify that packages give correct answers? Will SAS Technical Support help me if a package does not work correctly?

A: No. The accuracy of the package is the sole responsibility of the package author. Contact the author if you have problems using the package or if you think a computation is incorrect.

Q: What is the best way to contact the author of a package?

A: Authors should include contact information as part of the package documentation. If a package resides on the SAS/IML File Exchange, you can communicate with the author by posting comments.

Q: The author of the package is ignoring me! Please help!

A: Sorry to hear that. Post a question to the SAS/IML Support Community at `https://communities.sas.com/sas-iml`. There are many helpful people there.

Q: Where are packages installed?

A: A package is installed in a *collection*. A package can belong to the PRIVATE, PUBLIC, or SYSTEM collection. The collection determines the directory in which a package is installed. The following statements display the root directories for each collection:

```
proc options value
     option=(imlpackageprivate imlpackagepublic imlpackagesystem);
run;
```

You can also use the PACKAGE INFO statement to display the installation directory for an installed package.

Q: I run SAS/IML programs from SAS Studio. Can I install packages from that environment?

A: Yes. SAS Studio can access directories on your local computer. Put the ZIP file for the package in a location that is visible from SAS Studio. Then run the PACKAGE INSTALL statement as usual.

Q: Can I install and use packages from the free SAS® University Edition?

A: Yes. The installation guide for SAS University Edition tells you how to create a shared folder on your local machine. SAS University Edition runs in a virtual machine that can access that folder by using the alias **`/folders/myfolders`**. Put the ZIP file in that shared folder. Then use the following statements to install a package:

```
proc iml;
package install "/folders/myfolders/polygon.zip"; /* SAS University Edition */
```

Q: I noticed that packages are experimental in SAS/IML 14.1. Are there any known limitations?

A: There are two known issues, which have been fixed in preparation for the next release. The issues in SAS/IML 14.1 are as follows:

- You cannot use the SUBMIT statement in a file that gets loaded by a package.
- The PACKAGE INSTALL statement cannot read ZIP files that are created by using the Windows built-in compression program (**RMB►Send to►Compressed (zipped) Folders**).

Q: What compression utilities can I use to create a package ZIP file?

A: In a Linux environment, the PACKAGE statement works with ZIP files that were created by using the `zip -r` command. In a Windows environment, the PACKAGE statement has been tested with 64-bit versions of 7-Zip 15.11, PeaZip 5.8.1, PKZip 14.40, WinRAR 4.20, and WinZip 18.0. With PeaZip, you need to set the option **Extract all to ►No paths**.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Rick Wicklin
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513