Paper 018-2009

ODS Packages: Putting Some Zip in Your ODS Output

Eric Gebhart, SAS Institute Inc., Cary, NC

ABSTRACT

Packages and package templates add powerful new abilities to the Output Delivery System (ODS). Automatically create zip files and SAS packages that contain all the output generated from your SAS® program. ODS packages make it easy to e-mail your ODS output to others and can help keep your file system clear of file clutter.

With SAS® Integration Technologies, ODS packages can also publish to WebDAV, e-mail, queues, and subscriptions as well as archives.

ODS packages are also completely integrated with tagsets allowing for fully automatic generation of packages and the creation of some of the newer file formats like Open Document Format and Microsoft Office 2007.

Learn how easy it is to use ODS packages and find out how they will impact the future of ODS.

INTRODUCTION

ODS packages are some of the easiest and most useful new additions to ODS. In their most basic usage, they allow you to simply and easily create zip files from your ODS output. ODS packages can also use the package publishing framework provided by SAS Integration Technologies. When SAS Integration Technologies is installed, ODS packages supply a simple and powerful interface to all of the transports that SAS packages provide.

ODS packages can also be controlled by a new template definition called a *package template*. Package templates can make using packages even easier. And when package templates are combined with tagsets, the use of packages can become completely transparent. New destinations like Open Document Format and the Microsoft Office 2007 Office Open Format are now possible using all of these features of ODS packages.

This paper includes step-by-step explanations all of these features of ODS packages and how to use them.

ODS PACKAGES, THE BASICS

The simplest way to us ODS Packages is to create an ODS package sandwich. Just put the ODS PACKAGE OPEN statement at the top of your program and the ODS PACKAGE PUBLISH, and CLOSE statements at the end of your program. Of course nothing could be that simple, and this isn't that simple either. The other thing that has to happen is that each ODS destination that you wish to be a part of the package needs to be told about the package. That is simple, just add the option 'package' to your ODS destination statement. Here is a simple example:

```
ods package open;

ods html package;

 proc gplot data=sashelp.class;
     plot height*weight;
     by name;
 run;
 quit;

ods html close;

ods package publish archive properties(archive_name="example1.zip"
archive_path="./");

ods package close;
```

It looks fairly simple up until that ODS PACKAGE PUBLISH statement. This is how it works. The ODS PACKAGE OPEN statement creates an ODS package. It's not really doing much at this point; it's just waiting for an ODS destination to come over and shake hands. That's what happens when the ODS HTML statement gets to the package option. From then on, ODS HTML statement will tell the package about every file it creates. The package just keeps track of the files.. It doesn't do anything else until the next ODS PACKAGE statement. All destinations

that have connected with the package should be closed before the package can be published. Otherwise, some files might still be open and the package will not be able to copy them into the archive.

The ODS PACKAGE PUBLISH statement is telling the package to create an archive where the archive name is example1.zip, and the archive path is the current working directory. Both of these attributes are required. At this point, the ODS package will create an archive from the list of files it has created because the ODS HTML statement told it what files were being created.

Because the ODS package is nothing more than a list of files, it is possible to publish that package as many times as you want. But once the ODS package is closed, there is nothing else to do. The package is gone, along with the list of files that it was keeping.

SAS INTEGRATION TECHNOLOGIES, SAS PACKAGES, AND ZIP FILES

When the ODS package goes to publish the package, it looks to see if SAS Integration Technologies is available. If it is, then the Publishing Framework is used, and the PUBLISH statement creates a SAS package, which is actually a zip file with a metadata file inside. If SAS Integration Technologies is not available, then the ODS package creates a zip file using the zip file functionality that is built in to Base SAS.

To make things complicated, there is more to it than just that. If SAS Integration Technologies is available, there are other transports available besides the ability to create zip archives. There are also WebDav, e-mail, subscribers, and queues.

The properties for each of these transports are as documented in the SAS Publishing Framework documentation, which can be found here: http://support.sas.com/rnd/itech/doc9/dev_guide/app/pkgintf/pkg_publ.html. The purpose of this paper is to explain ODS packages. The Publishing Framework within SAS Integration Technologies is outside the scope of this paper.

It is important to know that ODS packages will work with or without a SAS Integration Technologies license, but without SAS Integration Technologies, the only thing that ODS packages can do is create a zip file. The other transports can be specified, but a warning will be issued and an archive will be created instead.

LOOKING INSIDE

This could be really easy and straightforward, and it is if you are willing to forget one of the two ways to look inside the archive /SAS Package that you just created with ODS packages. The easy and straightforward answer is to use unzip. On UNIX, I use unzip -t example1.zip to see the contents of the zip or package file. WinZip also works, with the usual frustrations of getting the extension to match up so double-clicking on it does the right thing.

The second way to read a SAS package, but not a zip file, is to use PROC DOCUMENT. If you have a SAS package or a zip file, the only difference is that a SAS package is a zip file with a binary metadata file inside. PROC DOCUMENT looks for that metadata file and is not at all happy when it can't find it. At any rate, here are two ways to look inside a SAS package:

```
unzip -t example1.zip
```

```
Archive:  example1.zip
   testing: PackageMetaData       OK
   testing: gplot18.gif           OK
   testing: gplot17.gif           OK
   testing: gplot16.gif           OK
   testing: gplot15.gif           OK
   testing: gplot14.gif           OK
   testing: gplot13.gif           OK
```

testing: gplot12.gif OK
testing: gplot11.gif OK
testing: gplot10.gif OK
testing: gplot9.gif OK
testing: gplot8.gif OK
testing: gplot7.gif OK
testing: gplot6.gif OK
testing: gplot5.gif OK
testing: gplot4.gif OK
testing: gplot3.gif OK
testing: gplot2.gif OK
testing: gplot1.gif OK
testing: gplot.gif OK
testing: sashtml.htm OK
No errors detected in compressed data of example1.zip.

```
proc document name=archive;
import archive="example1.zip" to myPackage;

list/levels=all;run;

dir myPackage;
list 'sashtml.htm'n/details; run;
```

The output from PROC DOCUMENT looks like this.

Listing of: \Work.Archive\
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\myPackage#1	Dir
2	\myPackage#1\'sashtml.htm'n#1	File
3	\myPackage#1\'gplot.png'n#1	File
4	\myPackage#1\'gplot1.png'n#1	File
5	\myPackage#1\'gplot2.png'n#1	File
6	\myPackage#1\'gplot3.png'n#1	File
7	\myPackage#1\'gplot4.png'n#1	File
8	\myPackage#1\'gplot5.png'n#1	File
9	\myPackage#1\'gplot6.png'n#1	File
10	\myPackage#1\'gplot7.png'n#1	File

```
 11 \myPackage#1\'gplot8.png'n#1                                File
 12 \myPackage#1\'gplot9.png'n#1                                File
 13 \myPackage#1\'gplot10.png'n#1                               File
 14 \myPackage#1\'gplot11.png'n#1                               File
 15 \myPackage#1\'gplot12.png'n#1                               File
 16 \myPackage#1\'gplot13.png'n#1                               File
 17 \myPackage#1\'gplot14.png'n#1                               File
 18 \myPackage#1\'gplot15.png'n#1                               File
 19 \myPackage#1\'gplot16.png'n#1                               File
 20 \myPackage#1\'gplot17.png'n#1                               File
 21 \myPackage#1\'gplot18.png'n#1                               File

Listing of: \Work.Archive\myPackage#1\'sashtml.htm'n#1
Order by: Insertion
Number of levels: 1

          Size
Type      in Bytes      Created           Modified      Symbolic Link    Template
================================================================================
file          4742  28JAN2009:00:33:30  28JAN2009:00:33:30

Label                                                    Page Break
================================================================================
MARKUP, Proc, Gplot, GPLOT, Plot of Height by Weight, GPLOT1, Plot of Height by
Weight, GPLOT2, Plot of Height by Weight, GPLOT3, Plot of Height by Weight,
GPLOT4, Plot of Height by Weight, GPLOT5, Plot of Height by Weight, GPLOT6, Plot
of Height by Weight
```

CLEANING UP THE MESS

One of the nice things about packages is that they can keep your work space nice and tidy. Instead of having all of your ODS output lying around, you can organize it into zip files. When the package is closed you can tell it to clean up all of those files. After all, they are in the zip files, so there's no need to leave them lying around. The ODS package CLOSE statement takes one other argument, clear. When 'clear' is added to the CLOSE statement, the package will find and delete all the files that were automatically added by the ODS destinations. In the case of the gplot example, sashtml.html and all of those gplot.png files will be deleted and you will have to show one file, example1.zip.

If you would like to add your own files to the package before you publish you can do that with the ODS PACKAGE ADD statement.

For example, if we wanted to add the SAS program that generated the zip file we could do this:

```
ods package add file="example1.sas";
```

In fact I did that early on, when clear deleted everything, and it deleted my test program. That's when I realized that maybe clear should not be so zealous, and that is why it does not delete files that are added with the ADD statement.

USING MULTIPLE PACKAGES

It is possible to have more than one ODS package open at a time. All they need is a name. ODS will keep track of one package without a name, but only one nameless package is allowed. Package naming works very much like IDs do for the ODS destinations.

To give a name to a package just include the name in parenthesis on the ODS PACKAGE OPEN statement.

ods package(mypkg) open;

To connect a destination to the named package the same syntax is used on the ODS destination statement.

```
ods rtf package(mypkg) file="test.rtf";
```

Here is an example that shows both an unnamed and named package in use at the same time.

```
 ods package open;
ods package(mypkg1) open;


ods html package(mypkg1);
ods rtf package(mypkg1) file="test.rtf";

 proc gplot data=sashelp.class;
     plot height*weight;
     by name;
 run;
 quit;


ods _all_ close;


ods package add file='example3.sas';

ods package(mypkg1) publish archive
  properties(archive_name='rtf.zip' archive_path=".");

ods package publish archive
  properties(archive_name='html.zip' archive_path=".");

ods package(mypkg1) close clear;
ods package close clear;
```

At this point we have covered the basics of creating SAS packages and zip files using the ODS package interface. This could be enough to make you a very happy camper. But if you aren't much of a camper, and your idea of camping is a five-star hotel, then you'll want to keep reading. Roll out the red carpet because package templates can make all of this even easier.

PACKAGE TEMPLATES

Package templates are one of the easiest template definitions ODS has. In their simplest form they provide defaults for how a package should behave, what its archive name and path should be, which transport it should use, and even if the package should clear the files when it closes. Here is a package template that will help us do everything much easier:

```
proc template;

    define package packages.test;
```

```
          publish = Archive properties ( path="./" archive_name="example4.zip" );

          clear = yes;

      end;
run;
```

That looks familiar. Now, to use it all we need is to name it when the package is opened. This program is the equivalent of the first example.

```
ods package template=test open;

ods html package;

 proc gplot data=sashelp.class;
     plot height*weight;
     by name;
 run;
 quit;

ods html close;

ods package publish;

ods package close;
```

This can make using a package a lot simpler, and if you still want control, the PUBLISH statement can still take properties and what you supply will supersede any settings that are in the template. The template only provides default values. But that isn't the only thing that package templates do; package templates can also organize your zip files for you.

In addition to default arguments for the package, OPEN, CLOSE, and PUBLISH statements, a package template can have a list of paths. A path is a directory that will be created within the zip file to hold certain files or types of files.

The same logical names that are used in the ODS HTML statement can be used here in the package template. Body, Contents, Pages, Frame, Stylesheet, Code, and Data can all be used here.

The following paths will cause the stylesheet file to be placed in a subdirectory named style, while all the other output files will go into the root directory of the zip file.

```
 path './'
                files = body contents frame code data
          ;

          path 'style/'
                 files = stylesheet;
          ;
```

This can be made even simpler by the addition of a default path. This will give the same results:

```
          default_path = "./";

          path 'style/'
                 files = stylesheet;
          ;
```

Yet another way to sort files into different directories is by mime type. This is a much more flexible and useful way to control where the output files go. Here is another version that accomplishes the same thing:

```
default_path = "./";

          path 'style/'
                 mimetypes= "text/css";
          ;
```

These paths make the zip file even more organized:

```
path 'images/'
                mimetypes = "image/bmp image/gif image/jpg image/png"
          ;

          path 'drawing/'
                mimetypes = "image/svg+xml"
          ;

          path 'rtf/'
                mimetypes = "text/rtf text/richtext"
          ;
```

Now we have a nice package template that can keep everything nicely organized and makes using packages as easy as pie. The only bit of cleverness comes in when we want to reference the images and the stylesheet file. When the zip file is unzipped, the path to the images and stylesheet might not be what it once was. There are a couple of ways to fool the html into doing the right thing, such as writing the files to a similar physical location or putting a URL on them.

```
proc template;

    define package packages.test;

          publish = Archive properties ( path="./" archive_name="example4.zip" );

          clear = yes;

          default_path = "./";

          path 'style/'
                mimetypes= "text/css";
          ;

          path 'images/'
                mimetypes = "image/bmp image/gif image/jpg image/png"
          ;

          path 'drawing/'
                mimetypes = "image/svg+xml"
          ;

          path 'rtf/'
                mimetypes = "text/rtf text/richtext"
          ;

     end;
run;


ods package template=test open;

ods html package file="example4.html" gpath="images/" stylesheet="example4.css"
(url="style/example4.css";
ods rtf package;

 proc gplot data=sashelp.class;
     plot height*weight;
     by name;
 run;
 quit;

ods html close;

ods package publish;

ods package close;
```

The resulting zip file looks like this:

```
unzip -t example4.zip

Archive:  example4.zip
    testing: PackageMetaData          OK
    testing: images/gplot18.png       OK
    testing: images/gplot17.png       OK
    testing: images/gplot16.png       OK
    testing: images/gplot15.png       OK
    testing: images/gplot14.png       OK
    testing: images/gplot13.png       OK
    testing: images/gplot12.png       OK
    testing: images/gplot11.png       OK
    testing: images/gplot10.png       OK
    testing: images/gplot9.png        OK
    testing: images/gplot8.png        OK
    testing: images/gplot7.png        OK
    testing: images/gplot6.png        OK
    testing: images/gplot5.png        OK
    testing: images/gplot4.png        OK
    testing: images/gplot3.png        OK
    testing: images/gplot2.png        OK
    testing: images/gplot1.png        OK
    testing: images/gplot.png         OK
    testing: rtf/sasrtf.rtf           OK
    testing: ./example4.html          OK
    testing: style/example4.css       OK
No errors detected in compressed data of example4.zip.
```

PACKAGES AND TAGSETS WORKING TOGETHER

There is one more layer that can make using ODS packages completely transparent. It only works with tagset destinations, but it actually combines packages with the destination in such a way that the use of packages is completely obscured. The only thing that is needed is the setting of a few attributes in the tagset. The first thing is that the tagset needs to specify a package template to use. Also, because mime types can be so important, each file that a tagset can create can now have a mime type assigned to it by the tagset. Another piece of the puzzle is how do we create a zip file with a package, without an ODS PACKAGE statement? In this case, the file specified on the ODS statement is the package file, and the tagset specifies the other files, the body file, the stylesheet file, the frame, contents, pages, data and code files. Here is a tagset which adds a few of these settings to the HTML tagset, and uses the package template from the last example. There is one minor change to the package template, the removal of the archive_name from the publish properties. It is apparently a bug that the template name supersedes the name given on the ODS statement. Removing the archive name from the template allows the ODS destination statement to provide the filename.

```
proc template;

    define package packages.test;

         publish = Archive properties ( path="./");

         clear = yes;

         default_path = "./";

         path 'style/'
                mimetypes= "text/css";
         ;

         path 'images/'
                mimetypes = "image/bmp image/gif image/jpg image/png"
         ;
```

```
          path 'drawing/'
                 mimetypes = "image/svg+xml"
          ;

          path 'rtf/'
                 mimetypes = "text/rtf text/richtext"
          ;

      end;
run;


 proc template;
     define tagset tagsets.myHTML;
        parent=tagsets.html4;

        /* specify the package template to use. */
        package=test;

        /* set the proper mimetypes for the files */
        default_mimetype = "text/html";
        stylesheet_mimetype = "text/css";

        /* specify the files that are to be created. */
        body = "test.html";
     end;
 run;
```

Using this tagset is very simple. In fact it is just like using the HTML destination; the only difference is what it creates. This program does everything the first example did, and more:

```
ods tagsets.myHTML file="HTML.zip";

 proc gplot data=sashelp.class;
     plot height*weight;
     by name;
 run;
 quit;

ods tagsets.myHTML close;
```

Looking inside the output file shows this:

```
unzip -t example5.zip
Archive:  example5.zip
    testing: PackageMetaData          OK
    testing: images/gplot18.png       OK
    testing: images/gplot17.png       OK
    testing: images/gplot16.png       OK
    testing: images/gplot15.png       OK
    testing: images/gplot14.png       OK
    testing: images/gplot13.png       OK
    testing: images/gplot12.png       OK
    testing: images/gplot11.png       OK
    testing: images/gplot10.png       OK
    testing: images/gplot9.png        OK
    testing: images/gplot8.png        OK
    testing: images/gplot7.png        OK
    testing: images/gplot6.png        OK
    testing: images/gplot5.png        OK
    testing: images/gplot4.png        OK
    testing: images/gplot3.png        OK
    testing: images/gplot2.png        OK
```

```
    testing: images/gplot1.png        OK
    testing: images/gplot.png         OK
    testing: ./test.html              OK
No errors detected in compressed data of example5.zip.
```

This last example is the basis of how some of the future ODS destinations will work. Open Document Format and the Microsoft Office Open file format are actually zip files with XML files inside them. It is not necessarily important for everyone to know that, but this sort of packaging is becoming a normal way of managing output types that are growing more and more complex and with XML files that grow at an alarming rate. Packaging all of that up into an easily managed zip file makes a lot of sense.

This paper started with a simple example using an ODS package to create a zip file from a simple ODS HTML report. It also showed how to use multiple destinations and multiple packages simultaneously. Package templates were then used to simplify the use of packages and to create more complex zip files, which kept the output organized. Finally, package templates and tagsets were combined to create a new type of ODS destination that creates a zip file as its only type of output. Using any or all of these techniques can be of great benefit when it comes to managing and distributing your automated reports.

CONTACT INFORMATION

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

Name: Eric Gebhart
Enterprise: SAS Institute
Address: SAS Campus Drive
City, State ZIP: Cary, NC, 27513
Work Phone: 919-632-1742
E-mail: Eric.Gebhart@sas.com
Web: http;//support.sas.com/rnd/base/ods/odsmarkup/