Good Documentation

Steve Simon

Good Documentation

Abstract: Every program that you write should have a brief documentation header at the very top of the program. The only exception is short throw-away programs that you run once and then delete. This presentation will review some of the elements that you might want to include in your documentation header.

- This presentation was written on 2019-06-05 and was last modified on 2020-06-26.

I want to talk briefly about good documentation practices. In particular, I want to talk about the documentation header, the documentation that you would place at the very top of any program you write. You should always include a documention header for any program you write. It doesn't matter if you are the only one who will ever see your code, because you yourself may end up reviewing your programs months or years after you originally wrote it. The documentation header will help you remember why you wrote the program and how to get it running again.

Documentation for THIS program

The program was written by Steve Simon. The creation and last modification dates appear on the previous slide. The program and any output produced by this program are placed in the public domain (no copyright). If you use this program, a brief acknowledgement of the author is appreciated, but not required.

This PowerPoint presentation was itself produced by a program, so it is only fair that I include some documentation. I cheated a bit because this documentation was not placed at the very top of the program. It is not intended to be a model for your program, but more as an effort to keep me from feeling too guilty about failing to practice what I preach.

Documentation for THIS program

- This program is written in R Markdown. It was run using R version 4.0.0 (2020-04-24). It requires the use of the knitr package. It produces a PowerPoint file, including speaker notes that you can use to talk about good documentation practices.
  - Why R Markdown? See http://blog.pmean.com/powerpoint-with-r-markdown/

Here is some additional information in my documentation header. It is a bit unusual to use R Markdown to produce a PowerPoint presentation. If you are curious about this, please refer to the entry on my blog listed at the bottom of this slide.

Items that you MIGHT include in a documentation header

- Who wrote the it?
- What does it do?
- Where can you find it?
- Where can you use it?
- When was it written?
- How was it written?

There are no agreed upon standards for documentation, and I don't plan on setting up my own standards. What you document and at what level of detail depends on a lot of factors. Here are some suggestions that you might consider.

Who wrote it?

- Original author
  - Contact information
- Based on...
- Current maintainer

Give yourself some credit when you write a program. If you don't intend for other people to use your program, you can stop here.

If other people are likely to use your program, most of them would like to acknowledge your work. If it is important enough, give them your email address or telephone number. That may imply a level of commitment that you are unwilling to offer. Fair enough.

You yourself should give credit if you have adapted or modified code from someone else. This is also an opportunity for the next person to learn from the same resource that you relied on.

If the person currently responsible for the maintenance of the program is different from the original author, be sure to specify that.

What does it do?

- Brief statement of purpose
- Specify the type of input
  - Give a simple working example
- Describe the output

If you yourself can remember what your program is supposed to do after it has been sitting idle on your computer for the past six months, you are a smarter person than I am. Do yourself a favor and write up at least one sentence to help jog your memory.

A single sentence may be enough for a specialized program that only works with a single specialized data set.

If your program is intended to be used with more than one data set or more than one set of input variables, describe this. Does the data set have to have a special form? If there are input variables, specify the order of input and any default values. If you can provide a simple working example, please do so. Those examples are often the most helpful part of your documentation.

Tell your audience what type of output to expect (text, graphics, etc.).

Where can you find it?

- Program name
- File path/URL
- Repository

This may seem a bit silly, but you should include the name of the file in your documentation header. There are a variety of ways that your filename can get lost. Including an explicit filename will also help you and others to give proper attribution to a different program that relies heavily on your original code.

Ideally, your program should run no mater where in the directory structure it resides, but you'd be surprised at how often relocating a program to a different folder will change how that program runs. So include the path name in addition to the program name if you suspect that this is important.

If the program is available on a particular website, definitely include the URL so that anyone who downloads the program will remember where they got it from.

Many programs are maintained on a repository. A repository offers version control, branching, and other features that make it easy for a team to collaborate on a program. A popular repository that I use is github. If you are using a repository, specify the location of the repository in the documentation header.

Where can you use it? (1/2)

- Public domain
- Strong copyright
  - No re-use
  - Contact me first
- Fair use provisions

You should let everyone know what restrictions you are placing on your computer program. At one extreme, you can place your program in the public domain. This means that anyone can use your program in any way without any restrictions.

You have to explicitly place your program in the public domain. If you fail to specify any details, you are automatically assumed to be the copyright holder. One exception is if you are doing work for an employer and your employer insists controlling copyright. Another exception is if you are doing consulting work under contract and that contract demands that you turn over copyright control of any program that you write to your client.

Even though it is not required, it is probably a good idea to assert copyright ownership. This could be as simple as a statement like "This program is copyrighted by Steve Simon in 2019." You can formally register copyright with the U.S. Copyright office or the corresponding copyright office in your own country. This provides a few additional legal protections, but probably is not worth it for a small program.

If you or your company asserts copyright, you can specify a bit more. If you don't want other people using your code put some dire warning like "Any use of this program without express written consent is strictly forbidden" or you can be a bit more open by asking someone to contact you to explore any re-use of your work.

Even if you assert copyright, others can, without your permission, still use your work under certain limited provisions. These are known as the fair use provisions of copyright law. The details are tricky and well beyond the scope of this document, but in short, fair use occurs if the use is only of a small portion of the program, if the use for educational purposes or for critical commentary, and/or if the use that does not impact the marketability of your original program. You have to factor all of these in aggregate. Just because you have an educational purpose, that does not mean that you can grab a large amount of the original copyrighted program.

Where can you use it? (2/2)

- Creative Commons, Free Software Foundation
  - No commercial use
  - No modifications/derivative uses
  - Share alike
  - Attribution required
- CITATION, LICENSE, CONTRIBUTING files

You can explicitly set the conditions under which others can use your work with a more permissive license, such as through the Creative Commons or the Free Software Foundation. You maintain copyright but allow some re-sue of your work without needing to get permission first. This is not a blanket approval of any use, however.

One of the restrictions might be that others can't use your work for a commercial purpose. This restricts re-use to education, non-profits, and other noncommercial endeavors.

You might ask that the program only be used without modifications. This forbids the development of derivative works, works that are adapted or changed in any way.

If others use and modify your work, the “share alike” provision insures that these derivative works cannot have any stronger restrictions than those that you placed on your work. If you give away your program for free, the derivative work has to be given away for free.

You can demand attribution. Anyone who uses your work give you a proper acknowledgement. You should define what a proper acknowledgement should be. It could just be a mention of your name, or your name and your website. If your program is part of a publication, you can ask that any use should also provide a proper bibliographic citation of that publication.

If there are a group of programs in one folder or repository, often the copyright details are included in a file called LICENSE. If you have a preferred format for citation of your work, this would be included in a file called CITATION. Some programmers want to encourage collaboration, and include a CONTRIBUTING file that documents the type of help that they want and the process by which improvements can be contributed.

When you use these special files, you can just reference them briefly in the documentation header. Just include a single sentence like “Information about copyright restrictions can be found in the LICENSE file.”

When was it written?

- Creation date
- Last modification date
- Version history

A program creation date can sometimes help provide context. It could be something like “That was when I had just finished the short course on macros.” If it is an old date, it could warn you that the program might be using obsolete features.

The last modification date also ocmes in handy. Compare that date to various email requests. If the date of the email is later than the last modfication date, you have a rock solid guarantee that you have not incorporated the changes suggested in that email. The converse, of course, may not be true.

You might be saying to yourself "There is no need for a modification date in the documentation header because you can always find the date using a directory command or file manager program. True enough, but there are lots of things that can happen that can change the date of a file even if it has not been modified.

The last modification date can also allow you to sync program output with your program code. A discrepancy in one direction would indicate that your output is based on the wrong version and a discrepancy in the other direction indicates that the most current version of your program is found somewhere else.

Some programs have version numbers and list the version history. This is often overkill.

Version control software will track the creation date, all modification dates, and even what modifications were made when. So don’t bother with dates in your documentation header if you are using version control. Just state the version control software you are using and where someone could find your repository.

How was it written?

- Programming language
  - Version
- Dependencies

Code for a program like SAS should be easy for you to recognize, but you’d be surprised how many potential users of your code would not know this right away. You would think that failure to recognize what language a program is written in would be a factor that disqualifies someone from using your program, but please be generous. A single line near the top of your program would save someone a lot of uncertainty.

Version numbers for your software may or may not be important. Definitely include a version number if you are using features that were only recently added to the software. Most research publications require documention of both the software itself and the version number.

If your program has any dependencies, such as special libraries or macros that need to be installed, document those here as well.

The case for minimal documentation

- Creation and maintenance effort
- Internal use only
- Other places for documentation

Documentation example #1, the complete header

```
/*%add_string
    Purpose: Add a text string to each variable in a list as either a prefix or suffix.
    dependence: %num_tokens

    Required arguments:
        words - the variable list
        str - the text string to add to each variable in the &words list

    Optional arguments:
        location - whether to add the text string as a prefix or suffix [prefix|suffix, default: suffix]
        delim - the character(s) separating each variable in the &words list [default: space]

    Examples:
        %put  %add_string(a b c, _max); *produces the text a_max b_max c_max;
        %put %add_string(a b c, max_, location=prefix);     *produces the text max_a max_b max_c;
        %put %add_string(%str(a,b,c), _max, delim=%str(,)); *produces the text a_max,b_max,c_max;

    Credit:
        source code from Robert J. Morris, Text Utility Macros for Manipulating Lists of Variable Names
          (SUGI 30, 2005) www2.sas.com/proceedings/sugi30/029-30.pdf

*/
```

Figure 1: Screenshot of documentation header for Jiangtang sas program

Here is an example of a documentation header that I found on the Internet. The print is a bit small, but let me describe the individual elements.

Documentation example #1, snippet 1 of 4.

```
%add_string

Purpose: Add a text string to each variable in a list as either a prefix or suffix.

dependence: %num_tokens
```

This is a SAS macro named add_string. The author explains the purpose of the program and notes that it relies on a second macro named num_tokens.

Documentation example #1, snippet 2 of 4.

```
Required arguments:
    words - the variable list
    str - the text string to add to each variable
          in the &words list

Optional arguments:
    location - whether to add the text string as
               a prefix or suffix [prefix|suffix,
               default: suffix]
    delim - the character(s) separating each
            variable in the &words list
            [default: space]
```

This macro has two required arguments, words and str. It has two optional arguments, location and delim and the user thoughtfully provides the default if these arguments are not specified.

Documentation example #1, snippet 3 of 4.

```
Examples:
  %put %add_string(a b c, _max);
    *produces the text a_max b_max c_max;
  %put %add_string(a b c, max_, location=prefix);
    *produces the text max_a max_b max_c;
  %put %add_string(%str(a,b,c), _max, delim=%str(,));
    *produces the text a_max,b_max,c_max;
```

This is very nice. The author gives three short and simple examples and shows what the output would look like.

Documentation example #1, snippet 4 of 4.

```
Credit:
  source code from Robert J. Morris, Text
  Utility Macros for Manipulating Lists of
  Variable Names (SUGI 30, 2005)
  www2.sas.com/proceedings/sugi30/029-30.pdf
```

The author also tells us that this macro was documented as a SAS Users Group conference and gives a hyperlink.

Documentation example #1, what's missing

- Author
- Creation date
- Copyright restrictions
  - Source: Jiangtang github repository

Perhaps it is listed elsewhere, but I did not see the name of the current author anywhere in this program. I know it is not Robert J. Morris. I did not also see a creation date. Perhaps the date of the SAS presentation would serve as a proxy for the creation date. I also did not see under what provisions I might be able to use this macro myself.

You might find some of this, however, in other files in the repository listed here.

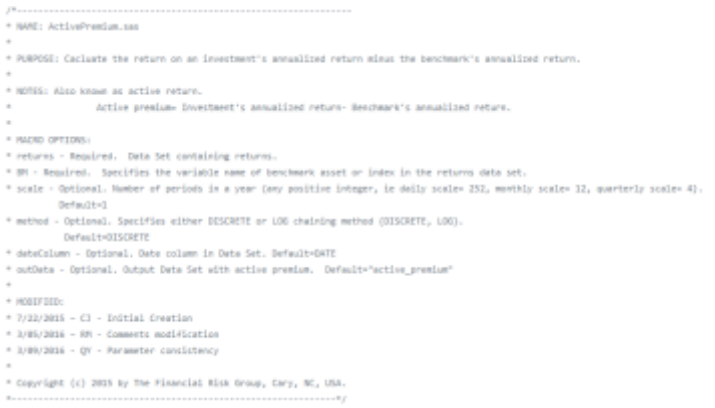

Figure 2: Screenshot of documentation header for Financial Risk Group sas program

Documentation, example #2

Documentation example #2, snippet 1 of 4.

```
/*-----------------------------------------------
* NAME: ActivePremium.sas
*
* PURPOSE: Cacluate the return on an investment's
* annualized return minus the benchmark's
* annualized return.
*
* NOTES: Also known as active return.
*        Active premium= Investment's annualized
*    return - Benchmark's annualized return.
```

Documentation example #2, snippet 2 of 4.

```
* MACRO OPTIONS:
* returns - Required.  Data Set containing
*           returns.
* BM - Required.  Specifies the variable
*      name of benchmark asset or index
*      in the returns data set.
* scale - Optional. Number of periods in a
*         year {any positive integer, ie
*         daily scale= 252, monthly scale
*         = 12, quarterly scale= 4}.
*         Default=1
```

Documentation example #2, snippet 3 of 4.

```
* method - Optional. Specifies either
*          DISCRETE or LOG chaining
*          method {DISCRETE, LOG}.
*          Default=DISCRETE
* dateColumn - Optional. Date column in
*              Data Set. Default=DATE
* outData - Optional. Output Data Set
*           with active premium.
```

```
*           Default="active_premium"
```

Documentation example #2, snippet 4 of 4.

```
* MODIFIED:
* 7/22/2015 - CJ - Initial Creation
* 3/05/2016 - RM - Comments modification
* 3/09/2016 - QY - Parameter consistency
*
* Copyright (c) 2015 by The Financial Risk Group,
* Cary, NC, USA.
*-----------------------------------------------*/
```

- Source: https://github.com/FinancialRiskGroup/SASPerformanceAnalytics

On your own (1 of 2)

On your own (2 of 2)

- Evaluate the documentation shown above, available at
  - https://www.hhs.gov/opa/performance-measures/claims-data-sas-program-instructions/index.html
- What elements did these programmers include in their documentation?
- What elements did the programmers leave out?
  - Are any of the omissions critical?

Further resources

The Pennsylvania State University. 1.6 - Guidelines for Formatting and Commenting SAS Programs. Available in html format.

```
/*
-------------------------------------------------------------------------------

   00.setup.sas
   OPA Measure Calculations
   Step 0: Define macro variables

   Summary  : Define macro variables used in subsequent programs.
              User enters environment- and data-specific info (directories,
              variable names). User specifies whether to perform measure
              calculations on postpartum women (and number of days postpartum).
              Measure calculations will be performed on all women population
              by default. User can also specify demographic variables for
              stratification of final measures.

   Data Reqs: 1) A sas data file containing women with one calendar year of claims
              2) Every record must contain at least:
                  a) patient ID
                  b) patient date of birth (or age at service date)
                  c) date of service

   Authors  : P. Hastings / phil -at- farharbor -dot- com
            : H. Monti / holly -at- farharbor -dot- com
            : Based on original code by B. Frederiksen -at- DHHS/OPA
   Version  : 2.01
   Date     : 2019-02-26
   Revisions: Now using contraceptive provision to describe measures.
              Updated OPA contact in setup helper. Updated names of revised claims code
              lookup tables. Updated LARC flag to exclude surveillance codes.
              Separated IUD and implant methods from LARC for MostMod report only.
              Updated the live birth date of service as the first date rather than
              the last date of live birth claim in the postpartum period.
              Added a flag for each contraceptive method used during the time period,
              and added these variables to final dataset. Calculating measures based on
              2018 specifications.
   History  : 2016-12-19(1.00), 2017-02-22(1.01), 2017-05-17(1.02),
              2018-04-20(1.03), 2018-08-20(1.03a)

-------------------------------------------------------------------------------

*/
```

Figure 3: Screenshot of documentation header for hhs sas program