

# Hello World! Getting Started with the SAS DS2 Language

Tricia Aanderud and Jonathan Boase, Zencos Consulting

## ABSTRACT

DS2 is an object-oriented programming language that is used for advanced data manipulation and data modeling applications. DS2 features extend the DATA step capabilities by adding variable scoping, user-defined methods, ANSI SQL data types and user-defined packages.

In this workshop, you'll write some DS2 programs to familiarize yourself with the language. You'll have some ideas of ways to upgrade or extend your existing code when you return to the office.

## INTRODUCTION

Datastep2 (DS2) is an object-oriented programming language introduced in SAS 9.4 that functions within a procedure. It is included with Base SAS so you can use it in SAS programming environments like SAS Enterprise Guide and SAS Studio. This hands-on workshop introduces you to the DS2 programming language. You will walk through five simple examples to gain an understanding of the language basics.

## WHY LEARN PROC DS2

First note that DS2 is not a replacement for the DATA step so it's quite possible that you can continue your entire SAS career without ever giving it a second thought. However, here's some reasons that you might want to add it to your toolbox.

- When working with databases, DS2 allows you to have more data types than the traditional character and numeric. You can keep the precision of BIGINT, FLOAT, DECIMAL, and so on. The extra precision is useful to those working with databases.
- Using the Methods and Packages within DS2, you can create reusable programs. DS2 features some pre-defined packages that you can incorporate with your programs as well.
- Take advantage of SAS FedSQL by embedding SQL code in your SET statement. SAS FedSQL is a SAS proprietary implementation of ANSI SQL:1999 core standard, which allows vendor-neutral access to many data sources.
- When a program becomes CPU-bound – you can use threading to spread the operation over multiple processors. This makes your program more efficient.
- Run threaded programs inside massively parallel databases like Teradata, Hadoop, and Greenplum. This does require some additional product licensing.

## EXAMPLE 1 – HELLO WORLD!

It's a tradition for an introduction to a new programming language to start with a Hello World example. In this example, you can see how a simple DATA step program was converted into a similar PROC DS2 program. The program simply prints a friendly message to the log.

| Simple DATA Step Program                           | Simple Proc DS2                                                                                                        |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <pre>data _null_;   put 'Hello World!'; run;</pre> | <pre>1 proc ds2; 2   data _null_; 3     method run();       put 'Hello World!';       end;   enddata; run; quit;</pre> |

Yes, this is a ridiculously simple example, but it illustrates three points about PROC DS2 code:

1. Enclose DS2 code with the PROC DS2 and RUN/QUIT statements.
2. This program uses a *Data program block*. The Data block is enclosed by DATA and ENDDATA statements. Program blocks must have at least one system method and can generate output data sets. (There are 3 program blocks – you will learn about the other program blocks later.)
3. The meat of this program is in the Method code block, which is where the code is executed. Methods blocks are enclosed with METHOD and END statements. This program uses a *system method* called RUN(). The RUN() method is like the DATA step in that it iterates once for each data row and has an implicit output at the end.

## UNDERSTANDING METHODS

Method blocks are the code blocks within a program. Method blocks cannot contain other method blocks and all method identifiers must be unique. There are two method code blocks: *system* and *user-defined*.

- **System Methods:** There are three system methods: INIT, RUN, and TERM. RUN allows you to execute code. As you might imagine from their names, the INIT system method initializes code and the TERM method finalizes code. These are called implicitly or explicitly.
- **User-Defined Methods:** The user-defined (or custom) method executes only when called by name. These are reusable code blocks that you created. We will do this in the next exercise.

In the following example, you can see each of the system methods in use:

| Explanation                                                                                                                                       | Using Proc DS2 System Methods                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| Start the DS2 procedure and program block                                                                                                         | <pre>proc ds2; data _null_;</pre>                                |
| Declare the variable used in the program. An Error appears in the log if you skip this step.                                                      | <pre>dcl double rows;</pre>                                      |
| The INIT system method initializes code. This code is a simple PUT statement to demonstrate that it only runs once, unlike the RUN system method. | <pre>method INIT(); put 'Hello, how many rows here?'; end;</pre> |
| The RUN system method sets the data set and then uses the declared variable to count the rows in the data set.                                    | <pre>method RUN(); set DS2LIB.CARS; rows+1; end;</pre>           |
| The TERM system method is the final step. It displays the answer to the question along with the completed row count.                              | <pre>method TERM(); put 'It has:' rows=; end;</pre>              |
| End the program block and DS2 procedure                                                                                                           | <pre>enddata; run; quit;</pre>                                   |

## CHECK THE LOG

Here's how the log appears. Notice that the INIT and TERM system methods only ran once. In a DATA step without conditional statements, it would repeat the statement for each iteration of the data set.

This exercise introduced you to the basics of a DATA program block that used the three system methods. You should understand the structure of a DS2 program and how to use the system methods.

```

28      !   proc ds2;
29              data _null_;
30              dcl double rows;
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45              enddata;
46              run;
46      !   quit;
Hello how many rows in the table?
It has rows=428
NOTE: Execution succeeded. No rows affected.

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.17 seconds
      cpu time            0.17 seconds

```

## EXAMPLE 2 – USER DEFINED OR CUSTOM METHODS

User-Defined methods allow you to create your own method. These code blocks allow you to organize your code. These methods must be called to execute.

In this example, you'll create a method that calculates your mortgage payment. Notice that there are two steps – first step you create your method. In the second step, you run your method.

| Explanation                                                                                                      | Using Proc DS2 Custom Methods                                                                                                              |
|------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Start the DS2 procedure and program block                                                                        | <pre>proc ds2;</pre>                                                                                                                       |
| Notice that we are outputting to a data set so the overwrite=yes option allows the program to run a second time. | <pre>data ds2lib.scenario_results   /overwrite=yes;</pre>                                                                                  |
| Declare the variable created in the program called monthly_payment.<br>This is a global variable.                | <pre>dcl double monthly_payment having format dollar12.2;  keep scenario_num monthly_payment;</pre>                                        |
| We only want to keep 2 variables from this calculation                                                           |                                                                                                                                            |
| Create the custom method and add the expected parameters                                                         | <pre>*STEP ONE Create a custom method to calculate the monthly payment based on input variables;</pre>                                     |
| Declare any variables used in this method. All variables are local.                                              | <pre>method calc_Pmt(   double home_price   , double down_pmt   , double APR) returns double;</pre>                                        |
| Create the calculations using the parameters and declared variables.                                             | <pre>dcl double loan_amt; dcl double mth_loan_pmt;  loan_amt=(home_price-down_pmt); mth_loan_pmt=mort(loan_amt,., (APR/100)/12,360);</pre> |
| Show which variable is returned from the method.                                                                 | <pre>return mth_loan_pmt; end;</pre>                                                                                                       |
| Using a RUN method, call the calc_Pmt custom method.                                                             | <pre>*STEP TWO Add the data to calculate the payment using method;</pre>                                                                   |
| Assign the returned variable to the montly_payment variable you created above.                                   | <pre>method run();   set ds2lib.MORTGAGE_SCENARIOS;   monthly_payment=     calc_Pmt(home_price,down_pmt,APR);   output; end;</pre>         |
| End the program block and DS2 procedure                                                                          | <pre>enddata; run; quit;</pre>                                                                                                             |

## EXAMPLE 3 – USING PACKAGES

The second code block type, Packages, are collections of user-defined methods that are stored on disk and can be shared and reused in other DS2 programs. There are two steps for this example: we create the package containing the custom method from Exercise 2 and then we execute that method from the package.

For another great package example, [see this blog post from Maria Nicholson](#) where she creates a custom package for product warranty and refund calculations.

### STEP ONE: CREATE THE PACKAGE

| Explanation                                                             | Creating DS2 Packages                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Start the DS2 procedure                                                 | <code>proc ds2;</code>                                                                                                                                                                                                                                                                                             |
| Create a package block that is saved to the DS2LIB library.             | <code>package ds2lib.my_package /overwrite=yes;</code>                                                                                                                                                                                                                                                             |
| This is the method that you wrote in the last exercise. Let's reuse it! | <pre> *Reuse your custom method from Example 2.; method calc_Pmt(   double home_price, double down_pmt, double APR)   returns double;    dcl double loan_amt;   dcl double mth_loan_pmt;    loan_amt=(home_price-down_pmt);   mth_loan_pmt=mort(loan_amt,..,(APR/100)/12,360);    return mth_loan_pmt; end; </pre> |
| End the package block and DS2 procedure                                 | <code>endpackage</code><br><code>run; quit;</code>                                                                                                                                                                                                                                                                 |

### STEP TWO: CALL THE PACKAGE

| Explanation                                                                                          | Calling Custom Methods from DS2 Packages                                                                                                 |
|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Start the DS2 procedure and data program block                                                       | <code>proc ds2;</code><br><br><code>data ds2lib.package_results;</code>                                                                  |
| Declare an instance of my_package named package_object()                                             | <pre> dcl package ds2lib.my_package   package_object(); dcl double monthly_payment having format dollar12.2; </pre>                      |
| When you call calc_Pmt method, use the object.method() notation used in object oriented programming. | <pre> method run(); set ds2lib.mortgage_scenarios; monthly_payment=package_object.calc_Pmt(home_price,down_pmt,APR); output; end; </pre> |
| Otherwise the code looks identical to the RUN method.                                                |                                                                                                                                          |
| End the package block and DS2 procedure                                                              | <code>enddata;</code><br><code>run; quit;</code>                                                                                         |

## EXAMPLE 4 – PARALLEL PROCESING WITH THREADS

The third program block in DS2, Thread, can be useful if you have a bottleneck at the CPU processing level. Threading takes advantage of idle CPU resources by allowing multiple versions of your code to run against your data simultaneously across multiple CPUs, causing your program to complete in less real time.

In this example, we will run the package from Exercise 3 as a threaded program. This is also a two-step procedure where you create the thread in the first step and then call it in the second step.

### STEP ONE: CREATE THE THREAD

| Explanation                                                       | Creating DS2 Threads                                                                                                                                   |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Start the DS2 procedure                                           | <code>proc ds2;</code>                                                                                                                                 |
| Create a thread block.                                            | <code>thread ds2lib.my_thread/overwrite=yes;</code>                                                                                                    |
| Declare package and variables as before.                          | <pre> dcl package ds2lib.my_package package_object();  dcl double rowcount; dcl double monthly_payment having format dollar12.2; drop rowcount; </pre> |
| Let's reuse the method from the package we created in exercise 3. | <code>method run();</code>                                                                                                                             |
| This time, let's also count the rows.                             | <pre> set ds2lib.big_mortgage_scenarios; monthly_payment= package_object.calc_Pmt(home_price,down_pmt,APR); rowcount+1; output; end; </pre>            |
| Call a term method to show the row count for each thread.         | <pre> method term(); put '**Thread' _threadid_ 'processed' rowcount 'rows: '; end </pre>                                                               |
| End the thread block and DS2 procedure                            | <pre> endthread; run; quit; </pre>                                                                                                                     |

### STEP TWO: CALL THE THREAD

| Explanation                                                                     | Calling DS2 Threads                                                |
|---------------------------------------------------------------------------------|--------------------------------------------------------------------|
| Start the DS2 procedure and data program block                                  | <pre> proc ds2; data ds2lib.thread_results /overwrite=yes; </pre>  |
| Declare an instance of my_thread named thread_object()                          | <code>dcl thread ds2lib.my_thread thread_object();</code>          |
| When you call the thread object, use the same name you assigned in declaration. | <pre> method run();  set from thread_object threads=4; end; </pre> |
| You can set how many threads you want to use. We are using 4 in this example.   |                                                                    |
| End the package block and DS2 procedure                                         | <pre> enddata; run; quit; </pre>                                   |

## EXAMPLE 5 – COMBINING DS2 WITH SQL

Here's a trick to use when working with a database. You can insert a FedSQL command in the SET statement to query a database (or SAS datasets) for data to use in your DS2 program. This is also useful for subsetting your data, as DS2 does not have a built-in WHERE SET statement command.

In this example, we join two tables from the database to get the most current APR based on scenario.

| Explanation                                                                                                        | Combining DS2 with SQL                                                                                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Start the DS2 procedure and data program block                                                                     | <pre>proc ds2;  data ds2lib.FedSQL_results/overwrite=yes;</pre>                                                                                                                                                                                                                           |
| Declare package object and output variable.                                                                        | <pre>dcl package ds2lib.my_package package_object();  dcl double monthly_payment having format dollar12.2;</pre>                                                                                                                                                                          |
| Use the RUN method. SQL is used to join 2 tables to create a temporary input dataset from the database at runtime. | <pre>method run(); set {SELECT       t1.*       ,t2.Today_APR     FROM MyDB.big_mortgage_scenarios t1       LEFT JOIN MyDB.daily_rates t2       ON (t1.scenario_num=t2.scenario_num)     };  monthly_payment= package_object.calc_Pmt(home_price,down_pmt,today_APR ); output; end;</pre> |
| End the block and DS2 procedure                                                                                    | <pre>enddata run; quit;</pre>                                                                                                                                                                                                                                                             |

## CONCLUSION

PROC DS2 is a powerful tool for your SAS programming toolbox. It extends your ability to work with MPP databases and allows the use of object-oriented programming techniques. This paper provided an overview of the capabilities. It should be enough to get you started on your journey.

## REFERENCES

- Wilson, K. "Reasons to love PROC DS2". SAS Users Blog. Feb 2016. Available at: <http://blogs.sas.com/content/sgf/2016/02/19/reasons-love-proc-ds2/>
- Nicholson, M. "Proc DS2 – First Impressions", Zencos Blog. Jan 2016. Available at: <https://www.zencos.com/blog/proc-ds2-first-impressions/>
- Jordan, M. 2016. [Mastering the SAS DS2 Procedure: Advanced Data Wrangling Techniques](#). SAS Press, Cary NC.
- Matsey, B and Nguyen, T. The Power of DS2 Programming. SESUG 2015. Available at: <http://analytics.ncsu.edu/sesug/2015/PSA-202.pdf>

## ACKNOWLEDGMENTS

Thanks to the Zencos senior programming staff for sitting through our dry-run workshop. Also thanks to Maria Nicholson for the multiple reviews.

## RECOMMENDED READING

- SAS 9.4 DS2 Language Reference, Sixth Edition

## CONTACT INFORMATION

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

Tricia Aanderud  
Zencos Consulting  
[taanderud@zencos.com](mailto:taanderud@zencos.com)  
<http://www.zencos.com>

Jonathan Boase  
Zencos Consulting  
[jboase@zencos.com](mailto:jboase@zencos.com)  
<http://www.zencos.com>

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.