sas

Mastering the SAS® DS2 Procedure

Advanced Data Wrangling Techniques

SAS® University Edition

Mark Jordan

From *Mastering the SAS® DS2 Procedure.*
Full book available for purchase here.

Contents

From *Mastering the SAS® DS2 Procedure: Advanced Data Wrangling Techniques*, by Mark Jordan. Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

From *Mastering the SAS® DS2 Procedure.*
Full book available for purchase here.

Chapter 2: Introduction to the DS2 Language

2.1 Introduction

In this chapter, we will describe the basic components and construction of DS2 programs. Along the way, we'll note similarities and differences between DS2 data programs and traditional Base SAS DATA steps. We'll also convert an existing DATA step to a DS2 data program and execute our first DS2 program using PROC DS2.

2.2 DS2 Programming Basics

2.2.1 General Considerations

I like to describe DS2 as a next-generation language that combines the flexibility, control, and power of DATA step programming, the rich ANSI SQL data palette, and the benefits of object-based code modularity. At first glance, the DS2 language is comfortingly similar to the DATA

step. It is fundamentally a high-level imperative, procedural language that is designed for manipulating rectangular data sets and that includes features for working with arrays, hash objects, and matrices. Like the DATA step, most DS2 data program statements begin with a keyword, and all statements end with a semicolon. However, there are significant differences. Table 2.1 highlights those.

Table 2.1: DATA Step versus DS2

DATA Step	DS2
There are almost no reserved words.	All keywords are reserved words.
Data rows are processed individually and sequentially in single compute threaded.	Several data rows can be processed in parallel, using multiple concurrent compute threads.
All variables referenced in a DATA step are global in scope.	Variables referenced in a data program can be global or local in scope.
All variables referenced in a DATA step are in the program data vector (PDV) and will become part of the result set unless explicitly dropped.	Variables with local scope are not added to the PDV and are never part of the result set.
Creating reusable code with variable encapsulation requires the use of a separate procedure, PROC FCMP, which has its own syntax.	Reusable code modules with variable encapsulation are possible using standard PROC DS2 syntax in a package program.
The DATA step can consume a table produced by an SQL query as input to the SET statement.	DS2 can directly accept the result set of an SQL query as input to the SET statement.
The DATA step can process only double-precision numeric or fixed-width character data. DBMS ANSI SQL data types must be converted to one of these data types before processing can occur.	DS2 processes most ANSI SQL data types in their native format at full precision.

2.2.2 Program Structure

A quick comparison of a DATA step and the equivalent DS2 data program clearly show the languages are closely related, but that DS2 data programs are more rigidly structured:

```
data _null_;
   Message='Hello World!';
   put Message=;
run;
proc ds2;
data _null_;
   method run();
      Message='Hello World!';
      put Message=;
   end;
enddata;
run;
quit;
```

The primary structural difference is that DS2 programs are written in code blocks. In Base SAS, the DS2 language is invoked with a PROC DS2 block, which begins with a PROC DS2 statement and ends with a QUIT statement:

```
proc ds2;
   <ds2 program blocks>
quit;
```

Within a PROC DS2 block, you can define and execute three fundamental types of program blocks.

Table 2.2: DS2 Program Blocks

Program Block	Brief Description
Data	The heart of the DS2 language, data programs manipulate input data sets to produce output result sets. They can accept input from tables, thread program result sets, or SQL query result sets.
Package	Package programs create collections of variables and methods stored in SAS libraries, enabling an object-oriented approach to development. Easy and effective reuse of proven code modules can ensure standardization of important proprietary processes, decrease time required to write new programs, and improve overall code quality.

Program Block	Brief Description
Thread	Thread programs manipulate input data sets to produce output result sets that are returned to a data program. Used to simultaneously process several rows of data in parallel, threads can accept input from tables or SQL queries.

A more detailed description of each of these program blocks is provided in Section 2.2.3. Each program block is delimited by the appropriate DATA, PACKAGE, or THREAD statement and the corresponding ENDDATA, ENDPACKAGE, or ENDTHREAD statement. DS2 uses RUN group processing and requires an explicitly coded RUN statement to cause the preceding program block to execute:

```
proc ds2;
   package package_name;
      <ds2 programming statements to create the package here>
   endpackage;
   run;
   thread thread_name;
      <ds2 programming statements to create the thread here>
   endthread;
   run;
   data output_dataset_name;
      <ds2 programming statements to process data here>
   enddata;
   run;
quit;
```

Each program block consists of a combination of global declarative statements, followed by one or more uniquely named executable method blocks. In DS2, executable statements are valid only in the context of a method block. Method blocks are delimited by METHOD and END statements:

```
proc ds2;
   data output_dataset_name;
      <global declarative statements>
      method method_name(<method parameters>);
         <local variable declarations>
         <executable DS2 programming statements>
      end;
   enddata;
   run;
quit;
```

2.2.3 Program Blocks

A brief description of each of the three program blocks is provided here to help you interpret the simple programs included in this chapter. Most of this book is dedicated to the data program. Package programs are discussed in detail in Chapter 5, and thread programs in Chapter 6.

2.2.3.1 Data Programs

A DS2 data program begins with a DATA statement, ends with an ENDDATA statement, includes at least one system method definition, and can generate a result set. It is the fundamental programming tool in the DS2 language. As in a Base SAS DATA step, the DS2 data program DATA statement normally lists the name or names of the table or tables to which the result set will be written. Using the special table name _NULL_ to suppress the result set is optional. If no destination table is named in a Base SAS DATA step, SAS directs the result set to the WORK library, using an automatically generated data set name (DATA1, DATA2, and so on). A DS2 data program without a destination table name sends its results set to the Output Delivery System (ODS) for rendering as a report, much like an SQL query.

```
data;
   set crs.banks;
run;

proc ds2;
data;
   method run();
      set crs.banks;
   end;
enddata;
run;
quit;
```

The SAS log for the traditional DATA step indicates that the result set was written to a data set named DATA1 in the WORK library:

```
NOTE: There were 3 observations read from the data set CRS.BANKS.
NOTE: The data set WORK.DATA1 has 3 observations and 2 variables.
```

The output from the DS2 data program appears instead in the Results tab:

Figure 2.1: Output of the DS2 Data Program

Name	Rate
Carolina Bank and Trust	0.0318
State Savings Bank	0.0321
National Savings and Trust	0.0328

2.2.3.2 Package Programs

A DS2 package program begins with a PACKAGE statement, ends with an ENDPACKAGE statement, and generates a package as a result. DS2 packages are used to store reusable code, including user-defined methods and variables. Packages are stored in SAS libraries and look like data sets. However, the contents of the package are merely a couple of rows of clear text header information followed by more rows containing encrypted source code. Packages make creating and sharing platform-independent reusable code modules easy and secure, and they provide an excellent means for users to extend the capabilities of the DS2 language.

Packages can be used for more than just sharing user-defined methods–they are the "objects" of the DS2 programming language. Global package variables (variables declared outside the package methods) act as state variables for each instance of the package. So, each time you instantiate a package, the instance has a private set of variables that it can use to keep track of its state. Packages can also accept constructor arguments to initialize the package when it is instantiated. DS2 packages allow SAS users to easily create and reuse objects in their DS2 programs.

2.2.3.3 Thread Programs

A DS2 thread program begins with a THREAD statement, ends with an ENDTHREAD statement, and generates a thread as a result. Much like DS2 packages, threads are stored in SAS libraries as data sets and their contents consist of clear text header information followed by encrypted source code. Threads are structured much like a DS2 data program, in that they contain at least one system method definition and can include package references and user-defined methods. Once a thread is created, it can be executed from a DS2 data program using the SET FROM statement. The THREADS= option in the SET FROM statement allows several copies of the thread program to run in parallel on the SAS compute platform for easy parallel processing, with each thread returning processed observations to the data program as soon as computations are complete.

2.2.4 Methods

Methods are named code blocks within a DS2 program, delimited by a METHOD statement and an END statement. Method blocks cannot contain nested method blocks, and all method identifiers (names) must be unique within their DS2 data, package, or thread program block. There are two types of methods:

1. *system methods* execute automatically only at prescribed times in a DS2 program. They cannot be called by name.
2. *user-defined methods* execute only when called by name.

2.2.5 System Methods

There are three system methods that are included in every DS2 data program, either implicitly or explicitly. These methods provide a DS2 data program with a more structured framework than the SAS DATA step. In the Base SAS DATA step, the entire program is included in the implicit, data-driven loop. In a DS2 data program, the RUN method provides the implicit, data-driven loop that will be most familiar to the traditional DATA step programmer. The INIT and TERM methods are not included in the loop, and provide a place to execute program initialization and finalization code.

System methods execute automatically and do not accept parameters. You must explicitly define at least one of these methods into your data or thread program or the program will not execute. If you do not write explicit code for one or more system method blocks, the DS2 compiler will create an empty version of the missing system method for you at compile time. An empty method contains only the appropriate METHOD statement followed by an END statement.

2.2.5.1 The INIT Method

The INIT method executes once and only once, immediately upon commencement of program execution. It provides a standard place to execute program initialization routines. The following DATA step and DS2 data programs produce the same results, but the DS2 data program does not require any conditional logic:

DATA step:

```
data _null_;
   if _n_=1 then do;
      put 'Execution is beginning';
   end;
run;
```

DS2 data program:

```
proc ds2;
data _null_;
   method init();
      put 'Execution is beginning';
   end;
enddata;
run;
quit;
```

2.2.5.2 The RUN Method

The RUN method best emulates the performance of a traditional SAS DATA step. It begins operation as soon as the INIT method has completed execution and acts as a data-driven loop. The RUN method iterates once for every data row (observation) in the input data set. The RUN method is the only method that includes an implicit output at the END statement. This DATA step and DS2 data program produce the same results:

```
data new_data;
   if _n_=1 then do;
      put 'Execution is beginning';
   end;
   set crs.one_day;
run;
```

```
proc ds2;
data new_data;
   method init();
      put 'Execution is beginning';
   end;
   method run();
```

```
      set crs.one_day;
   end;
enddata;
run;
quit;
```

2.2.5.3 The TERM Method

The TERM method executes once, and only once, immediately after the RUN method completes execution and before the data or thread program terminates execution. It provides an appropriate place to execute program finalization code. This DATA step and DS2 data program would produce the same results, but the DATA step requires the use of the END= SET statement option, the associated automatic variable, and a conditional logic decision to accomplish what the DS2 data program does without requiring any additional resources or code:

```
data new_data;
   if _n_=1 then do;
      put 'Execution is beginning';
   end;
   set crs.one_day end=last;
   if last=1 then do;
      put 'Execution is ending;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      put 'Execution is beginning';
   end;
   method run();
      set crs.one_day;
   end;
   method term();
      put 'Execution is ending';
   end;
enddata;
run;
quit;
```

2.2.6 User-Defined Methods

In DS2, you can easily define and use your own reusable code blocks. These code blocks are called user-defined methods, and they can accept parameter values either by reference or by value. When all parameters are passed into a method by value, the values are available inside the method for use in calculations, and the method can return a single value to the calling process–much like a Base SAS function. This data program uses a user-defined method to convert temperatures from Celsius to Fahrenheit:

```
proc ds2;
/* No output DATA set. Results returned as a report (like SQL) */
data;
   dcl double DegC DegF;
   /* Method returns a value */
```

```
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         output;
      end;
   end;
enddata;
run;
quit;
```

Figure 2.2: Output of Temperature Conversion

DegC	*DegF*
0	32
15	59
30	86

If one or more parameters are passed by reference, the values are available inside the method for use in calculations, and those values can be modified by the method at the call site, much like a Base SAS call routine. In DS2, parameters passed by reference are called IN_OUT parameters. A method that has IN_OUT parameters cannot return a value, but can modify several of its IN_OUT parameters during execution.

This data program uses a user-defined method to convert temperatures from Fahrenheit to Celsius, but passes the temperature parameter in by reference:

```
proc ds2;
data;
   dcl double Tf Tc;
   /* Method modifies a value at the call site */
   method f2c(in_out double T);
   /* Fahrenheit to Celsius (Rounded) */
      T=round((T-32)*5/9);
   end;
   method init();
      do Tf=0 to 212 by 100;
         Tc=Tf;
         f2c(Tc);
         output;
      end;
   end;
enddata;
run;
quit;
```

Figure 2.3: Output of Temperature Conversion Program

Tf	*Tc*
0	-18
100	38
200	93

When calling this type of method, you must supply a variable name for IN_OUT parameter values; otherwise, constant values will result in a syntax error:

```
proc ds2;
/* No output DATA set. Results returned as a report (like SQL) */
data;
   dcl double Tf Tc;
   /* Method modifies a value at the call site */
   method f2c(in_out double T);
   /* Fahrenheit to Celsius (Rounded) */
      T=round((T-32)*5/9);
   end;
   method init();
   /* Method f2c requires a variable as a parameter */
   /* Passing in a constant causes an error         */
      f2c(37.6);
   end;
enddata;
run;

quit;
```

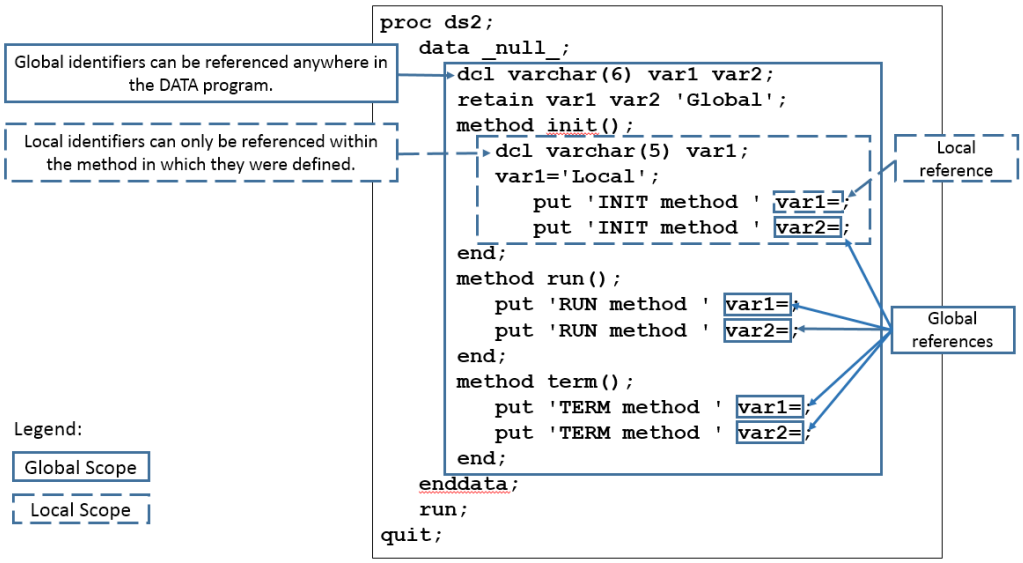
SAS Log:

```
ERROR: Compilation error.
ERROR: In call of f2c: argument 1 is 'in_out'; therefore, the
argument must be a modifiable value.
```

2.2.7 Variable Identifiers and Scope

In a DS2 program, all objects, variables, and code blocks must have identifiers (names). Within the DS2 program, an identifier's scope is either global or local, and identifiers are unique within their scope. An identifier's scope determines where in the program that identifier can be successfully referenced. Figure 2.4 shows a DS2 data program with variable identifiers of both global and local scope and indicates which values will be returned when referenced.

Figure 2.4: DS2 Data Program Variable Scope



If the program illustrated in Figure 2.4 is executed, the following results are produced in the SAS log:

```
INIT method  var1=Local
INIT method  var2=Global
RUN method  var1=Global
RUN method  var2=Global
TERM method  var1=Global
TERM method  var2=Global
```

Variable identifiers are also global in scope if the variable is undeclared or is introduced to the program via a SET statement. In DS2, an undeclared variable is created whenever a variable is first referenced in a program statement other than in a SET or DECLARE statement, such as an assignment statement. The use of undeclared variables in DS2 is discouraged; doing so will produce warnings in the SAS log. Although this might at first seem strange to a SAS DATA step programmer, if you've ever executed a long-running SAS program only to be greeted by the dreaded **`NOTE: Variable var is uninitialized`** in the SAS log, you will understand the benefit of this new behavior. Generally, that message means you've mistyped a variable name somewhere in your code, and the processing time for this run of the DATA step has been wasted.

You can control DS2 behavior for undeclared variables with the system option DS2SCOND= or SCOND= option in the PROC DS2 statement. The default is WARNING, but NONE, NOTE, and ERROR are all valid values. When writing, prototyping, or testing code, I prefer to have this option set to ERROR, so that DS2 programs containing undeclared variables will fail to compile, will not execute, and will produce this message in the SAS log:

```
ERROR: Compilation error.
ERROR: Line nn: No DECLARE for referenced variable var; creating it
as a global variable of type double.
```

In a DS2 data program, variable scope plays one additional important role: only global variables are included in the PDV, and only PDV variables are eligible to become part of the data program result set. You can explicitly remove global variables from the program result set using DROP or KEEP statements. Variables with local scope are never included in the PDV, so there is no need to drop them–they will never appear in the program result set.

For example, in the following DS2 program, the variables Total and Count are declared globally and have global scope. The variables Category and Amount are introduced via the SET statement and also have global scope. All of these variables can be referenced in both the RUN and TERM methods, and all are included in the program result set.

```
proc ds2;
   data;
      dec double Total Count;
      method run();
         set crs.one_day (keep=(Payee Amount));
         Total+Amount;
         Count+1;
      end;
      method term();
         put Total= Count=;
      end;
   enddata;
   run;
quit;
```

SAS Log:

```
Total=7230.5 Count=6
```

Figure 2.5: Report Produced by the Data Program

Obs	Total	Count	Payee	Amount
1	9.60	1	Misc	$9.60
2	19.91	2	Ice Cream	$10.31
3	37.91	3	Services	$18.00
4	230.50	4	Big Retailer	$192.59
5	3230.50	5	Misc	$3,000.00
6	7230.50	6	Misc	$4,000.00

In the next DS2 program, the variables Total and Count are locally declared in the RUN method. As a result, they have scope that is local to RUN and can be referenced only by the RUN method. When the TERM method attempts to reference variables Total and Count, they are not available in the PDV, so the DS2 compiler treats these as new, undeclared variables. Warning messages are produced in the SAS log and, because undeclared variables have global scope, Total and Count

are included in the PDV and the program result set. However, because the global versions of these variables were never assigned a value, Total and Count contain missing values in the output:

```
proc ds2;
   data;
      method run();
         declare double Total Count;
         set crs.one_day (keep=(Payee Amount));
         Total+Amount;
         Count+1;
      end;
      method term();
         put Total= Count=;
      end;
   enddata;
   run;
quit;
```

SAS Log:

```
Total=. Count=.
WARNING: Line nn: No DECLARE for referenced variable total; creating
it as a global variable of type double.
WARNING: Line nn: No DECLARE for referenced variable count; creating
it as a global variable of type double.
```

Figure 2.6: Report Produced by the Data Program Showing Missing Values

Obs	Payee	Amount	Total	Count
1	Misc	$9.60	.	.
2	Ice Cream	$10.31	.	.
3	Services	$18.00	.	.
4	Big Retailer	$192.59	.	.
5	Misc	$3,000.00	.	.
6	Misc	$4,000.00	.	.

If we delete the TERM method from the program, the only reference to the variables Total and Count are the local variables in the RUN method, so they will not be included in the PDV at all. No warnings about undeclared variables are issued in the SAS log, and the result set contains only the global variables Payee and Amount:

```
proc ds2;
   data;
      method run();
         declare double Total Count;
         set crs.one_day (keep=(Payee Amount));
         Total+Amount;
         Count+1;
      end;
```

```
   enddata;
   run;
quit;
```

Figure 2.7: Report Produced by the Data Program with Local Variables Excluded

Obs	Payee	Amount
1	Misc	$9.60
2	Ice Cream	$10.31
3	Services	$18.00
4	Big Retailer	$192.59
5	Misc	$3,000.00
6	Misc	$4,000.00

User-defined methods can accept parameters. Parameters passed by value are treated as variables with local scope within the method. For example, in the following program, the user-defined method **`fullname`** has two parameters, **`first`** and **`last`**, which act as local variables. There is also one locally declared variable, **`FinalText`**. The main data program has three globally declared variables, **`WholeName`**, **`GivenName`**, and **`Surname`**, which will be included in the PDV. The result set contains only the global variables **`WholeName`**, **`GivenName`**, and **`Surname`**.

```
proc ds2;
   data;
      declare varchar(100) WholeName;
      method fullname(varchar(50) first, varchar(50) last)
                     returns varchar(100);
         dcl varchar(100) FinalText;
         FinalText=catx(', ',last,first);
         Return FinalText;
      end;
      method run();
         set crs.customer_sample (keep=(GivenName Surname));
         WholeName=fullname(GivenName, Surname);
      end;
   enddata;
   run;
quit;
```

Figure 2.8: Report Produced by the Data Program

Income levels

WholeName	*GivenName*	*Surname*
Schmidt, William	William	Schmidt
Laverty, Daniel	Daniel	Laverty
Grayson, Sarah	Sarah	Grayson
Hastings, Eldon	Eldon	Hastings

If you have ever stored snippets of code in a SAS program file for inclusion in a traditional DATA step, you have probably experienced what I refer to as *PDV contamination*. When the included code references a variable that already existed in the main program, or when it includes new variable references, PDV values for existing variables can inadvertently be modified by the included code. Also, unwanted variables can be added to the PDV and appear in the output data set.

When reusing DS2 methods, the method's local variables never affect the PDV, a concept often referred to as *variable encapsulation*. Because method parameters and locally declared variables are local in scope, they are encapsulated in your method code and won't contaminate the PDV. In a later chapter, we will store our user-defined methods in a DS2 package for simple reuse in future programs. Because of variable encapsulation, you will never need to worry about PDV contamination when reusing properly written DS2 methods.

2.2.8 Data Program Execution

DS2 data programs are delivered to the DS2 compiler for syntax checking, compilation, and execution. At compile time, resources are reserved for the PDV, the code is compiled for execution and, if an output data set is being produced, the output data set descriptor is written. After compilation, execution begins with the INIT method code, and it is automatically followed by the RUN and TERM method code. Only system methods execute automatically; any user-defined methods must be called from the INIT, RUN, or TERM methods or else the user-defined method will not be executed.

2.3 Converting a SAS DATA Step to a DS2 Data Program

2.3.1 A Traditional SAS DATA Step

Here is a traditional SAS DATA step with three subsections, which we will convert into a DS2 data program:

```
data _null_;
   /* Section 1 */
   if _n_=1 then
      do;
         put '**********';
         put 'Starting';
```

```
         put '**********';
      end;

   /* Section 2 */
   set crs.banks end=last;
   put Bank Rate;

   /* Section 3 */
   if last then
      do;
         put '**********';
         put 'Ending';
         put '**********';
      end;
run;
```

2.3.2 Considerations

1. Section 1 consists of a DO group of statements that will be executed only when _N_=1. The automatic variable _N_ counts the number of times the DATA step has iterated, so this block will execute only one time, when the DATA step first starts execution.
2. Section 2 consists of unconditionally executed statements. These statements should execute once for every observation in the input data set. In this section, the SET statement uses the END= option to create **`last`**, a temporary variable containing an end-of-file indicator. The variable **`last`** is initialized to zero and remains 0 until the SET statement reads the last observation of the last data set listed, when it is set to 1. As an automatic variable, **`last`** is automatically flagged for DROP, and will not appear in the output data set.
3. Section 3 consists of a DO group of statements that will execute only if the variable **`last`** contains a value other than 0 or missing.

If we think about this, Section 1 code sounds like a great candidate for the INIT system method, Section 2 for the RUN method, and Section 3 for the TERM method.

2.3.3 The Equivalent DS2 Data Program

Here is a DS2 data program equivalent to the original SAS DATA step:

```
proc ds2 ;
   data _null_;

      /* Section 1 */
      method init();
         put '**********';
         put 'Starting';
         put '**********';
      end;

      /* Section 2 */
      method run();
         set crs.banks;
         put Bank Rate;
      end;
```

```
      /* Section 3 */
      method term();
         put '**********';
         put 'Ending';
         put '**********';
      end;
   enddata;
   run;
quit;
```

2.4 Review of Key Concepts

- All DS2 programs are structured in blocks.
- There are three types of DS2 program blocks: data, package, and thread.
- A program block begins with the appropriate DATA, PACKAGE, or THREAD statement, and ends with the corresponding ENDDATA, ENDPACKAGE, or ENDTHREAD statement. The remainder of the program consists of a combination of global declarative statements and method definitions. All executable statements must be part of a method block definition.
- There are three system methods: INIT, RUN, and TERM. Every data and thread program must contain explicit coding for one of these methods. System methods execute automatically and do not accept parameters.
- You can write user-defined methods, keeping the following in mind:
  - User-defined methods do not execute automatically; they execute only when called.
  - User-defined methods can accept parameters with values passed either by value or by reference (IN_OUT parameters).
  - A method that has no IN_OUT parameters can return a value, much like a SAS function.
  - Method IN_OUT parameter values can be modified at the call site, much like a SAS CALL routine.
  - User-defined methods can be stored for easy reuse in a DS2 package.
- Variables created in a DS2 program should be declared using a DECLARE (DCL) statement. Where the variable is declared determines the variable's scope.
- Variables introduced to a DS2 program via a SET statement, declared in the global program space (before method definitions begin), or which appear undeclared in the program code will have global scope. Global variables can be referenced anywhere inside the DS2 program, are part of the PDV, and are included in the program result set by default.
- Variables declared inside a METHOD block and method parameter variables are local in scope, and can be referenced only within that method. Local variables are never included in the PDV and cannot become part of the program result set.

From *Mastering the SAS® DS2 Procedure: Advanced Data Wrangling Techniques*, by Mark Jordan. Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Index

N

O

P

From *Mastering the SAS® DS2 Procedure: Advanced Data Wrangling Techniques*, by Mark Jordan. Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

About This Book

Purpose

This book will take you from complete novice to confident competence with the new SAS programming language, DS2.

Is This Book for You?

Are you comfortable with traditional Base SAS DATA step and SQL processing? Want to supercharge your data preparation? Then DS2, the new SAS programming language that integrates the power and control of the DATA step with the ease, flexibility, and rich data type palette of SQL might be just what you are looking for! DS2 plays well with multiple data sources, but is exceptionally well suited for manipulating data on massively parallel platforms (MPPs) like Hadoop, Greenplum, and Teradata.

DS2 gives you the power to accomplish tasks such as these:

- process traditional SAS data sets and DBMS data tables containing full-precision ANSI data types, all in the same DS2 data program
- easily create and share reusable code modules
- create custom objects and use object-oriented programming techniques
- safely and simply process multiple rows of data simultaneously in Base SAS
- execute DS2 code in-database on Hadoop, Greenplum, and Teradata

Prerequisites

This book assumes reader proficiency in these topics:

- DATA step programming:
  - SAS libraries
  - accessing data with a LIBNAME statement
  - reading and writing SAS data sets
  - the role of the program data vector (PDV) in DATA step processing
  - conditional processing techniques

    - arrays
    - iterative processing (DO loops)
- macro processing:
    - assigning values to macro variables
    - resolving macro variables in SAS code
    - timing of macro process execution versus execution of other SAS code
- SQL joins

Scope of This Book

This book covers the following topics:

- the types of processes for which the language was designed and the conditions indicating when DS2 would be a good choice for improving process performance
- programming statements and functions shared between traditional DATA steps and DS2 data programs
- DATA step functionality that is not available in DS2
- new DS2 functionality that is not available in the traditional DATA step
- manipulating ANSI data types at full precision in DS2
- converting traditional Base SAS DATA steps to DS2 data programs
- creating reusable code modules in DS2 and packaging them for easy reuse
- using DS2 packages to create object-oriented programs
- creating DS2 thread programs for parallel processing of data records
- executing DS2 thread programs distributed on MPP DBMS platforms

About the Examples

Software Used to Develop the Book's Content

SAS 9.4M3

The majority of the programs in this book use only Base SAS and were developed with SAS University Edition.

- The programs that read and write to DBMS also required these solutions:
    - SAS/ACCESS Interface to Teradata.
    - SAS In-Database Code Accelerator for Teradata.

All DBMS data was stored in Teradata Express 15.0.

The primary reference used in writing this book were the following sections of the online documentation for SAS 9.4:

- SAS® 9.4 DS2 Language Reference
- SAS® 9.4 Statements
- SAS® 9.4 Functions and CALL Routines
- SAS® 9.4 FedSQL Language Reference
- DS2 Programming: Essentials course notes

Example Code and Data

To access the book's example code and data, go to the author page at http://support.sas.com/publishing/authors. Select the name of the author. Then, look for the cover and select Example Code and Data.

If you are unable to access the code through the website, send email to saspress@sas.com.

SAS University Edition

If you are using SAS University Edition to access data and run your programs, check the SAS University Edition page to ensure that the software contains the product or products that you need to run the code.

Most of the programs in this book require only Base SAS to execute and will run just fine in SAS University Edition. However, a few programs read from or write to Teradata tables. These programs cannot be executed in SAS University Edition because the SAS/ACCESS Interface to Teradata is not included. A list of the SAS software products that are included with SAS University Edition is available at http://support.sas.com/software/products/university-edition/index.html.

Output and Graphics Used in This Book

All output and SAS graphics in this book were produced by executing the associated program code in SAS Studio.

Additional Help

Although this book illustrates many analyses regularly performed in businesses across industries, questions specific to your aims and issues may arise. To fully support you, SAS and SAS Press offer you the following help resources:

- For questions about topics covered in this book, contact the author through SAS Press:
  - Send questions by email to saspress@sas.com; include the book title in your correspondence.
  - Submit feedback on the author's page at http://support.sas.com/author_feedback.
- For questions about topics in or beyond the scope of this book, post queries to the relevant SAS Support Communities at https://communities.sas.com/welcome. The Base SAS programming community is the appropriate community for questions about DS2.

- SAS maintains a comprehensive website with up-to-date information. One page that is particularly useful to both the novice and the seasoned SAS user is the Knowledge Base. Search for relevant notes in the "Samples and SAS Notes" section of the Knowledge Base at http://support.sas.com/resources.
- Registered SAS users or their organizations can access SAS Customer Support at http://support.sas.com. Here you can pose specific questions to SAS Customer Support; under *Support*, click *Submit a Problem*. You must provide an email address to which replies can be sent, identify your organization, and provide a customer site number or license information. This information can be found in your SAS logs.

Keep in Touch

We look forward to hearing from you. We invite questions, comments, and concerns. If you want to contact us about a specific book, please include the book title in your correspondence.

Contact the Author through SAS Press

- By email: saspress@sas.com
- Via the web: http://support.sas.com/author_feedback

Purchase SAS Books

- Visit sas.com/store/books.
- Phone 1-800-727-0025
- Email sasbook@sas.com

Subscribe to the SAS Learning Report

Receive up-to-date information about SAS training, certification, and publications via email by subscribing to the SAS Learning Report monthly eNewsletter. Read the archives and subscribe today at http://support.sas.com/community/newsletters/training!

Publish with SAS

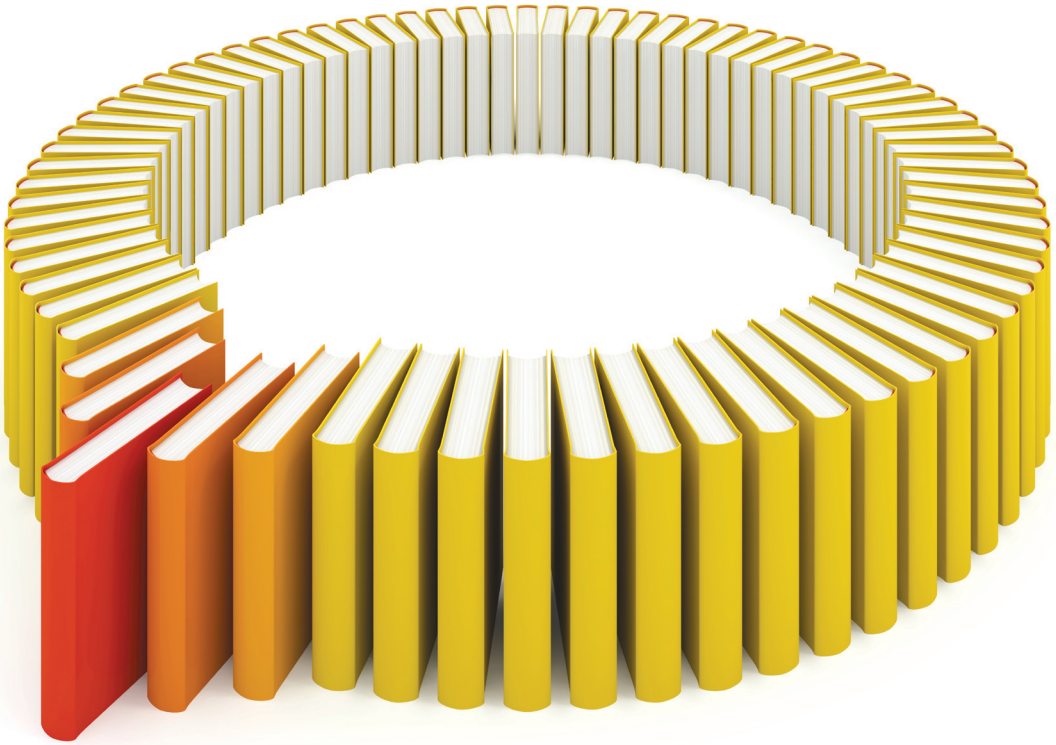
SAS is recruiting authors! Are you interested in writing a book? Visit http://support.sas.com/saspress for more information.

About The Author

A self-avowed technophile, Mark Jordan grew up in northeast Brazil as the son of Baptist missionaries. He served 20 years as a US Navy submariner, pursuing his passion for programming as a hobby. Upon retiring from the Navy, he turned his hobby into a dream job, working as a SAS programmer for 9 years in manufacturing and financial services before coming to SAS Institute in 2003. Mark teaches a broad spectrum of Foundation SAS programming classes, and has authored and co-authored the several SAS training courses.

When he isn't teaching SAS programming, Mark sporadically posts "Jedi SAS Tricks" on the SAS Training Post blog, enjoys playing with his grandchildren and great-grandchildren, hanging out at the beach and reading science fiction novels. His secret obsession is flying toys – kites, rockets, drones – though he tries (unsuccessfully) to convince his wife they are for the grandkids. Mark currently lives in Toano, VA with his wife, Lori, and their cat, the amazing Tiger Man. You can read his blog at http://go.sas.com/jedi and follow him on Twitter @SASJedi.

Learn more about this author by visiting his author page at http://support.sas.com/jordan. There you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more.



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.

sas
THE POWER TO KNOW®

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. © 2013 SAS Institute Inc. All rights reserved. S107969US.0413