

The DS2 Procedure

SAS® Programming Methods at Work

Peter Eberhardt



From *The DS2 Procedure: SAS® Programming Methods at Work*. Full book available for purchase [here](#).

Contents

| | |
|---|-------------|
| Preface..... | vii |
| About This Book | ix |
| About The Author | xiii |
| Chapter 1: Solving Bigger Problems..... | 1 |
| big data. Big data. BIG DATA..... | 1 |
| PROC DS2..... | 2 |
| Problem Space | 2 |
| Clarity..... | 2 |
| Scope | 2 |
| Modularity and Encapsulation | 3 |
| Data Types | 4 |
| Data Space | 4 |
| Embedded SQL..... | 4 |
| Threaded Data Access..... | 4 |
| In-Database Processing | 5 |
| Our First DS2 Programs | 5 |
| PROC DS2 as a DATA Step Equivalent | 5 |
| Chapter 2: Object-Oriented Programming for SAS Programmers | 15 |
| Background and Definition | 15 |
| Dog Class | 15 |
| An Example of OOP | 16 |
| Moving Forward | 20 |
| Chapter 3: Variables and Data Types | 23 |
| Variable Names..... | 25 |
| DECLARE Statement..... | 26 |
| DATA Step Conversion—Numerics | 27 |
| DATA Step Conversion—Characters..... | 28 |

| | |
|--|-----------|
| DATA Step Conversion—Date Types | 29 |
| DATA Step Conversion—Binary Types | 29 |
| DATA Step Conversion—Saving Your Table | 29 |
| More about Dates | 30 |
| Operations and Assignments | 30 |
| IF Operator | 32 |
| SELECT Operator | 33 |
| Arrays | 34 |
| Temporary Arrays | 34 |
| Variable Arrays | 34 |
| Deferred Dimensioning | 35 |
| Array Dimensions | 36 |
| Array Assignment | 37 |
| Missing Values and NULL Values | 40 |
| ANSI Mode and SAS Mode | 40 |
| Testing for Missing or NULL Values | 41 |
| Chapter 4: Scope..... | 43 |
| Scope | 43 |
| The Program Data Vector (PDV) | 44 |
| KEEP and DROP | 45 |
| Scope in Other Programming Blocks | 45 |
| Chapter 5: Methods..... | 47 |
| Defining Methods | 47 |
| System-Defined Methods | 47 |
| User-Defined Methods | 49 |
| Recursion | 59 |
| Chapter 6: Packages | 61 |
| User-Defined Packages..... | 61 |
| Instantiation | 62 |
| Using a Package Variable | 63 |
| Package as Object | 64 |
| Packages and Scope..... | 67 |
| Package as Method Parameter and Method Return Variable | 67 |
| System-Defined Packages | 68 |
| FCMP Package | 68 |

| | |
|--|------------|
| TZ Package | 71 |
| Chapter 7: An Example | 75 |
| Problem | 75 |
| The Hash Package | 78 |
| Four Steps to Creation | 78 |
| Lookup and Retrieval | 81 |
| Problem Revisited | 81 |
| Threaded Read | 82 |
| Parameterized Threaded Read | 82 |
| Chapter 8: Data Sources | 87 |
| Overview | 87 |
| Sample Tables | 88 |
| SET | 89 |
| SET Data Set Options | 89 |
| SET with No BY Statement | 89 |
| SET with BY Statement | 89 |
| SET with FedSQL as a Data Source | 92 |
| Merge | 93 |
| Merge with FedSQL as a Data Source | 95 |
| Threads | 96 |
| Thread Methods | 98 |
| Thread Parameters | 98 |
| SQLSTMT Package | 99 |
| SQLSTMT Binding | 100 |
| SQLSTMT – Returning the Rows | 100 |
| References | 101 |
| Index | 103 |



From *The DS2 Procedure: SAS® Programming Methods at Work*. Full book available for purchase [here](#).

Chapter 1: Solving Bigger Problems

| | |
|---|----------|
| big data. Big data. BIG DATA. | 1 |
| PROC DS2 | 2 |
| Problem Space | 2 |
| Clarity | 2 |
| Scope | 2 |
| Modularity and Encapsulation | 3 |
| Data Types | 4 |
| Data Space | 4 |
| Embedded SQL | 4 |
| Threaded Data Access | 4 |
| In-Database Processing | 5 |
| Our First DS2 Programs | 5 |
| PROC DS2 as a DATA Step Equivalent | 5 |

big data. Big data. BIG DATA.

It seems that not a day goes by that we do not hear a familiar chant; even the most techno-Luddites chant it—“big data. Big data. BIG DATA.” Although there is no doubt that the volumes of data are growing, big data is the smaller of our problems. Yes, data are big, but *how we handle* that big data is an even bigger problem. If the problems that we have today were the same as the ones that we had 10 or even five years ago, our bigger and better hardware could easily handle them.

Today, we have far more complex problems. Today, the mega-retailer is no longer happy with data about the profitability of a product by store. It wants to know who is buying what, when and where are they are buying it, in what combinations are they buying it, and what can be offered at check-out to increase the basket value. This is a complex problem, and bigger and better hardware does not solve it. The complex and mercurial nature of today’s problems means that we have to develop complex yet flexible solutions. How can we, as SAS developers, develop more complex and flexible solutions? One way is to use PROC DS2.

PROC DS2

The DATA step has served SAS programmers well over the years. Although it is powerful, it has not fundamentally changed since its inception. SAS has introduced a significant programming alternative to the DATA step—PROC DS2—a new procedure for your object-oriented programming environment. PROC DS2 is basically a new programming language based on the DATA step language. It is a powerful tool for advanced problem solving and advanced data manipulation. PROC DS2 makes it easier to develop complex and flexible programs for complex and flexible solutions. These programs are robust and easier to understand, which eases maintenance down the road.

Starting with SAS 9.4, PROC DS2 is part of the Base SAS package. For users in a high-performance analytics environment, there is PROC HPDS2. However, in this book, only PROC DS2 is discussed.

Problem Space

PROC DS2 deals with this more complex problem space by using many object-oriented programming (OOP) constructs. With OOP constructs, SAS programmers can develop more robust and flexible programs using the following:

- clarity
- scope
- modularity and encapsulation
- data types

Clarity

In DS2, you must be clear with each identifier that you are using. An identifier is one or more tokens or symbols that name programming language entities such as variables, labels, method names, package names, and arrays, as well as data source objects such as table names and column names. To ensure clarity, in DS2, identifiers are declared using a DECLARE statement. The DECLARE statement clearly states both the name and data type of the identifier. Before you can use an element in a DS2 program, you must tell DS2 the name and data type of the element. The benefit (besides making the programmer think more clearly about the nature of the program!) is that because the program does not compile if an invalid identifier is used, misspellings and other hard-to-detect errors can be addressed and corrected at the beginning.

Scope

In programming, scope is the area in which a variable is visible. In other words, scope lets you know where a variable can be accessed. In DS2, there are two levels of scope:

- global
- local

Global variables have global scope. That is, they are accessible from anywhere in the program. Local variables have local scope. That is, they are accessible only from within the block in which the variable

was declared and only while that block is executing. Each variable in any scope must have a unique name, but variables in different scopes can have the same name. This enables you to use consistent and meaningful variable names in different parts (or methods) of your program without overwriting values. The benefit is that you can more easily isolate worker variables (e.g., a DO loop variable, an intermediate calculation, etc.) from variables that will ultimately be written out to result sets.

Modularity and Encapsulation

A programming block is a section of a DS2 program that encapsulates variables and code. Programming blocks enable modularity and encapsulation by using modular and reusable code to perform specific tasks. This, in turn, can lead to shorter development time and the standardization of often-repeated or business-specific programming tasks. Layered programming blocks enable advanced encapsulation and abstraction of behavior, which enhances the readability and understandability of a program.

In addition, a programming block defines the scope of identifiers within that block. An identifier declared in the outermost programming block has global scope. An identifier declared in a nested block has local scope.

Table 1.1¹ lists some of the most common programming blocks, adapted from the *SAS 9.4 DS2 Language Reference Manual*.

Table 1.1: Common Programming Blocks

| Block | Delimiters | Notes |
|--------------|----------------------|--|
| Procedure | PROC DS2...QUIT | |
| Data program | DATA...ENDDATA | Variables that are declared at the top of a data program have global scope within the data program. In addition, variables that the SET statement references have global scope. Unless you explicitly drop them, global variables in the data program are included in the program data vector (PDV). Note: Global variables exist for the duration of the data program. |
| Method | METHOD...END | A method is a sub-block of a data program, package, or thread program. Method names have global scope within the enclosing programming block. Methods contain all of the executable code. PROC DS2 has three system-defined methods: INIT(), RUN(), and TERM(). Variables that are declared at the top of a method have local scope. Local variables in the method are not included in the PDV. Note: Local variables exist for the duration of the method call. |
| Package | PACKAGE...ENDPACKAGE | Variables that are declared at the top of a package have global scope within the package. Package |

4 The DS2 Procedure: SAS Programming Methods at Work

| Block | Delimiters | Notes |
|--------|--------------------|---|
| | | variables are not included in the PDV of a data program that is using an instance of the package. Note: Package variables exist for the duration of the package instance. |
| Thread | THREAD...ENDTHREAD | Variables that are declared at the top of a thread have global scope within the thread program. In addition, variables that the SET statement references have global scope. Unless you explicitly drop them, global variables in the thread program are included in the thread output set. Note: Thread variables exist for the duration of the thread program instance. They can be passed to the data program using the SET FROM statement. |

Data Types

Unlike the DATA step, which has two data types—numeric (double-precision floating-point) and fixed-length character—DS2 has many data types. This allows DS2 programs to interact better with external databases.

Data Space

No surprise here, you have to deal with a big data space. DS2 helps you by providing three major features:

- embedded SQL
- threaded data access
- in-database processing

Embedded SQL

DS2 can access data through a SET statement just like the DATA step. In addition, data can be accessed through embedded SQL statements.

Threaded Data Access

DS2 can access data through a SET statement or through embedded SQL statements. DS2 also has threaded access to the data. The effectiveness of threaded access is determined, to a large extent, by how the back-end database manages threads.

In-Database Processing

If your data is in one of the supported databases, DS2 can process inside the database. This topic is not covered in this book.

Our First DS2 Programs

It seems de rigueur to start all programming language tutorials with a “Hello World” example. Because SAS developers are focused on real world problems and getting accurate results, let’s fast-forward and say “hello” to some simple data conversions.

PROC DS2 as a DATA Step Equivalent

Before you really take advantage of DS2, let’s look at a simple DATA step that creates a table, and then let’s look at the equivalent in DS2. The example data and program creates a SAS data set with data points representing temperatures in degrees Celsius. The following DATA step creates a SAS data set named dsDegC and uses parameters defined in the macro variables. One thousand observations (&NObs) are generated between -40 (&min) and 40 (&max). To verify that the DATA step and DS2 both create the same data, the seed value (&seed) is set to be passed into a random number generator.

Parameters

```
%let NObs = 1000;
%let min  = -40;
%let max  = 40;
%let seed = 123456;
```

DATA Step

```
data dsDegC (keep=degC)
    dsAvgC (keep=avgC)
;
    label degC = 'Temp in Celsius';
    label avgC = 'Average Temp in Celsius';
    format degC F3.;
    format avgC F5.2;
    call streaminit(&seed);
    Min = &min; Max = &max;
    sum = 0;
    do obs = 1 to &NObs;
        u = rand("Uniform"); /* U[0,1] */
        degC = min + floor((1+Max-Min)*u); /* uniform integer in Min..Max */
        output dsDegC;
        sum = sum + degC;
    end;
    avgC = sum / (obs-1);
    output dsAvgC;
run;
```

DS2

```
proc DS2 scond=error; ❶
data ds2DegC_1 (keep=(degC) overwrite=YES) ❷
    ds2AvgC_1 (keep=(avgC) overwrite=YES)
;
declare integer degC having label 'Temp in Celsius' format F3.; ❸
declare double avgC having label 'Average Temp in Celsius' format F5.2;
method run(); ❹
    declare int min max obs; ❺
    declare double u sum;
    streaminit(&seed);
    Min = &min; Max = &max;
    sum = 0;
    do obs = 1 to &NObs;
        u = rand('UNIFORM');
        degC = min + floor((1+Max-Min)*u); /* uniform integer in Min..Max */
        output ds2DegC_1;
        sum = sum + degC;
    end;
    avgC = sum / (obs-1);
    output ds2AvgC_1;
end;
enddata;
run;
quit;
```

The heart of the program, with the exception of the output data set name, is the same in both the DATA step and DS2.

```
do obs = 1 to &NObs;
    u = rand("Uniform"); /* U[0,1] */
    degC = min + floor((1+Max-Min)*u); /* uniform integer in Min..Max */
    output dsDegC;
    sum = sum + degC;
end;
```

However, the DS2 program appears to be more complex, requiring more statements to get to the heart of the program.

- ❶ DS2 is a new procedure in SAS 9.4 terminated by the QUIT statement. The `scond=error` option specifies that any undeclared identifiers should cause an error. There is also a new SAS option called `DS2COND` that can be set to `ERROR`. A best practice is to set `DS2COND=ERROR` in the configuration file so that it is always set.
- ❷ Unlike the DATA step, DS2 does not automatically overwrite existing tables. The `overwrite=YES` data set option tells DS2 to drop the data set if it exists before creating it. This is standard in SQL.

- ③ All identifiers must be declared with a name and data type. The label and format are optional. The variables `degC` and `avgC` are declared outside of the method so they are global in scope. Only global variables can be written to the output tables.
- ④ All executable code must reside in a method. `method run()` is one of the system-defined DS2 methods.
- ⑤ `min`, `max`, and `obs` are integer variables. Because they are declared inside `method run()`, they are local in scope. Local variables are not written to the output tables.

The original DATA step has three distinct phases:

The first phase is initialization (setting the starting values):

```
call streaminit(&seed);
Min = &min; Max = &max;
sum = 0;
```

The second phase is processing (executing the DO loop):

```
do obs = 1 to &NObs;
  u = rand("Uniform");
  degC = min + floor((1+Max-Min)*u);
  output dsDegC;
  sum = sum + degC;
end;
```

The third phase is termination (calculating the average):

```
avgC = sum / (obs-1);
output dsAvgC;
```

In this simple DATA step, it is easy to enforce the one-time nature of the initialization and termination phases of the program. However, in many DATA steps, you must add programming logic to enforce these phases. DS2 simplifies and clarifies these phases.

Initialization, Processing, and Termination

DS2 simplifies and clarifies the three phases (initialization, processing, and termination) using three system-defined methods `INIT()`, `RUN()`, and `TERM()`. The first refinement of the DS2 program demonstrates this:

```
proc DS2 scnd=error;
data ds2DegC_2 (keep=(degC) overwrite=YES)
  ds2AvgC_2 (keep=(avgC) overwrite=YES)
  ;
declare integer degC having label 'Temp in Celsius' format F3.;
declare double avgC having label 'Average Temp in Celsius' format F5.2;
declare int min max NObs; ❶
declare double sum;
retain sum nobs;
```

8 The DS2 Procedure: SAS Programming Methods at Work

```
method init(); ❷
    streaminit(&seed);
    Min = &min; Max = &max;
    nob = &NObs;
    sum = 0;
end;

method run();
    declare double u;
    declare int obs;
    do obs = 1 to NObs;
        u = rand('UNIFORM');
        degC = min + floor((1+Max-Min)*u);
        output ds2DegC_2;
        sum = sum + degC;
    end;
end;

method term(); ❸
    avgC = sum / nob;
    output ds2AvgC_2;
end;

enddata;
run;
quit;
```

- ❶ More variables now have global scope. They are no longer just inside a method and have only local scope. All three methods use global variables.
- ❷ `method init()` is a system-defined method. It is automatically called at the start of the program. This replaces the `if _n_ = 1` block that is common in many DATA steps. This method can be used to initialize variables and invoke processing.
- ❸ `method term()` is a system-defined method. It is automatically called after `method run()` completes. It can be used to perform any wrap-up processing (in this case, calculating the average).

User-Defined Method

DS2 enables you to create your own methods to encapsulate logic. In the DS2 program, there is a formula (`min + floor((1+Max-Min)*u)`) that is used in more than one place. You can simply repeat the calculation. Or, even better, you can encapsulate the logic in a method. In this way, if you want to change the formula, you change it only once, as seen in the following example:

```
proc DS2 scnd=error;
data ds2DegC_3 (keep=(degC) overwrite=YES)
    ds2AvgC_3 (keep=(avgC) overwrite=YES)
    ;
```

```

declare integer degC having label 'Temp in Celsius' format F3.;
declare double avgC having label 'Average Temp in Celsius' format
F5.2;
declare integer min max NObs;
declare double sum;
retain sum nob;

method getRange(integer min, integer max, double u) returns integer; ❶
    return(min + floor((1+Max-Min)*u)); ❷
end;

method init();
    streaminit(&seed);
    Min = &min; Max = &max;
    nob = &NObs;
    sum = 0;
end;

method run();
    declare double u;
    declare int obs;
    do obs = 1 to nob;
        u = rand('UNIFORM');
        degC = getRange(min, max, u); ❸
        output ds2DegC_3;
        sum = sum + degC;
    end;
end;

method term();
    avgC = sum / nob;
    output ds2AvgC_3;
end;

enddata;
run;
quit;

```

- ❶ `getRange` takes three positional arguments—two integers (min and max) and double `u`. It returns an integer value.
- ❷ The return statement sends the `getRange` method's result to the caller. The formula is embedded in the return statement.
- ❸ The `getRange` method is invoked to calculate the `degC` value rather than using the formula directly.

Packages Make Methods Reusable

In the previous example, you saw how a method can be defined to replace a formula or algorithm that occurs in many places in a program. You can also define a method that can be invoked in many DS2

10 The DS2 Procedure: SAS Programming Methods at Work

programs—this is called a package. In its simplest form, a package is a collection of related methods that is saved to a table that can be accessed by other DS2 programs.

```
proc DS2 scond=error;
package range /overwrite=YES; ❶
    method getRange(integer min, integer max, double u) returns
integer;
        return(min + floor((1+Max-Min)*u));
    end;
endpackage;
run;
quit;

proc DS2 scond=error; ❷
data ds2DegC_4 (keep=(degC) overwrite=YES)
    ds2AvgC_4 (keep=(avgC) overwrite=YES)
    ;
declare integer degC having label 'Temp in Celsius' format F3.;
declare double avgC having label 'Average Temp in Celsius' format
F5.2;
declare integer min max nobs;
declare double sum;
retain sum nobs;

declare package range range(); ❸

method init();
    streaminit(&seed);
    Min = &min; Max = &max;
    nobs = &NObs;
    sum = 0;
end;

method run();
    declare double u;
    declare int obs;
    do obs = 1 to nobs;
        u = rand('UNIFORM');
        degC = range.getRange(min, max, u); ❹
        output ds2DegC_4;
        sum = sum + degC;
    end;
end;

method term();
    avgC = sum / nobs;
    output ds2AvgC_4;
end;

enddata;
run;
quit;
```

- ❶ A package is a collection of methods. Typically, the methods are logically related (for example, all of the methods are used to calculate a range of values). The package is saved to a table so that it can be used by other DS2 programs. In this example, the package is saved in the Work library. Once a package is tested and debugged, it is saved to a permanent library.
- ❷ PROC DS2 is invoked a second time to demonstrate the use of packages defined outside the PROC.
- ❸ All identifiers in a DS2 program need to be declared. In this line, an entity (variable) called `range` is declared. The `range` variable initiates an instance of a `range` package that was defined in a previous DS2 program. Although the variable `range` and the package `range` have the same name, it is not required.
- ❹ The `getRange()` method is called. It is in the `range` package referenced by the `range` variable.

The previous examples demonstrate clarity, specifically because they separate processing steps into different methods—`init()`, `term()`, and `getRange()`. Furthermore, encapsulation is used; first, computational formulas are moved into methods. Second, methods are moved into a package that can be accessed by other DS2 programs.

Accessing Data—SET statement

In the following example, the table that was created in the previous example is read and a new data set is created. Temperatures are in degrees Fahrenheit.

```
proc DS2 scond=error;
package conv /overwrite=yes; ❶

method C_to_F(integer C) returns double;
/* convert degrees fahrenheit to degrees celsius */
return 32. + (C * (9. / 5.));
end;

method F_to_C(double F) returns double;
/* convert degrees fahrenheit to degrees celsius */
return (F - 32.) * (5. / 9.);
end;

endpackage;
run;
quit;

proc DS2 scond=error;
data ds2DegF_5 (keep=(degF) overwrite=YES)
    ds2AvgF_5 (keep=(avgF) overwrite=YES)
    ;
declare double degF having label 'Temp in Fahrenheit' ' ' format F6.1;
declare double avgF having label 'Avg Temp in Fahrenheit' format F6.1;
declare double sum;
declare integer cnt;
declare package conv cnv(); ❷
retain sum cnt;
```

12 The DS2 Procedure: SAS Programming Methods at Work

```
method init();
    sum = 0;
    cnt = 0;
end;

method run();
    set ds2DegC_1;      ❸
    degF = cnv.C_to_F(degC);  ❹
    sum = sum + degF;
    cnt = cnt + 1;
    output ds2DegF_5;
end;

method term();
    avgF = sum / cnt;
    output ds2AvgF_5;
end;

enddata;
run;
quit;
```

- ❶ A new package is created with temperature-conversion methods.
- ❷ A new instance of the package is created and called `cnv`.
- ❸ The table created in the previous example is read. The `run()` method iterates over all of the rows in the table.
- ❹ The `C_to_F()` method is invoked.

Accessing Data—Threads

The last enhancement to this example shows how processing goes from sequential using the SET statement to concurrent using threads. You can use threaded processing on a single machine with multiple cores or parallel processing on back-end databases.

```
proc ds2;
    thread temps /overwrite=yes;  ❶
        method run();  ❷
            set ds2DegC_1;
        end;
    endthread;
run;
quit;

proc DS2 scnd=error;
data ds2DegF_6 (keep=(degF) overwrite=YES)
    ds2AvgF_6 (keep=(avgF) overwrite=YES)
    ;
declare double degF having label 'Temp in Fahrenheit' format F6.1;
declare double avgF having label 'Avg Temp in Fahrenheit' format F6.1;
declare double sum;
```



```

declare integer cnt;
declare package conv cnv();
declare thread temps temps; ❸
retain sum cnt;

method init();
    sum = 0;
    cnt = 0;
end;

method run();
    set from temps threads=4; ❹
    degF = cnv.C_to_F(degC);
    sum = sum + degF;
    cnt = cnt + 1;
    output ds2DegF_6;
end;

method term();
    avgF = sum / cnt;
    output ds2AvgF_6;
end;

enddata;
run;
quit;

```

- ❶ A thread is created in the Work library. The `overwrite=yes` option deletes an existing thread of the same name if one exists.
- ❷ The method `run()` iterates on the input data.
- ❸ The thread must be declared before it is used.
- ❹ DS2 launches four threads to read the data.

14 *The DS2 Procedure: SAS Programming Methods at Work*

¹ SAS Institute Inc. 2015. *SAS® 9.4 DS2 Language Reference, Fifth Edition*. Cary, NC: SAS Institute Inc.

From *The DS2 Procedure: SAS® Programming Methods at Work*, by Peter Eberhardt. Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *The DS2 Procedure: SAS® Programming Methods at Work*. Full book available for purchase [here](#).

Index

A

- accessing data 11–13, 81–86
- ACCUMULATE() method 18
- accumulator 17
- ADD() method 19, 65, 81
- adding data to Hash package 81
- amt attribute 18, 19, 65
- ANSI mode 40–41
- arrays
 - assignment of 37–40
 - deferred dimensioning 35–36
 - dimensions of 36–37
 - temporary 34
 - variable 34–35
- assignments
 - about 30–31
 - of arrays 37–40
- ATTRIB statement 23
- avgC variable 7
- avgF variable 66

B

- big data 1
- BIGINT 24
- BINARY() 24
- Binary data types 24, 29
- binding SQMSTMT 100
- bindResults() method 83
- bmi method 52
- BY GROUP 92
- by reference parameters 55–57
- BY statement
 - SET statement with 89–92
 - SET statement with no 89
- by value parameters 55

C

- calling methods 79
- CHAR() 24
- Character data types 24, 28–29
- clarity 2
- clauses

- HAVING 34
- ORDER BY 92
- select 99
- WHERE 84, 87–88, 89
- cnt attribute 18, 19, 65
- complete key 79, 80
- completing declarations 80
- constructor 64–65
- conversion, of DATA steps 27–28, 28–29
- C_to_F() method 12, 63

D

- data
 - accessing 11–13, 81–86
 - adding to Hash package 81
 - big 1
 - complete key and 79
 - defining 80
 - options for SET statement 89
 - partial key and 79
 - types of 4, 23–42, 24
- data() method 80
- data sources
 - about 87–88
 - sample tables 88–89
 - SET statement 89–93
- data space 4–5
- DATA step
 - conversion of 27–28, 28–29
 - DS2 procedure as an equivalent to 5–13
 - FCMP procedure with functions in a 69
- datasource parameter 79
- DATE 24
- Date data types 24, 29, 30
- DECIMAL() 24
- declarations, completing 80
- DECLARE PACKAGE statement 79, 80
- DECLARE statement 2, 23, 26–27, 34, 99
- DECLARE THREAD statement 96
- DECLARE TX statement 36
- declaring Hash package 79
- deferred dimensioning 35–36

- defineData() method 80
- definekey() method 80
- defining
 - data 80
 - keys 80
 - methods 47–59
- degC variable 7
- degF variable 66
- dimensioning, deferred 35–36
- dimensions, of arrays 36–37
- dot(.) notation 63
- DOUBLE/FLOAT 24
- drop command 84
- DROP= option 89
- DROP statement 45
- DS2
 - examples 75–86
 - FCMP package with functions in 70–71
 - methods, compared with FCMP procedure 71
- DS2 procedure
 - See also specific topics*
 - about 2
 - as a DATA step equivalent 5–13
- DS2COND option 24
- duplicate parameter 79

E

- Eberhardt, Peter 68
- embedded SQL 4
- encapsulation 3–4
- end= option, SET statement 49
- ENDTHREAD statement 96
- examples
 - DS2 75–86
 - Hash package 78–81
 - of object-oriented programming (OOP) 16–20
 - TZ package 72–73

F

- FCMP package 68–71
- FCMP procedure
 - compared with DS2 methods 71
 - with functions 69
 - with functions in a DATA step 69
- FedSQL
 - MERGE statement with as data source 95
 - SET statement with data source as 92–93
- fetch() method 83

- find() method 81
- firstOfMonth method 58
- FORMAT statement 23, 26
- forward reference 54–55
- functions
 - FCMP package in DS2 with 70–71
 - FCMP procedure in a DATA step with 69
 - FCMP procedure with 69
 - IFC() 32–33
 - IFN() 32–33

G

- get methods 19
- getAmt() method 65
- getAvg() method 19, 65, 66
- getCnt method 65
- GETLOCALTIME method 71
- getMax method 20
- getMin method 20
- GETOFFSET method 71
- getRange() method 9, 11
- GETTIMEZONEID method 71
- GETTIMEZONENAME method 72
- GETUTCTIME method 72

H

- HASH object. *See* Hash package
- Hash package
 - adding data 81
 - declaring 79
 - example 78–81
- hashexp parameter 79
- Hash/Hash iterator package 68
- HAVING clause 34
- HPDS2 procedure 2
- HTTP package 68

I

- identifier 2
- IF DONE statement 49
- IF operator 32–33
- IF statement 51
- IFC() function 32–33
- IFN() function 32–33
- IF/THEN statement 32–33
- IN= option 89
- in-database processing 5
- inDate parameter 58
- INFORMAT option 26
- inFromDate parameter 83

INIT() method 3, 7–8, 44, 47, 77, 84, 98
 initialization 7–8
 IN_OUT statement 55
 INPUT statement 87–88
 instantiation 62–63
 INTEGER 24
 inToDate parameter 83
 ISO 8601 date formats and functions 30

J

JSON package 68

K

KEEP= option 89
 KEEP statement 45
 keys, defining 80

L

LABEL option 26
 LENGTH statement 23
 Logger package 68
 lookup and retrieval 81

M

Matrix package 68
 MERGE statement
 about 87–88, 93–95, 100
 with FedSQL as data source 95
 method parameter, packages as 67
 method return variable, packages as 67
 methods
 about 3
 ACCUMULATE() 18
 ADD() 19, 65, 81
 bindResults() 83
 bmi 52
 calling 79
 C_to_F() 12, 63
 data() 80
 defineData() 80
 definekey() 80
 defining 47–59
 fetch() 83
 find() 81
 firstOfMonth 58
 forward reference 54–55
 get 19
 getAmt() 65
 getAvg() 19, 65, 66
 getCnt 65

GETLOCALTIME 71
 getMax 20
 getMin 20
 GETOFFSET 71
 getRange() 9, 11
 GETTIMEZONEID 71
 GETTIMEZONENAME 72
 GETUTCTIME 72
 INIT() 3, 7–8, 44, 47, 77, 84, 98
 modularity and 53–54
 overloading 18, 51–53
 packages and 9–11
 REPLACE() 81
 RUN() 3, 7–8, 12, 13, 44, 47, 49, 58, 76, 82, 98
 scope and 57–59
 setAmt 65
 setCnt 65
 system-defined 47–49
 TERM() 3, 7–8, 19, 47, 98
 thread 98
 TOISO8601 72
 TOLOCALTIME 72
 TOTIMESTAMPZ 72
 TOUTCTIME 72
 user-defined 8–9, 49–59

missing values 40–42

modularity

 about 3–4

 methods and 53–54

multidata parameter 79

N

naming variables 25–26
 NCHAR() 24
 NULL values 40–42
 Numeric() 24
 Numeric data types 24, 27–28
 NVARCHAR() 24

O

object-oriented programming (OOP)

 about 2

 background of 15–16

 defined 15

 example of 16–20

 future of 20–21

objects, packages as 64–67

OOP. *See* object-oriented programming (OOP)

operations 30–31

options

- DROP= 89
- DS2COND 24
- end= 49
- IN= 89
- INFORMAT 26
- KEEP= 89
- LABEL 26
- overwrite=yes 13
- RENAME= 89
- SCOND 24
- WHERE= 89

- ORDER BY clause 92

- ordered parameter 79

- OUTPUT statement 98

- overloading methods 51–53

- overwrite=yes option 13

P

packages

- about 61
- instantiation 62–63
- as method parameter 67
- as method return variable 67
- methods and 9–11
- as objects 64–67
- scope and 67
- system-defined 68–73
- user-defined 61–67
- using package variables 63–64

- parameterized threaded read 82–86

parameters

- datasource 79
- duplicate 79
- hashexp 79
- inDate 58
- inFromDate 83
- inToDate 83
- method 67
- multidata 79
- ordered 79
- by reference 55–57
- suminc 79
- thread 98–99
- by value 55

- partial key 79, 80

- PDV (Program Data Vector) 3, 44–45

- problem space 2–4

procedures

- DS2 2, 5–13
- FCMP 69, 71
- HPDS2 2

- processing 7–8

- Program Data Vector (PDV) 3, 44–45

- programming blocks

- common 3–4
- scope in other 45

Q

- QUIT statement 6

R

- recursion 59

- RENAME= option 89

- REPLACE() method 81

- retrieval, lookup and 81

- RETURN statement 9, 50

- returning rows 100

- rows, returning 100

- RUN() method 3, 7–8, 12, 13, 44, 47, 49, 58, 76, 82, 98

S

- SAS mode 40–41

- saving tables 29

- SCOND option 24

scope

- about 2–3, 43–44
- methods and 57–59
- in other programming blocks 45
- packages and 67
- Program Data Vector (PDV) 44–45

- Secosky, Jason 68

- select clause 99

- SELECT statement 33–34, 93

- SET FROM statement 4, 96

SET statement

- about 4, 11–12, 12–13, 82, 87–88
- Data set options 89
- end= option 49
- with FedSQL as data source 92–93
- with no BY statement 89
- with BY statement 89–92

- setAmt method 65

- setCnt method 65

- SMALLINT 24

- SQL, embedded 4

- sqlstmt package 82–86

- SQLSTMT package
 - about 68, 99
 - binding 100
 - returning the rows 100
- statements
 - BY 89–92
 - ATTRIB 23
 - DECLARE 2, 23, 26–27, 34, 99
 - DECLARE PACKAGE 79, 80
 - DECLARE THREAD 96
 - DECLARE TX 36
 - DROP 45
 - ENDTHREAD 96
 - FORMAT 23, 26
 - IF 51
 - IF DONE 49
 - IF/THEN 32–33
 - IN_OUT 55
 - INPUT 87–88
 - KEEP 45
 - LENGTH 23
 - MERGE 87–88, 93–95, 100
 - OUTPUT 98
 - QUIT 6
 - RETURN 9, 50
 - SELECT 33–34, 93
 - SET 4, 11–13, 49, 82, 87–93
 - SET FROM 4, 96
 - THREAD 96
 - VARARRAY 34–35, 36
 - VARARRAY OUT 36
- suminc parameter 79
- system-defined methods 47–49
- system-defined packages 68–73
- T**
- tables
 - sample 88–89
 - saving 29
- temporary arrays 34
- temps variable 66
- TERM() method 3, 7–8, 19, 47, 98
- termination 7–8
- testing for missing or NULL values 41–42
- THREAD statement 96
- threaded data access 4
- threaded read
 - about 82
 - parameterized 82–86
- threads
 - about 12–13, 96–98
 - methods 98
 - parameters 98–99
- TIME() 24
- Time data types 24
- TIMESTAMP() 24
- TINYINT 24
- TOISO8601 method 72
- TOLOCALTIME method 72
- TOTIMESTAMPZ method 72
- TOUTCTIME method 72
- %TSLIT() macro 31
- TZ package
 - about 68, 71–72
 - example of 72–73
- U**
- Unicode UTF-8 29
- user-defined methods 8–9, 49–59
- user-defined packages 61–67
- V**
- values, missing and NULL 40–42
- VARARRAY OUT statement 36
- VARARRAY statement 34–35, 36
- VARBINARY() 24
- VARCHAR() 24
- variable arrays 34–35
- variables
 - about 3
 - avgC 7
 - avgF 66
 - data types and 23–42
 - degC 7
 - degF 66
 - naming 25–26
 - package 63–64
 - temps 66
- W**
- WHERE clause 84, 87–88, 89
- WHERE= option 89

About This Book

Purpose

PROC DS2: SAS® Programming Methods at Work introduces a new SAS procedure. New in SAS 9.4, PROC DS2 provides programming concepts that can be used for the first time by many SAS programmers. Through examples, the book helps SAS DATA step programmers learn about PROC DS2 features such as data types, methods, packages, and threads.

Is This Book for You?

PROC DS2: SAS® Programming Methods at Work is for anyone interested in learning this new SAS procedure. This book explains the basic concepts from the ground up.

Prerequisites

PROC DS2: SAS® Programming Methods at Work assumes that the reader has at least a basic understanding of SAS DATA step programming.

Scope of This Book

PROC DS2: SAS® Programming Methods at Work covers the basic concepts of PROC DS2 so that DATA step programmers can quickly take advantage of the new procedure.

It discusses more advanced uses of PROC DS2 for high-performance procedures or in-database computing.

About the Examples

Software Used to Develop the Book's Content

PROC DS2: SAS® Programming Methods at Work was developed using the third maintenance release for SAS 9.4.

Example Code and Data

All code in this book was created using the SAS program files that are provided. No other external data are needed.

You can access the example code and data for this book by accessing the author's page at <http://support.sas.com/publishing/authors>. Click the name of the author. Then, look for the thumbnail image of this book. Select **Example Code and Data** to display the SAS program files that are included in this book.

If you are unable to access the example code and data, send email to saspress@sas.com.

SAS University Edition

If you are using SAS University Edition to access data and run your programs, then check the SAS University Edition page at <http://support.sas.com/software/products/university-edition/index.html> to ensure that you have the products that you need to run the example code.

Output and Graphics Used in This Book

In the book, all output goes to the default Results window. No special drivers are required.

Additional Help

Although this book illustrates many analyses that are regularly performed in businesses across industries, you might have specific questions. To fully support you, SAS and SAS Press offer you the following 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 beyond the scope of this book, post questions to the relevant SAS Support Communities at <https://communities.sas.com/welcome>.
- 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 & 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**. 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, 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

For a complete list of books available through SAS, visit sas.com/store/books.

- Phone: 1-800-727-0025
- Email: sasbook@sas.com

Subscribe to the SAS Training and Book Report

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

Publish with SAS

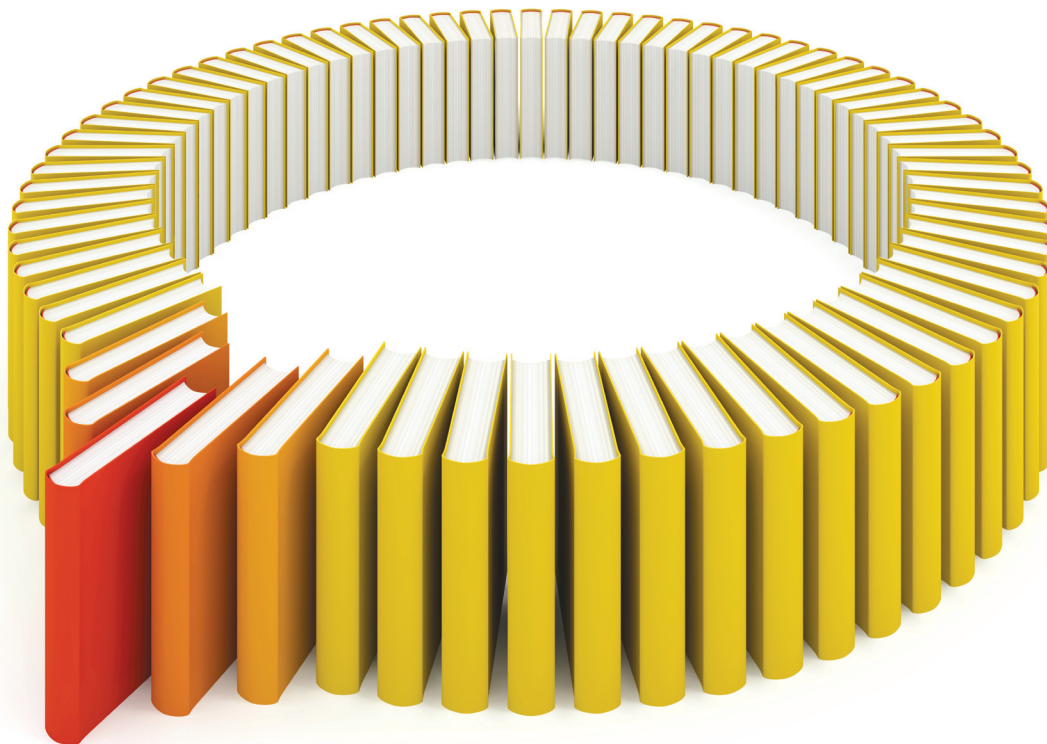
SAS is recruiting authors! Are you interested in writing a book? Visit <http://support.sas.com/saspress> for more information.

About The Author



Peter Eberhardt has been a SAS consultant since 1982; his company, Fernwood Consulting Group, Inc., is a SAS Alliance Partner. In addition to providing consulting services to the Financial and Health Care sectors, Peter has been an active contributor to the SAS user community as a speaker at user groups across Canada, the United States, the Caribbean, and China. He has also served on the Content Advisory Team for SAS Global Forum and as the Academic Chair for SESUG (2008, 2012) and PharmaSUG China (2015).

Learn more about this author by visiting his author page at <http://support.sas.com/eberhardt>. 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.


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