PharmaSUG 2017 – Paper BB09

Expansion of Opportunities in Programming: DS2 Features and Examples of Usage Object Oriented Programming in SAS®

Serhii Voievutkyi, Experis Clinical a Manpower Group Company, Kyiv, Ukraine

ABSTRACT

DS2 is a SAS programming language that allows us process data in a way that we could not do before in the traditional DATA step. Proc DS2 is a powerful instrument for advanced problem solving and data manipulation. It also includes additional data types, ANSI SQL types, programming structure elements, threaded processing and user-defined methods. DS2 language is based on object-oriented concepts, which allows programmer to use packages and methods. This presentation will demonstrate how to modify the traditional DATA step with using DS2 to resolve different issues and how object-oriented programming can be used in statistical programming.

INTRODUCTION

PROC DS2 is a powerful instrument for advanced data manipulation. DS2 allows you to create methods and packages that are based on object-oriented idea. This tool includes advanced features, which effectively combine with traditional DATA step. It enables you to do calculation with greater precision by using many new data types, to have well-structured program with reusable code modules, to save time when processing big data by avoiding required additional sorting steps for using FIRST/LAST and MERGE statements, to submit SQL code directly in the step without calling PROC SQL. DS2 procedure gives an opportunity to run more than one process concurrently, using threaded processing. These components provide the programmer with strong instruments that can be used to reduce the time for code development and program execution.

This paper gives brief overview of fundamental concepts of the DS2 language that are necessary to understand examples in this presentation. The purpose of this paper is to present examples how traditional DATA step can be modified to use DS2 statement, to show how to apply specific DS2 components and discuss lessons learned.

OVERWIEW OF DS2 CONCEPTS

DS2 is a procedural language that shares core features with the DATA step and at the same time has extended capabilities. It has different objects such as variables and scope, different datatypes, methods, packages, table I/O statements, control flow statements, and parallel programming statements. DS2 enables you to use embedded FedSQL syntax directly in DATA step to exchange data interactively between DS2 and any supported database. To use described objects as well as DATA step, all the code is placed within the framework PROC DS2 and QUIT in the program. In this paragraph, we will review several of listed components that are related to further examples. For a complete introduction to the language, please appeal to SAS DS2 Language Reference (available on the SAS website).

DS2 PROGRAMMING BLOCKS AND SCOPE

One of the important DS2 characteristic that differentiates it from DATA step is Scope concept. Scope describes where in a program a variable can be accessed. Global variables have global scope and are accessible from anywhere in the program. Local variables have local scope and are accessible only from within the program or block in which the variable was declared. Each variable in any given scope must have a unique name, but variables in different scopes can have the same name. When scopes are nested, if a variable in an outer scope has the same name as a variable in an inner scope, the variable within the outer scope is hidden by the variable within the inner scope [1].

Within DS language, there are several programming blocks:

Block	Delimiter	Scope
Data program	DATA... ENDDATA	Variables that are declared at the top of this programming block have global scope within the data program. In addition, variables that the SET statement references have global scope. Unless explicitly dropped, global variables in a data program are included in the program data vector (PDV). Note: Global variables exist for the duration of the data program.
Package	PACKAGE... ENDPACKAGE	Variables that are declared at the top of this programming block have global scope within the package. Package-scope variables are not included in the PDV of a data program that is using an instance of the package. Note: Package-scope global variables exist for the duration of the package instance.
Thread program	THREAD... ENDTHREAD	Variables that are declared at the top of this programming block have global scope within the thread program. In addition, variables that the SET statement references have global scope. Unless explicitly dropped, global variables in a thread program are included in the thread output set. Note: Thread-scope global variables exist for the duration of the thread program instance, but they can be passed to the SET FROM statement in 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 block. Variables that are declared at the top of this programming block have local scope. Local variables are not included in the PDV. Note: Local variables exist for the duration of the method call.
DO loop	DO...END	Not applicable.

Table 1. DS2 Programming Blocks and Scope [1].

DATA TYPES AND VARIABLE DECLARATION

Another powerful component of DS2 language is availability of different Data Types. Unlike Base SAS, which supports only 2 data types: numeric (data is signed, fractional, limited to 8 bytes, has approximate precision) and character (data is fixed length), DS2 supports many of the ANSI SQL data types that are native to the data sources that SAS supports.The ability to avoid data type conversions enables you to move data efficiently to/from a database or other data source. The summarized list of all data types you can find below:

Data Type	Description
BIGINT	Stores a large signed, exact whole number, with a precision of 19 digits. The range of integers is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Integer data types do not store decimal values; fractional portions are discarded.
BINARY(n)	Stores fixed-length binary data, where n is the maximum number of bytes to store. The maximum number of bytes is required to store each value regardless of the actual size of the value.
CHAR(n)	Stores a fixed-length character string, where n is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value. If char (10) is specified and the character string is only five characters long, the value is right padded with spaces.
DATE	Stores a calendar date. A date literal is specified in the format yyyy-mm-dd: a four-digit year (0001 to 9999), a two-digit month (01 to 12), and a two-digit day (01 to 31). For example, the date September 24, 1975 is specified as 1975-09-24. DS2 complies with ANSI SQL:1999 standards regarding dates. However, not all data sources support the full range of dates. For example, dates between 0001-01-01 and 1582-12-31 are not valid dates for a SAS data set or an SPD data set.

DECIMAL| NUMERIC(p,s)	Stores a signed, exact, fixed-point decimal number, with user-specified precision and scale. The precision and scale determines the position of the decimal point. The precision is the maximum number of digits that can be stored to the left and right of the decimal point, with a range of 1 to 52. The scale is the maximum number of digits that can be stored following the decimal point. Scale must be less than or equal to the precision. For example, decimal (9,2) stores decimal numbers up to nine digits, with a two-digit fixed-point fractional portion, such as 1234567.89.
DOUBLE	Stores a signed, approximate, double precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point.
FLOAT	Stores a signed, approximate, double precision, floating-point number. Data defined as FLOAT is treated the same as DOUBLE.
INTEGER	Stores a regular size signed, exact whole number, with a precision of ten digits. The range of integers is -2,147,483,648 to 2,147,483,647. Integer data types do not store decimal values; fractional portions are discarded. Note: Integer division by zero does not produce the same result on all operating systems. It is recommended that you avoid integer division by zero.
NCHAR(n)	Stores a fixed-length character string like CHAR but uses a Unicode national character set, where n is the maximum number of multibyte characters to store. Depending on the platform, Unicode characters use either two or four bytes per character and support all international characters.
NVARCHAR(n)	Stores a varying-length character string like VARCHAR but uses a Unicode national character set, where n is the maximum number of multibyte characters to store. Depending on the platform, Unicode characters use either two or four bytes per character and can support all international characters.
REAL	Stores a signed, approximate, single-precision, floating-point number.
SMALLINT	Stores a small signed, exact whole number, with a precision of five digits. The range of integers is -32,768 to 32,767. Integer data types do not store decimal values; fractional portions are discarded.
TIME(p)	Stores a time value. A time literal is specified in the format hh:mm:ss[.nnnnnnnnn]; a two-digit hour 00 to 23, a two-digit minute 00 to 59, and a two-digit second 00 to 61 (supports leap seconds), with an optional fraction value. For example, the time 6:30 a.m. is specified as 06:30:00. When supported by a data source, the p parameter specifies the seconds precision, which is an optional fraction value that is up to nine digits long.
TIMESTAMP(p)	Stores both date and time values. A timestamp literal is specified in the format yyyy-mm-dd hh:mm:ss[.nnnnnnnnn]: a four-digit year 0001 to 9999, a two-digit month 01 to 12, a two-digit day 01 to 31, a two-digit hour 00 to 23, a two-digit minute 00 to 59, and a two-digit second 00 to 61 (supports leap seconds), with an optional fraction value. For example, the date and time September 24, 1975 6:30 a.m. is specified as 1975-09-24 06:30:00. When supported by a data source, the p parameter specifies the seconds precision, which is an optional fraction value that is up to nine digits long.
TINYINT	Stores a very small signed, exact whole number, with a precision of three digits. The range of integers is -128 to 127. Integer data types do not store decimal values; fractional portions are discarded.
VARBINARY(n)	Stores varying-length binary data, where n is the maximum number of bytes to store. The maximum number of bytes is not required to store each value. If varbinary(10) is specified and the binary string uses only five bytes, only five bytes are stored in the column.
VARCHAR(n)	Stores a varying-length character string, where n is the maximum number of characters to store. The maximum number of characters is not required to store each value. If varchar(10) is specified and the character string is only five characters long, only five characters are stored in the column.

Table 2. Data Types [1].

In usual SAS DATA step data types can be assigned using LENGTH, FORMAT, or ATTRIB statements, or it can be implicitly determined based on the result of a calculation, but DS2 uses new **DECLARE** statement to define the variable. A DECLARE statement is allowed only at the top of the programming block in which it is used and has next syntax:

```
DECLARE [PRIVATE] <data-type> <variable-list> [HAVING <having-clause>];
```

The DECLARE statement can be used to specify scalar variables and temporary arrays. More than one variable and array can be specified in a DECLARE statement. In DS2, the DECLARE statement is also used for package and thread declarations.

METHODS

Methods are basic program execution units. In DS2, the method structure is used to group related program statements in one syntactically identifiable location. The group of statements in the method can then be easily invoked, or executed, multiple times. DS2 methods are similar to functions, in languages such as C, and methods in Java. In Base SAS, a user-written function that is created by the FCMP procedure is equivalent to a method. In addition, the label target of a LINK statement is functionally similar to a rudimentary method. A method defines a scoping block. Therefore, any parameters and any variable declarations in the method body are local to the method. There are two types of methods in DS2: ***system*** methods, and ***user-defined*** methods. A DS2 program can contain the following three system methods: ***INIT*** (for any initializations), ***RUN*** (to represent entire DATA Step program), and ***TERM*** (for finalization code). These methods must be defined without any parameters and without a return value [1].

PACKAGES

A DS2 package is a collection of methods and variables that can be used in DS2 programs. A DS2 package supports a set of related tasks and is designed for reuse. The package can be thought of as a library that contains the methods. There are two types of packages: ***User-defined*** packages (these packages you can use to store methods for any purpose) and DS2 ***Predefined*** packages.
Here is a list of predefined packages:

- ***FCMP*** (supports calls to FCMP functions and subroutines from within the DS2 language);
- ***Hash and hash iterator*** (enables you to quickly and efficiently store, search, and retrieve data based on unique lookup keys);
- ***HTTP*** (constructs an HTTP client to access HTTP web services);
- ***JSON*** (enables you to create and parse JSON text);
- ***Logger*** (provides a basic interface (open, write, and level query) to the SAS logging facility);
- ***Matrix*** (provides a powerful and flexible matrix programming capability);
- ***SQLSTMT*** (provides a way to pass FedSQL statements to a DBMS for execution and to access the result set returned by the DBMS);
- ***TZ*** (provides a way to process local and international time and date values) [1].

THREADED PROCESSING

Typically, DS2 code runs sequentially. That is, one process runs to completion before the next process begins. It is possible to run more than one process concurrently, using **threaded processing**. In threaded processing, each concurrently executing section of code is said to be running in a **thread**. DS2 threading works well both on a machine with multiple cores and within a massively parallel processing (MPP) database.
To enable DS2 code to run in threads:

1. Create the thread by enclosing your DS2 code between THREAD...ENDTHREAD statements.
2. Create one or more instances of the thread in a DS2 program by using a DECLARE THREAD statement.
3. Execute the thread or threads by using a SET FROM statement.

It is important that types are exactly match with a data program and a thread program. For the decimal data type, both the precision and scale must match. For character strings, the column size, the character set, and whether the string is of varying length or fixed length, must all match [1].

THREADS

Typically, one process runs to completion before the next process begins. DS2 can run more than one process concurrently. DS2 threading works well both on a machine with multiple cores and within a massively parallel processing (MPP) database. Thread processing can help you to run program faster. Here you can see how you can update traditional DATA step to use threads. 1) You need to create the thread by enclosing your DS2 code between THREAD...ENDTHREAD statements. 2) Copy core code from traditional DATA step between DATA and RUN statements to method run statement. 3) Declare derived variables. 4) Create DATA step declaring thread and using the SET FROM statement in the RUN method as in example below. In this example, the thread program starts in four threads.

Traditional DATA step:

```
data step1;
  set step0;
  bmi = (weight / (height * height)) * 703;
  /*..Derivation of other variables */
run;
```

DS2 code that uses four threads:

```
proc ds2;
  THREAD thread_name / overwrite=yes;
    declare double bmi; *other derived variables also should be declared;
    method run();
      set step0;
      bmi = (weight / (height * height)) * 703;
      /*..Derivation of other variables*/
    end;
  ENDTHREAD;
run;

  data step1 / overwrite = yes;
     dcl thread thread_name thr;
     method run();
       set from thr threads = 4;
     end;
   enddata;
   run;
quit;
```

Threads are most effective when used on data with a large number of records and data step has many calculations.

WITHOUT SORTING

Traditional DATA step required sorting for BY statement. However, it not required in PROC DS2. You can use FIRST, LAST, MERGE and SET statements without sorting. Sorting big data takes time, so program with DS2 will run faster. DS2 indicates the beginning and end of a BY group by creating two temporary variables for each BY variable: FIRST.variable and LAST.variable. The value of these variables is either 0 or 1. DS2 sets the value of FIRST.variable to 1 when it reads the first row in a BY group, and sets the value of LAST.variable to 1 when it reads the last row in a BY group. These temporary variables are available for DS2 programming but are not added to the result data set.

Traditional DATA step:

```
proc sort data = step0;
  by visit param aval;
run;
```

```
data step1;
  length anlfl $1;
  set step0;
  by visit param aval;
  if first.param then anlfl = 'Y';
  *derivation of other variables;
run;
```

DS2 code without sorting:

```
proc ds2;
  data step1 / overwrite = yes;
  declare char(1) anlfl;*other derived variables also should be declared;
  method run();
    set step0;
    by visit param aval;
    if first.param then anlfl = 'Y';
    *derivation of other variables;
   end;
   enddata;
   run;
quit;
```

DS2 code without sorting also can be used with threads:

```
proc ds2;
  thread thread_name / overwrite = yes;
    declare char(1) anlfl; *other derived variables also should be
declared;
    method run();
      set step0;
       by visit param aval;
       if first.param then anlfl = 'Y';
      *deriviation of other variables;
    end;
  endthread;
run;

  data step1 / overwrite = yes;
     dcl thread thread_name t;
     method run();
       set from t threads = 4;
     end;
   enddata;
   run;
quit;
```

Be careful to use MERGE or SET statements with two or more data sets using BY. Results in DS2 may differ from results in traditional DATA step.

SQL IN DS2

SET statement can use SQL code within PROC DS2 to read a table as in this example:

```
Set {select * from tbl};
```

Some environments might preserve the order imposed by the ORDER BY clause, but others do not. DS2 wraps the embedded SQL with additional code. In addition, the ORDER BY clause is not honored because DS2 cannot detect and produce FIRST or LAST information. It is better to use BY statement outside SQL code. In example below, DS2 creates table STEP2 variables and contains two united tables: STEP1 and derived table using SQL. Code sorts data set by two variable ID and AVAL.

```
proc ds2;
  data step2 / overwrite = yes;
  method run();
    set step1
         {select t1.*, t2.aval
         from step0a as t1 left join step0b as t2 on t1.id = t2.id
         };
         By id aval;
  end;
  enddata;
  run;
quit;
```

Another example shows how to update MERGE statement in traditional DATA step to use DS2:

Traditional DATA step:

```
proc sort data = step1a;
  by param;
run;
proc sort data = step1b;
  by param;
run;

data step2;
  merge step1a(in = ina) step1b(keep = param aval);
  by param;
  if ina;
  *derivation of other variables;
run;
```

DS2 code:

```
proc ds2;
  data step2 / overwrite=yes;
  method run();
    set {select t1.*, t2.aval
         from step1a as t1 left join ste1b as t1
         on t1.param = t2.param
         };
     *derivation of other variables;
     end;
  enddata;
  run;
quit;
```

METHODS AND PACKAGES

When you use DS2 methods and packages, you approach writing SAS programs differently than you do when programming in Base SAS. These DS2 language constructs follow a more structured programming and object-oriented approach. A method can be thought of as a module that contains a sequence of instructions to perform a specific task. DS2 methods can exist only within a data program, thread program, or package. Thus, methods enable you to break up a complex problem into smaller modules. A DS2 package bundles data and methods into a named object that can be stored and reused by other

DS2 programs. Methods can works in two ways: return the value or modify the value of the argument variable. User-defined methods that have the same name can exist in the same scope if their argument signatures are unique. That is, if only the return type differs, then the overloading is ambiguous. In an example below, you can see package for conversions with user-defined methods. This example shows that you can use different methods with the same name. One method F2C works without rounding while others work with rounding, two of them return a value and one of them modifies the value of the argument variable.

Traditional DATA step:

```
proc ds2;

package convert / overwrite = yes;

  method f2c(double f) returns double;
    /*conversion of temperature from Fahrenheit to Celsius*/
    return (f - 32) * 5 / 9;
  end;

  method in2cm(double inch) returns double;
    return inch * 2.54;
  end;

  method lb2kg(double lb) returns double;
    return lb / 2.2046;
  end;

  method f2c(double f, int r) returns double;
    return round((f - 32) * 5 / 9, 0.1 ** r);
  end;

  method f2c(double f, int r, in_out double c );
    c = round((f - 32) * 5 / 9, 0.1 ** r);
  end;

endpackage;

data step1;
  dcl package convert cnv();
  dcl double x y z having label 'Temperature in Celsius';

  method run();
    x = cnv.f2c(55);
    y = cnv.f2c(55, 3);
    cnv.f2c(55, 0, z);
  end;

enddata;
run;

quit;
```

Next example is more difficult. Here variable X will be rounded to 1 decimal place. Y – 4 decimal places and Z – 5 decimal places. The _NEW_ operator constructs an instance of the CONVERT package. THIS statement means that this variable is a global variable in the package.

```
proc ds2;
package convert/ overwrite = yes;
  dcl int rnd;

  method convert(int rnd);
    this.rnd = rnd;
  end;

  method f2c(double f) returns double;
    /*conversion of temperature from Fahrenheit to Celsius*/
     return round((f - 32) * 5 / 9,0.1**rnd);
  end;

endpackage;

data step1;
dcl package convert cnv(1) cnv2(2);
dcl double x y z having label 'Temperature in Celsius';

method run();
  x = cnv.f2c(55);
  cnv = _new_ convert(4);
  cnv2 = _new_ convert(5);
  y = cnv.f2c(55);
  z = cnv2.f2c(55);
end;

enddata;
run;
quit;
```

Next example shows predefined package HASH. In the following example, the first table program generates the data1 table. The second table program creates a hash package h with one key, k, and two data, d1 and d2. Then each row from the data1 table is read and the keys and data are added to hash package h. Finally, the data values stored in hash package h are written to the out1 table. The third table program writes the contents of the out1 table.[1]

/* Generate and output 5 rows for table data1. */

```
proc ds2;
  data data1(overwrite=yes);
    declare double k d1 d2;
    method init();
      declare int i;
      do i = 1 to 5;
        k = i; d1 = i*10; d2 = i*2; output;
      end;
    end;
  enddata;
  run;

  data _null_;
    declare double k d1 d2;
    declare package hash h(0, '', 'descending');

    /* Define key and data variables for hash h. */
    method init();
      h.defineKey('k');
```

```
        h.defineData('d1');
        h.defineData('d2');
        h.defineDone();
      end;

      /* Read rows from table data1.
       * Add key and data values from rows to hash h. */
      method run();
        set data1;
        h.add();
      end;

      /* Add additional key and data values to hash h.
       * Output hash h to table out1. */
      method term();
        k = 11; d1 = 110; d2 = 22; h.add();
        k = 12; d1 = 120; d2 = 24; h.add();
        k = 13; d1 = 130; d2 = 26; h.add();
        k = 14; d1 = 140; d2 = 28; h.add();
        h.output('out1');
      end;

    enddata;
    run;

    /* Outputs rows from table out1. */
    data finalout;
      method run();
        set out1;
      end;
    enddata;
    run;
  quit;
```

Data set FINALOUT contains records from data set DATA1 and additional records that were added in method term.

CONCLUSION

PROC DS2 is a powerful instrument than can be organized in object-oriented structure. Threads and avoid sorting allow you to run program faster. Improved performance was observed with a large number of observations (millions) and when there are many calculations within data set. On some data DS2 code worked 10 times faster. DS2 code is similar to DATA step and it is easy to update code from DATA step to DS2.

REFERENCES

[1] SAS Institute, Inc. 2013. SAS® 9.4 DS2 Language Reference. Available at http://support.sas.com/documentation/cdl/en/ds2ref/69739/PDF/default/ds2ref.pdf

CONTACT INFORMATION

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

Serhii Voievutskyi
Expiris Clinical a Manpower Group Company
Kyiv, Ukraine
+380997324971
Serhii.Voievutskyi@intego-group.com