PROC DS2: What's in it for you?

Viraj R Kumbhakarna, MUFG Union Bank N.A., San Francisco, CA

ABSTRACT

In this paper, we explore advantages of using PROC DS2 procedure over the data step programming in SAS®. DS2 is a new SAS proprietary programming language that is appropriate for advanced data manipulation. We explore use of PROC DS2 to execute queries in databases using FED SQL from within the DS2 program. Several DS2 language elements accept embedded FedSQL syntax, and the run-time generated queries can exchange data interactively between DS2 and supported database. This action enables SQL preprocessing of input tables, which effectively allows processing data from multiple tables in different databases within the same query thereby drastically reducing processing times and improving performance. We explore use of DS2 for creating tables, bulk loading tables, manipulating tables, and querying data in an efficient manner.

We explore advantages of using PROC DS2 over data step programming such as support for different data types, ANSI SQL types, programming structure elements, and benefits of using new expressions or writing one's own methods or packages available in the DS2 system. We also explore high-performance version of the DS2 procedure, PROC HPDS2, and show how one can submit DS2 language statements for execution to either a single machine running multiple threads or to a distributed computing environment, including the SAS LASR Analytic Server thereby massively reducing processing times resulting in performance improvement.

The DS2 procedure enables users to submit DS2 language statements from a Base SAS session. The procedure enables requests to be processed by the DS2 data access technology that supports a scalable, threaded, high-performance, and standards-based ways to access, manage, and share relational data. In the end, we empirically measure performance benefits of using PROC DS2 over PROC SQL for processing queries in-database by taking advantage of threaded processing in supported data databases such as Oracle

INTRODUCTION

In this paper, we explore what is the PROC DS2 procedure? What are the similarities and differences between PROC DS2 procedure and a SAS data step? When to use DS2? And other questions one may have regarding the PROC DS2 procedure.

WHAT IS DS2?

DS2 is a SAS programming language that is great for advanced data manipulation, it is included with Base SAS and shares core features with the SAS DATA step. The PROC DS2 procedure enables users to submit DS2 language statements from a Base SAS session. DS2 shares its core features with a SAS data step. However DS2 also adds variable scoping, user-defined methods, ANSI SQL data types, and user-defined packages. In addition, the procedure also enables the DS2 language to be used in large, complex applications.

The DS2 SET statement accepts embedded FedSQL syntax, and the runtime-generated queries can exchange data interactively between DS2 and any supported database. This allows SQL preprocessing of input tables, by effectively combining the power of the two different languages.

COMPARISON OF SAS DATA STEP AND PROC DS2

PROC DS2 shares its core components with a SAS data step. Although, there are also differences between a DS2 procedure and a SAS data step.

In a data step the executable code resides in the DATA and PROC step, whereas the executable code resides in the methods. There is no concept of scope in a data step, whereas in DS2, variables that are declared in methods have local scope and all others have global scope. Declaring variables is not required in a data step. The variables are created on the fly by assignment. Data type of the variable is determined by the context it is first used. All variables have global scope. In DS2, variables need to be explicitly declared using the DECLARE statement, which also determines data type and scope attribute of the variables. The data types supported in data step are mainly character and numeric. Numeric data is signed fractional, limited to bytes and has approximate precision. Character data is fixed length. In PROC DS2, almost all ANSI SQL data types are supported. Numeric types of varying sizes and precision. Character data types can be fixed or variable length. DS2 also supports ANSI date, time and timestamp data types, but can also process SAS date, time and datetime values using conversion functions. SQL language statements cannot be written in a SAS data step, they are available in PROC SQL. Whereas SQL select statements

can directly be written in and used as input for a DS2 SET statement. In addition the SQLSTMT predefined package provides a way to pass SQL statements to a DBMS for execution and to access the result set returned by DBMS.

CONSTRUCT OF DS2 PROGRAM

In this section, we will discuss the construct of a DS2 program, the syntax of a basic DS2 program, and what are the semantics of a DS2 program.

DS2 PROGRAM DEFINTION AND SYNTAX

A DS2 program is a set of DS2 statements that runs in the DS2 procedure. The getting started sample programs demonstrate that a DS2 program can serve many purposes, including but not limited to the following:

- To define and store one or more packages or threads, in permanent or temporary locations.
- To create one or more data sets or tables, in permanent or temporary locations.
- To run one or more data programs, using any, all, or none of the above components.
- Any combination of the above. That is, you can create data, packages, and threads, plus run one or more data programs within a single DS2 program.

The order and number of programming blocks in a DS2 program does not matter, as long as the program compiles and contains enough RUN statements to execute the program. DS2 procedure consists of the following framework. When you write a program, place your code within the following framework:

```
PROC DS2 <option(s)>;
...DS2 language statements
RUN;
RUN CANCEL;
QUIT;
```

PROC DS2 specifies that the subsequent input is DS2 language statements. RUN CANCEL will cancel the previous DS2 language statements. Please see below for a sample DS2 program:

```
proc ds2;
data _null_;
   method init();
      dcl varchar(40) str;
      str = 'PROC DS2: What is in it for you?';
      put str;
   end;
enddata;
run;
quit;
```

Following results will be observed in the SAS log if above code is executed:

```
40   proc ds2;
41   data _null_;
42      method init();
43         dcl varchar(40) str;
44         str = 'PROC DS2: What is in it for you?';
45         put str;
46      end;
47   enddata;
48   run;
PROC DS2: What is in it for you?
NOTE: Execution succeeded. No rows affected.
49   quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.09 seconds
      cpu time            0.09 seconds
```

DS2 PROGRAM SYMANTICS

A DS2 program consists of a list of declarations followed by a list of method statements. Here is an example of a simple declare list:

```
declare varchar(40) str;
declare int x;
```

Here is an example of a simple method statement list:

```
method init();
end;
method run();
end;
method term();
end;
```

Combining the two lists creates a simple DS2 program.

```
declare varchar(40) str;
declare int x;
method init();
end;
method run();
end;
method term();
end;
```

The program illustrates how to declare an identifier, either in a DECLARE statement or in a METHOD statement. It also illustrates how the high-level structure of a DS2 program consists of a sequence of variable declarations followed by a sequence of METHOD statements.

VARIABLE DECLARATION STATEMENTS

A variable declaration allocates memory space and identifies that memory with an identifier, called the variable name. The declaration, both explicitly or implicitly, allocates memory for the variable and designates the type of data that can be saved at that memory location. In DS2, one can declare variables by using the DECLARE statement. A DECLARE statement performs the following actions:

- Assigns an identifier to a memory location. That identifier becomes the variable name.
- Designates the type of data that the variable can hold.
- Allocates a specified amount of memory to the variable.
- More than one variable can be declared in one DECLARE statement

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.

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:

```
data _null_;
method init();
   end;
method run();
   end;
```

```
method term();
   end;
enddata;
```

Every DS2 program will contain either implicitly or explicitly, these three methods. If you do not define any one of these methods in a DS2 program, the DS2 compiler will create an empty version of it (like those above). These methods are meant to provide a more structured framework than the SAS DATA Step implicit loop concept. In Base SAS, the entire DATA Step program was included in the implicit loop. In DS2, the implicit loop is represented by the RUN method, with the INIT and TERM methods providing initialization and finalization code, respectively.

When a DS2 program executes, here are the results:

1. The INIT method runs. Any initializations take place.
2. Variables in the program data vector which have not been retained will be set to the appropriate missing values. For more information, see the RETAIN Statement.
3. The RUN method executes.
4. Execution control then depends on the status of any input statement in the RUN method. Currently, the only input statement in DS2 is the SET statement. If the RUN method meets one of these conditions, then processing proceeds to Step 5. Otherwise, processing proceeds to Step 2 so that the RUN method can execute again:
5. No input statements
6. An input statement that has completed execution
7. The TERM method executes, and any final statements execute.
8. The INIT, RUN, and TERM methods must be defined without any parameters and without a return value. If you specify a parameter for the INIT, RUN, or TERM methods, an error will occur.
9. If you do not specify an OUTPUT statement in the DS2 program, the DS2 compiler will provide one with no parameters that executes at the end of the RUN method.
10. If you attempt to call the INIT, RUN, or TERM method directly from a DS2 program, an error will occur.
11. User-defined methods can be created by enclosing statements that you would like executed one or more times within METHOD and END statements. For more information about user-defined methods, see the METHOD Statement.

When using PROC DS2, DS2 programs are delimited by RUN statements. If additional DS2 code is found after a RUN statement, this code composes a new, distinct DS2 program from the DS2 program before the previous RUN statement

SCOPE OF DS2 IDENTIFIERS

In this section, we discuss what the scope of the DS2 identifiers is. What are the programming blocks in DS2? What is the scope of variables in DS2? What is a variable lifetime?

PROGRAMMING BLOCKS

A programming block is a section of code that begins and ends with and ordered pair of keywords. The following table depicts the programming blocks and the corresponding keyword delimiters which create programming blocks:

Block	Delimiters	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 subblock 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.

Figure 1. DS2 Programming Blocks and Scope

Within DS2, following terms are used for programming blocks:

- A data programming block or data program refers to code bounded by DATA…ENDDATA statements.
- A package programming block or package refers to the stored library of variables and methods bounded by PACKAGE…ENDPACKAGE statements. The variables and methods of a package can be used by DS2 programs, threads, or other packages.
- A thread programming block, or thread program, refers to a stored program that is bounded by the THREAD…ENDTHREAD statements. The thread program can be called by the SET FROM statement in a DS2 program or package.
- A DO programming block, or DO loop, refers to a sub-block of programming statements that are bounded by the DO and END statements.
- A method programming block or method block refers to a sub-block of programming statements that are bounded by the METHOD and END statements.

Some blocks can be nested. In following example, there is one data program, defined by the DATA and ENDDATA statements, and three nested method blocks, defined by the three method statements.

```
data _null_;
   declare int x;

   method init();
      declare double d;
   end;
```

```
   method run();
   end;

   method term();
   end;
enddata;
```

A variable declared in the outermost programming block is called a global variable, or a variable having global scope. A variable declared in any nested block is called a local variable, or a variable having scope that is local to that block. DS2 also assigns global scope to undeclared variables. In the preceding example, X is a global variable, and D is a variable that is local to the nested INIT method.

A DS2 program can have multiple subprograms followed by optional data program. The following restrictions apply:

- There can be only one data program and the data program must be the last subprogram.
- The ENDPACKAGE, ENDTHREAD, or ENDDATA statements are optional for the last subprogram of the DS2 program. These statements are required for all other subprograms.

VARIABLE LOOKUP

When a variable is referenced, DS2 will always search for the variable's declaration beginning in the block of the reference. Then, if it is not found there, it will search successively in any outer containing blocks or program. In this example, any reference to X in the INIT method will refer to the global declaration of X.

```
declare int x;
method init();
end;
```

Because methods are blocks, they can contain declarations themselves. In this example, any reference to X in the INIT method will refer to the local declaration of X, but any reference to X in the RUN method will refer to the global declaration of X.

```
declare int x;
method init();
   declare int x;
end;
method run();
end;
```

DEFINTION OF SCOPE

Scope can be considered an attribute of identifiers. Identifiers can refer to a number of program entities: method names, functions, data names, labels, or program variables. This section uses program variables as examples, but any identifier is subject to scoping rules.

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.

In DS2, a variable is accessible only as long as program execution is taking place within the scope of the variable. That is, the values of variables are accessible only when a statement in the scope of the variable is actively executing.

In the following example, the variable Z is in scope only while the INIT method is executing. Neither the RUN or the TERM methods can refer to it.

```
data;
     declare int z;     /* global z in global scope */
     method init();
        z = 10;            /* global z assigned 10 */
     end;
```

```
     method run();
     end;
     method term();
     end;
enddata;
```

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. For example, in the following program two different variables share the same name, Z. Global variable Z has global scope, and local variable Z has local scope. Within the local scope of method INIT, local variable Z hides global variable Z. Therefore, the assignment statement assigns 10 to local variable Z.

```
data;
      declare int Z;     /* global z in global scope */
      method init();
         declare int Z;  /* local z in local scope */
         Z = 10;         /* local z assigned 10 */
      end;
      method run();
      end;
      method term();
      end;
enddata;
```

VARIABLE LIFETIME

The lifetime of a variable is the time during which the variable exists. Global variables exist for the duration of the program. Local variables exist for the duration of the block in which the variable was declared. The value of a global variable will be set to a missing or null value before entry into the RUN method unless that global variable appears within a RETAIN statement in the current program block.

In the following example, the variable X exists only while the INIT method is executing and the variable Y exists for the duration of the data program.

```
data;
     declare double y;
     method init();
        declare double x;
     end;
enddata;
```

During a variable's lifetime, it can be overshadowed by a locally declared variable of the same name, as in this example:

```
data;
declare int x;
method init();
     declare int x;
end;
method run();
end;
```

Although the global variable X has lifetime for the entire program, it is not directly accessible from the INIT method because of the local declaration of X in the INIT method.

DS2 PACKAGES

DS2 package is a collection of methods and variables that can be used in DS2 programs. A DS2 package supports a 1set of related tasks and is designed for reuse. There are two types of packages:

USER DEFINED PACKAGES

These are packages that you can use to store methods for any purpose. A user can store methods that you create in user-defined packages. These packages can be thought of as libraries of your methods. Any type of method can be saved in a package. Once you have stored methods in a package (using the PACKAGE statement), you can access them by creating an instance of the package with only a DECLARE statement or with the _NEW_ operator.

PREDEFINED PACKAGES

SAS provides some predefined packages in DS2.

- 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.

To use a package, a DS2 program, another package, or a thread instantiates the package and accesses its methods.

THREADED PROCESSING IN DS2

Threaded processing is a great feature of DS2. Generally, DS2 code runs sequentially i.e. one process runs to completion before the next process begins. Although, using threaded processing it is possible to run more than one process concurrently. In threaded processing, each concurrently executing section of code is said to be running in a thread. It has been observed that DS2 threading works well both on a machine with multiple cores as well as within a massively parallel processing (MPP) database.

A DS2 program processes input data and produces output data. A DS2 program can run in two different ways: as a program and as a thread. When a DS2 program runs as a program, here are the results:

- Input data can include both rows from database tables and rows from DS2 program threads.
- Output data can be either database tables or rows that are returned to the client application.

When a DS2 program runs as a thread, here are the results:

- Input data can include only rows from database tables, not other threads.
- Output data includes the rows that are returned to the DS2 program that started the thread.

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

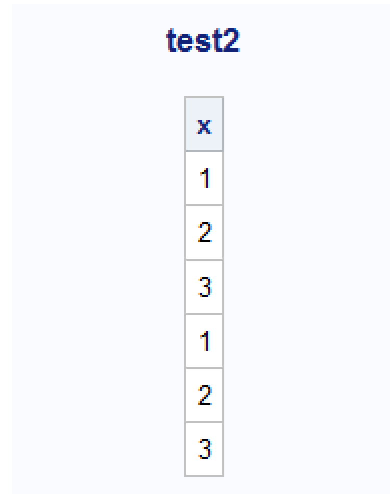
In this example, a very simple thread, T, is created by using the THREAD statement.

```
thread t;
       dcl int x;
       method init();
              dcl int i;
              do i = 1 to 3;
                     x = i;
                     output;
              end;
       end;
endthread;
```

In this DS2 program, an instance of T is declared, and two threads are executed, using the SET FROM statement in the RUN method. Each of the two threads generates three rows for x for a total of six rows in the output table.

```
data;
      dcl thread t t_instance;
      method run();
            set from t_instance threads=2;
            put 'x= ' x ;
      end;
      enddata;
run;
```

When you run the DS2 program, the SAS log might display the following output. Because of how threads are processed, the order of the output could be different.



test2

x
1
2
3
1
2
3

Figure 2. Threaded process output in DS2

If one computation thread can keep up with the I/O thread, then that single thread is used for all computation. A single reader feeds all threads. A SET statement in a thread program shares a single reader for that SET statement. Each row in the input table is sent to exactly one thread.

SAS IN DATABASE CODE ACCELERATOR

The SAS In-Database Code Accelerator publishes the DS2 thread program to the database and executes the thread program in parallel inside the database. If the data program does not contain any transformations requiring serialization, the SAS In-Database Code Accelerator also publishes and executes the data program in parallel inside the database

SAS FEDERATED QUERY LANGUANGE (FEDSQL)

SAS FedSQL is a SAS proprietary implementation of ANSI SQL:1999 core standard. It provides support for new data types and other ANSI 1999 core compliance features and proprietary extensions. FedSQL provides a scalable, threaded, high-performance way to access, manage, and share relational data in multiple data sources. When possible, FedSQL queries are optimized with multi-threaded algorithms in order to resolve large-scale operations.

For applications, FedSQL provides a common SQL syntax across all data sources. That is, FedSQL is a vendor-neutral SQL dialect that accesses data from various data sources without having to submit queries in the SQL dialect that is specific to the data source. In addition, a single FedSQL query can target data in several data sources and return a single result table.

A federated query is one that accesses data from multiple data sources and returns a single result set. The data remains stored in the data source. For example, in this query, data is requested from an Oracle table and from two Teradata tables:

```
select Ora1.employee Tera2.salary, Tera3.address
from Oracle.employee Ora1, Teradata.salary Tera2, Teradata.address Tera3
where Ora1.emp_id = Tera2.emp_id and Tera2.emp_id = Tera3.emp_id;
```

CONCLUSION

In conclusion, we have observed that DS2 is a very powerful language. It allows a great deal of additional functionality such as support for different data types allowing for greater precision in data processing, threaded application processing resulting in faster processing speeds in on a machine with multiple cores as well as within massively parallel processing databases. It offers support for in-database processing at the disposal of an application developer and accepts embedded FedSQL which allows users to connect to multiple tables within disparate databases within a single query and extract data for processing. DS2 is a powerful language. The DS2 language shares core features with the DATA step. However, capabilities of DS2 extend far beyond those of the DATA step.

REFERENCES

SAS Institute Inc. 2015. “SAS® 9.4 DS2 Language Reference, Fifth Edition". *Cary, NC: SAS Institute Inc.* *https://support.sas.com/documentation/cdl/en/ds2ref/68052/PDF/default/ds2ref.pdf*

SAS Institute Inc. 2014. “SAS® 9.4 FedSQL Language Reference, Third Edition". Cary, NC: SAS Institute Inc. *http://support.sas.com/documentation/cdl/en/fedsqlref/67364/PDF/default/fedsqlref.pdf*

ACKNOWLEDGMENTS

I would like to profusely thank John Hagen, Mike Zeffiro and Sunmin Park for supporting my work at MUFG Union Bank. I would like to thank Baskar Anjappan, Thomas Billings and Uma Karanam for encouraging me to write and publish a paper. I would also thank my employers MUFG Union Bank for providing me with an opportunity for attending the conference.

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.

CONTACT INFORMATION

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

Name: Viraj R Kumbhakarna
Enterprise: MUFG Union Bank N.A.
Address: 5600 Caprice Common
City, State ZIP: Fremont, CA, 94538
E-mail: vkumbhakarna@gmail.com

DISCLAIMER

The contents of the paper herein are solely the author's thoughts and opinions, which do not represent those of MUFG Union Bank N.A. MUFG Union Bank N.A. does not endorse, recommend, or promote any of the computing architectures, platforms, software, programming techniques or styles referenced in this paper.

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.

The output/code/data analysis for this paper was generated using SAS software. Copyright, SAS Institute Inc. SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc., Cary, NC, USA.

Copyright 2015, SAS Institute Inc., Cary, NC, USA. All Rights Reserved. Reproduced with permission of SAS Institute Inc., Cary, NC