A Brief Introduction to DS2

By Mark Jordan

Email: Mark.Jordan@sas.com
Twitter: @SASJedi
Blog: http://sasjedi.tips
Author Info: http://support.sas.com/jordan

Objectives

- Describe the DS2 programming language.
- Identify when it is most appropriate to use the DS2 language.
- Compare the DS2 DATA program to Base SAS DATA step execution.
- Understand basic DS2 Syntax
  - Explain DS2 variable declaration and its effect on variable scope.
  - Name three DS2 system methods and the conditions under which they execute.
- Parallel Processing in DS2
  - Describe the similarities and differences between DS2 DATA and THREAD programs
  - Convert a DS2 DATA program to a DS2 thread.
  - Execute DS2 threads from a DS2 DATA program

What Is DS2?

DS2 is a new SAS programming language.

- Included with Base SAS with DATA step-like syntax.
- Implemented in Base SAS as the DS2 procedure.
- It provides syntax for advanced data manipulation techniques.

Base SAS DATA Step

```
data _null_;
   Text='Hello, World!';
   put Text=;
run;
```

DS2 DATA program

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


What's Different About DS2?

DS2 natively supports ANSI SQL data types for precise data manipulation.

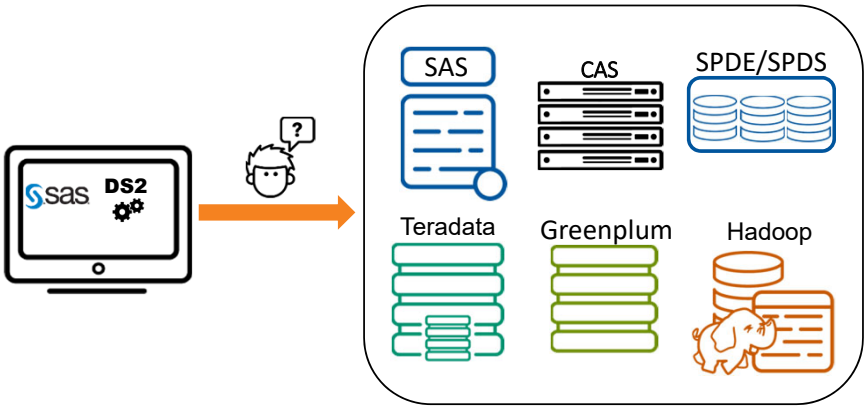
Examples:

Data Type	Examples
Fractional Numerics	DECIMAL, DOUBLE, FLOAT, REAL
Integer Numerics	BIGINT, INTEGER, SMALLINT, TINYINT
Date and Time	DATE, TIME, TIMESTAMP
Character	CHAR, NCHAR, VARCHAR, NVARCHAR

SAS numeric variables are processed as DOUBLE.
SAS character variables are processed as CHAR.

What's Different About DS2?

- DS2 can define and directly manipulate many ANSI data types.
- Types that are available for storage depend on the destination.

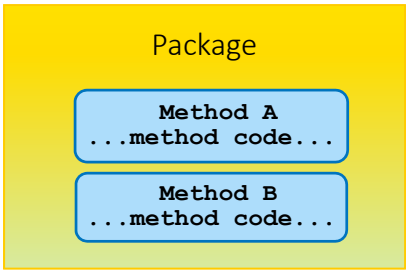


5

sas

What's Different About DS2?

DS2 improves the extensibility and reusability of code through the use of methods and packages.



Methods and packages can be predefined or user-defined.

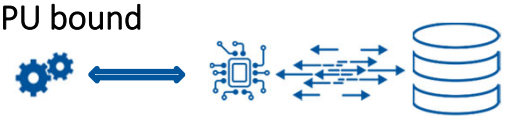
6

sas

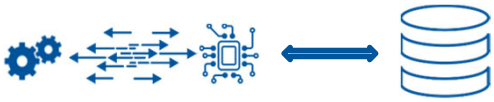
What's Different About DS2?

DS2 was designed from the ground up to mitigate the toughest **processing bottlenecks** in modern data processing environments:

- A process where data is delivered more quickly than computations can be completed is **CPU bound**



- A process where computations can be completed faster than the next row of data can be delivered is **I/O bound**



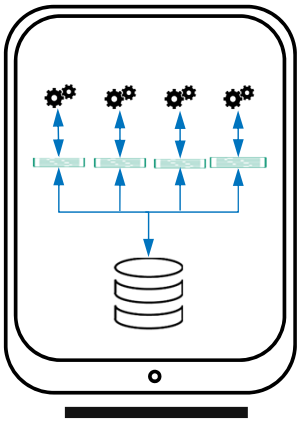
7

sas

Processing Terminology

Threading

- Threaded application processing mitigates CPU-bound processes.
  - Computations are conducted on multiple data records in parallel.
  - I/O can proceed more smoothly.



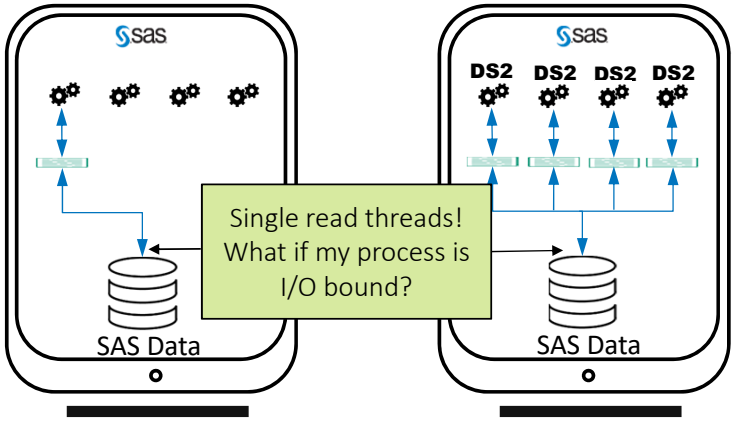
If the SAS log reports real (elapsed) time about equal to CPU time, the process is likely CPU bound. Threading DS2 on the SAS platform can reduce the elapsed time required to execute these programs.

8

sas

What's Different About DS2?

- The Base SAS DATA step is single threaded.
- DS2 can execute multi-threaded in Base SAS providing relief for CPU bound processes.

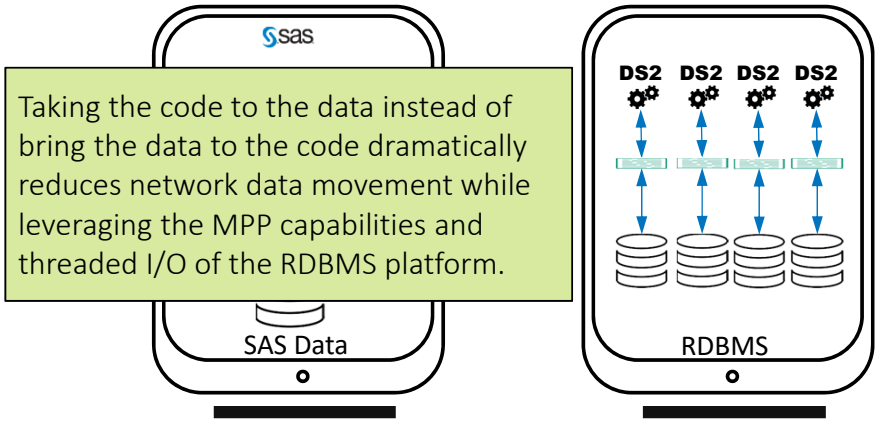


9

sas

What's Different About DS2?

With the SAS In-Database Code Accelerator, DS2 threads can be executed in parallel on Massively Parallel Processing (MPP) RDBMS hardware:

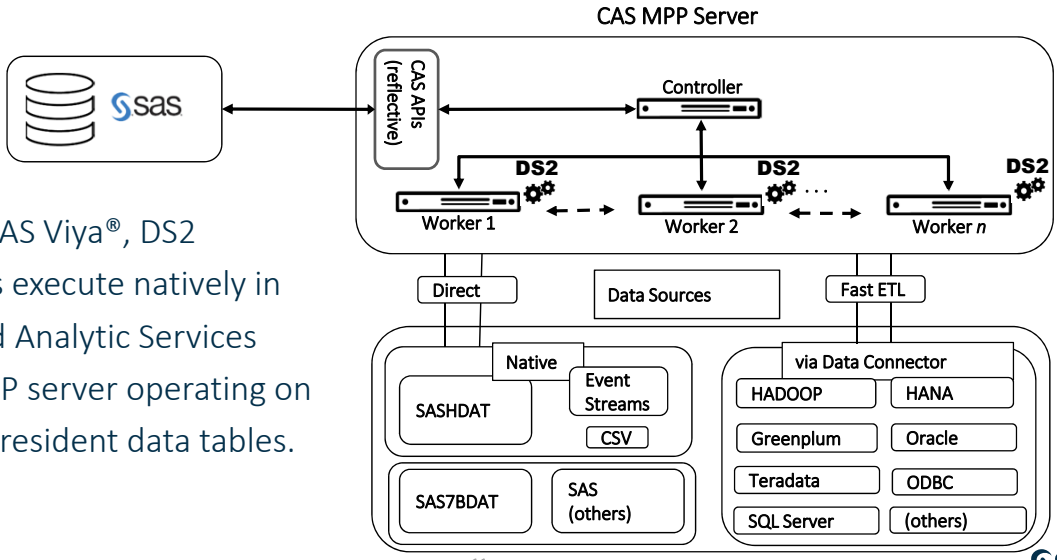


10

sas

What's Different About DS2?

Or with SAS Viya®, DS2 programs execute natively in the Cloud Analytic Services (CAS) MPP server operating on memory-resident data tables.



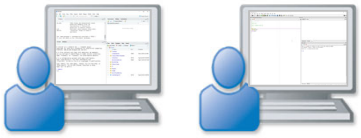
11

How Are DS2 Programs Created?

DS2 programs are written and executed in Base SAS using the DS2 procedure (PROC DS2).

DS2 programs can also be executed in SAS® Viya™ on CAS from

- a SAS 9 client using PROC DS2 code
- Python, Lua, Java, R, etc. using CAS actions



12

When Should I Use DS2?

DS2 uses a special threaded driver to access data. Drivers are available for the following data sources:

- Amazon Redshift
- Aster
- CAS
- DB2 (UNIX and PC)
- Pivotal Greenplum
- Hadoop (Hive and HDMD)
- Impala
- Memory Data Store (MDS)
- MS SQL Server
- MYSQL
- Netezza
- Oracle
- PostgreSQL
- SAP (read only)
- SAP HANA
- SAP IQ
- SAS data sets
- SASHDAT files
- SPD Engine
- SPD Server
- Teradata
- Vertica
- ODBC-compliant databases

Drivers for SAS data are included with Base SAS
Drivers for 3rd party data are included with the SAS/ACCESS Interface

13

sas

When to Use DS2?

DS2 programs are best suited for applications that do the following:

- are computationally complex (threaded processing in Base SAS)
- can execute in massively parallel processing (MPP) platforms
  - CAS
  - With the SAS In-Database Code Accelerator: Teradata, Hadoop and Greenplum
- require the precision that the DS2 data types offer
- leverage the convenient reusability of DS2 methods and packages

14

sas

Basic DS2 Syntax

- DS2 includes syntax for three types of programs:
  - package programs
  - thread programs
  - DATA programs
- PROC DS2 uses RUN-group processing.

Packages are the objects of the DS2 programming language. Packages and Object Oriented Programming with DS2 are not covered in this presentation.

```
proc ds2;
package work.pgk;
   <more program statements>
endpackage;
run;
thread work.thread;
   <more program statements>
endthread;
run;
data _null_;
   <more program statements>
enddata;
run;
quit;
```

15

Basic DS2 Syntax

Methods

- Methods are named, executable blocks of code.
  - The METHOD statement names the method.
  - The END statement terminates the method.
- All executable code must be encapsulated in a method block.

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

ds2_d01

DS2 DATA Programs

DATA programs are analogous to traditional SAS DATA step

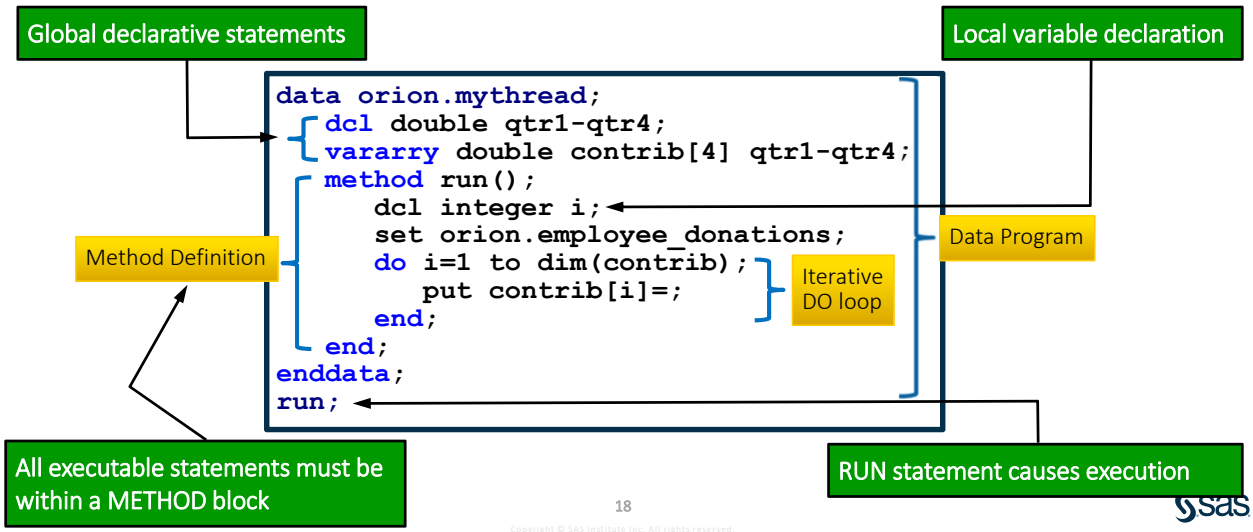
- Enforce a more structured programming syntax
- Require all executable code to be included in a method definition
- Can include variables global to the DATA program (and in the PDV) or local to a method (not in the PDV)
- Require at least explicitly written system method

17

sas

DS2 DATA Program Structure

- DATA program block structure



18

sas

DS2 Packages

DS2 packages are collections of methods and variables stored in SAS libraries.

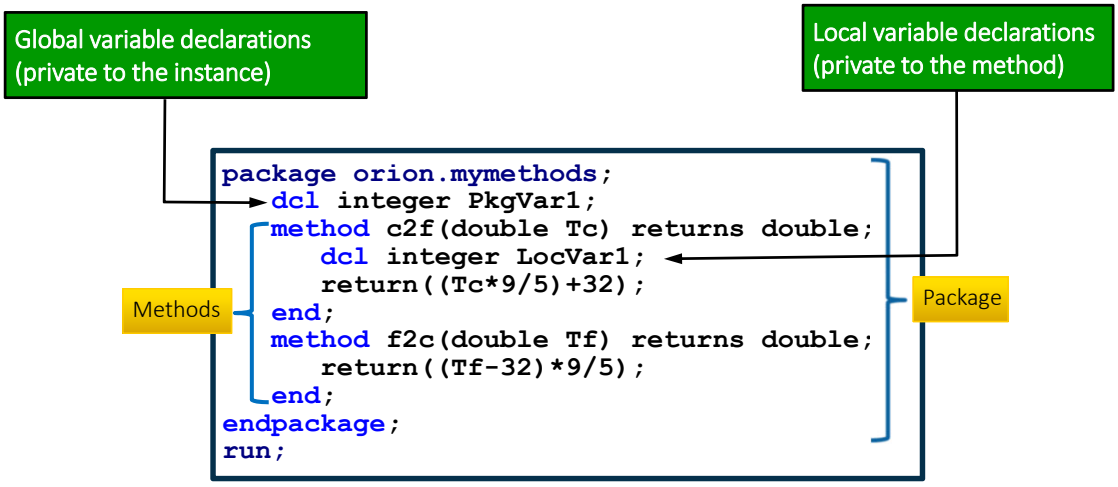
- Simplify re-using code in DS2 DATA and Thread programs.
- Predefined packages ship with SAS to extend DS2 capabilities.
- User-defined packages enable easy, secure sharing of proprietary algorithms.
- Packages enable object-oriented programming for the DS2 language
  - Can accept parameters when instantiated
  - Can include user-defined constructor and destructor methods
  - Global variables in a package are private to the package instance and do not affect the PDV of the DATA program in which they are used

19

sas

DS2 PACKAGE Program Structure

- PACKAGE program block structure



20

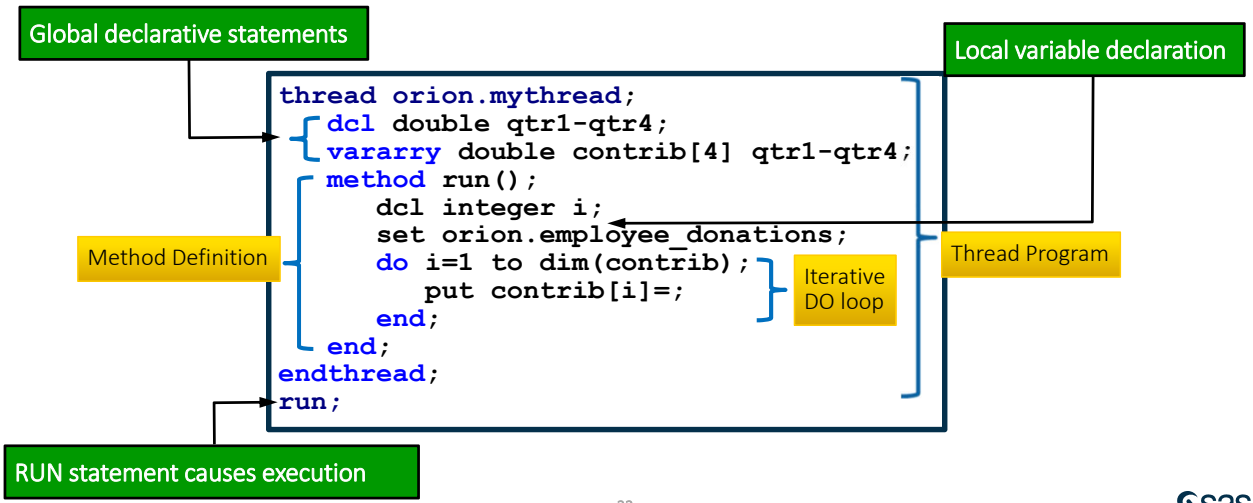
sas

DS2 Thread Programs

- Threads are programs stored in SAS libraries and used for parallel processing in DS2
- Similar to DATA programs in that they
  - Enforce a more structured programming syntax
  - Require all executable code to be included in a method definition
  - Require at least explicitly written system method
  - Global variables appear in the PDV of the DATA program in which they are used
- Similar to DS2 packages in that they
  - can be re-used when stored in permanent SAS libraries.
  - can contain user-defined methods.
  - can accept parameters

DS2 Thread Program Structure

- THREAD program block structure



Basic DS2 Syntax

```
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
```

System methods execute automatically.

- INIT() – once at start
- RUN() – once for every row
- TERM() – once at termination

The data set orion.banks has three observations. The RUN method executes three times.

Only the RUN method includes an implicit OUTPUT statement.

ds2_d02

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

System methods

- execute automatically
- cannot be explicitly called

DS2 DATA programs must contain user-written code for at least one system method.

ds2_d02

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

System methods

- execute automatically
- cannot be explicitly called
- do not accept arguments.

ds2_d02

Basic DS2 Syntax

User-defined methods have the following features:

- can accept arguments

```
proc ds2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double Degc DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

ds2_d03

Basic DS2 Syntax

User-defined methods have the following features:

- can accept arguments
- can return a value

```
proc ds2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double Degc DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

ds2_d03

Basic DS2 Syntax

User-defined methods have the following features:

- execute when referenced

```
proc ds2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double Degc DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

ds2_d03

Basic DS2 Syntax

User-defined methods have the following features:

- execute when referenced
- can be referenced multiple times

```
proc ds2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double DegC DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

Partial SAS Log

```
degc=0  degf=32
degc=15 degf=59
degc=30 degf=86
```

ds2_d03

Basic DS2 Syntax

Declarative statements must come before executable statements.

- Global declaratives affect the entire program, and are placed before any method definitions.
- Local declaratives affect only the method in which they appear, and are placed before the executable statements.

```
proc ds2;
data test;
   dcl double MyVar;
   retain MyVar 0;
   method run();
      dcl integer i;
      set banks;
      do i=1 to 3;
         MyVar=10**i;
      end;
   end;
enddata;
run;
quit;
```

The DECALRE (DCL) statement is the only permissible local declarative statement.

30

Basic DS2 Syntax

General form of the DECLARE statement:

DECLARE|DCL *data-type variable-list*
<HAVING LABEL '*string*' | FORMAT | INFORMAT>;

Examples:

```
/* Declare three DOUBLE variables formatted dollar12.2*/
dcl double Var1 Var2 Var3 having format dollar12.2;

/* Declare a high-precision fixed point numeric variable */
dcl decimal(35,5) Var1;

/* Declare a fixed-width character variable labeled 'My Text'*/
dcl char(25) Var1 having label 'My Text';
```

Where a variable is declared in a DS2 program determines the variable's scope.

31

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   dcl double MyVar;
   method term();
      do i=1 to 3;
         MyVar=10**i;
         put MyVar=;
      end;
   end;
enddata;
run;
quit;
```

undeclared variable i

Undeclared Variables

- Variables not read in from a SET statement and not referenced in a DECLARE statement are undeclared.
- The numeric default type is DOUBLE.
- The character default type is CHAR.
- Undeclared variables produce warnings in the SAS log.

ds2_d04

32

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   dcl double MyVar;
   method init();
      dcl char(12) MyVar;
      MyVar='Just Testing';
      put MyVar=;
   end;
   method term();
      do i=1 to 3;
         MyVar=10**i;
         put MyVar=;
      end;
   end;
enddata;
run;
quit;
```

undeclared

Variable Scope

- Undeclared variables have global scope – they will appear in the PDV
- They can be referenced in any method in the program.

It is good practice to declare index variables local to the method which contains the DO loop.

ds2_d04

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   dcl double MyVar;
   method init();
      dcl char(12) MyVar;
      MyVar='Just Testing';
      put MyVar=;
   end;
   method term();
      do i=1 to 3;
         MyVar=10**i;
         put MyVar=;
      end;
   end;
enddata;
run;
quit;
```

Variable Scope

- Globally declared variables have global scope – they will appear in the PDV
- They can be referenced in any method in the program.

MyVar declared as global to the DATA program

MyVar referenced in the RUN method

ds2_d04

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   dcl double MyVar;
   method init();
      dcl char(12) MyVar;
      MyVar='Just Testing';
      put MyVar=;
   end;
   method term();
      do i=1 to 3;
         MyVar=10**i;
         put MyVar=;
      end;
   end;
enddata;
run;
quit;
```

Variable Scope

- Locally declared variables have local scope and will not appear in the PDV.
- They may only be referenced by the method in which they were declared.

Local MyVar can only be referenced within the INIT method

ds2_d04

sas

Basic DS2 Syntax

```
proc ds2;
data _null_;
   dcl double MyVar;
   method init();
      dcl char(12) MyVar;
      MyVar='Just Testing';
      put MyVar=;
   end;
   method term();
      do i=1 to 3;
         MyVar=10**i;
         put MyVar=;
      end;
   end;
enddata;
run;
quit;
```

double-precision floating-point numeric (8 bytes)

fixed-width character variable (12 bytes)

Encapsulation of Local Variables

- Variables of different scope may have the same name without interfering with each other.

ds2_d04

sas

Converting a Traditional DATA step to DS2

Selecting a method in DS2:

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;
   /* Section 2 */
   set orion.banks end=last;
   put _all_;
   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;




enddata;
run;
quit;
```

ds2_d05

sas

Converting to DS2

Converting Section 1

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;
   /* Section 2 */
   set orion.banks end=last;
   put _all_;
   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
/* Section 1 */
   method init();
      Text='**> Starting';
      put Text;
   end;




enddata;
run;
quit;
```

ds2_d05

sas

Converting to DS2

Converting Section 2

```
data _null_;
   /* Section 1 */
   if _n_ = 1 then do;
      Text='**> Starting';
      put Text;
   end;
   /* Section 2 */
   set orion.banks end=last;
   put _all_;
   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
/* Section 1 */
   method init();
      Text='**> Starting';
      put Text;
   end;
 /* Section 2 */
   method run();
      set orion.banks;
      put _all_;
   end;




enddata;
run;
quit;
```

ds2_d05

sas

Converting to DS2

Converting Section 3

```
data _null_;
   /* Section 1 */
   if _n_ = 1 then do;
      Text='**> Starting';
      put Text;
   end;
   /* Section 2 */
   set orion.banks end=last;
   put _all_;
   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
/* Section 1 */
   method init();
      Text='**> Starting';
      put Text;
   end;
 /* Section 2 */
   method run();
      set orion.banks;
      put _all_;
   end;
 /* Section 3 */
   method term();
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

ds2_d05

sas

Converting an Existing Business Process to DS2

An existing DATA step program is used to calculate quarterly employee charitable contributions, including Orion Star's 25% increase.

To run this process as multi-threaded in Base SAS, do the following:

- convert the DATA step to DS2
- convert the DS2 DATA program to a DS2 thread
- execute the thread from a DS2 DATA program

41

sas

Convert a DATA Step to a DS2 DATA Program

We'll use PROC DSTODS2 for the initial conversion.

Note there are some issues to address in the converted program:

```
proc dstods2 in="ds2_d06b.sas"
             out="ds2_d06c.sas";
run;
```

ds2_d06a

```
data new1;
   array contrib[*] qtr1-qtr4;
   set orion.employee_donations;
   where find(recipients,'%');
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;
run;
```

ds2_d06b

```
data NEW1;
   keep EMPLOYEE_ID QTR:  TOTAL;
   vararray double CONTRIB[*] QTR1-QTR4;
   method run();
      set ORION.EMPLOYEE_DONATIONS;
      /* WHERE FIND (RECIPIENTS,'%') */;
      TOTAL = 0.0;
      do Q = 1.0 to DIM(CONTRIB);
         TOTAL + CONTRIB[Q] * 1.25;
      end;
      ;
      _return:
      ;
   end;
enddata;
```

ds2_d06c

PROC DSTODS2 converts what it can, and inserts the rest of the unconverted code as comments as close to the original position as possible. For example, DS2 does not have a WHERE statement so the DS2 DATA program includes that code as a comment.

sas

Convert a DATA Step to a DS2 DATA Program

First, let's remove that unused _return label and add RUN

```
data new1;
   array contrib[*] qtr1-qtr4;
   set orion.employee_donations;
   where find(recipients,'%');
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;
run;
```

Now, what about that commented out WHERE statement?

```
data NEW1;
   keep EMPLOYEE_ID QTR:  TOTAL;
   vararray double CONTRIB[*] QTR1-QTR4;
   method run();
      set ORION.EMPLOYEE_DONATIONS;
      /* WHERE FIND (RECIPIENTS,'%') */;
      TOTAL = 0.0;
      do Q = 1.0 to DIM(CONTRIB);
         TOTAL + CONTRIB[Q] * 1.25;
      end;
   end;
enddata;
run;
```


sas

Convert a DATA Step to a DS2 DATA Program

Replace the data set name and WHERE comment with an SQL query on the SET statement:

```
data new1;
   array contrib[*] qtr1-qtr4;
   set orion.employee_donations;
   where find(recipients,'%');
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;
run;
```

Functional! Now, how to address the two undeclared variables (TOTAL and Q)?

```
data NEW1;
   keep EMPLOYEE_ID QTR:  TOTAL;
   vararray double CONTRIB[*] QTR1-QTR4;
   method run();
      set {select *
           from ORION.EMPLOYEE_DONATIONS
           WHERE FIND (RECIPIENTS,'%')};
      TOTAL = 0.0;
      do Q = 1.0 to DIM(CONTRIB);
         TOTAL + CONTRIB[Q] * 1.25;
      end;
   end;
enddata;
run;
```


sas

Convert a DATA Step to a DS2 DATA Program

Let's declare TOTAL as global (desired in the output) and Q as local to the RUN method (not desired in the output):

```
data new1;
   array contrib[*] qtr1-qtr4;
   set orion.employee_donations;
   where find(recipients,'%');
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;
run;
```

Now that Q is local, do I really need that KEEP statement?

```
data NEW1;
   dcl double TOTAL;
   keep EMPLOYEE_ID QTR:  TOTAL;
   vararray double CONTRIB[*] QTR1-QTR4;
   method run();
      dcl integer q;
      set {select *
              from ORION.EMPLOYEE_DONATIONS
              WHERE FIND (RECIPIENTS,'%')};
      TOTAL = 0.0;
      do Q = 1.0 to DIM(CONTRIB);
         TOTAL + CONTRIB[Q] * 1.25;
      end;
   end;
enddata;
run;
```


Convert a DATA Step to a DS2 DATA Program

If I leverage the SQL SELECT list, I won't need the KEEP statement:

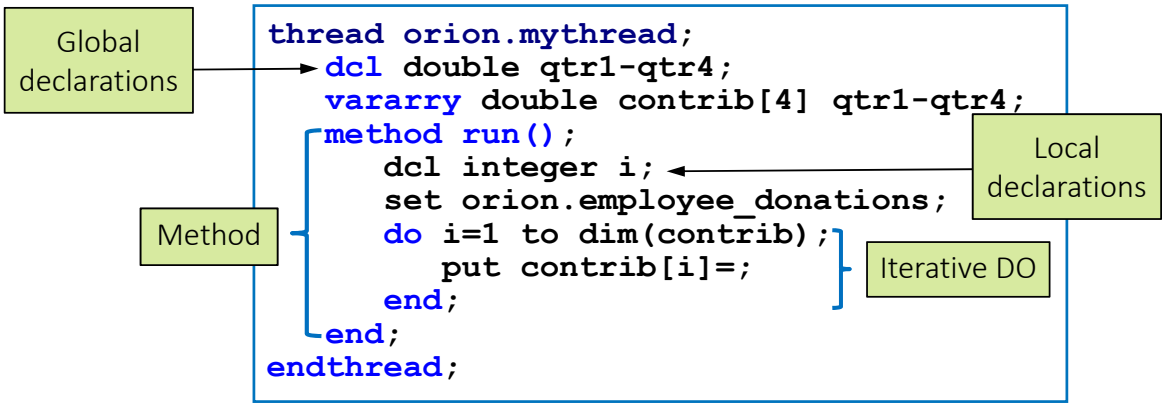
```
data new1;
   array contrib[*] qtr1-qtr4;
   set orion.employee_donations;
   where find(recipients,'%');
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;
run;
```

Ready to run!

```
data NEW1;
   dcl double TOTAL;
   vararray double CONTRIB[*] QTR1-QTR4;
   method run();
      dcl integer q;
      set {select EMPLOYEE_ID,QTR1,QTR2,QTR3, QTR4
              from ORION.EMPLOYEE_DONATIONS
              WHERE FIND (RECIPIENTS,'%')};
      TOTAL = 0.0;
      do Q = 1.0 to DIM(CONTRIB);
         TOTAL + CONTRIB[Q] * 1.25;
      end;
   end;
enddata;
run;
```

DS2 Thread Program Structure

- THREAD program block structure



Convert a DATA Program to a Thread Program

Now, convert the DATA statement to a THREAD statement.

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   method run();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
    from orion.employee_donations
    where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
enddata;
run;
quit;
```

```
proc ds2;
thread work.th_Donation/overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   method run();
      dcl int q;
      set {select Employee_ID, qtr1
                , qtr2, qtr3, qtr4
      from orion.employee_donations
      where find(recipients,'%')>0};
      Total=0;
      do q=1 to dim(contrib);
         Total+contrib[q]*1.25;
      end;
   end;
endthread;
run;
quit;
```

ds2_d07

Convert a DATA Program to a Thread Program

Convert an ENDDATA statement to an ENDTHREAD statement.

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   method run();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
    from orion.employee_donations
    where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
enddata;
run;
quit;
```

```
proc ds2;
thread work.th_Donation/overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   method run();
      dcl int q;
      set {select Employee_ID, qtr1
                , qtr2, qtr3, qtr4
      from orion.employee_donations
      where find(recipients,'%')>0};
      Total=0;
      do q=1 to dim(contrib);
         Total+contrib[q]*1.25;
      end;
   end;
endthread;
run;
quit;
```

ds2_d07

The THREAD program is executed to create a stored thread for parallel execution in a subsequent DS2 DATA program.

Using DS2 Threads

To use a thread in a DS2 DATA program, perform the following tasks:

- Declare an instance of the thread.
- Invoke parallel thread execution with a SET FROM statement.
  - Use the THREADS= option to specify the number of threads to execute.

Output rows are returned to the DS2 DATA program.

- Processing of thread results by additional statements before or after the SET FROM statement is single threaded.
- When you execute multiple threads, the order of rows that are returned might vary from run to run.

Using DS2 Threads

The THREADS= option specifies the number of threads to execute in parallel.

- Specifying more threads than are available (over threading) will eventually negatively affect performance.

Without THREADS=, execution in Base SAS is single-threaded.

Using DS2 Threads

Execute the thread in a DATA statement.

```
proc ds2;
data new /overwrite=yes ;
   dcl thread work.th_Donation th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;
```

Declare a thread instance.

ds2_d07

Using DS2 Threads

Execute the thread in a DATA statement.

```
proc ds2;
data new /overwrite=yes ;
   dcl thread work.th_Donation th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;
```

The SET FROM statement executes the thread.

ds2_d07

Using DS2 Threads

Execute the thread in a DATA statement.

```
proc ds2;
data new /overwrite=yes ;
   dcl thread work.th_Donation th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;
```

Specify the number of threads.

ds2_d07

Executing DS2 Threads in Base SAS

This demonstration illustrates Base SAS threaded application processing in a DS2 DATA program.

ds2_d08a/ds2_d08b

Questions?

Email: Mark.Jordan@sas.com
Twitter: @SASJedi
Blog: http://sasjedi.tips
Author Page: http://support.sas.com/jordan