OOPs! You Did it Again with Proc DS2

Object Oriented Programming and SAS
DS2 is a new object oriented programming language introduced in SAS 9.4 that functions within a procedure (PROC).

It is included with Base SAS so you can use it in SAS programming environments like SAS Enterprise Guide and SAS Studio.

DS2 is not necessarily meant to be a replacement for the Data Step, so don’t think that you have to convert all your existing programs to DS2.

However, DS2 does have advantages in certain areas: additional data types, ANSI SQL types, programming structure elements, and user defined methods and packages.
**Proc DS2**

**An Updated Data Step**

- New supported data types that follow most ANSI standards. This includes VARCHAR, INTEGER, FLOAT, BINARY, and other DATETIME data types.
- RUN/INIT/TERM are automatic system methods run for every DS2 program.
- Executable code must be in a method.

```sas
proc ds2;
  data cars_avg_mpg / overwrite=yes;
  dcl double avg_mpg;
  method init();
    put 'DS2 start';
  end;
  method run();
    set data.cars;
    avg_mpg = (MFG_City+MFG_Highway)/2;
  end;
  method term();
    put 'DS2 end';
  end;
  enddata;
  run;
  qmit;
```

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.
Proc DS2

User Defined Methods

- Users can create their own methods to keep their program organized between calculations and processes.
- User defined methods must be called in order to execute.
- User defined methods can be defined within a reusable user defined package (class).

```sas
proc ds2;
  data employees_full / overwrite=yes;
  dcl char(32) first_name;
  dcl char(32) last_name;

  method get_first(char(32) full_name) returns char;
    return scan(full_name, 1, ' ');
  end;

  method get_last(char(32) full_name) returns char;
    return scan(full_name, 2, ' ');
  end;

  method run();
    set work.employees;
    first_name = get_first(name);
    last_name = get_last(name);
  end;

enddata;
run;
quit;
```
Proc DS2

Parallel Processing with Threads

- Even though it still uses the implicit loop when processing a data step, DS2 can spread that process out across multiple threads.

- Threads can be useful if you have a bottleneck at the CPU processing level.

What You Need to Know

Log displays how many rows processed through each thread.

Business Case

User Defined DS2 Package
Business Problem

If you’ve owned a product for 30 days or less then you can return it for a full refund of the purchase price.

Warranty Refund Calculation

From 31 to 365 days, you can return it but your refund amount will be the purchase price less $10.00 times the number of days you’ve owned it or $0, whichever is higher.

After 365 days you receive no refund.
Build a reusable module to be called many times by other programmers, encapsulates the particulars of the warranty, and allows the flexibility to make changes to the assumptions.

- Define a package called `warranty` and save it permanently for later use.
- Include a set of warranty properties:
  - Daily Rate – daily dollar amount subtracted from original purchase price
  - Min Days – number of days since purchase up to which a full refund can be granted
  - Max Days – number of days since purchase after which zero refund is granted
  - Return Date – date product is returned for refund

- When the warranty package is instantiated, automatically set default values for the properties:
  - Daily Rate = $10.00
  - Min Days = 30
  - Max Days = 365
  - Return Date = today

- Allow programmers to override the default property values, as required.
- Include a refund method that accepts the purchase price and date purchased and returns the number of days elapsed since purchase and the refund amount.
A DS2 user defined package is saved as encrypted text within a SAS data set. Save the package permanently by using a SAS libname as the first level of the package name.

The warranty package properties dailyRate, minDays, maxDays, and returnDate are declared as variables global to the package. Within the methods you'll notice that these package global variables are identified using "this." to distinguish them from any possible local method variables of the same name.

DS2 allows method overloading. Note that there are three definitions of the warranty method. The first takes no arguments and is the default constructor. It will assign the package properties default values upon instantiation. The other two warranty method definitions provide options for instantiating the warranty package with user-defined values for dailyRate, minDays, maxDays, and returnDate.

DS2 allows you to pass arguments to methods by value or by reference. Use the "in_out" keyword to indicate by reference. The number of days elapsed since purchase and the refund amount are to be returned to the calling DATA step, so those arguments are designated as "in_out".

```sas
libname example 'C:\Projects\Enrico\blog\ds2';
%proc ds2;
package example.warranty / override=yes;
do double dailyRate;
do double minDays;
do double maxDays;
do double returnDate;

method warranty();
  this.dailyRate=10.00;
  this.minDays=30;
  this.maxDays=60;
  this.returnDate=today();
end;

method warranty(double dailyRate, double minDays, double maxDays, double returnDate);
  this.dailyRate=dailyRate;
  this.minDays=minDays;
  this.maxDays=maxDays;
  this.returnDate=returnDate;
end;

method warranty(double dailyRate, double returnDate);
  this.dailyRate=dailyRate;
  this.minDays=30;
  this.maxDays=60;
  this.returnDate=returnDate;
end;

method refund(double purchasePrice, double datePurchased, in_out double days, in_out double refundAmount);
  days=this.returnDate-datePurchased;
  if days <= this.minDays then /* full refund */
    refundAmount=purchasePrice;
  else if days > this.maxDays then /* no refund */
    refundAmount=0;
  else /* reduced refunded amount */
    refundAmount=max(0,purchasePrice-(this.dailyRate*days));
end;
endpackage;
run;
quit;
```
Use Proc DS2 to create the data set `work.refunds`.

The warranty package default constructor is used to instantiate the object `w`.

Two new variables, `refund` and `elapsedDays`, are declared globally so that they can be included as columns in `work.refunds`.

For each row in `example.products`, the refund method is called. It computes `refund` and `elapsedDays` using the `purchase_price` and `date_of_purchase`.

Records are implicitly output to `work.refunds`.

Reference the library where the warranty package is stored.

Create global instances of the `refund` and `elapsedDays` variables. They are output to `work.refunds`.

Use the `refund` method to compute `refund` and `elapsedDays` using the `purchase_price` and `date_of_purchase`.

Convert SAS date to double type.

Use the `refund` method to compute `elapsedDays` and `refund` using the `purchase_price` and `date_of_purchase`.
The `work.refunds` DS2 code block can be modified to override the default property values by passing user-defined values to the constructor.

Alternatively, the value of a property can be changed through an assignment statement prior to the refund method call.

```sas
proc ds2;

data work.refunds / overwrite=yes;

dcl package example.warranty v(20.00, to_double(date'2015-12-31'));

dcl double refund having format dollar20.2;

dcl double elapsedDays;

method run();
  set example.products;
  w.maxDays=100;
  w.refund(purchase_price, to_double(date_of_purchase), elapsedDays, refund);
end;

run;

quit;
```