PhUSE 2006

Paper TS03

What's New in SAS® 9.1.3 for Clinical Programmers

Dave Smith, SAS, Marlow, UK

ABSTRACT

SAS 9 has brought a wealth of new features for a huge variety of industries, and it is often difficult to spot the details that will help programmers in their daily task of completing new drug submissions. This paper will highlight some of the new features that could assist clinical programmers up to SAS 9.1.3 Service pack 4. Focus areas will include Base SAS, SAS Enterprise Guide 4 and the Add-in to Microsoft Office.

INTRODUCTION

This paper is directed at those who are performing the day to day tasks associated with clinical programming, especially those who either have SAS 9 or are looking for reasons to upgrade from SAS 8. Most of the topics covered are available to users with standard modules (Base, Stat, Graph); where this is not the case this will be indicated.

BASE SAS

The following section covers features common to the core of the SAS language; the first two parts concentrate on improving programming efficiency; the third on improving output and the last part on adherence to standards.

HASH TABLES

Hash tables appeared with SAS 9.0 and are a highly efficient way of joining tables, especially where one table is large and the other small. They are more efficient than using SAS formats as a look-up and much more flexible, as multiple tables can be joined in one step and complex keys can be used.

The basic procedure is to declare a hash on the table being accessed and then either perform a look-up or iterate over the table (i.e. process each row in turn). The declaration step defines the keys that the hash will be using and the data that is accessed.

For a simple example, consider adding treatment codes to a demography table. Instead of sorting the input datasets and merging them, the hash is declared on the treatment table. Note that the TREAT variable must be defined in advance with a length statement to avoid errors in the log. It is also important for efficiency to only declare the hash on the first iteration of the dataset.

```
data new;
    set rawdata.demography ;
    length treat $14. ;
    if _n_=1 then do;
           declare hash th(dataset: "rawdata.treatment");
           th.defineKey('CENTER','INVEST','PATIENT');
           th.defineData('TREAT');
           th.DefineDone( );
    end;
    rc=th.find();
    if rc=0 then output;
    drop rc;
run;
```

The find() function returns a zero if the values of CENTER, INVEST and PATIENT in the demography table match up with the hash table, and the TREAT variable is appended.

The data definition gives all the variables being returned; if you want all the variables in the dataset then you can use `th.defineData(all: 'yes').`

For comparison, on small tables the above code ran in 0.03 seconds, two sorts and a merge took a total of 0.11 seconds and proc sql took 0.90 seconds.

A second example which shows the use of hash iterators is presented below. This example creates lab shift values from a lab table with normal and alert range values and the treatment table in a single data step. The documentation is largely in the form of comments in the code, but it is important to consider what is being achieved here.

1) The lab data is being augmented with a flag for each test and visit indicating whether the test is outside the normal or alert values
2) The baseline value of the flag is retained as is the most extreme value of the flag post-baseline.
3) The data are re-organised to a one row per test structure
4) The treatment codes are appended

Note that the different hashes have different keys, which increases the number of merges that can be performed in a single step. The iterators are defined to allow passes through the data referred to by the hash. The check() function looks to see if a key combination is present, the add() function adds to the object and the replace() function is used to update the current values.

```
data test1;
    if _n_=1 then do;
           ** Set up between-row variables **;
           retain lmh 0 templmh 0 baseline_lmh 0 ;
           ** You need to declare hash key variables before using them **;
           length CENTER 8. invest 8. patient 8. labtest $8 visit 8. treat $14. ;

           ** Create the first hash on the labs dataset **;
           declare hash labsHash(hashexp: 9, dataset: "staging.labs_with_ranges",
           ordered: "a");
           labsHash.defineKey('CENTER','INVEST','PATIENT','LABTEST');
           labsHash.defineData('CENTER','INVEST','PATIENT','LABTEST','VISIT',
           'RESULT','HI_NORM','HI_ALERT','LO_NORM','LO_ALERT');
           labsHash.DefineDone( );

           ** Create the second hash for the output table **;
           ** This allows the change in structure from one per visit to **;
           ** one per test **;
           declare hash outputHash(hashexp: 4, ordered: "a" );
           outputHash.defineKey('CENTER','INVEST','PATIENT','LABTEST');
           outputHash.defineData('CENTER','INVEST','PATIENT','LABTEST','RESULT',
           'HI_NORM','HI_ALERT','LO_NORM','LO_ALERT','LMH','BASELINE_LMH', 'TREAT');
           outputHash.DefineDone( );

           ** Create the third hash for the treatment table **;
           declare hash treatHash(hashexp: 4, dataset: "raw.treatmnt",
           ordered: "a");
           treatHash.defineKey('CENTER','INVEST','PATIENT');
           treatHash.defineData('CENTER','INVEST','PATIENT','TREAT');
           treatHash.DefineDone( );

           ** Set everything to missing. This prevents uninitialised **;
           ** messages in the log **;
           call missing (center, invest, patient, labtest, visit, result, hi_norm,
           hi_alert,lo_norm,lo_alert,treat);

           ** Create the first hash iterator on the labs table **;
           declare hiter labs_iter("labsHash");
           ** Create the second hash iterator on the output table **;
           declare hiter output_iter("outputHash");
    end;

    ** Now iterate through the labs table **;
    rc=labs_iter.first();
    do while (rc=0);
           ** See if the current key values are in the output table **;
           crc=outputHash.check();
           if crc ne 0 then do;
                  ** If this is a new set of keys in the output hash, lookup the
                  treatment code **;
```

```
                frc=treatHash.find();
                ** then add the keys with the treatment to the output hash **;
                outputHash.add();
                ** set the default low/high alert flag for a new test **;
                lmh=0;
        end;

        ** Now a little business logic to identify flag status **;
        if .<LO_NORM<RESULT<HI_NORM>. then templmh=0;
        else if HI_NORM<=RESULT<HI_ALERT then templmh=1;
        else if RESULT>=HI_ALERT then templmh=2;
        else if LO_NORM>=RESULT>LO_ALERT then templmh=-1;
        else if RESULT<=LO_ALERT then templmh=-2;

        ** For baseline set the baseline flag **;
        if visit=0 then do;
                baseline_lmh=templmh ;
                lmh=0 ;
        end;
        else do;
                ** After baseline, set the most extreme flag value **;
                if abs(templmh)>abs(lmh) then lmh=templmh ;
        end;

        ** As you go through the lab table update the output hash **;
        rc=outputHash.replace();
        ** go and get another row from the labs table **;
        rc=labs_iter.next();
    end;

    ** Finally, now the output hash contains what we need, write the output **;
    ** table. This could have been done directly into the table, but this **;
    ** method cuts down on the i/o overhead **;
    rc=output_iter.first();
    do while (rc=0);
        output;
        rc=output_iter.next();
    end;
run;
```

To achieve the same without hash tables would be likely to take multiple sorts and data steps with by processing. Whether the relative obscurity of the hash table code compared to more standard methods is outweighed by its greater efficiency is a matter for discussion, but the flexibility and power of hash tables is clear. In circumstances where the code is to be thoroughly validated and then used repeatedly and efficiency is vital hash tables should be considered as the first option.

REGULAR EXPRESSIONS

With SAS 9.1 SAS supports the use of pattern matching functions, either as regular expressions or Perl regular expressions. These can be used to validate text structures and to search and replace substrings in a very code-efficient manner.

There are two similar sets of functions with slightly different syntaxes in the expression text. Perl gives a richer pattern matching set, although this can lead to relatively increased complexity. It has been said that Perl is a write-only language because reading the sequence of characters and translating that back into business rules can be tortuous; it is therefore recommended that full and careful documentation is employed!

Regular expressions are best illustrated by example, and the first of these would be to check the structure of a study number. The valid structure is ddd-ddd-dddd where each d represents a numeric digit.

```
data test ;
input studyID $1-14 ;
if _n_=1 then regID=prxparse("/\d{3}-\d{3}-\d{4}/");
if prxmatch(regid,studyID)=0 then put studyID " is not a valid studyID";
cards;
```

```
123-456-7890
654_654-2765
88-765-0987
766-876-987
;;;;
```

The output given is

```
654_654-2765  is not a valid studyID
88-765-0987  is not a valid studyID
766-876-987  is not a valid studyID
```

The key to this code is the Perl expression `\d{3}-\d{3}-\d{4}`. Each `\d` represents a numeric digit and the `{3}` indicates exactly three of them. The syntax of Perl expressions is given in the SAS help, and there are plenty of web sources. In this example the regular expression is parsed into a variable called regID and can then be used throughout the rest of the datastep; this ensures that the parsing can be done in a single iteration. The prxmatch function then validates the data against the parsed expression and returns a zero if the pattern does not match.

Another example would be where genotypic information needs to be in a certain format and validated to contain the correct letters (cagt). Here some data have been entered in a format that proc allele would not accept, and needs to be altered:

```
data test2 ;
length genotype2 $8;
input genotype $1-2 ;
regID=prxparse("/^[GATC]{2}$/i");
if prxmatch(regid,genotype)=0 then genotype2="";
else genotype2=prxchange('s/(\D)(\D)/$1-$2/',-1,genotype);
put genotype= genotype2= ;
cards;
aa
ga
GG
gt
ax
tt
;;;;
```

The output given is

```
genotype=aa genotype2=a-a
genotype=ga genotype2=g-a
genotype=GG genotype2=G-G
genotype=gt genotype2=g-t
genotype=ax genotype2=
genotype=tt genotype2=t-t
```

Here the prxparse function validates that there are two characters in the first two places in the input field and that only GATC are valid values, independent of case. The prxchange function then changes the pattern from two consecutive characters to two characters separated by a dash. The form of the substitution expression is s/ *old*/*new*/ and each section of the input text is bracketed so that it can then be referred to by $1 for the first segment and $2 for the second segment.

The examples given above used the Perl syntax; the equivalent functions for regular expressions would be RXPARSE, RXMATCH, RXCHANGE etc. The main advantage of these versions is that the syntax is simpler, but they do also include a function to validate that a word is a valid SAS name.

Code efficiency is greatly enhanced by using these expressions; you certainly could perform many of these tasks using traditional functions but the code would be lengthy and difficult to read.

ODS STAT GRAPHICS

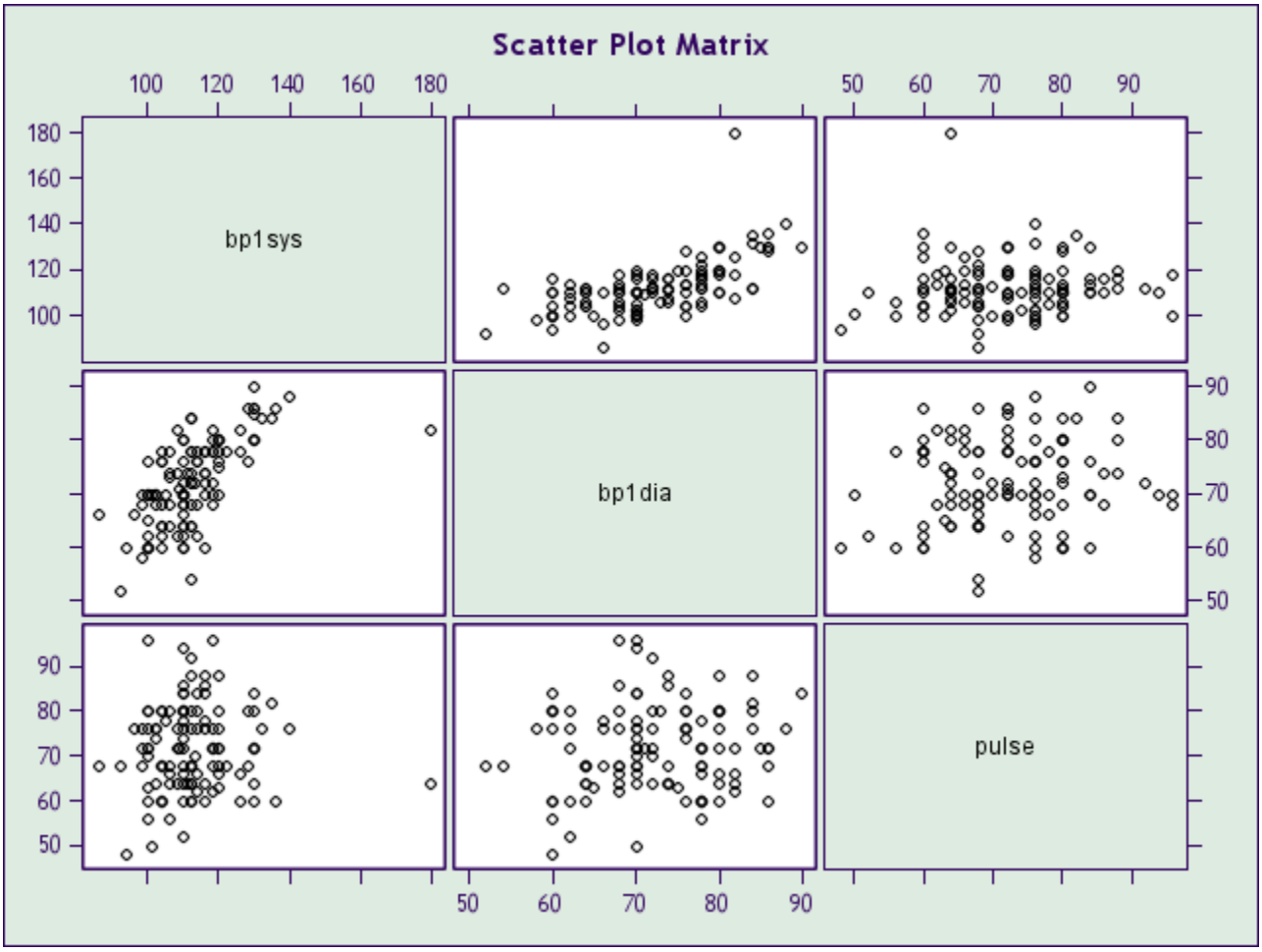
The introduction of ODS Stat graphics brings a welcome improvement to the appearance of statistical displays. They are currently experimental, and are due to go production with the 9.2 release.

To use them all that is required is to turn the option on with the statement

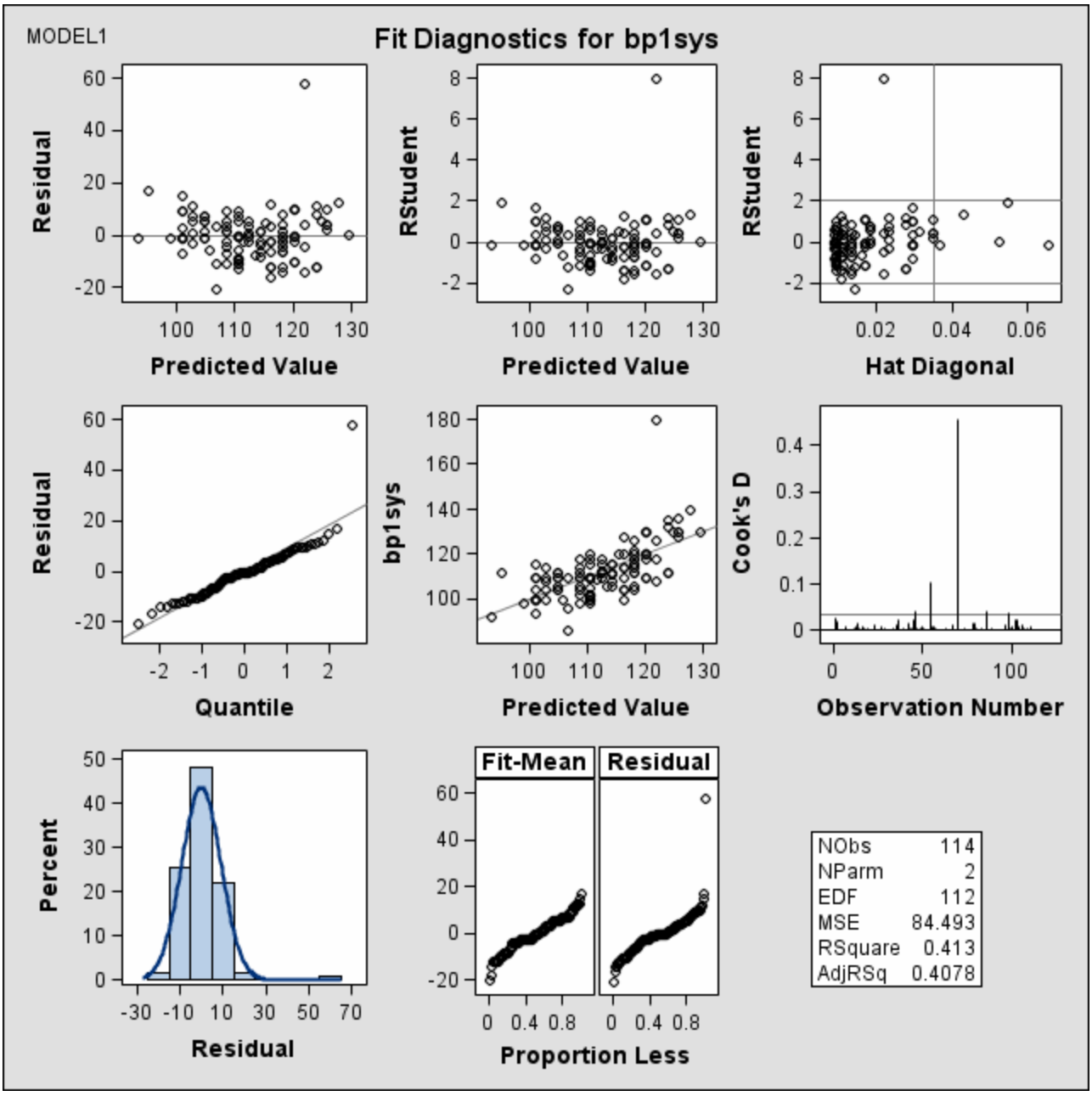
```
ods graphics on;
```

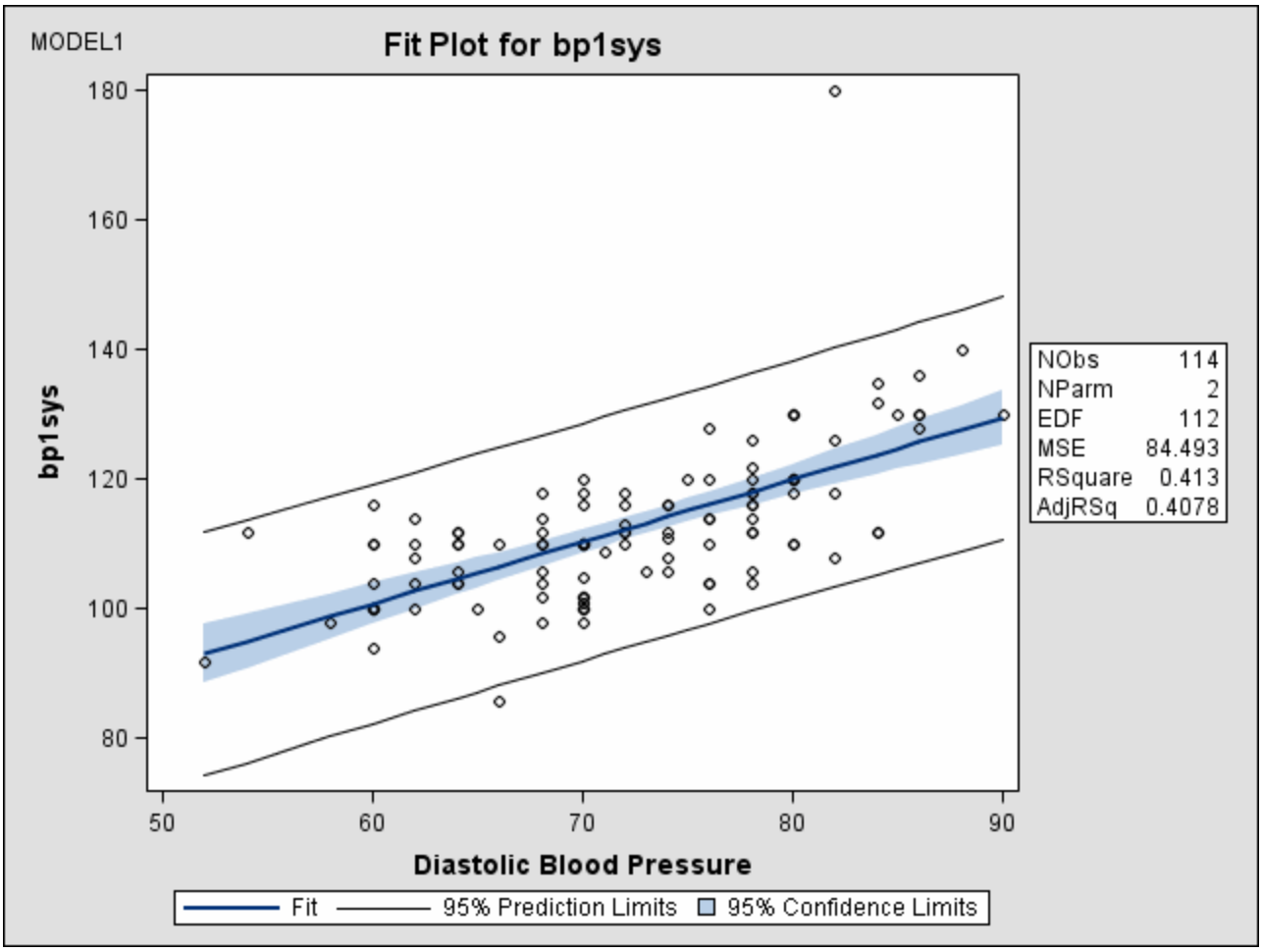
The results are much more attractive: for example using proc corr as follows

```
ods graphics on;
proc corr data=sasuser.vitals plots;
var bp1sys bp1dia pulse ;
run;
ods graphics off;
```



Enhancements have been made to several procedures including REG, PHREG, LIFETEST, GLM, MIXED, and PRINCOMP, some of the output from Proc REG is shown below:


MODEL1
Fit Diagnostics for bp1sys
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Hat Diagonal
Residual
Quantile
bp1sys
Predicted Value
Cook's D
Observation Number
Percent
Residual
Fit-Mean
Residual
Proportion Less
NObs 114
NParm 2
EDF 112
MSE 84.493
RSquare 0.413
AdjRSq 0.4078



Further reading will be indicated at the end of the paper.

PROC CDISC

The CDISC procedure continues to be enhanced as the standards emerge, and updates are made available regularly on http://support.sas.com . The latest enhancements relate to the SDTM model, and the procedure will now validate that the structures of the files comply with the SDTM guidelines.

The syntax is straightforward:

```
PROC CDISC          MODEL=SDTM ;

SDTM                SDTMVersion = "3.1" ;

DOMAINDATA          DATA = results.AE
                    DOMAIN = AE
                    CATEGORY = EVENTS ;
RUN;
```

The procedure tests the input at two levels. At the metadata level the procedure

- Verifies that all required variables are present in the dataset
- Reports as an error any variables in the dataset that are not defined in the domain
- Reports a warning for any expected domain variables which are not in the dataset
- Notes any permitted domain variables which are not in the dataset
- Verifies that all domain variables are of the expected data type and proper length
- Detects any domain variables which are assigned a controlled terminology specification by the domain and do not have a format assigned to them

At the data level the procedure

- Verifies that all required variable fields do not contain missing values
- Detects occurrences of expected variable fields that contain missing values
- Detects the conformance of all ISO-8601 specification assigned values; including date, time, datetime, duration, and interval types
- Notes correctness of yes/no and yes/no/null responses

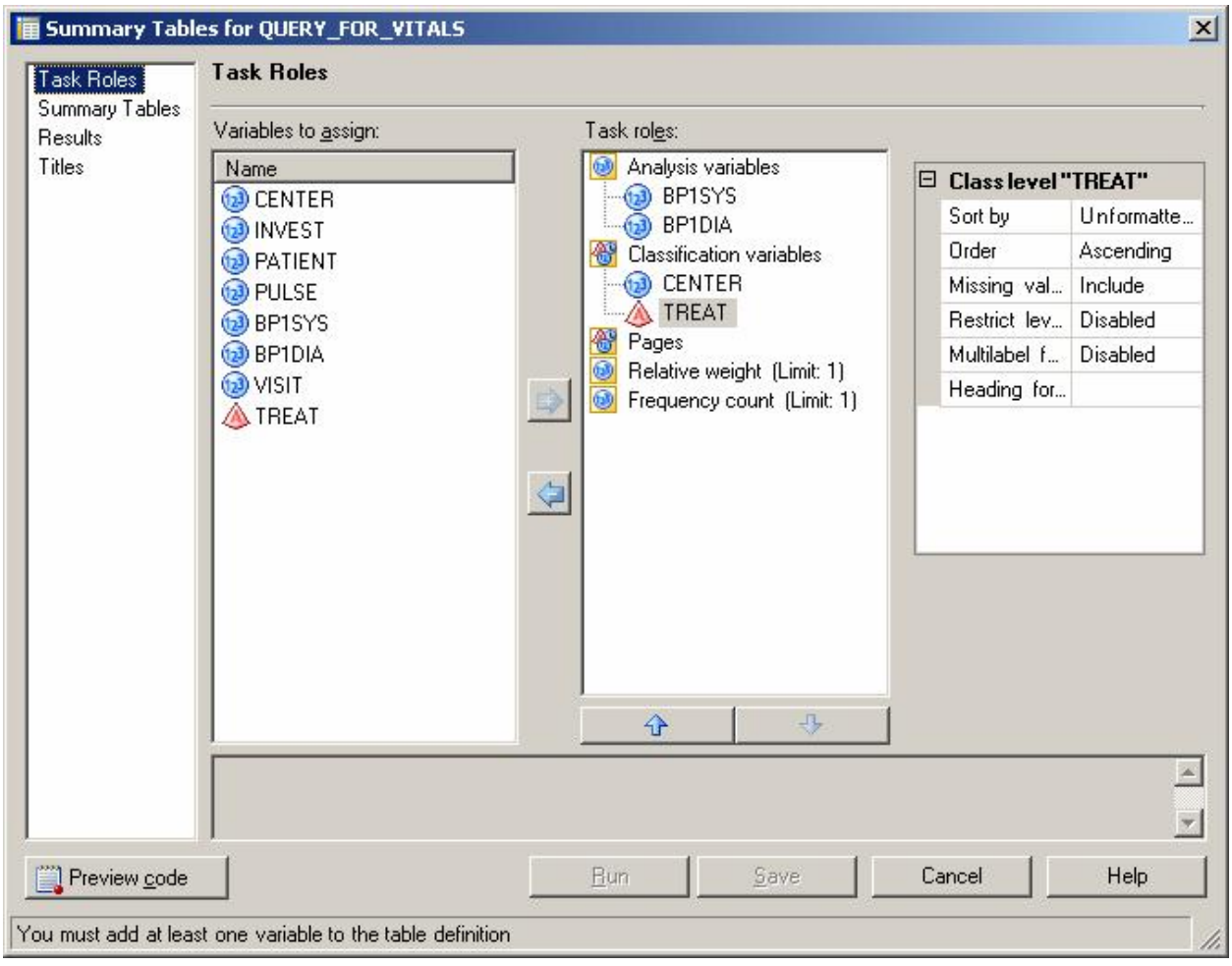
This is a major benefit where SDTM structures are being created. As new standards are released proc CDISC will continue to develop to support them. The next planned version will support CRT-DDS, with development well underway.

ENTERPRISE GUIDE

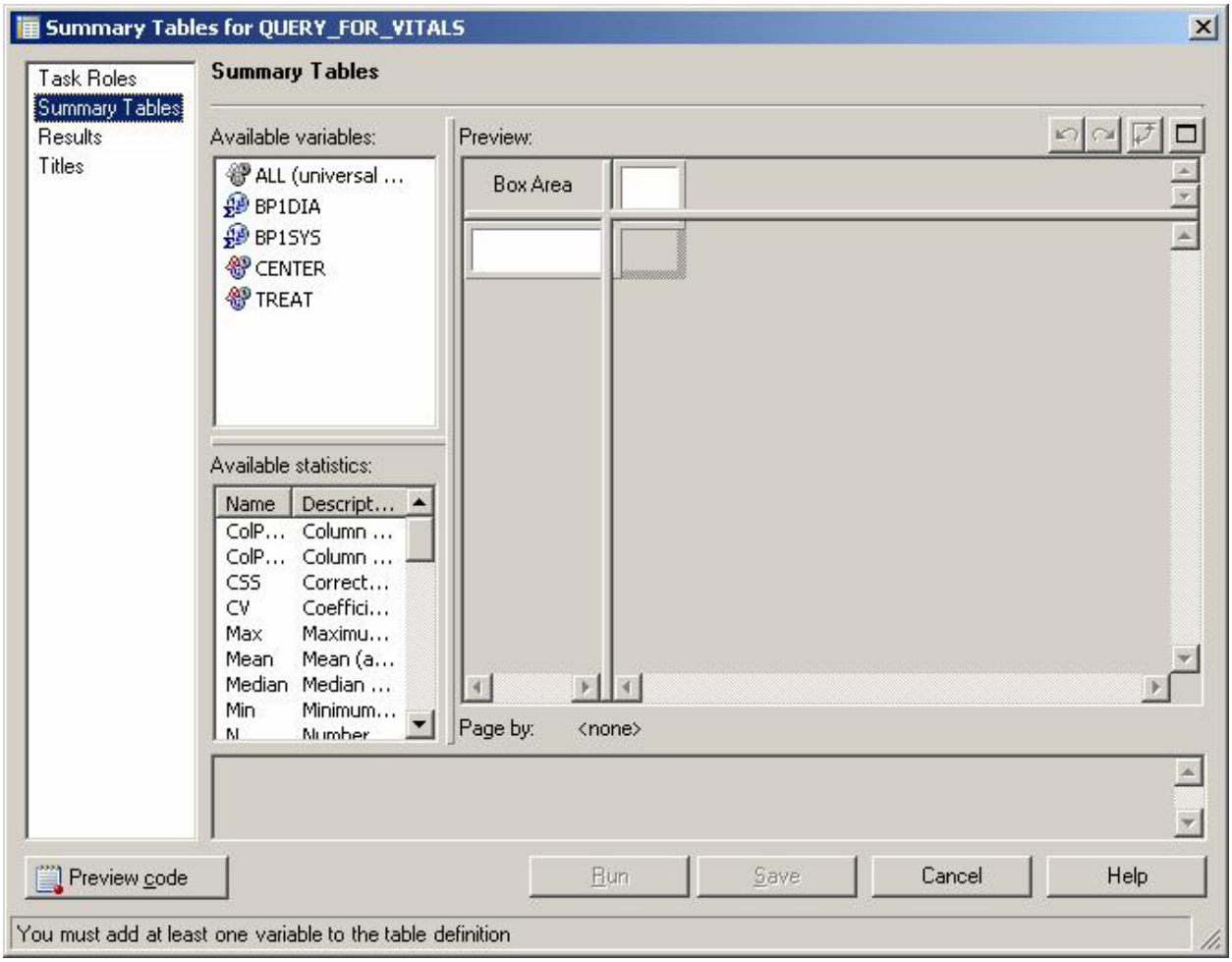
PROC TABULATE MADE EASY

It is a common conception that proc tabulate is a very useful procedure that has been greatly hampered by being difficult to use. Once the complexity of the table increases beyond simple cross-tabulations the amount of trial and error that is generally employed to produce the desired output leads to many programmers simply avoiding the procedure altogether and producing the output by a combination of data step and proc report. With the advent of the wizard in Enterprise Guide the tabulate procedure is revitalized, as it is now very easy to produce the tabulations that are required first (or second) time.

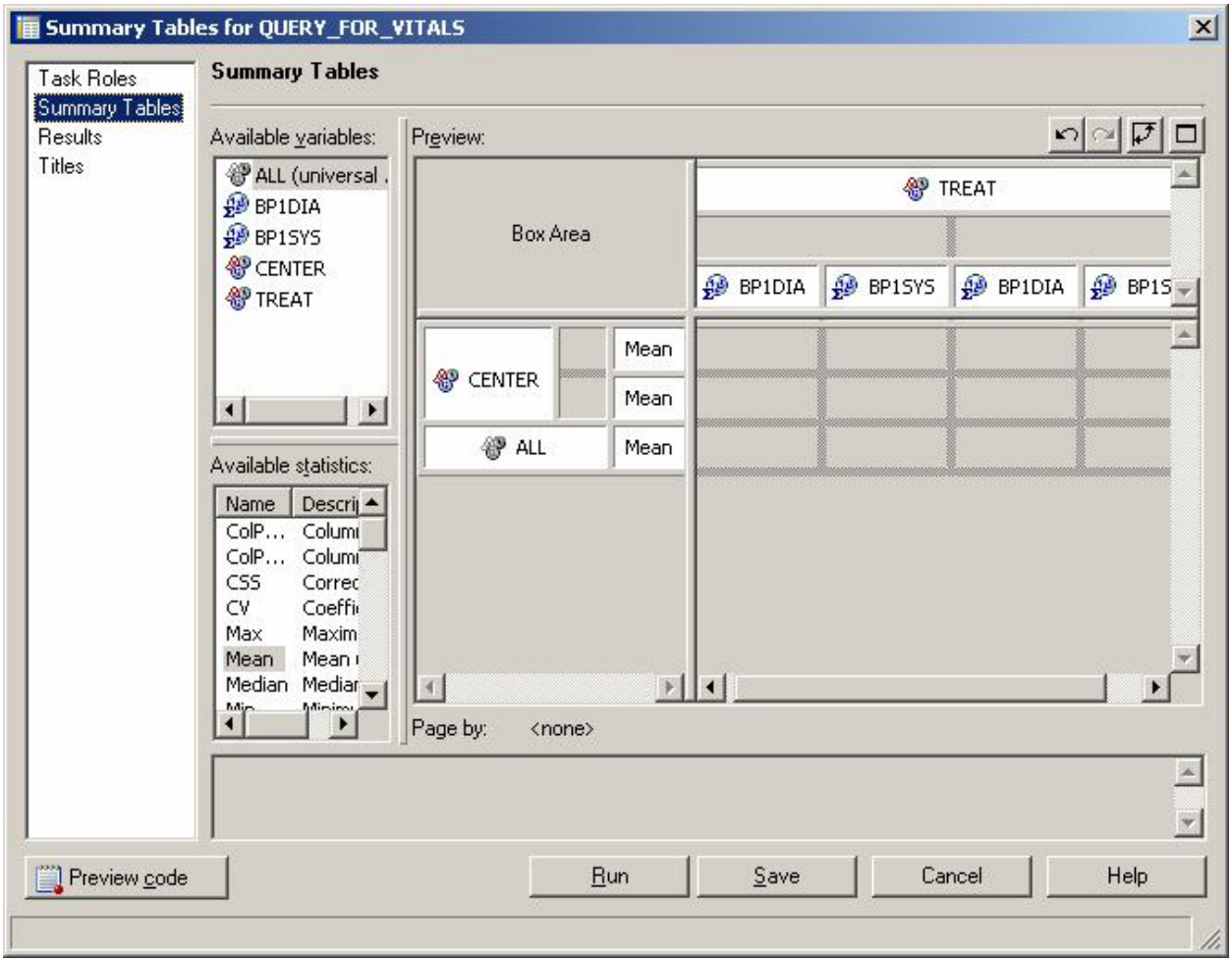
In the screenshot below the analysis and classification variables are selected by drag and drop and the order of the output can be selected for the class levels.



The next screenshot shows the empty palette for the table. Variables (or All) can be dragged onto the palette, and icons appear to show where the items are being placed (above, left, right, below). Statistics are available to drag onto the palette from the left.



The next screenshot shows the completed table for the example, created in a few seconds. If the individual elements of the table are selected a right mouse gives a menu of formatting options to further customize the table such as data value format, borders and shading, and header values.

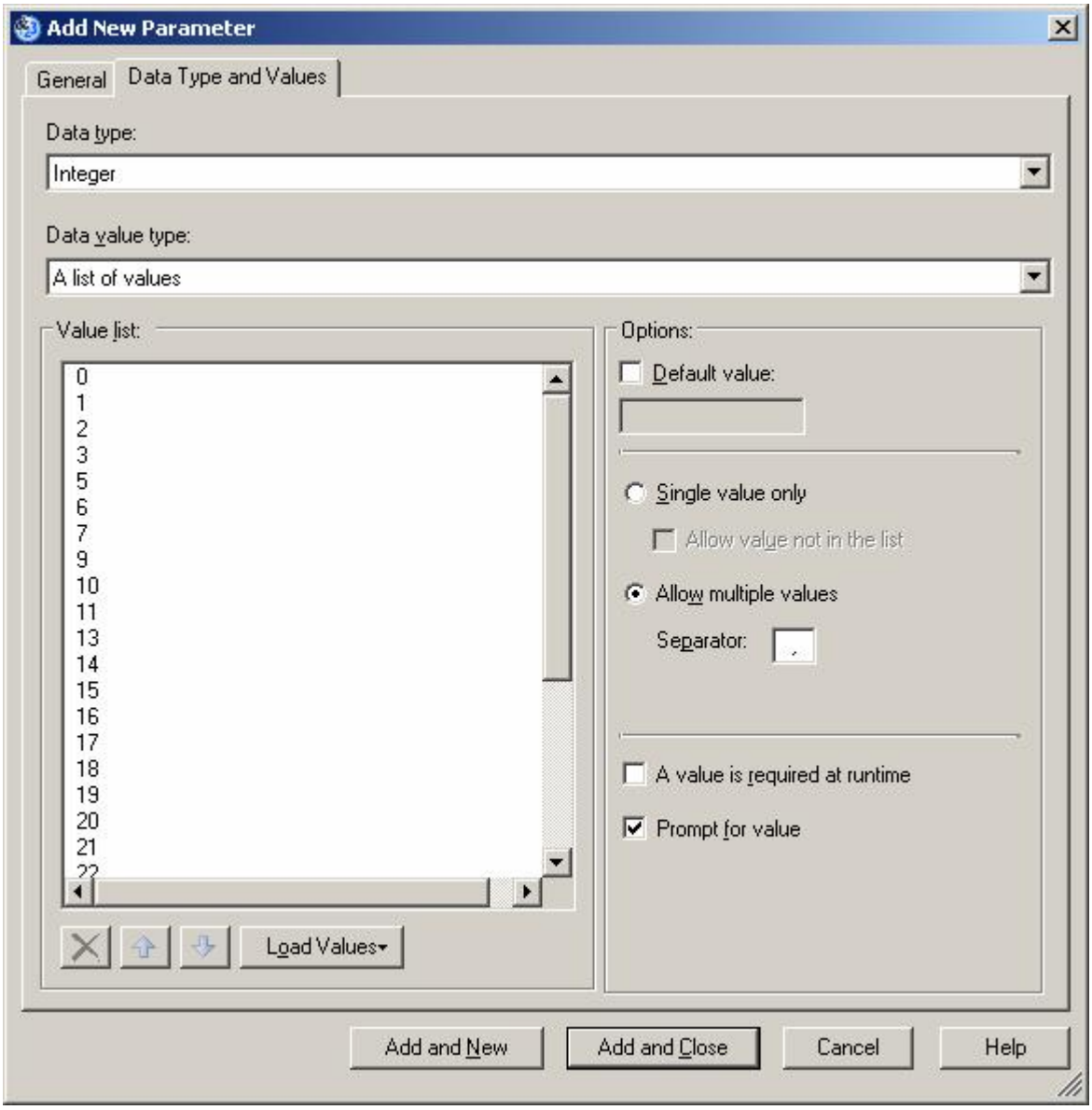


The resulting table is shown below.

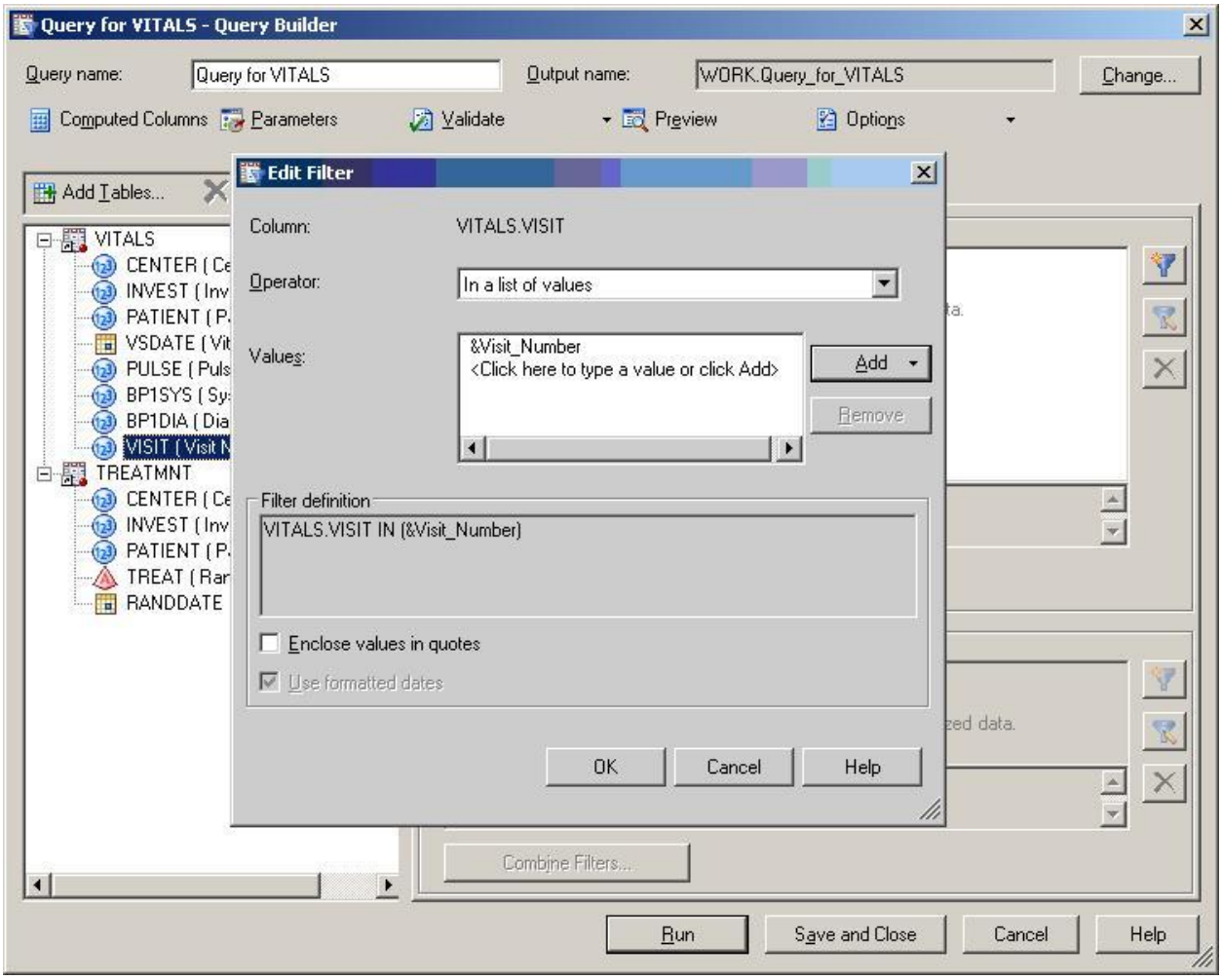
		Randomized treatment assignment			
		Drug A		Drug B	
		Diastolic Blood Pressure	Systolic Blood Pressure	Diastolic Blood Pressure	Systolic Blood Pressure
Center					
145	Mean	72.4	112.7	73.9	114.5
227	Mean	71.5	112.7	70.9	113.9
379	Mean	73.2	112.8	73.5	114.3
All	Mean	72.7	112.8	73.3	114.3

PARAMETERS

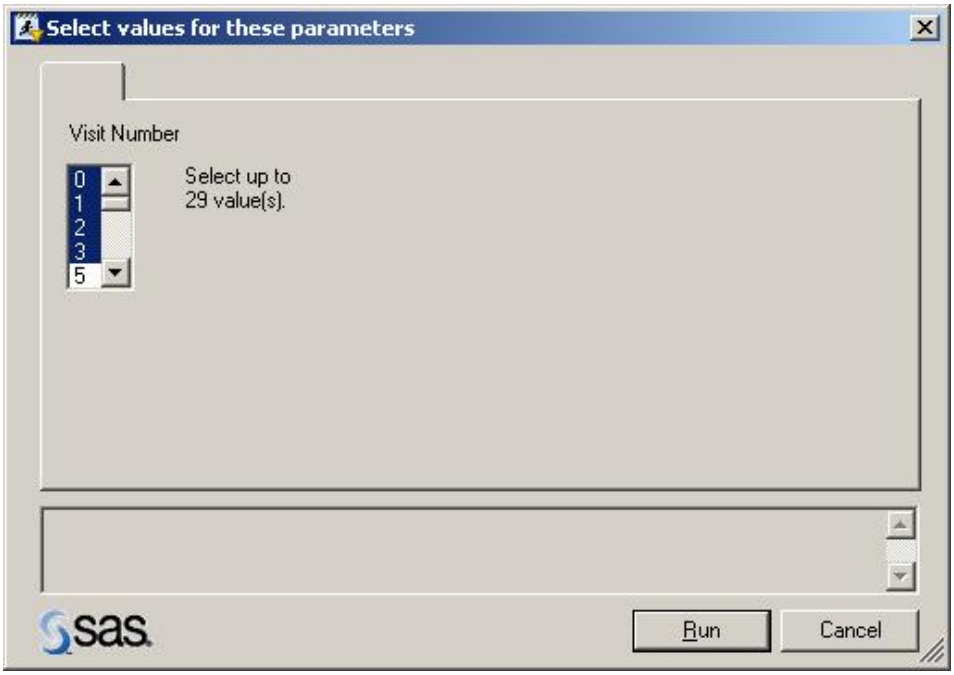
Parameters were introduced in Enterprise Guide 4.1 and allow the creation of macro variables which can be used across the project. For example, if the previous example was to be limited by visit numbers, a parameter of visit number could be created. Once the variable name is selected, the data type can be selected and optionally the list of valid values can be loaded up from the input data sources.



The query can now be augmented with a filter, selecting the new parameter from drop-down list.



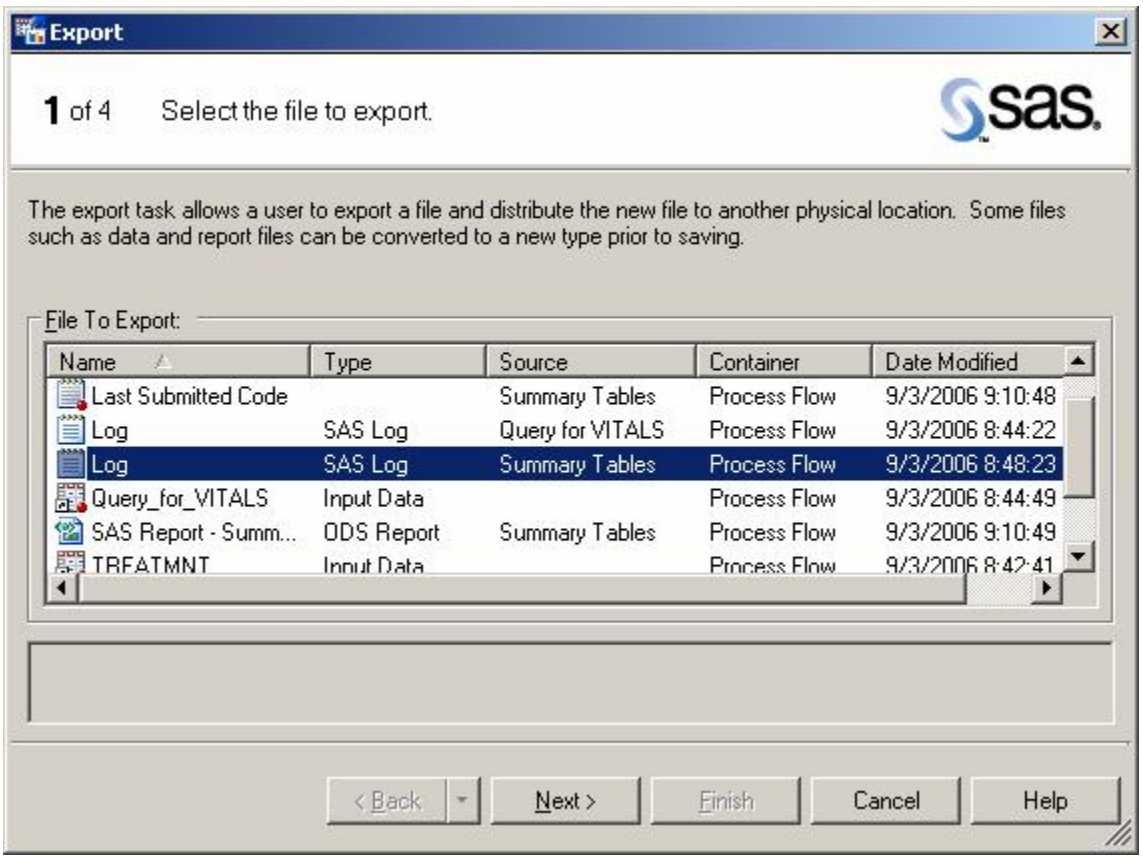
When the query is re-run, a dialog box appears



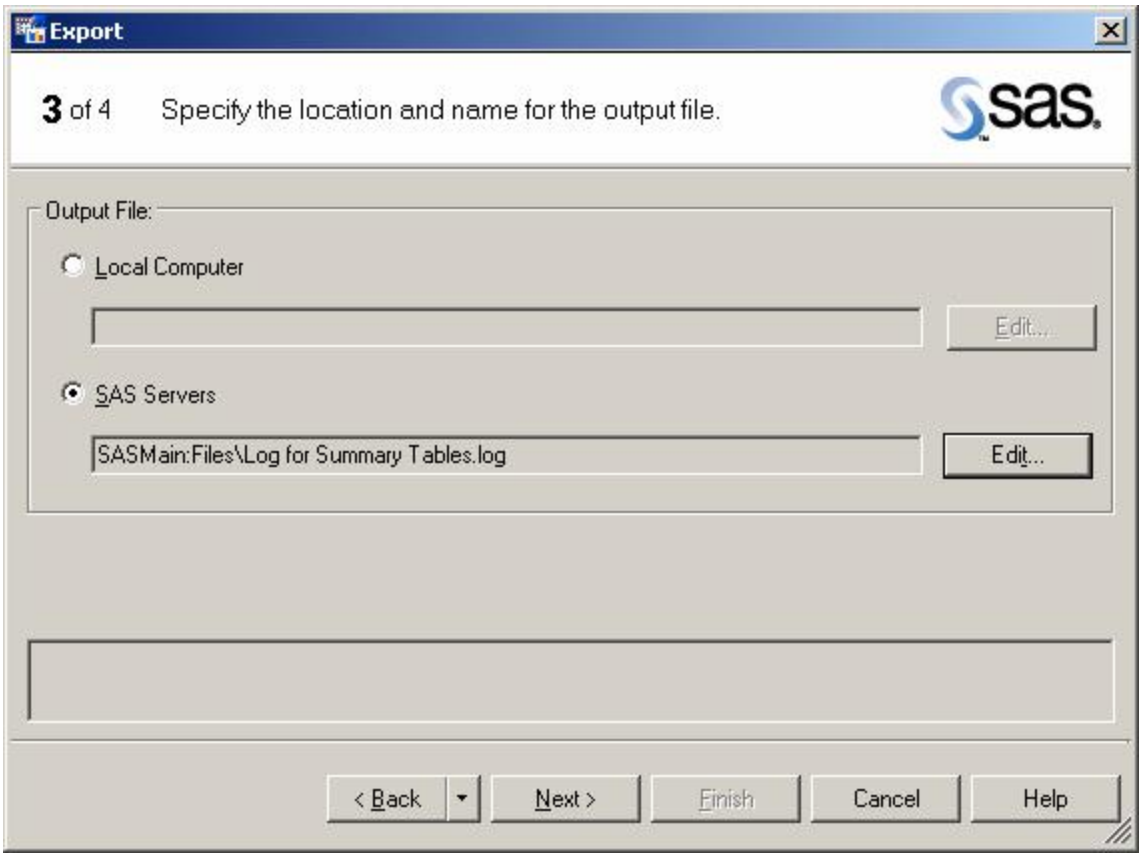
When this is run the subset is applied to the whole process. The parameter can also be used elsewhere, for example in a title. When a stored process is created from this the parameter is automatically assumed to be a parameter in the stored process.

COMPLIANCE FEATURES

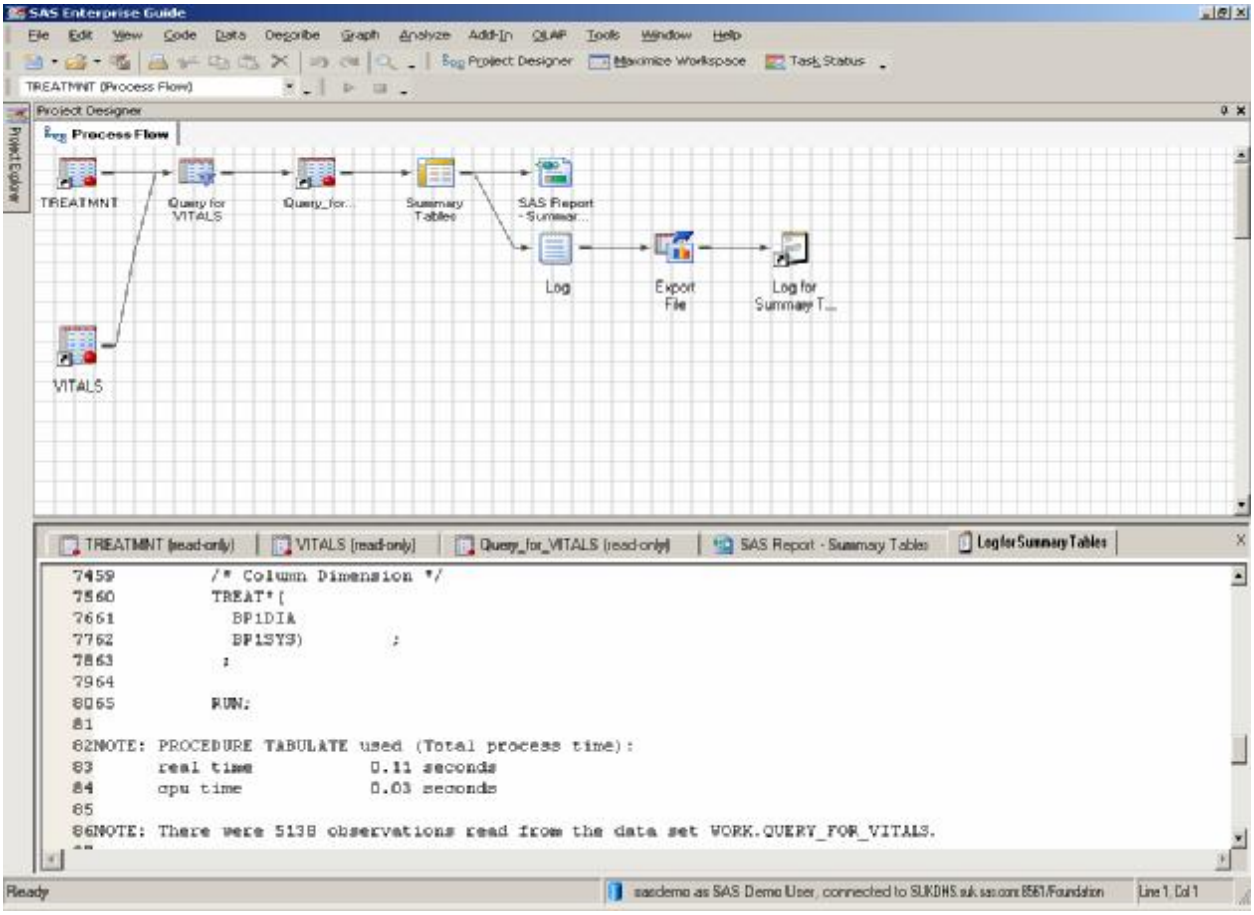
Enterprise Guide has always made the SAS log available as part of the associations of the icons in the process flow, but this is less useful in a pharmaceutical context, because the code, log and output usually need to be stored on a server or shared drive in a defined location, making the use of Enterprise Guide projects nearly impossible. Enterprise Guide 4.1 introduces the concept of exporting components as a step in the project. This applies to the log, the last submitted code, the output and the input data. The wizard guides users through the export process with appropriate options for the output types (i.e. .log or .txt for log files, several options for data export) and then gives options for output location.



The options include saving to the SAS server in the Files location (which can be customized)



The export of the log then appears as a step in the process flow.



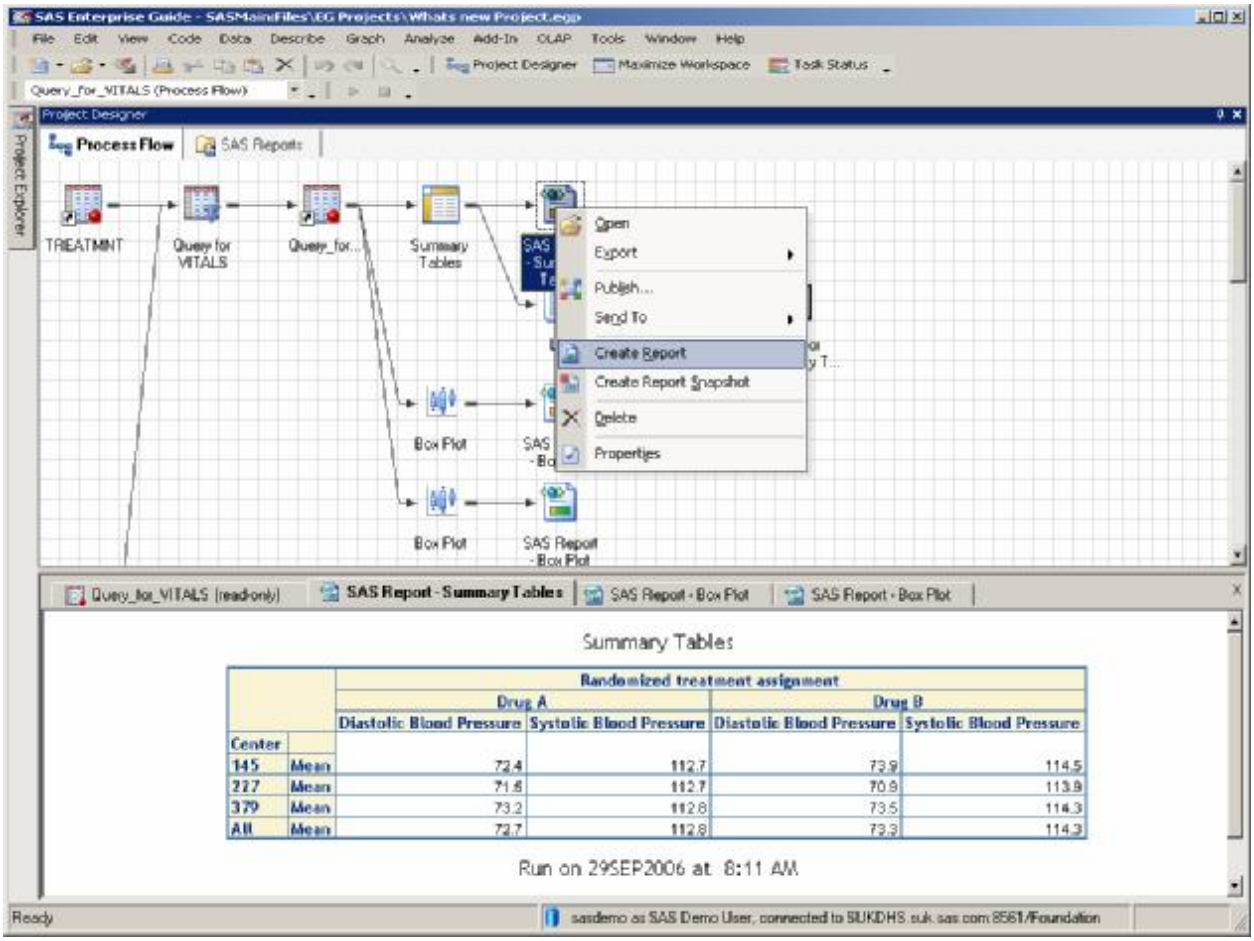
Enterprise Guide 4.1 also introduces the project log, where the log of the entire project is recorded. This can then be exported.

This is still different from the traditional methods of observing compliance common in most pharmaceutical companies, where a single program creates a single log and a single set of output; what this does do is allow the use of Enterprise Guide projects in the compliant environment. This could be useful where a set of reports are regularly run by data managers that require parameter selection (for example of centre or patient) but there is insufficient complexity or regularity of use to justify the effort in application development to provide a front-end.

THE REPORT OBJECT

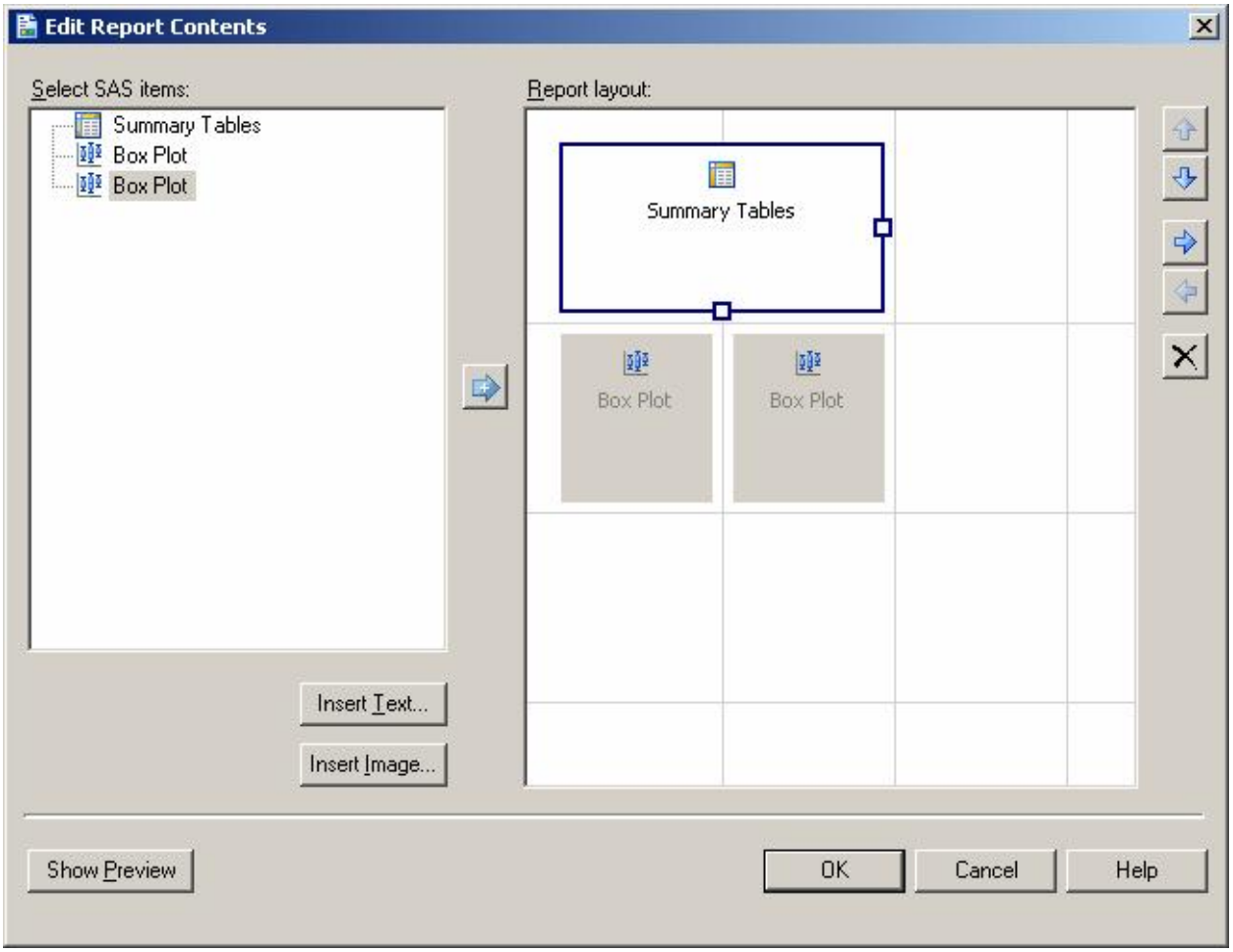
The report object was introduced with Enterprise Guide 4.1, and enables reports created from different SAS procedures to be combined in a structured layout and embedded images and text. With SAS 9.1.3 Service pack 4 these reports can then be published directly to SAS Web Report Studio (which is available as part of the SAS BI Server offering).

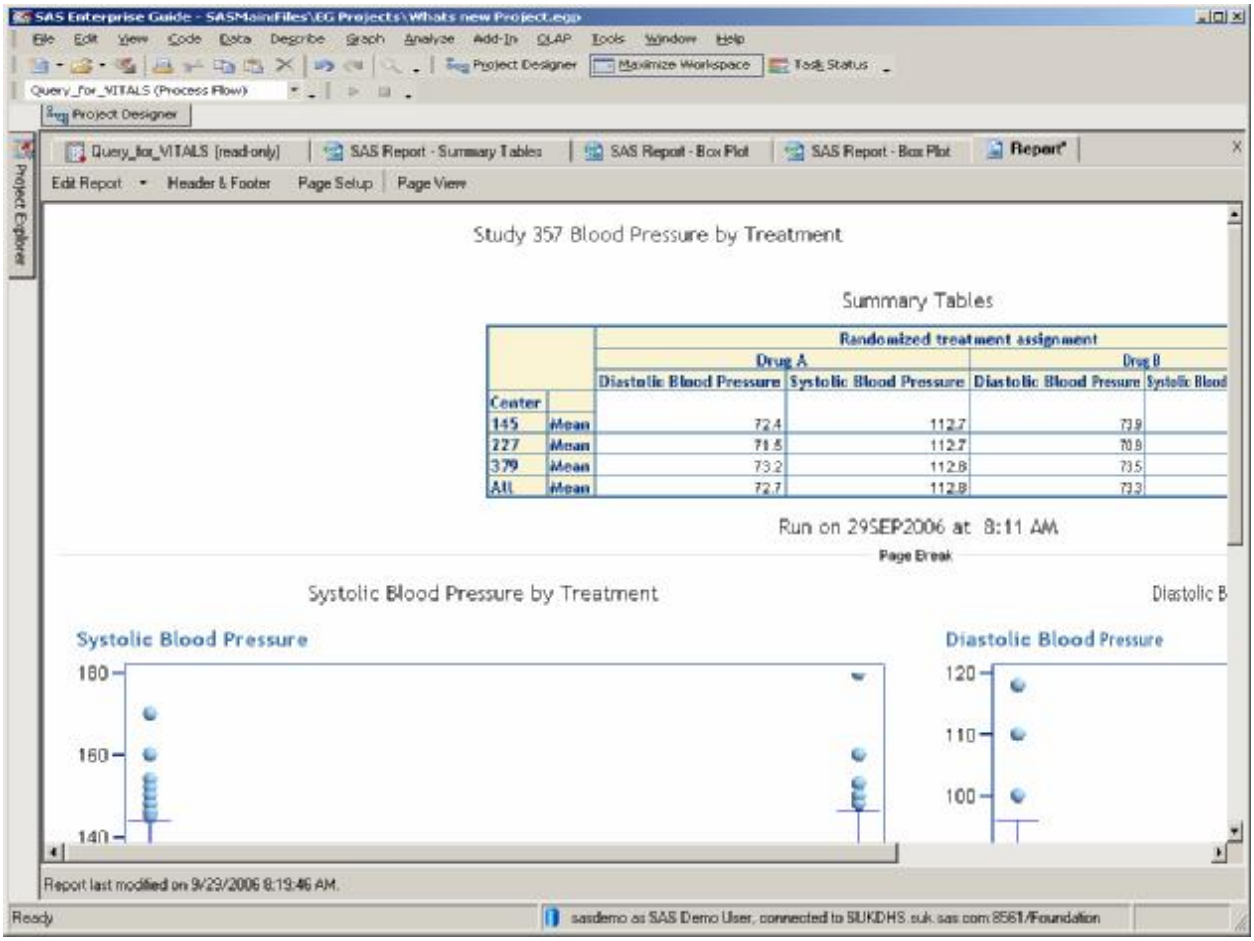
For example, in the previous example the report object was selected as the output type, so a right-click brings up the option to create a report.



This allows the report to be edited. Other report objects from the project can then be dragged onto a design palette.

Items can be stretched across different parts of the grid to achieve custom layouts, text and images can be inserted into the report and custom headers and footers can be added.


Edit Report Contents
Select SAS items:
Summary Tables
Box Plot
Box Plot
Report layout:
Summary Tables
Box Plot
Box Plot
Insert Text...
Insert Image...
Show Preview
OK
Cancel
Help



The report can then be published to Web Report Studio, through a simple wizard, and the report is then instantly available to Web Report Studio or Portal users.

STORED PROCESSES

Whereas reports created as report objects have static content, SAS provides the capability for dynamic content through stored processes. A stored process is an encapsulation of code and data, with the possibility of using additional user parameters to select the analysis to run. These parameters appear in the code as macro variables and are substituted in the manner you would expect in SAS Macro.

An Enterprise Guide project or set of project steps can be turned into a stored process using a wizard, and the stored process is then available across the enterprise through Enterprise Guide, Web Report Studio, the Information Delivery Portal or the Add-in to Microsoft Office. In each of these an automatic parameter window will appear if required.

A security model is applied to stored processes, controlled by the SAS Management Console. This ensures that individuals only see the reports and data to which they are entitled.

An example of where this might be used by clinical programmers is to provide a parameterized report to data managers to give details of study status without having to develop an application around the report.

ADD-IN TO MICROSOFT OFFICE

The Add-in to Microsoft Office (which is available as part of the SAS BI Server offering) provides a SAS tab in the menu bar of Microsoft Word, Excel and PowerPoint to bring the power of SAS to the Windows desktop. Reports and analyses can be run from the menu on either SAS data or the contents of an Excel spreadsheet. These analyses are identical to those available through Enterprise Guide, and subject to the same restrictions (i.e. if a user is prevented from running complex statistical routines in Enterprise Guide they will also not see them in Microsoft Office). Additionally stored processes can be run into the spreadsheet, presentation or document. This is where the application for clinical programming has greatest potential; the in-text tables for a submission can be surfaced as a sequence of stored processes. These can then be refreshed in the document as the data becomes finalized without altering the text around the tables, giving a great efficiency saving.

CONCLUSION

There are a great many enhancements in recent releases of SAS for the clinical programmer, and this is only a small subset. Most of the immediate benefit comes from Base SAS and Enterprise Guide, which (depending upon architectures) should be readily available to most existing clinical programmers on upgrading to SAS 9.1.3 SP4.

RECOMMENDED READING

An introduction to ODS Stat graphics http://www2.sas.com/proceedings/sugi29/204-29.pdf

CONTACT INFORMATION

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

Dave Smith
SAS Institute
Wittington House
Henley Road
Medmenham
Marlow
Bucks
SL7 2EB

Work Phone: 01628 404379
Fax: 01628 490550
Email: david.smith@suk.sas.com
Web: http://www.sas.com