Questions & Answers
OASUS Fall 2006 General Meeting

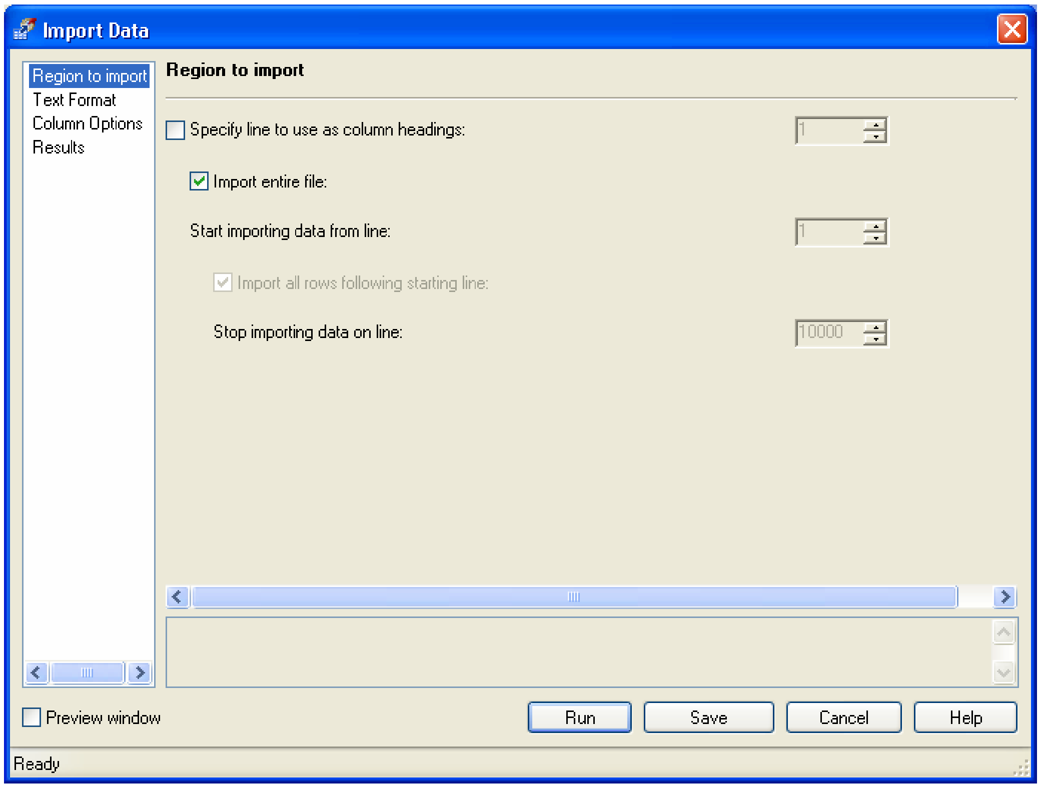
Question #1 How do I access flat files in Enterprise Guide?

To access flat files in Enterprise Guide 4.1, you have to first convert those files into a format that SAS can use to process the data. To do so, we propose three approaches.

1) Import Data Task

The Import Data task allows you to create SAS data sets from flat files (but also from PC-based database files). The Import Data task generates DATA step code. Various options can be set for each file to import:

Region to import



Import Data

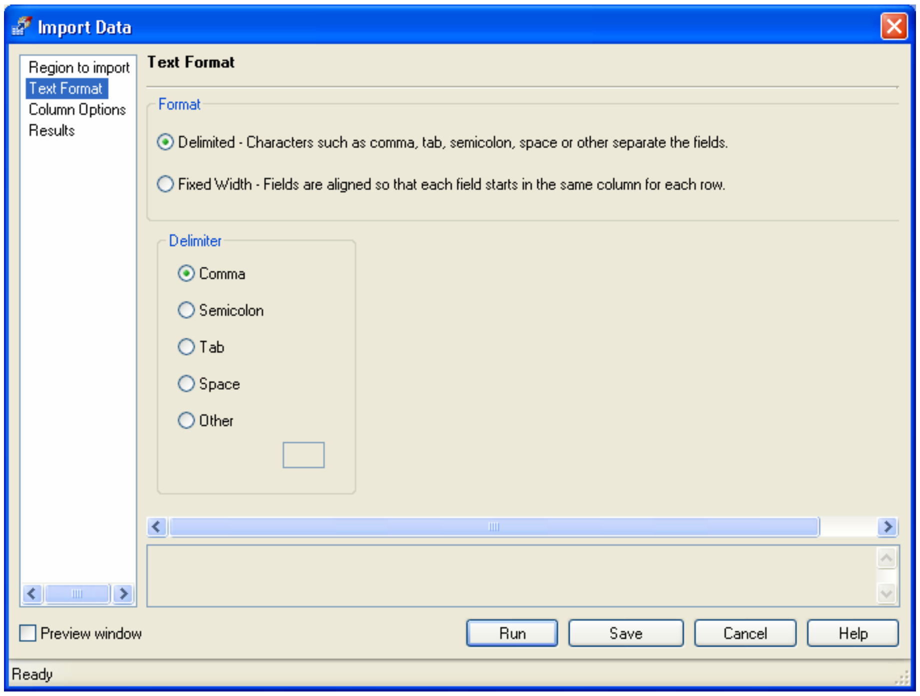
Region to import
Text Format
Column Options
Results

Region to import

- [ ] Specify line to use as column headings: 1
- [✓] Import entire file:
- Start importing data from line: 1
- [✓] Import all rows following starting line:
- Stop importing data on line: 10000

[ ] Preview window

Format of the text



[✓ Delimited - Characters such as comma, tab, semicolon, space or other separate the fields.]

Fixed Width - Fields are aligned so that each field starts in the same column for each row.

[✓ Comma]

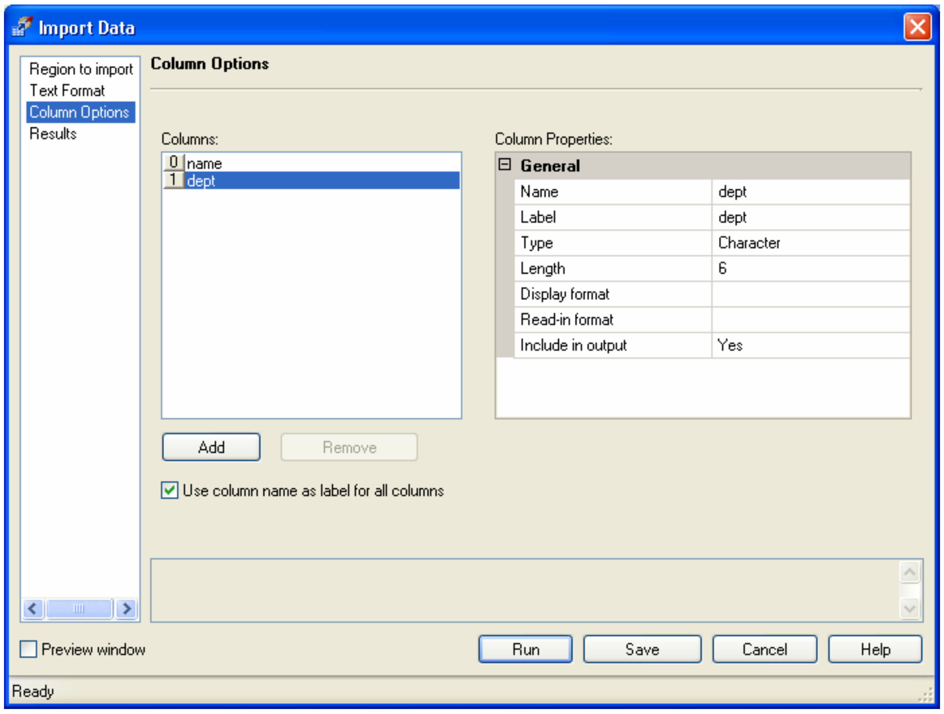
Semicolon

Tab

Space

Other

Column specifications.



Columns:

0	name
1	dept

Column Properties:

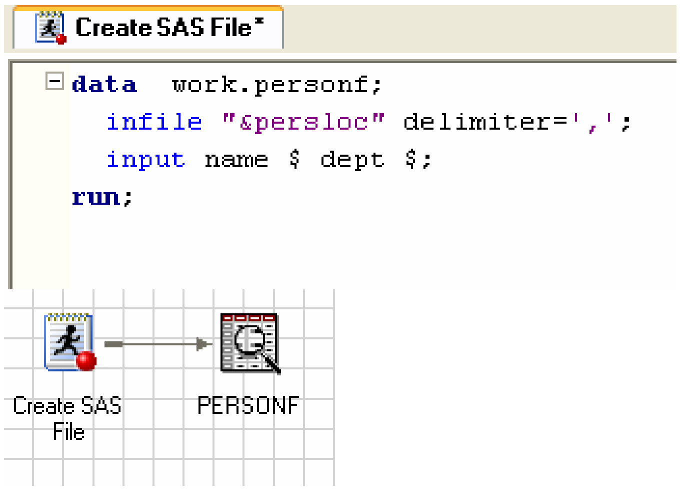
General

Name	dept
Label	dept
Type	Character
Length	6
Display format	
Read-in format	
Include in output	Yes

[✓ Use column name as label for all columns]

2) Data File created with code

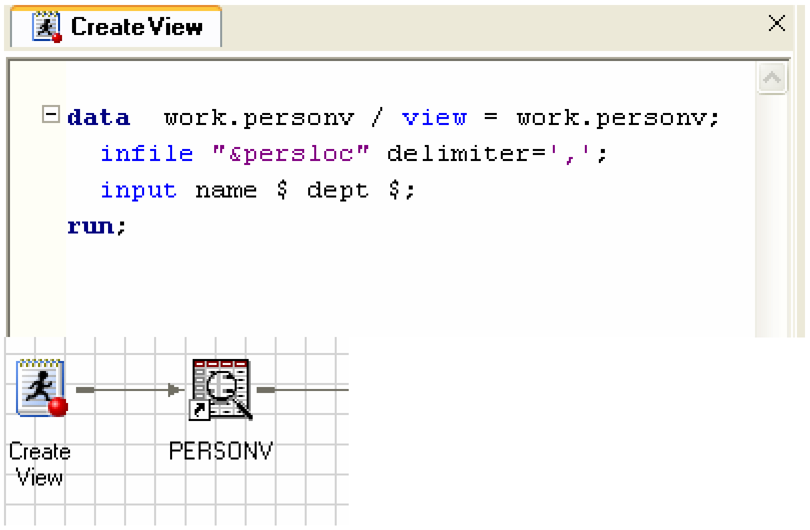
Enterprise Guide provides a facility to write and execute any SAS programs. Therefore, it is possible to write a DATA step by hand to achieve the same results. For example:



```
data  work.personf;
   infile "&persloc" delimiter=',';
   input name $ dept $;
run;
```

3) DATA step view

This is similar to approach #2, but is more dynamic. A DATA step view stores only the description of the data to output in the form of a DATA step. When a view is accessed, the associated DATA step gets executed. To access a flat file, all is required is to open a previously defined DATA step view. The advantage of a view is that the most current data is always accessed.



```
data  work.personv / view = work.personv;
   infile "&persloc" delimiter=',';
   input name $ dept $;
run;
```

Question #2 Can I assign an array to a macro variable?

An array is a set of elements that can be processed as a group. Typically, you would refer to elements of an array by the array name and a subscript. Although the macro language does not support arrays, it is possible to mimic arrays using different techniques. Two popular techniques are discussed here.

Technique #1: List of values in a macro variable.

This technique stores a list of values separated with some character in a single macro variable. The advantage of this approach is that the array elements are hold into one and only one variable. However, the array is limited by the maximum size macro variables can take (in SAS 9.1.3 for Windows, 64k is the current storage limit for in-memory macro variables) and care must be taken to break and scan the elements.

The following example illustrates the techniques:

Data Structure

	%SCAN(					)
	&ARRAY,1	&ARRAY,2	&ARRAY,3	&ARRAY,4	&ARRAY,5	&ARRAY,6
&ARRAY	10	20	30	40	50	60

Macro

```
/* This macro list each element of a macro array
implemented as a list of values on the SAS log */
%macro ListArray(array);

    %let i=1;

        %do %while (%scan(&array,&i) NE %str( ));
          %put array element &i = %scan(&array,&i);
          %let i=%eval(&i+1);
        %end;

%mend;
%ListArray(10  20   30  40   50   60);
```

Results

```
array element 1 = 10
array element 2 = 20
array element 3 = 30
array element 4 = 40
array element 5 = 50
array element 6 = 60
```

References

Carpenter, A. (2005), “Storing and Using a List of Values in a Macro Variable”, *Proceedings of the Thirtieth Annual SAS Users Group International Conference*, 028-30.

Technique #2: List of macro variables.

Like arrays in the DATA step, this approach utilizes one variable per array element. Each variable that makes up an array follows a common naming convention so that the array index becomes embedded in the macro variable name (for example &myvar1 &myvar2 &myvar2). A special macro variable is also required to indicate the number of elements in the array (for example &myvarn). The following figure illustrates the concept:

Data Structure

&MYVAR1	&MYVAR2	&MYVAR3	&MYVAR4	&MYVAR5	&MYVAR6
10	20	30	40	50	60

&MYVARn	6

Macro

```
/* This macro list each element of a macro array
implemented as a list of macro variables on the SAS log
*/

%macro ListArray( Array, Dim);
       %do i=1 %to &Dim;
         %put array element &i = &&&Array&i;
       %end;
%mend;

%let MyArray1=10;
%let MyArray2=20;
%let MyArray3=30;
%let MyArray4=40;
%let MyArray5=50;
%let MyArray6=60;
%let MyArrayDim=6;

%ListArray(MyArray,&MyArrayDim);
```

Results

```
array element 1 = 10
array element 2 = 20
array element 3 = 30
array element 4 = 40
array element 5 = 50
array element 6 = 60
```

The advantage of this approach is that the array element can hold almost anything and is only limited to the amount of memory available to the SAS session. The main disadvantage is that the number of macro variables can become very large.

References

Carpenter, A. (), “Resolving and Using &&var&i Macro Variables”, *Proceedings of the 22nd Annual SAS Users Group International Conference*, 077-22.

Fehd, Ronald (2004), “Array: Construction and Usage of Arrays of Macro Variables”, *Proceedings of the 29th Annual SAS Users Group International Conference*, 070-29.

Clay, Ted (2006), “Tight Looping With Macro Arrays”, *Proceedings of the 31st Annual SAS Users Group International Conference*, 040-31.

Question #3 How to run SAS in batch on a workstation when all the programs and the data files are on a remote server?

First, to run SAS in batch, you have to invoke the SAS executable using a command with the following notation:

```
"SAS Executable"
    -CONFIG "<location of SAS Config file>"
    -SYSIN “<location of SAS program to run>"
    -LOG "<location of SAS log>"
```

Second, you use the SYSIN option to specify the location of a SAS program to run which can be anywhere on the network as long as it is accessible by SAS. Note that there are many other options that can be specified on the command line.

The following example is for the Windows platform and invokes SAS in batch to run a program stored on a shared drive:

```
"C:\Program Files\SAS\SAS 9.1\sas.exe"
      -CONFIG "C:\Program Files\SAS\SAS 9.1\nls\en\SASV9.CFG"
      -SYSIN "\\filer10\team9\sas\test.sas"
      -LOG "c:\temp\mylog.log"
```

Finally, accessing data files can be done by providing SAS with a fully qualified file name where the data resides. Typically, this is done with FILENAME statement. In the case of remote files, you have to provide the location on the network where the file resides. In Windows, you would need network drive or, preferably, a fully qualified name that follows UNC conventions.

For example:

```
FILENAME myfile  "\\filer10\team9\data\test.txt" ;
```

Question #4 How to export 2 SAS variables in Excel into one column with a character to separate the two?

The first step is to concatenate the two variables using a DATA step. The following example implements a solution with the CATX function.

```
DATA test;

  set sashelp.class;
  sexname= catx('-',sex,name);

run;
```

The second step is to export the resulting dataset to an Excel spreadsheet. In the following example, this is achieved using the EXPORT procedure.

```
PROC EXPORT DBMS=EXCEL DATA= test
            OUTFILE= "c:\temp\textexcel.xls" REPLACE;
RUN;
```

Question #5 How to minimise space on SAS Data Set?

There are several techniques to reduce the space on a SAS data set. We are showing here three techniques that are particularly effective. They can be used alone or in combination depending on the situation. In all the examples used to illustrate those techniques, we use a data set called BIG created as followed:

```
data BIG;
      do i=1 to 10000;
            code1=1;
            code2=2;
            code3=3;
            code4=4;
            code5=5;
            texta="textaaaaaaaaaaaaa";
            textb="textbbbbbbbbbbbbb";
            textc="textccccccccccccc";
            textd="textddddddddddddd";
            texte="texteeeeeeeeeeeee";
            text=cat(texta,textb,textc,textd,texte);
            output;
      end;
run;
```

1) Turn compression on

Using the COMPRESS= system or data set option, any SAS data set created on disk will be compressed. SAS data set compression can greatly reduce the size of SAS data sets. To use the COMPRESS= system or data set option, set the option to either "YES" or "BINARY". COMPRESS=YES uses an algorithm that works better with SAS data sets that primarily have character variables. On the other hand, COMPRESS=BINARY uses a different algorithm that works better with SAS data sets that have many variables including many numeric variables.

Another option to use with COMPRESS= is REUSE=. Specifying this option allows SAS to reuse space within the compressed SAS data set that has been freed by deleting an observation. Otherwise, SAS cannot reclaim the deleted space.

The following example shows how to compress an existing data set. Running this example on the Windows platform shows a 63% compression rate.

```
data BIG_COMPRESSED (COMPRESS=YES);
      set BIG;
run;
```

2) Reduce the length of a variable

Reducing the length of a variable to just what is needed to store the data can be done using the LENGTH statement. It is particularly important in the case of numeric variables because SAS by default uses a length of 8 bytes, but you may need much less to represent numbers in your application. With character variables, SAS may be able to derive the optimum length prior to running the DATA step. If not, SAS will use a length of 200 characters which may be way too much. Again, using the LENGTH to specify the maximum number of bytes needed may yield very interesting results.

The following example optimises the length of 5 numeric variables and of one character variable that would be otherwise assigned a length of 200. This yields a reduction in storage requirement of 66%.

```
data BIG_OPTIMISED;
   length code1 code2 code3 code4 code5 3 ;
   length text $100;
   set big;
run;
```

3) Retain only the required variables

Because SAS by default outputs all variables, it is easy to end up with way too many variables stored in a data set than is required. It is recommended to delete any variables that you do not need. Use the data set DROP= option to identify which variables to delete or use the KEEP= option to identify which variables to retain. Both will accomplish the same thing, one will be easier to use than the other depending on the number of the existing variables you want to eliminate. Deleting unneeded variables can have a dramatic impact on the size of the SAS data set. In the following example, dropping 5 character variables reduces the storage requirements by 30%.

```
data BIG_REDUCED (DROP=i texta textb textc textd texte);
  set BIG;
run;
```

References

Lafler, Kirk Paul (2000), “Efficient SAS® Programming Techniques”, *Proceedings of the 25th Annual SAS Users Group International Conference*, 146-25.

Smith, Curtis A. (2000), “By Hook or By Crook: Overcoming Resource Limitations”, *Proceedings of the 25th Annual SAS Users Group International Conference*, 291-25.

Smith, Curtis A. (2000), “Programming Tricks For Reducing Storage And Work Space”, *Proceedings of the 27th Annual SAS Users Group International Conference*, 23-27.