How Smooth is (PROC) SYLK?

Howard Schreier, U.S. Dept. of Commerce, Washington DC

Abstract

PROC SYLK (an experimental component of Version 9 of SAS® Software) allows SAS to import not just data, but also formulas and formatting, from Excel and other compliant spreadsheet applications. PROC SYLK also provides a programming language for a batch spreadsheet processing environment entirely internal to SAS. This paper will explain the basic operation of PROC SYLK, evaluate the programming language, and illustrate the procedure's strengths and weaknesses.

Introduction

Before there was PROC SYLK, there was SYLK. Symbolic link (SYLK) format is a set of conventions for representing the content of a computer spreadsheet in a text file. Thus, SYLK files are similar in nature and purpose to DIF (Data Interchange Format) files and to CSV (comma-separated value) files. However, SYLK files incorporate both cell values and formulas (DIF files record one or the other, depending on what each cell is set to display when the conversion occurs; CSV files record values only). SYLK files also preserve unrounded values with formatting information to facilitate rounding for presentation.

The SYLK format was developed by Microsoft in the 1980's. It has not evolved, and thus lacks support for some spreadsheet features which came later. It is also poorly documented. There is a brief technical description which appears to have originated within Microsoft and is now available on a number of third-party websites, but not on microsoft.com; see the References. Throughout this paper, "SYLK" not preceded by "PROC" refers to this text file structure; only references to "PROC SYLK" describe or characterize the SAS procedure.

Historically, SYLK has been supported as an input and/or output format by a number of applications. Since PROC SYLK only reads SYLK files and does not write them, we are interested only in those applications which create SYLK files. Excel is among these, and since it is likely the SYLK-capable application of most interest to SAS users, the rest of this paper will deal with SYLK files generated by Excel to the exclusion of all other SYLK-capable applications.

The terms "format" and "formatting" have different meanings in different contexts. I will try to clarify by using the term "content formatting" to refer to rules for converting numeric values to character strings and the term "appearance formatting" to refer to aspects such as typography (typeface, size, weight, etc.), color, shading and bordering, and so forth.

There are some fundamental differences between SAS and Excel which make the goal of PROC SYLK, to port Excel spreadsheet models into SAS implementations, something like pushing a square peg into a round hole.

1. In Excel, data values (constants) and formulas coexist in the characteristic grid structure (worksheet). SAS segregates the two: data values are stored in SAS datasets, while formulas are maintained in program code (using languages such as the DATA step language or SQL).

2. Excel permits cell (or even finer) granularity for data typing and for formatting attributes. SAS imposes uniformity within a column for both typing and metadata.

3. An Excel work product is likely to intersperse detail with summary data, and to include formatting to make the worksheet serve as the report. SAS solutions tend to treat detail data, summary data (such as subtotals and totals), and reports as separate entities.

4. In Excel, the formula dependency relationships among cells can be intricate, as long as cycles are avoided; in particular, the formula in a particular cell can reference cells which appear above or below it. SAS typically processes the observations in a dataset from top to bottom.

PROC SYLK is characterized by SAS as "experimental" in both Versions 9.0 and 9.1, and it clearly has rough edges. I presume that a fully supported production version would incorporate needed corrections and changes, so in the present discussion I will not dwell on such issues. Rather, I will look at the fundamental design, role, and

value of PROC SYLK.

All of the examples in this paper were processed using Version 9.1.2 in a Windows 2000 environment.

Overview

PROC SYLK is really two different things. (1) It is a mechanism for importing data, formulas, and formatting information which originated in an Excel spreadsheet into SAS; call this SYLK-import mode. (2) It is a programming language embedded in SAS; call this standalone mode. The two modes can be used separately, and later we will see examples of such usage. However, by invoking PROC SYLK twice, once in SYLK-import mode and once in standalone mode, one can port a model or report from a spreadsheet implementation to a SAS implementation. PROC SYLK supports this back-to-back sequence in that the first invocation generates the code for the second. The two modes and two steps are necessary because SAS, unlike Excel, does not store formulas and data together [Square Peg # 1].

This is the sequence of operations; parenthetical references are to symbols in Figure 1.

Begin with an Excel spreadsheet (1), stored in the native Excel file structure. Use Excel (A) to save it as a SYLK file (2). PROC SYLK (B) reads this file and creates three outputs:

- A SAS dataset (3) containing all of the constants, with missing values in the places where formula results would appear. The dataset header incorporates labels and numeric formats.
- PROC SYLK code (4) to calculate results for all of the formulas in the SYLK representation of the spreadsheet. The code is not in the form of a DATA step or any other established SAS language (such as SQL or IML). Rather, it is in PROC SYLK's own language. So, PROC SYLK actually programs itself.
- PROC TEMPLATE code (5), basically to take care of appearance formatting.

When the generated code is submitted, PROC SYLK (C) runs a second time and performs the calculations indicated by the formulas. The result is a second SAS dataset (6). Finally, a DATA step can generate a report (7) based on this dataset and the template.

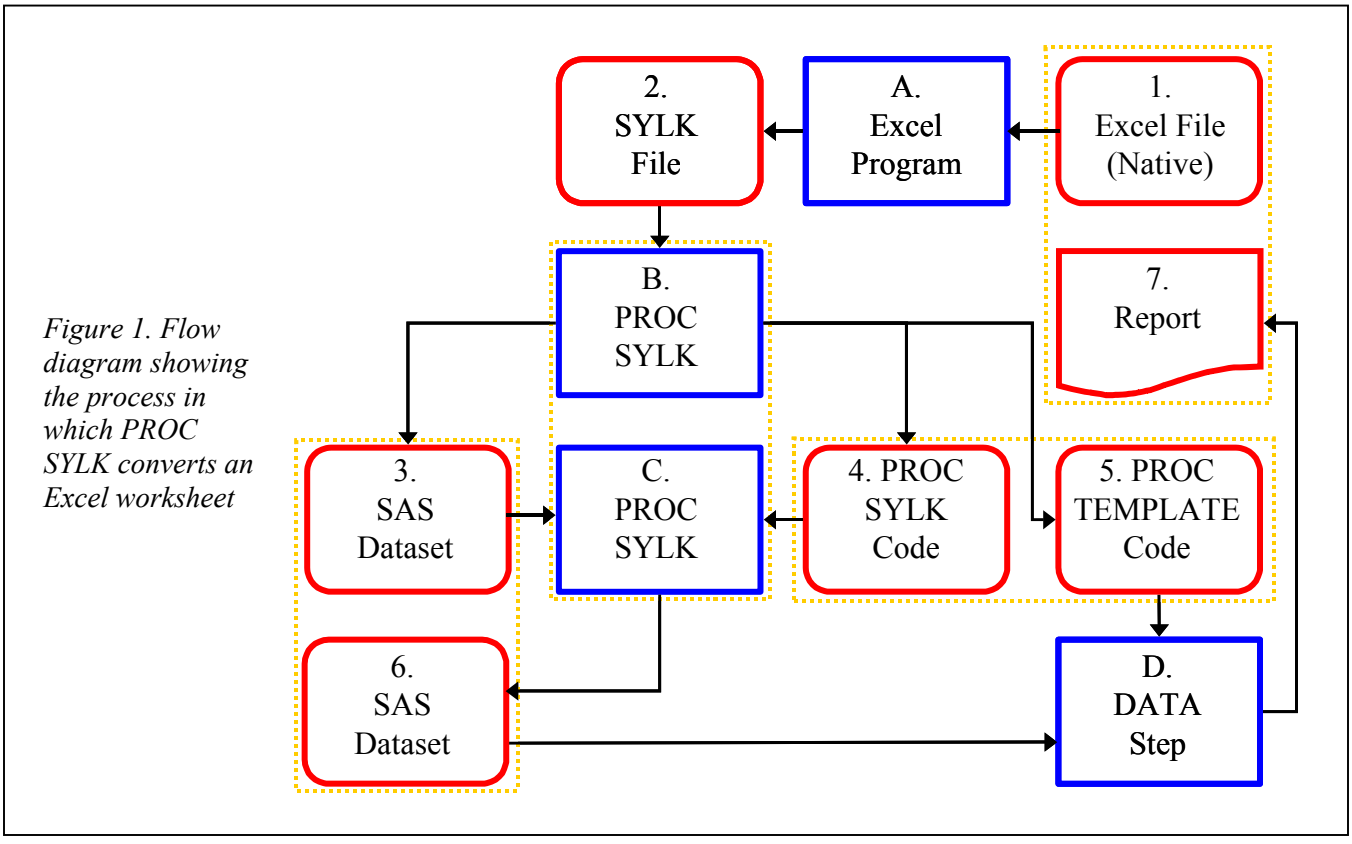

Figure 1. Flow diagram showing the process in which PROC SYLK converts an Excel worksheet

In looking at the overall process as depicted in this flowchart, note that PROC SYLK is run twice (B and C); that the two SAS datasets (3 and 6) have identical structures (number of observations, as well as variable names, types, and attributes); that the generated PROC SYLK and PROC TEMPLATE code (4 and 5) are actually stored in the same file; and that the report (7) more or less resembles the original Excel worksheet (1).

Here is a simple example. The original spreadsheet is shown in Figure 2. It is a very typical spreadsheet, but has been designed carefully to avoid pitfalls; notice the strict segregation of character and numeric values by column [Square Peg # 2] and the placement of the summary at the bottom rather than the top [Square Peg # 4].

The formula for the difference for February is

```
=D4-C4
```

Other formulas in that column are analogous. The formula for total is

```
=SUM(D3:D8)
```

All of the other values are constants.

The format string for the rightmost three columns is

```
0.0
```

and that for the first column is

```
mmm-yy
```

The SYLK file appears in Appendix 1. The exact conventions which it uses to represent the Excel worksheet are really not important to the PROC SYLK user. They are documented (see the References) and can be discerned to a degree simply by studying the file.

	A	B	C	D	E
1	Month		Predicted	Actual	Difference
2					
3	Jan-04		50.8	58.1	7.3
4	Feb-04		51.5	48.8	-2.7
5	Mar-04		52.3	58.2	5.9
6	Apr-04		53.1	64.6	11.5
7	May-04		53.9	67.1	13.2
8	Jun-04		54.7	64.0	9.3
9					
10		**Total**		**360.8**	

Figure 2. Screen clip showing original Excel worksheet used in Example 1

Here is the user-written PROC SYLK step which begins the process:

```
title 'Example 1, Part 1 of 2';

proc sylk;
read sylk    = intro
     out     = intro
     sasfile = intro ;
run;

proc contents data=intro; run;

proc print data=intro label;                run;

proc print data=intro      ; format _all_; run;
```

Recall that PROC SYLK has two modes of operation. In this case it is coded to operate in SYLK-import mode. The two modes are not toggled by an option in the PROC statement, as we might expect. Rather, it is the presence of the READ statement which controls the mode of operation. Its presence disables any other statements between the PROC statement and the RUN statement (or other end-of-step boundary).

The three parameters in the READ statement tell the procedure to read a SYLK file named INTRO.SLK from the current working directory, to create a SAS dataset named INTRO in the WORK library, and to store the generated code in a file named INTRO.SAS in the current working directory.

This is part of the PROC CONTENTS report:

```
Example 1, Part 1 of 2

     Alphabetic List of Variables and Attributes

#    Variable    Type    Len    Format     Label

1    x1          Num       8    monyy5.    Month
2    x2          Char      6
3    x3          Num       8    11.1       Predicted
4    x4          Num       8    11.1       Actual
5    x5          Num       8    11.1       Difference
```

Since PROC SYLK is operating in SYLK-import mode, the output SAS dataset reflects only spreadsheet cells which hold constants. The dataset does have “cells” (observation-variable intersections) for the formulas, but these are not populated (ie, they are missing values).

Here is the output from PROC PRINT, with the LABEL option in effect:

```
Example 1, Part 1 of 2

Obs    Month     x2       Predicted        Actual    Difference

 1         .                    .             .             .
 2     JAN04                 50.8          58.1             .
 3     FEB04                 51.5          48.8             .
 4     MAR04                 52.3          58.2             .
 5     APR04                 53.1          64.6             .
 6     MAY04                 53.9          67.1             .
 7     JUN04                 54.7          64.0             .
 8         .                    .             .             .
 9         .    Total           .             .             .
```

Here is the output from PROC PRINT, without the LABEL option and with stored formats disabled:

```
Example 1, Part 1 of 2

Obs      x1      x2       x3       x4     x5

 1         .                .        .       .
 2     16071             50.8    58.11       .
 3     16102             51.5    48.76       .
 4     16131             52.3    58.21       .
 5     16162             53.1    64.63       .
 6     16192             53.9    67.06       .
 7     16223             54.7    64.01       .
 8         .                .        .       .
 9         .    Total       .        .       .
```

Here is the generated code (INTRO.SAS):

```
/*- SAS file transferred from .SYLK file -*/

/*- create Table Template -*/
proc template;
  define table table.sylk;
    translate _val_=. into "";
    column x1 x2 x3 x4 x5 _ROW_;

    define x1; format= monyy5.; label= 'Month'; just= C;
    end;
    define x2; just= C;
      cellstyle _row_ = 9 as {  font_weight=bold },
                1 as {  foreground=black };
    end;
    define x3; format= 11.1; label= 'Predicted'; just= C;
    end;
    define x4; format= 11.1; label= 'Actual'; just= C;
      cellstyle _row_ = 9 as {  font_weight=bold },
                1 as {  foreground=black };
    end;
    define x5; format= 11.1; label= 'Difference'; just= C;
    end;
    define _ROW_; label='row'; end;

    header tabletitle;
    define tabletitle;
      text 'Imported SYLK File - intro.slk';
    end;

  end;
run;

proc sylk data=intro out=intro_out end=last;
  if _row_ ge 2 and _row_ le 7   then x5 = x4-x3;
  if _row_ eq 9 then do;
    x4 = SLKBLOCK("SUM","REL(x4,-7,x4,-2)");
  output;
  end;
  else output;
run;
```

There are two steps. The first is a PROC TEMPLATE step. It serves to tweak the appearance of the report pro-

duced later. It includes some code which pertains to content formatting; notice the suppression of the periods representing missing numeric values.

The CELLSTYLE statements deal with appearance formatting. They carry forward the boldface usage in the original spreadsheet.

The second step is a PROC SYLK step which takes as input the SAS dataset created in the SYLK-import phase and creates a second dataset in which placeholder missing values are replaced with the results of the formulas. The code is a bit clumsy, especially with regard to the OUTPUT statements. It is equivalent to:

```
proc sylk data=intro out=intro_out;
  if _row_ ge 2 and _row_ le 7   then x5 = x4-x3;
  if _row_ eq 9 then x4 = SLKBLOCK("SUM","REL(x4,-7,x4,-2)");
  output;
run;
```

Now it's a bit easier to see what's going on. The first conditional assignment statement handles the "Difference" column (X5) in the detail rows; the other handles the summary of the "Actual" column (X4). The unusual-looking SLKBLOCK function is discussed below.

So we run the generated code and display the results:

```
title 'Example 1, Part 2 of 2';

%include 'intro.sas';

proc print data=intro_out; format _all_; run;

ods html file='intro.htm';

data _null_; set intro_out;
file print ods=(template='table.sylk');
put _ods_;
run;

ods html close;
```

Here is the resulting dataset, first without benefit of labels and formats:

```
Example 1, Part 2 of 2

Obs      x1      x2       x3       x4        x5

 1         .               .        .         .
 2     16071              50.8     58.11      7.31
 3     16102              51.5     48.76     -2.74
 4     16131              52.3     58.21      5.91
 5     16162              53.1     64.63     11.53
 6     16192              53.9     67.06     13.16
 7     16223              54.7     64.01      9.31
 8         .               .        .         .
 9         .    Total      .     360.78       .
```

Imported SYLK File - intro.slk				
Month	x2	Predicted	Actual	Difference
JAN04		50.8	58.1	7.3
FEB04		51.5	48.8	-2.7
MAR04		52.3	58.2	5.9
APR04		53.1	64.6	11.5
MAY04		53.9	67.1	13.2
JUN04		54.7	64.0	9.3
	Total		360.8	

Figure 3 (left). Screen clip of the HTML output from Example 1

Figure 4 (right). Screen clip of the original Excel work-sheet for Example 2

	A	B	C
1	Month	Predicted	Actual
2	Jan-04	50.8	58.1
3	Feb-04	51.5	48.8
4	Mar-04	52.3	58.2
5	Apr-04	53.1	64.6
6	May-04	53.9	67.1
7	Jun-04	54.7	64.0

Figure 3 shows the ODS HTML output, reflecting the template which was automatically created.

Specialized Usage

PROC SYLK does not have to be used in the back-to-back fashion illustrated above. It can be used in SYLK-import mode only, to import data from Excel, or in standalone mode only, as a spreadsheet "flavored" programming language which uses SAS datasets for both input and output..

PROC SYLK as a Data Import Utility

This usage makes sense if the spreadsheet stored as a SYLK file contains constants only (no formulas). Figure 4 shows such a variation on our earlier example. Note the left-aligned value, which is text (that is, not a numeric value); this is asking for trouble [Square Peg # 2].

Here is the PROC SYLK code:

```
title 'Example 2';

proc sylk;
read sylk = justread
     out  = justread ;
run;

proc print data=justread; run;
```

Notice that in this case the SASFILE= option is not coded, because we do not need or want to the second-phase PROC SYLK code or the template. In fact, PROC SYLK will still generate the code and store it in a default location, but since no %INCLUDE is invoked automatically, it can be ignored.

This is the resulting SAS dataset:

```
Example 2

Obs       x1              x2              x3     x2_c

 1      JAN04            50.8            58.1
 2      FEB04            51.5            48.8
 3      MAR04               .            58.2    52.3
 4      APR04            53.1            64.6
 5      MAY04            53.9            67.1
 6      JUN04            54.7            64.0
```

Notice how the non-numeric value ("52.3") is stored in a separate variable. The "_c" suffix indicates that it is a character variable.

PROC SYLK as a Programming Tool

Now we will see the usage of PROC SYLK as a standalone programming language, not connected in any way to a SYLK file. Instead, the demonstration data will be loaded in a DATA step:

```
title 'Example 3';

data justcalc;
input  Month : monyy7. Predicted Actual     ;
format month   monyy7. predicted actual 8.1 ;
cards;
Jan2004  50.8  58.11
Feb2004  51.5  48.76
Mar2004  52.3  58.21
Apr2004  53.1  64.63
May2004  53.9  67.06
Jun2004  54.7  64.01
;
```

Let PROC SYLK calculate the monthly differences between ACTUAL and PREDICTED, and a total for ACTUAL.

```
title 'Example 3A';

proc sylk data=justcalc out=justcalc_out1 end=last;
Stub = put(month,monyy7.);
drop       month;
Difference = actual - predicted;
format difference 8.1;
output;
if last then do;
   output /;
   stub = 'Total';
   predicted = .;
   actual = SLKBLOCK("SUM","REL(actual,-5,actual,0)");
   difference = .;
   output;
   end;
run;

proc print data=justcalc_out1; id stub; run;
```

There are a few departures from what we saw earlier (in the PROC SYLK code generated by PROC SYLK in Example 1). The months are converted to character values so that they can be combined in a single column (STUB) with the "Total" label. Because the two extra rows (the one for the total and the empty one preceding it) are not present in the input, we let end-of-file trigger a conditional block of code to generate them. We also have appropriate variable names, so we dispense with variable labels. Here is the result:

```
Example 3A

Stub       Predicted      Actual    Difference

JAN2004         50.8         58.1          7.3
FEB2004         51.5         48.8         -2.7
MAR2004         52.3         58.2          5.9
APR2004         53.1         64.6         11.5
MAY2004         53.9         67.1         13.2
JUN2004         54.7         64.0          9.3
                   .            .            .
Total              .        360.8            .
```

However, consider that this approach is simply not SAS-like. Why not make the PROC SYLK step simpler, take advantage of a SAS reporting tool to do some of the work, and create a dataset which is suitable for general use (unlike the one which has an empty observation and summary data grafted onto it) [Square Peg # 3].

```
title 'Example 3B';

proc sylk data=justcalc out=justcalc_out2;
Difference = actual - predicted;
format difference 8.1;
output;
run;

proc print data=justcalc_out2; sum actual; run;
```

Of course this could have been done in a DATA step or a PROC SQL statement. Here is the result, augmented with the total courtesy of PROC PRINT:

```
Example 3B

Obs      Month     Predicted      Actual    Difference

 1     JAN2004         50.8         58.1          7.3
 2     FEB2004         51.5         48.8         -2.7
 3     MAR2004         52.3         58.2          5.9
 4     APR2004         53.1         64.6         11.5
 5     MAY2004         53.9         67.1         13.2
 6     JUN2004         54.7         64.0          9.3
                              ========
                                 360.8
```

The PROC SYLK Programming Language

The PROC SYLK programming language resemble the SAS DATA step language to a great extent. However, the capabilities differ, and neither one is a subset of the other.

The input and output capabilities of PROC SYLK are limited. External files cannot be read or written, so the INFILE and INPUT statements are not supported; the FILE and PUT statements are used only to write to the log and to the procedure output file. There can be only one input dataset and one output dataset. The input dataset is named using the DATA= option of the PROC statement, so the versatility found in the DATA step's SET, MERGE, UPDATE, and MODIFY statements is not available.

A number of DATA step statements are supported by PROC SYLK, but with limitations. For example, the PROC SYLK version of the ARRAY statement is limited to one-dimensional arrays, and lower bounds cannot be specified. On the other hand, a PROC SYLK array can encompass a combination of variables and constants, a feature not found in the DATA step.

The PROC SYLK programming language includes some useful statements which would undoubtedly be wel-

comed by DATA step programmers. For example, the RETAINBY statement provides automatic reinitialization of retained variables at BY group boundaries.

Probably the most significant such language extension found in PROC SYLK is the capability to create and store user-written functions and subroutines. Exploration and assessment of this feature is beyond the scope of this paper. I will just point out that the feature has been used by the PROC SYLK developers at SAS to support conversion of a number of Excel functions which do not have counterparts in the existing SAS function library. The PROC SYLK manual enumerates these and is a useful reference on the subject of Excel-SAS function correspondence.

Finally, we come to the essence of the SYLK programming language: a handful of functions which allow a program to reference cells in other rows of the data matrix and to summarize groups of cells. It is these functions which most distinguish PROC SYLK from the DATA step.

There are four functions which allow a SYLK program to reach into other observations. The ACELLN function

```
ACELLN (VV, NN)
```

where VV is the name of a variable and NN is the number of an observation, returns a numeric value from that location in the dataset. The RCELLN function is similar, except that the second argument is an offset, or relative observation number. It is added to the number of the current observation to get the address of the cell from which the value is to be retrieved. The functions ACELLC and RCELLC are analogous, but return character values.

The SLKBLOCK function is sort of a "superfunction" which arranges for suitable functions to operate over vectors or matrices. It provides a way to translate what Excel calls "array arguments". The syntax (somewhat simplified) is one of

```
SLKBLOCK('function', 'of variable')
SLKBLOCK('function', 'abs(leftvar, lowobs, rightvar, highobs)')
SLKBLOCK('function', 'rel(leftvar, lowobs, rightvar, highobs)')
```

Notice that the arguments are quoted strings. The first, 'function', must designate a function which can accept multiple arguments; typically it is one of the SAS statistical summary functions (SUM, MIN, MAX, etc.). The 'of variable' construct cumulatively references the designated variable; that is, the function will operate over all values of the variable, from the first observation to the current observation. The 'abs' and 'rel' constructs reference matrices. LEFTVAR and RIGHTVAR indicate the first and last columns, designated either by name or by number. LOWOBS and HIGHOBS are the first and last rows, in either absolute or relative terms.

Here is a brief example illustrating the cell reference and SLKBLOCK functions.

```
title 'Example 4';

data functiondemo;
input a b c;
cards;
11 12 13
21 22 23
31 32 33
;

proc sylk data=functiondemo out=functiondemo_out;
c = rcelln(a,-1);
d = slkblock('mean', 'rel(b,-1,c,-1)');
output;
run;

proc print data=functiondemo_out; run;
```

Result:

```
Example 4
```

Obs	a	b	c	d
1	11	12	0	0.0
2	21	22	11	6.0
3	31	32	21	16.5

The assignment statement for C uses the RCELLN function to overwrite the original values by looking back one row in the A column.. The second assignment statement creates a new variable (D) and populates it with the average of B and C from the preceding observation. Hence 16.5 is the average of 22 and 11. I consider the zeroes in the first row to be errors; they ought to be missing values.

These functions (ACELLN, RCELLN, ACELLC, RCELLC, and SLKBLOCK) allow spreadsheet "flavored" formulas to be coded in PROC SYLK programs. However, they are quite limited in the current experimental release of PROC SYLK (see below).

Assessment

PROC SYLK attempts to construct a bridge from Excel to SAS, one which permits not just data importation but also preservation of formatting and translation of formulas. However the fundamental differences between SAS and Excel, enumerated earlier and cited throughout this paper, make the process awkward and in ways undermine the value of the whole exercise. It's those square pegs and round holes.

I will try to assess the success of PROC SYLK, and its usefulness, in four areas: importing data from Excel, importing formatting information from Excel, importing formulas from Excel, and providing a standalone programming environment which is a useful addition to the SAS toolkit.

Importing Data

Many SAS users who must import data from Excel find that the biggest pitfall is mixed types (numeric and text values) interspersed in the same column. In Version 9.0, the response of PROC SYLK was not very satisfactory; it resulted in the loss or truncation of some values. In Version 9.1, PROC SYLK instead creates two variables when it detects mixed types, one for the numeric values and one for the character values. The suffix "_c" differentiates the name of the character column. There is no loss of information, and one can write code to handle the problem in some appropriate fashion. This seems superior to the way other data import tools handle this chronic problem.

Importing Formatting

A SYLK file saves both the unformatted value of a numeric cell and its format (as specified in Excel's Format>Cells>Number tab). Strictly speaking, it does not save the unformatted value (which would be machine-specific, raising complications for cross-platform usage); rather, it uses something akin to the SAS BEST15. format to preserve a generous number of significant digits. If PROC SYLK encounters multiple formats within a column, it can and will preserve and apply just one of them (the first encountered) as a SAS numeric format for the SAS dataset variable it creates from that column [Square Peg # 2].

In addition to this content formatting, there is appearance formatting (size, color, alignment, rotation, etc). In general, these aspects of display can be controlled in SAS through ODS templates. PROC SYLK does attend to appearance formatting in the ODS templates which it creates, but in a limited way. We will see that there are attributes which PROC SYLK (a) does preserve and others which it (b) does not preserve. The latter category includes attributes which PROC SYLK (c) cannot preserve as well as those which it probably (d) should not preserve. Let's go back to Example 1 and examine some specifics of appearance formatting there. Look at Figures 2 and 3.

> The last line in the original spreadsheet was boldfaced, and that attribute was (a) faithfully preserved in the ODS HTML output.
>
> The typeface in the original is Times New Roman, but the output (b) does not reflect this.
>
> The background color in the original is white, but the output does not reflect this. However, there is a good reason: SYLK files do not record background colors, so PROC SYLK (c) cannot possibly specify them in templates.

Finally, consider foreground color. In Figure 2, it is dark blue. PROC SYLK does not pick that up. However, that is (d) probably for the better. If an Excel spreadsheet were designed with a light foreground color and a dark background color, preserving the foreground color without the original background color might result in output with extremely poor contrast.

So we have seen that PROC SYLK does, in a limited way, preserve appearance formatting. Now let's briefly consider how it does so. The key is the CELLSTYLE statements in the PROC TEMPLATE code which PROC SYLK generates. In Example 1, we see these beginning with

```
cellstyle _row_ = 9
```

In other words, they have extremely specific hard-coded triggers. This limits their practical usefulness. This issue is discussed in greater depth below.

There are numerous tools available to import information from Excel into SAS. Those based on modern markup languages have more promise than does PROC SYLK for the faithful porting of appearance formatting.

Translating Formulas

We have discussed what PROC SYLK does or does not do in terms of importing constants and formats. Now we will turn to the formulas which materialize in the SAS code generated by PROC SYLK when it operates in SYLK-import mode.

It seems that this code "not ready for prime time", and probably never will be. That's not really a failure of implementation as much as it is a consequence of the fundamental differences between SAS and Excel [Square Pegs #1 and # 3].

The fundamental problem is that the PROC SYLK code generated by PROC SYLK almost always includes IF statements which inspect absolute record numbers, rather than events like control breaks (changes in key values) or end-of-file.

To illustrate, we'll go back to Example 1, but we'll create a new dataset, with some additional months' data. Here is the code which creates the test dataset:

```
title 'Example 5';

data moremonths;
input  x1: monyy7. x2 $ x3-x5      ;
format x1  monyy5.      x3-x5 10.1 ;
label
 x1 = 'Month'
 x2 = ' '
 x3 = 'Predicted'
 x4 = 'Actual'
 x5 = 'Difference';
cards;
.         .      .      .       .
Jan2004   .      50.8   58.11   .
Feb2004   .      51.5   48.76   .
Mar2004   .      52.3   58.21   .
Apr2004   .      53.1   64.63   .
May2004   .      53.9   67.06   .
Jun2004   .      54.7   64.01   .
Jul2004   .      55.5   70.49   .
Aug2004   .      56.3   77.58   .
Sep2004   .      57.2   87.46   .
Oct2004   .      58     80.89   .
Nov2004   .      58.9   74.15   .
Dec2004   .      59.8   71.37   .
.         .      .      .       .
.         Total  .      .       .
;

proc print data=moremonths label; run;
```

The dataset looks like this when printed with the LABEL option in effect:

Obs	Month	x2	Predicted	Actual	Difference
1	.		.	.	.
2	JAN04		50.8	58.1	.
3	FEB04		51.5	48.8	.
4	MAR04		52.3	58.2	.
5	APR04		53.1	64.6	.
6	MAY04		53.9	67.1	.
7	JUN04		54.7	64.0	.
8	JUL04		55.5	70.5	.
9	AUG04		56.3	77.6	.
10	SEP04		57.2	87.5	.
11	OCT04		58.0	80.9	.
12	NOV04		58.9	74.2	.
13	DEC04		59.8	71.4	.
14	.		.	.	.
15	.	Total	.	.	.

It has the exact same structure and metadata as the intermediate dataset from Example 1, but it has six additional detail observations, for the months of July through December. Now we'll run the PROC SYLK step which was generated by PROC SYLK in Example 1, changing only the dataset names in the PROC statement:

```
title 'Example 5A';

proc sylk data=moremonths out=moremonths_out1;
  if _row_ ge 2 and _row_ le 7   then x5 = x4-x3;
  if _row_ eq 9 then do;
    x4 = SLKBLOCK("SUM","REL(x4,-7,x4,-2)");
  output;
  end;
  else output;
run;

proc print data=moremonths_out1 label; run;
```

The result of course is a disaster:

```
Example 5A

Obs     Month      x2        Predicted         Actual     Difference
 1          .                     .               .              .
 2      JAN04                  50.8             58.1            7.3
 3      FEB04                  51.5             48.8           -2.7
 4      MAR04                  52.3             58.2            5.9
 5      APR04                  53.1             64.6           11.5
 6      MAY04                  53.9             67.1           13.2
 7      JUN04                  54.7             64.0            9.3
 8      JUL04                  55.5             70.5              .
 9      AUG04                  56.3            360.8              .
10      SEP04                  57.2             87.5              .
11      OCT04                  58.0             80.9              .
12      NOV04                  58.9             74.2              .
13      DEC04                  59.8             71.4              .
14          .                     .               .              .
15          .     Total           .               .              .
```

An unwanted January-June subtotal has obliterated the August figure, the full-year total is missing, and the differences are not calculated for the last six months. We could adapt the obviously inflexible code to the new specifics by changing the 7's to 13 and the 9 to 15. We noted earlier a similar rigidity in the PROC TEMPLATE code, so it would have to be repaired in a parallel fashion. That would generate the correct results, but each time the "depth" of the input dataset changed, this precise sort of revision would have to be carried out. That's obviously not an acceptable maintenance regimen. A macro-based solution could be crafted, but that carries its own burdens. It is arguably better to make the code data-driven:

```
title 'Example 5B';

proc sylk data=moremonths out=moremonths_out2;
if not missing(x1) then x5 = x4-x3;
if x2='Total' then x4 = SLKBLOCK("SUM","of x4");
output;
run;

proc print data=moremonths_out2 label; run;
```

This produces the correct results.

```
Example 5B

Obs     Month      x2        Predicted        Actual     Difference
 1          .                     .              .             .
 2      JAN04                  50.8           58.1           7.3
 3      FEB04                  51.5           48.8          -2.7
 4      MAR04                  52.3           58.2           5.9
 5      APR04                  53.1           64.6          11.5
 6      MAY04                  53.9           67.1          13.2
 7      JUN04                  54.7           64.0           9.3
 8      JUL04                  55.5           70.5          15.0
 9      AUG04                  56.3           77.6          21.3
10      SEP04                  57.2           87.5          30.3
11      OCT04                  58.0           80.9          22.9
12      NOV04                  58.9           74.2          15.3
13      DEC04                  59.8           71.4          11.6
14          .                     .              .             .
15          .     Total           .          822.7             .
```

However, this dataset is not particularly useful or efficient in the SAS environment [Square Peg #3]. Why not just delete the empty observations and the observation for the total and use a SAS reporting tool display the total when it is needed?

Also, consider that this problem would not be a problem at all in the Excel environment. Because Excel data and formulas are so tightly integrated, when one inserts additional detail rows, references adjust and summary formulas stretch their ranges. But when we port to SAS, the data and the formulas are divorced even though the formulas depend on the data in excessively rigid ways [Square Peg # 1].

Programming

Now we turn to the standalone programming language offered by PROC SYLK. As noted above, it has some nice features, many of which would be welcomed in the DATA step.

However, it is the SLKBLOCK function and the four cell reference functions which are the essence of the PROC SYLK programming language, so we'll focus the evaluation on them. We'll proceed by working through a more demanding problem than the one presented earlier (as Example 3), yet one which is easily handled by Excel. Here is sample data:

```
title 'Example 6';

data forma;
input  month: monyy7. actual     ;
format month  monyy7. actual 8.1 ;
cards;
Jan2004  58.11
Feb2004  48.76
Mar2004  58.21
Apr2004  64.63
May2004  67.06
Jun2004  64.01
Jul2004  70.49
Aug2004  77.58
Sep2004  .
Oct2004  80.89
Nov2004  74.15
Dec2004  71.37
;

proc print data=forma; run;
```

The missing value for September is intentional. Begin with an attempt to calculate a three-term centered moving average:

```
title 'Example 6A';

proc sylk data=forma out=centeredma;
if      SLKBLOCK("n"   ,"REL(actual,-1,actual,1)")=3 then
  cma = SLKBLOCK("mean","REL(actual,-1,actual,1)");
output;
run;
```

The –1 and 1 indicate the span of cells to sum, in this case from one above the current row to one below. This is logical, consistent with the Excel formulas, and syntactically correct in the PROC SYLK language. Unfortunately, forward references (formulas dependent on observations not yet encountered) are not supported in PROC SYLK, at least not yet. In the log, one sees:

```
ERROR: Argument 4 to REL must be negative
```

That is truly unfortunate. Such look-ahead capability is something of a holy grail among DATA step programmers. If it worked, it would make PROC SYLK an attractive alternative to the DATA step for certain situations (such as computation of centered moving averages).

The fact that it does not work is not surprising. Supporting look-ahead would greatly complicate the sequence of processing. Excel will support just about any chain of dependency within a spreadsheet, as long as it does not introduce cycles. SAS relies on top-to-bottom processing; look-ahead would probably require some drastic re-engineering of PROC SYLK's internals [Square Peg # 4].

So we'll change gears and implement a three-term trailing moving average instead. By definition, it looks back only. The code:

```
title 'Example 6B';

proc sylk data=forma out=trailma;
if          SLKBLOCK("n"   ,"REL(actual,-2,actual,0)")=3 then
  trailma = SLKBLOCK("mean","REL(actual,-2,actual,0)");
output;
run;

proc print data=trailma; run;
```

Here are the results:

```
Example 6B

Obs      month      actual    trailma

 1    JAN2004        58.1    19.3700
 2    FEB2004        48.8    35.6233
 3    MAR2004        58.2    55.0267
 4    APR2004        64.6    57.2000
 5    MAY2004        67.1    63.3000
 6    JUN2004        64.0    65.2333
 7    JUL2004        70.5    67.1867
 8    AUG2004        77.6    70.6933
 9    SEP2004           .       .
10    OCT2004        80.9       .
11    NOV2004        74.2       .
12    DEC2004        71.4    75.4700
```

The results for March through December are as expected, numerically correct, and appropriately missing where one of the terms is missing. January and February present a different story. TRAILMA should be missing for

both, because the computation would reach back into the previous year. The fact that values appear for these observations indicates that the predicate of the IF statement evaluates as TRUE. In other words, the N statistic "sees" three values where in fact there are fewer. The computed results are consistent with this misbehavior; they reflect division by 3 even though the numerators reflect fewer than 3 terms.

We can expect this misbehavior to be fixed in a production release. Meanwhile, let's attempt a workaround. We could employ the MAX function to code the IF predicate so that it will not even attempt to make an out-of-range reference:

```
title 'Example 6C';

proc sylk data=forma out=trailma2;
if           SLKBLOCK("n"    ,"ABS(actual,max(_n_-2,1),actual,_n_)")=3 then
  trailma = SLKBLOCK("mean","REL(actual,-2           ,actual,0   )");
output;
run;
```

The result is disappointing:

```
ERROR: Argument 2 to ABS must be an constant integer.
```

However, some analysis will justify this result. Note that the entire compound second argument to SLKBLOCK is quoted; it is passed as an entity to SLKBLOCK, which interprets it and passes the series of values which it represents to the N function. But an expression we have constructed in the PROC SYLK environment can generally incorporate subexpressions which will not make sense in the environment of the mediating SLKBLOCK function, so we should not expect the code to work.

But we can turn to a more carefully conceived approach:

```
title 'Example 6D';

proc sylk data=forma out=trailma2(drop=block);
block = compress( "ABS(actual,"          ||
                  put(max(_n_-2,1),6.)  ||
                  ",actual,"            ||
                  put(_n_,6.)           ||
                  ")"
                );
if SLKBLOCK("n",block)=3 then
 trailma = SLKBLOCK("mean","REL(actual,-2,actual,0)");
output;
run;
```

Here we build a string (BLOCK) in the PROC SYLK environment which conforms to the rules for forming arguments for the SLKBLOCK function. Specifically, as PROC SYLK loops through the 12 observations in our dataset, the computed values of BLOCK are:

```
ABS(actual,1,actual,1)
ABS(actual,1,actual,2)
ABS(actual,1,actual,3)
ABS(actual,2,actual,4)
ABS(actual,3,actual,5)
ABS(actual,4,actual,6)
ABS(actual,5,actual,7)
ABS(actual,6,actual,8)
ABS(actual,7,actual,9)
ABS(actual,8,actual,10)
ABS(actual,9,actual,11)
ABS(actual,10,actual,12)
```

Alas, we are disappointed yet again:

```
ERROR: Argument 2 to SYLKBLOCK function not valid.
```

This time the failure is less understandable. SLKBLOCK should not care whether the string it receives was generated in the calling environment as a constant or as the value of an expression.

We can work around these limitations and devise a solution:

```
title 'Example 6E';

proc sylk data=forma out=trailma2;
if           _n_>=3 and
             SLKBLOCK("n"    ,"REL(actual,-2,actual,0)")=3 then
  trailma = SLKBLOCK("mean","REL(actual,-2,actual,0)");
output;
run;

proc print data=trailma2; run;
```

This generates correct and appropriate output:

```
Example 6E

Obs      month      actual    trailma

  1    JAN2004        58.1       .
  2    FEB2004        48.8       .
  3    MAR2004        58.2    55.0267
  4    APR2004        64.6    57.2000
  5    MAY2004        67.1    63.3000
  6    JUN2004        64.0    65.2333
  7    JUL2004        70.5    67.1867
  8    AUG2004        77.6    70.6933
  9    SEP2004           .       .
 10    OCT2004        80.9       .
 11    NOV2004        74.2       .
 12    DEC2004        71.4    75.4700
```

Leaving our example, and instead generalizing, it seems that the insistence of SLKBLOCK on hard-coded constants within its argument strings limits the usefulness of this function, which is arguably at the core of the things which PROC SYLK is intended to do. RCELLN and the other cell-reference functions have similar limitations. Together with look-ahead references, this is the frontier where enhancement would mean the most. Another worthwhile goal is to have those boundary detection features which are implemented for datasets as a whole also be extended to BY groups, *a la* the RETAINBY statement.

Conclusions

PROC SYLK does more or less work as advertised. It will take a SYLK file representing a spreadsheet and generate a SAS dataset and SAS code which together implement the original model. However, the code tends to be extremely lacking in generality. One can manually refine the code, but even then the usefulness is often limited in comparison to a traditional SAS solution. These limitations are a consequence of fundamental differences between the spreadsheet paradigm and that of SAS.

PROC SYLK does appear to have some niche strengths:

- Importing mixed character and numeric values from Excel with no loss of information
- Providing a programming language with power for referencing across rows. The value of the language will be increased if the spreadsheet-like cell reference and data-reduction functions are enhanced to provide look-ahead capability, execution-time evaluation of arguments, and awareness of dataset and BY group boundaries.

References

SAS Institute Inc. 2002. *The SYLK Procedure* (Version 9.0; Experimental). Cary, NC: SAS Institute Inc. http://support.sas.com/documentation/onlinedoc/base/procsylk.pdf

SAS Institute Inc. 2003. *The SYLK Procedure*. (Version 9.1; Experimental). Cary, NC: SAS Institute Inc. http://support.sas.com/documentation/onlinedoc/base/91/sylk.pdf

DelGobbo, Vincent. 2002. *Techniques for SAS Enabling Microsoft Office in a Cross-Platform Environment*. (SUGI 27 Proceedings). Cary, NC: SAS Institute Inc. http://www2.sas.com/proceedings/sugi27/p174-27.pdf

comp.apps.spreadsheets FAQ (Section 14.6: SYLK format) http://www.faqs.org/faqs/spreadsheets/faq/

MCK. 1986. *Summary of all currently used sylk fields* (sylksum.doc). Microsoft Corp. http://www.wotsit.org (search for " SYLK")

Spreadsheet Symbolic Link Format (sylk.txt: The SYLK File Format). http://www.wotsit.org (search for " SYLK")

Acknowledgments

Jack Hamilton provided helpful comments on a draft version of this paper.

Contact Information

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

Howard Schreier

U.S. Dept. of Commerce
Mail Stop H-2015
Washington DC 20230

202-482-4180

Howard_Schreier@ita.doc.gov

http://howles.com/

Appendix 1: SYLK File for Example 1

```
ID;PWXL;N;E
P;PGeneral
P;P0
P;P0.00
P;P#,##0
P;P#,##0.00
P;P#,##0_);;\(#,##0\)
P;P#,##0_);;[Red]\(#,##0\)
P;P#,##0.00_);;\(#,##0.00\)
P;P#,##0.00_);;[Red]\(#,##0.00\)
P;P"$"#,##0_);;\("$"#,##0\)
P;P"$"#,##0_);;[Red]\("$"#,##0\)
P;P"$"#,##0.00_);;\("$"#,##0.00\)
P;P"$"#,##0.00_);;[Red]\("$"#,##0.00\)
P;P0%
P;P0.00%
P;P0.00E+00
P;P##0.0E+0
P;P#\ ?/?
P;P#\ ??/??
P;Pm/d/yyyy
P;Pd\-mmm\-yy
P;Pd\-mmm
P;Pmmm\-yy
P;Ph:mm\ AM/PM
P;Ph:mm:ss\ AM/PM
P;Ph:mm
P;Ph:mm:ss
P;Pm/d/yyyy\ h:mm
P;Pmm:ss
P;Pmm:ss.0
P;P@
P;P[h]:mm:ss
P;P_("$"* #,##0_);;_("$"* \(#,##0\);;_("$"* "-"_);;_(@_)
P;P_(* #,##0_);;_(* \(#,##0\);;_(* "-"_);;_(@_)
P;P_("$"* #,##0.00_);;_("$"* \(#,##0.00\);;_("$"* "-"??_);;_(@_)
P;P_(* #,##0.00_);;_(* \(#,##0.00\);;_(* "-"??_);;_(@_)
P;P0.0
P;P0.000
P;FArial;M200
P;FArial;M200
P;FArial;M200
P;FArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;ETimes New Roman;M240;L33
P;ETimes New Roman;M240;SB;L33
P;ETimes New Roman;M480;L33
P;ETimes New Roman;M480;SB;L33
P;ETimes New Roman;M360;L33
P;ETimes New Roman;M360;SB;L33
F;P0;DG0G8;SM15;M465
```

```
B;Y10;X6;D0 0 9 5
O;L;D;V0;K47;G100 0.001
F;W1 1 11
F;W3 3 14
F;W4 4 10
F;W5 5 15
F;P22;FG0G;C1
F;P0;FG0L;C2
F;P36;FF1G;C3
F;P36;FF1G;C4
F;P36;FF1G;C5
F;P22;FG0R;Y1;X1
C;K"Month"
C;X2;K" "
F;P36;FF1R;X3
C;K"Predicted"
F;P36;FF1R;X4
C;K"Actual"
F;P36;FF1R;X5
C;K"Difference"
F;P22;FG0R;Y2;X1
F;P36;FF1R;X4
F;P36;FF1R;X5
C;Y3;X1;K37987
C;X3;K50.8
C;X4;K58.11
C;X5;K7.31;ERC[-1]-RC[-2]
C;Y4;X1;K38018
C;X3;K51.5
C;X4;K48.76
C;X5;K-2.74;ERC[-1]-RC[-2]
C;Y5;X1;K38047
C;X3;K52.3
C;X4;K58.21
C;X5;K5.91;ERC[-1]-RC[-2]
C;Y6;X1;K38078
C;X3;K53.1
C;X4;K64.63
C;X5;K11.53;ERC[-1]-RC[-2]
C;Y7;X1;K38108
C;X3;K53.9
C;X4;K67.06
C;X5;K13.16;ERC[-1]-RC[-2]
C;Y8;X1;K38139
C;X3;K54.7
C;X4;K64.01
C;X5;K9.31;ERC[-1]-RC[-2]
F;SDM16;Y10;X2
C;K"Total"
F;SDM16;X4
C;K360.78;ESUM(R[-7]C:R[-2]C)
F;P37;FF3G;X6
E
```

Appendix 2: SYLK File for Example 2

```
ID;PWXL;N;E
P;PGeneral
P;P0
P;P0.00
P;P#,##0
P;P#,##0.00
P;P#,##0_);;\(#,##0\)
P;P#,##0_);;[Red]\(#,##0\)
P;P#,##0.00_);;\(#,##0.00\)
P;P#,##0.00_);;[Red]\(#,##0.00\)
P;P"$"#,##0_);;\("$"#,##0\)
P;P"$"#,##0_);;[Red]\("$"#,##0\)
P;P"$"#,##0.00_);;\("$"#,##0.00\)
P;P"$"#,##0.00_);;[Red]\("$"#,##0.00\)
P;P0%
P;P0.00%
P;P0.00E+00
P;P##0.0E+0
P;P#\ ?/?
P;P#\ ??/??
P;Pm/d/yyyy
P;Pd\-mmm\-yy
P;Pd\-mmm
P;Pmmm\-yy
P;Ph:mm\ AM/PM
P;Ph:mm:ss\ AM/PM
P;Ph:mm
P;Ph:mm:ss
P;Pm/d/yyyy\ h:mm
P;Pmm:ss
P;Pmm:ss.0
P;P@
P;P[h]:mm:ss
P;P_("$"* #,##0_);;_("$"* \(#,##0\);;_("$"* "-"_);;_(@_)
P;P_(* #,##0_);;_(* \(#,##0\);;_(* "-"_);;_(@_)
P;P_("$"* #,##0.00_);;_("$"* \(#,##0.00\);;_("$"* "-"??_);;_(@_)
P;P_(* #,##0.00_);;_(* \(#,##0.00\);;_(* "-"??_);;_(@_)
P;P0.0
P;P0.000
P;FArial;M200
P;FArial;M200
P;FArial;M200
P;FArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;EArial;M200
P;ETimes New Roman;M240;L33
P;ETimes New Roman;M240;SB;L33
```

```
P;ETimes New Roman;M480;L33
P;ETimes New Roman;M360;L33
P;ETimes New Roman;M280;L33
P;ETimes New Roman;M240;L33
F;P0;DG0G8;SM19;M465
B;Y7;X4;D0 0 6 3
O;L;D;V0;K47;G100 0.001
F;W1 1 11
F;W2 2 14
F;W3 3 10
F;W4 4 9
F;P22;FG0G;C1
F;P36;FF1G;C2
F;P36;FF1G;C3
F;P36;FF1G;C4
F;P22;FG0R;Y1;X1
C;K"Month"
F;P36;FF1R;X2
C;K"Predicted"
F;P36;FF1R;X3
C;K"Actual"
F;P36;FF1R;X4
C;Y2;X1;K37990
C;X2;K50.8
C;X3;K58.11
C;Y3;X1;K38018
C;X2;K51.5
C;X3;K48.76
C;Y4;X1;K38047
C;X2;K"52.3"
C;X3;K58.21
C;Y5;X1;K38078
C;X2;K53.1
C;X3;K64.63
C;Y6;X1;K38108
C;X2;K53.9
C;X3;K67.06
C;Y7;X1;K38139
C;X2;K54.7
C;X3;K64.01
E
```