Paper AD02_05

So, Your Data are in Excel!

Ed Heaton, Westat

Abstract

You say your customer sent you the data in an Excel workbook. Well then, I guess you'll have to work with it. This paper will discuss some of the quirks you will find when your data are stored in an Excel workbook. It will cover such things as

- naming conventions,
- character length issues,
- numeric precision,
- date, time, and datetime values,
- mixed data types, and
- caching.

This paper will demonstrate - through the Display Manager - SAS code and techniques to make your life with Excel more predictable and your work less prone to error. We will use the `excel` engine that is available in SAS 9.1.3 when you license SAS/ACCESS for PC Files.

The Basics of using Excel as a Database

Excel is not a relational database; it's a spreadsheet. A relational database contains tables where each column holds the value for an attribute. Every value in that column holds the same attribute, but for a different entity. Each row holds the attributes for a single entity. SAS expects its datasets to be relational.

Excel has no such expectations. So, if we want to store our data in Excel and then read it with SAS, we will have to assume the responsibility of keeping the tables relational.

This can be a problem – mostly because Excel determines the data type of data for each cell rather than for a column. However, many people like the ease and convenience of entering their data in Excel.

These are problems we will find when using Excel as a relational database:

- Excel sets data types automatically based on the data entered.
- Data types are set at the cell level rather than at the column level.
- A worksheet holds no more than 256 columns and 65,536 rows.
- A cell can contain no more than 32,767 characters. (Okay, this seems to be a good thing.)
- Excel dates go back only to 1 January 1900. (Well, Excel does claim that there was a 0 January 1900!)
- Excel has one data type (date-time) to store both dates and times of day. (This is not really an Excel shortcoming; it's just not the SAS standard.)
- Excel puts a dollar sign at the end of its worksheet name. (Again, this is simply an annoyance from the SAS perspective.)

Connecting to your Excel Workbook through a LibName Statement

You can connect to a Microsoft Excel workbook – version 5, 95, 97, 2000, or 2002 – with code similar to the following. (*Note: items in italics will vary for your situation.*)

```
LibName test excel "\\path\fileName.xls" ;
```

In fact, you don't even need to specify the engine.

```
LibName test "\\path\fileName.xls" ;
```

Suppose we have an Excel file called **Demo.xls** in the **H:\ExcelToSas** folder. Then…

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" ;
```

will create a libref to the workbook. Then I will see a SAS library called **Xlslib** in SAS **Explorer**. This icon will have a little globe in the lower right-hand corner to tell us that it's not really a library of SAS datasets.

Names that aren't SASsy

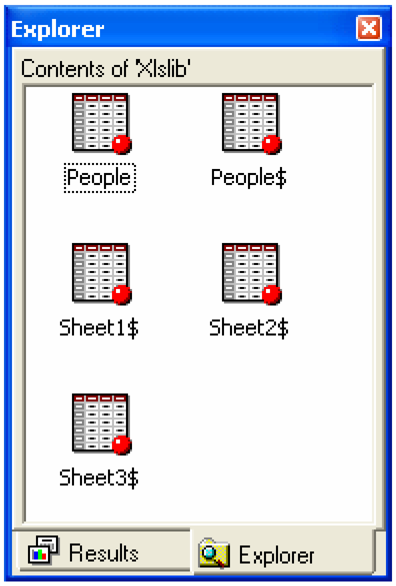
SAS names can contain at most 32 characters and only letters, digits, and underscores. Furthermore, they cannot start with a digit. Microsoft Excel does not have these restrictions. If you want to read tables that have names that don't conform to the SAS standard you must use the `validVarName=any` System Option in conjunction with SAS name literals. Now, Excel worksheet names end with a dollar sign. You don't see the dollar sign in Excel; but it's there. So, we need to submit the `Options` statement as follows.

```
Options validVarName=any ;
```

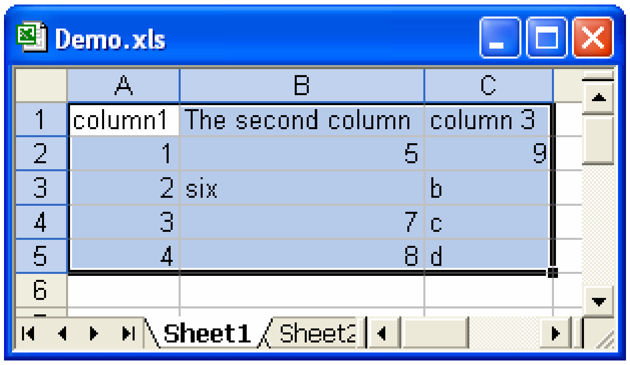
Then we must use a name literal to refer to the worksheet. Name literals are quoted and followed immediately by the letter en as below. Do not put a space between the ending quotation mark and the letter en.

```
Proc print data=xlsLib."Sheet1$"n ;
Run ;
```

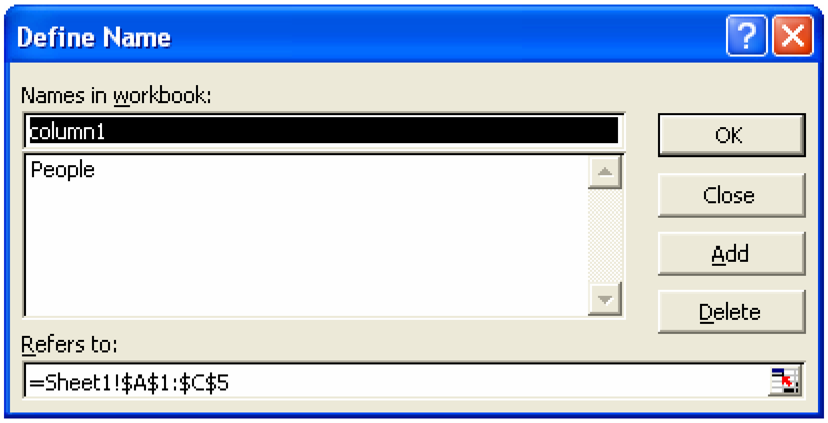
SAS name literals specify unconventional names for both datasets (tables) and variables (columns). You are still limited to 32 characters in the name.



If you don't want to use name literals, you can add named ranges to your Excel workbook and use SAS-compliant names for these named ranges. **People** – in this example – is a named range. We know it's a named range because it doesn't end with a dollar sign. You can add a named range to your Excel workbook by selecting all of the cells containing your data – including the column headers – and then pressing the **Ctrl** and **F3** keys at the same time. You will get a window that looks like the following.



	A	B	C
1	column1	The second column	column 3
2	1	5	9
3	2	six	b
4	3	7	c
5	4	8	d
6			



Names in workbook:

column1

People

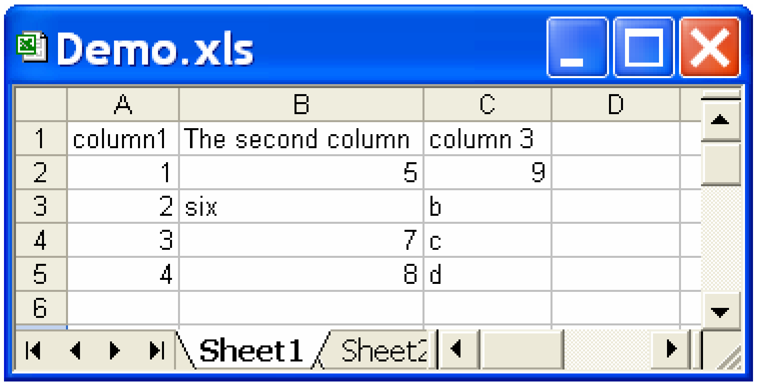
Refers to:

=Sheet1!$A$1:$C$5

Here we already have one named range called **People**. The content of the cell at the top-left corner of the selected range is in the name field as a suggestion for the name of this range. To add the selected range, simply type a name – replacing **column1** – and press the **Enter** key. Let's call this named range **foo**. Now we don't need the `validVarName=any` option to refer to the dataset.

```
Proc print data=xlsLib.foo ;
Run ;
```

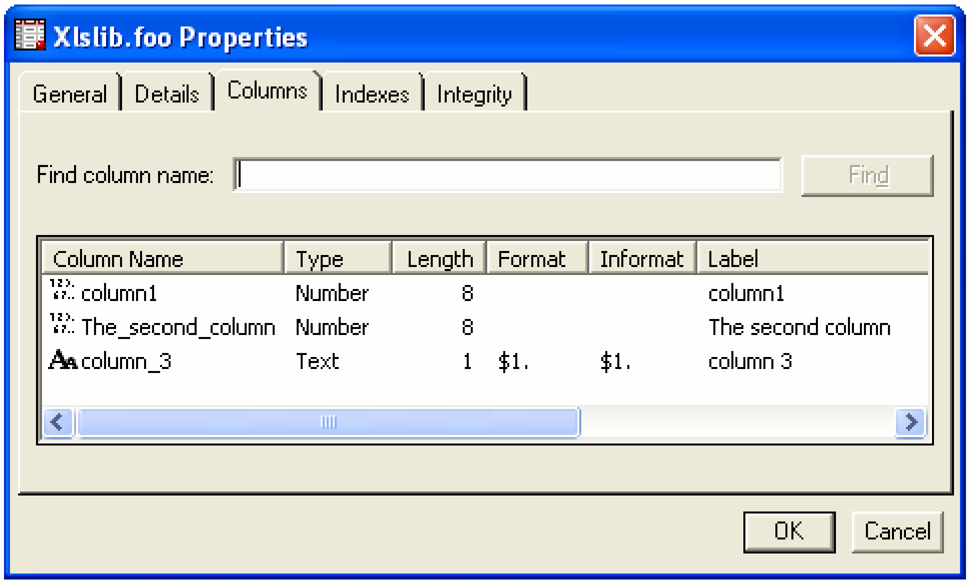
We can create named ranges in Excel that are composed of ranges that are not contiguous. However, the `excel` engine will not recognize a fragmented named range.



	A	B	C	D
1	column1	The second column	column 3	
2	1	5	9	
3	2	six	b	
4	3	7	c	
5	4	8	d	
6				

Sheet1 of **Demo.xls** has three columns of data. The first row contains column headers and some of these column names do not conform to SAS standards. Without the `validVarName=any` System Option, SAS will automatically convert the names to something that conforms to the SAS standard.

If we right-click on the **foo** dataset in SAS Explorer and select **View Columns**, we see that the column names have been changed – underscores were substituted for blanks. This is because we do not have `validVarName=any` in place.



Column Name	Type	Length	Format	Informat	Label
column1	Number	8			column1
The_second_column	Number	8			The second column
column_3	Text	1	$1.	$1.	column 3

We also see that the Excel names are preserved as variable labels. We can override this feature by specifying the `dbSasLabel=none` option on the `LibName` statement. Then we will get no variable labels.

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" dbSasLabel=none ;
```

If you want to strip the variable labels off for just one worksheet in just one SAS step, you can do that with the `dbSasLabel=none` dataset option.

```
Proc contents data=xlsLib.foo( dbSasLabel=none ) ;
Run ;
```

If your data start in the first row of the worksheet, you will need to tell SAS as follows. Otherwise, your first row of data will be missing.

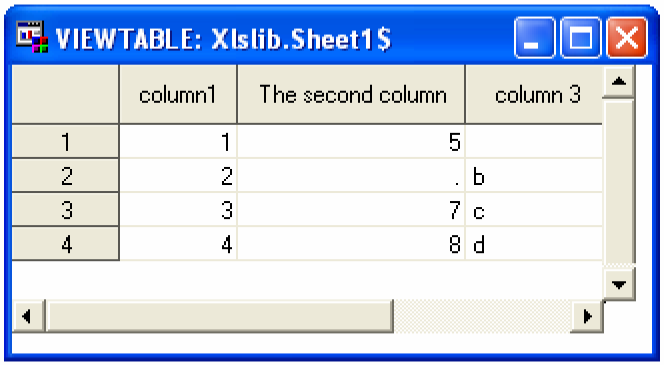
```
LibName xlsLib "H:\ExcelToSas\Demo.xls" header=no ;
```

When you have no column headers, SAS uses **F1**, **F2**, **F3**, etc. for the variable names. And yes, by default SAS will add **F1**, **F2**, **F3**, etc. for variable labels unless we specify `dbSasLabel=none`.

If you use the `header=no` option in the `LibName` statement, it applies to every worksheet in the Excel workbook. There seems to be no dataset option to specify no header row.

Columns with Both Numeric and Character Data

If we look at **Sheet1$**, we see that some values are missing. In the Excel worksheet, the second data value in the second column contains a character string – **six** – and SAS doesn't allow a single variable to be both numeric and text. So, SAS threw the character string away. Similarly, the first data value in the third column is the number 9 – which is not text – so SAS threw it away.

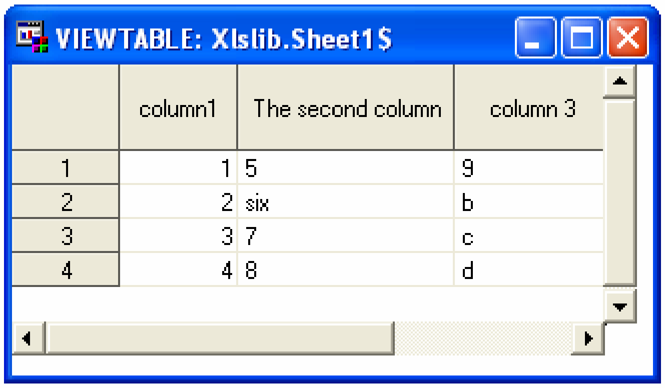


	column1	The second column	column 3
1	1	5	
2	2	.	b
3	3	7	c
4	4	8	d

How does SAS know which data type to retain when reading a column of Excel data? It uses the Microsoft Jet engine which scans the first eight rows of data and whichever data type is most common prevails. If there is a tie, Microsoft Jet makes the column numeric and throws away the character data.

For most work, we don't want SAS to throw away our data. Since we can't put letters in numeric fields, we need to convert the numbers to digits and bring in the whole column as text. SAS allows this with the `mixed=yes` option in the `LibName` statement.

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" mixed=yes ;
```



	column1	The second column	column 3
1	1	5	9
2	2	six	b
3	3	7	c
4	4	8	d

However, if the first eight rows in a column are all numbers, *Microsoft Jet* will make the column numeric and `mixed=yes` will not apply.

You can force the data type to character as you use the data with the `dbSasType=` dataset option. Just like other dataset options, this only changes the data type for the duration of the step. Let's see how this works to force **column1** to a 1-byte character variable.

```
Proc contents
    data=xlsLib."Sheet1$"n( dbSasType=( column1=char1 ) )
;
Run ;
```

Here's the output.

```
The CONTENTS Procedure

Data Set Name         XLSLIB.'Sheet1$'n   Observations          .
Member Type           DATA                Variables             3
Engine                EXCEL               Indexes               0
Created               .                   Observation Length    0
Last Modified         .                   Deleted Observations  0
Protection                                Compressed            NO
Data Set Type                             Sorted                NO
Label
Data Representation  Default
Encoding             Default


                   Alphabetic List of Variables and Attributes

#    Variable             Type    Len    Format    Informat    Label

2    The_second_column    Char      3    $3.       $3.         The second
column
1    column1              Char      1    $1.       $1.         column1
3    column_3             Char      1    $1.       $1.         column 3
```

Unfortunately, this will not help us for mixed columns where the first eight rows contain only numbers. The problem is that the data come to SAS with the character data already stripped off. So, even though we convert the numbers to characters, the original character data are already gone.

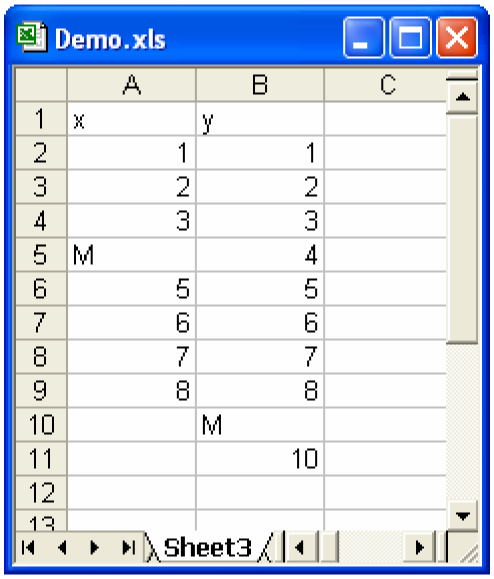
The solution to this problem is to scan more than the first eight rows. We don't have a SAS solution; this involves the Windows registry. If you are comfortable working with the Windows registry, here's the process.

1. Back-up the registry!
2. Find **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Excel**.
3. Double click on **TypeGuessRows**.
4. Change the value to **0**. (All rows are scanned when the value is **0**.)

If you don't know how to get to the *Windows* registry you shouldn't be doing this.

Special Missing Values

Suppose your worksheet has a column with mostly numbers but some of the cells have a letter that represents one of the SAS numeric missing values. We can deal with this quite nicely now that we've changed the settings for Microsoft Jet in the Windows registry.



	A	B	C
1	x	y	
2	1	1	
3	2	2	
4	3	3	
5	M	4	
6	5	5	
7	6	6	
8	7	7	
9	8	8	
10		M	
11		10	
12			

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" mixed=yes ;
Missing M ;
Data Sheet3( drop=_: ) ;
    Set xls."Sheet3$"n( reName=( x=_x y=_y ) ) ;
    x = input( _x , best. ) ;
    y = input( _y , best. ) ;
Run ;
LibName xls clear ;
```

Long Character Values

By default, the `excel` engine will scan each column of data to find the longest character field in the column and then set the character variable to that length in SAS – unless the length of the longest character field is greater than 1024 characters! By default, the `excel` engine will create no character variables longer than 1024 characters. This can be changed using the `dbMax_text=32767` option in the `LibName` satatement. (*See the next page for the output.)*

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" dbMax_text=32767 ;
Proc contents data=xls.People varNum ;
Run ;
```

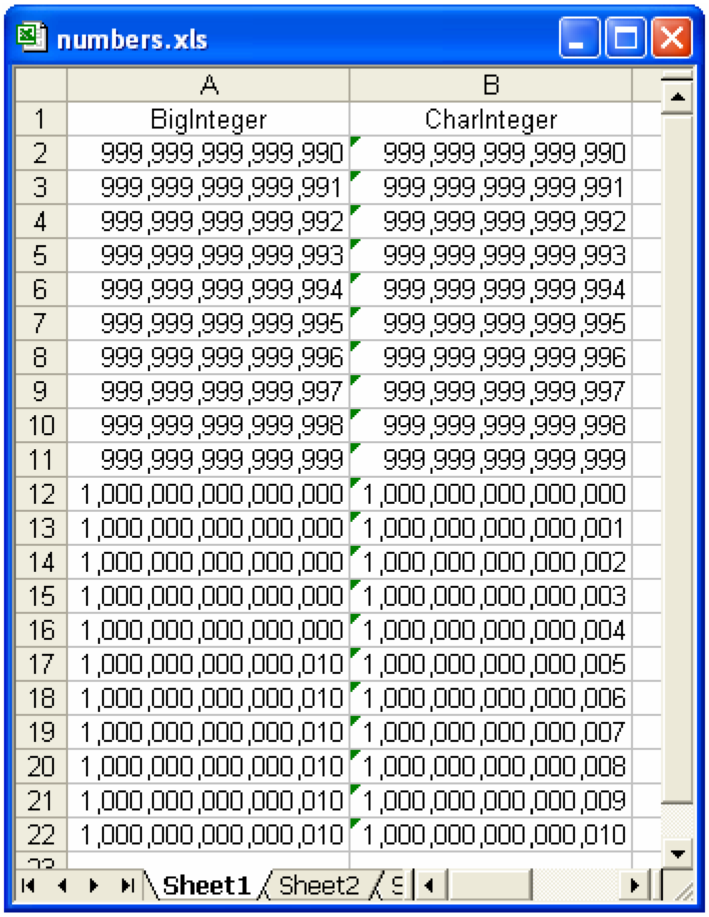
```
                    Variables in Creation Order

#    Variable     Type     Len    Format     Informat     Label

1    LastName     Char      10    $10.       $10.         LastName
2    FirstName    Char      16    $16.       $16.         FirstName
3    Title        Char      40    $40.       $40.         Title
4    Date         Num        8                            Date
5    Text         Char    5456    $5456.     $5456.       Text
```

It doesn't matter how large you set the `dbMax_text=` option – only that it is large enough. The `excel` engine will still scan the character columns and set the variable lengths to the longest occurrence of the data.

Numeric Precision

Excel numbers are not as precise as SAS numbers. Consider this Excel worksheet. The **BigInteger** column is numeric and the **CharInteger** column is text. The number in the first **CharInteger** column was typed into the first cell in the **BigInteger** column. Then that cell plus the next 20 cells down were selected. Finally, **Fill** and then **Series…** were selected from the **Edit** drop-down menu to create a series of numbers where each number is one more than the preceding number as shown in the **CharInteger** column.

Now, we see that Excel does not increment the numbers correctly after a quadrillion. Let's read this data with SAS to see if it's just an Excel display problem.



	A	B
1	BigInteger	CharInteger
2	999,999,999,999,990	999,999,999,999,990
3	999,999,999,999,991	999,999,999,999,991
4	999,999,999,999,992	999,999,999,999,992
5	999,999,999,999,993	999,999,999,999,993
6	999,999,999,999,994	999,999,999,999,994
7	999,999,999,999,995	999,999,999,999,995
8	999,999,999,999,996	999,999,999,999,996
9	999,999,999,999,997	999,999,999,999,997
10	999,999,999,999,998	999,999,999,999,998
11	999,999,999,999,999	999,999,999,999,999
12	1,000,000,000,000,000	1,000,000,000,000,000
13	1,000,000,000,000,000	1,000,000,000,000,001
14	1,000,000,000,000,000	1,000,000,000,000,002
15	1,000,000,000,000,000	1,000,000,000,000,003
16	1,000,000,000,000,000	1,000,000,000,000,004
17	1,000,000,000,000,010	1,000,000,000,000,005
18	1,000,000,000,000,010	1,000,000,000,000,006
19	1,000,000,000,000,010	1,000,000,000,000,007
20	1,000,000,000,000,010	1,000,000,000,000,008
21	1,000,000,000,000,010	1,000,000,000,000,009
22	1,000,000,000,000,010	1,000,000,000,000,010

```
LibName xlsLib ".\numbers.xls" ;
Data ;
    Set xlsLib.BigIntegers ;
    If ( _n_ eq 1 )
        then SasBigInteger = BigInteger ;
        Else SasBigInteger + 1 ;
Run ;
Proc print ;
    Format _numeric_ comma22. ;
Run ;
LibName xlsLib clear ;
```

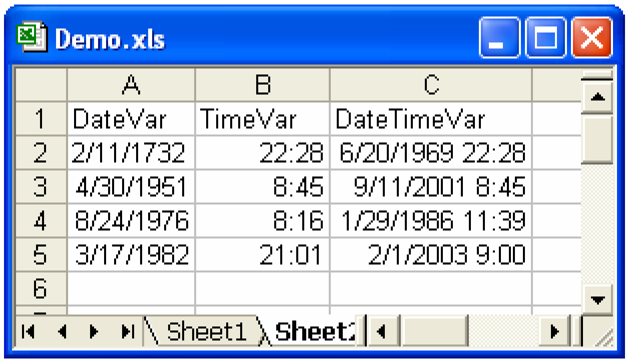
The output follows.

```
Obs                 BigInteger               SasBigInteger

  1          999,999,999,999,990          999,999,999,999,990
  2          999,999,999,999,991          999,999,999,999,991
  3          999,999,999,999,992          999,999,999,999,992
  4          999,999,999,999,993          999,999,999,999,993
  5          999,999,999,999,994          999,999,999,999,994
  6          999,999,999,999,995          999,999,999,999,995
  7          999,999,999,999,996          999,999,999,999,996
  8          999,999,999,999,997          999,999,999,999,997
  9          999,999,999,999,998          999,999,999,999,998
 10          999,999,999,999,999          999,999,999,999,999
 11        1,000,000,000,000,000        1,000,000,000,000,000
 12        1,000,000,000,000,000        1,000,000,000,000,001
 13        1,000,000,000,000,000        1,000,000,000,000,002
 14        1,000,000,000,000,000        1,000,000,000,000,003
 15        1,000,000,000,000,000        1,000,000,000,000,004
 16        1,000,000,000,000,010        1,000,000,000,000,005
 17        1,000,000,000,000,010        1,000,000,000,000,006
 18        1,000,000,000,000,010        1,000,000,000,000,007
 19        1,000,000,000,000,010        1,000,000,000,000,008
 20        1,000,000,000,000,010        1,000,000,000,000,009
 21        1,000,000,000,000,010        1,000,000,000,000,010
```

So we see that SAS numbers have greater precision than do Excel numbers. In fact, SAS integers are precise to 9,007,199,254,740,992, which is over nine times the precision of Excel. The good news is that we don't lose any precision when going from Excel to SAS. However, going from SAS to Excel can cause problems.

Date Value Anomalies between SAS and Excel

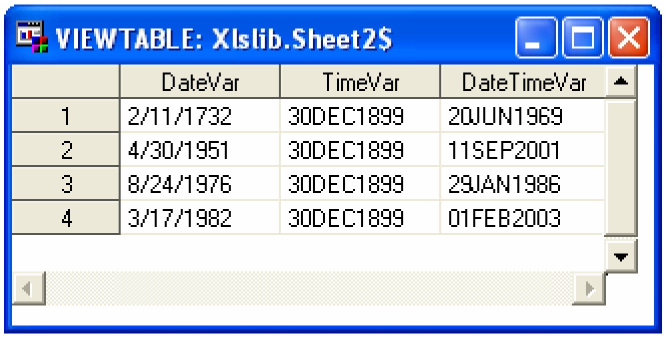
Excel dates are represented by positive integers from 1 through 65,380 that represent dates from 1/1/1900 through 12/31/2078. Since SAS dates are integers from -138,061 through 6,589,335 that represent dates from 1/1/1582 through 12/31/20000, the `excel` engine has no problem importing Excel dates. (Problems might occur when you try to move SAS datasets to Excel.) However, your Excel worksheet might have text cells that ***look*** like they have an Excel date, but don't. Consider this Excel worksheet.



Demo.xls

	A	B	C
1	DateVar	TimeVar	DateTimeVar
2	2/11/1732	22:28	6/20/1969 22:28
3	4/30/1951	8:45	9/11/2001 8:45
4	8/24/1976	8:16	1/29/1986 11:39
5	3/17/1982	21:01	2/1/2003 9:00
6			

The value in cell **A2** looks like an Excel date, but it isn't, it's a text string. Excel dates don't go back to 1732. So, this cell will show up as a missing value if we don't have `mixed=yes` in our `LibName` statement. With `mixed=yes`, the data type for the entire date column will be character when we look at it from SAS. So, we will probably want to use the `input()` function to convert this to a SAS date.



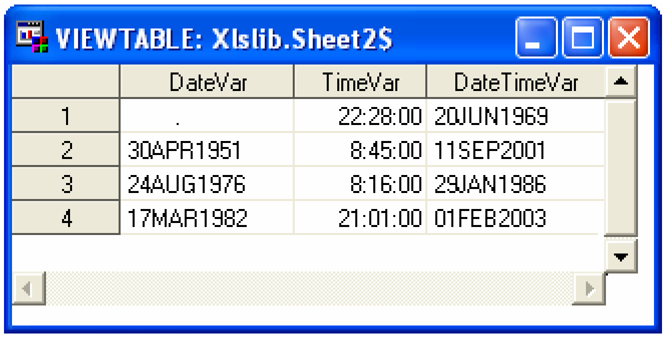
VIEWTABLE: Xlslib.Sheet2$

	DateVar	TimeVar	DateTimeVar
1	2/11/1732	30DEC1899	20JUN1969
2	4/30/1951	30DEC1899	11SEP2001
3	8/24/1976	30DEC1899	29JAN1986
4	3/17/1982	30DEC1899	01FEB2003

```
LibName xlsLib "H:\ExcelToSas\Demo.xls" mixed=yes ;
Data Sheet2( drop=_DateVar ) ;
    Format DateVar date9. ;
    Set xlsLib."Sheet2$"n( reName=( DateVar=_DateVar ) ) ;
    DateVar = input( _DateVar , mmddyy10. ) ;
Run ;
LibName xlsLib clear ;
```

Now, what happened to our times? The time in cell **B2** is represented in Excel as 0.9361111 but it gets to SAS as -21,915.06389. Then SAS applied the `date9.` format to the variable. This isn't what we want at all. By default, a column of time values will be converted to a value that is not correct as a SAS date, a SAS time, or a SAS datetime! SAS provides a `LibName` option to fix this problem.

```
LibName xlsLib "H:\ExcelToSas\Demo.xls"
    mixed=yes
    scanTime=yes
;
```


VIEWTABLE: Xlslib.Sheet2$

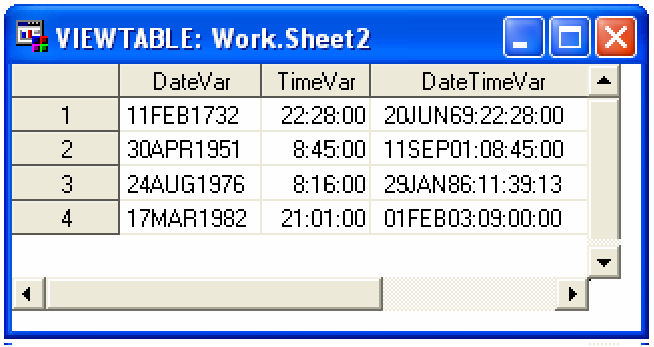
	DateVar	TimeVar	DateTimeVar
1	.	22:28:00	20JUN1969
2	30APR1951	8:45:00	11SEP2001
3	24AUG1976	8:16:00	29JAN1986
4	17MAR1982	21:01:00	01FEB2003

With **scanType=yes**, SAS scans a column and – if it contains only time values – converts the values correctly (The value from **B2** becomes 80,880.) and assigns the **time8.** format. Now we get the following when we look at the data from SAS.

SAS converts Excel's datetime fields to SAS dates and gives them a **date9.** format. We can correct this when we use the dataset with the **dbSasType=()** dataset option to tell SAS to read **DateTimeVar** as a datetime variable.

We need to import this data into SAS because of the different techniques used for the **DateVar** and **TimeVar** columns.

```
LibName xlsLib ".\Demo.xls" mixed=yes scanTime=yes ;
Data Sheet2( drop=_DateVar ) ;
    Format DateVar date9. ;
    Set xlsLib."Sheet2$"n(
        reName=( DateVar=_DateVar )
        dbSasType=( DateTimeVar=dateTime )
    ) ;
    DateVar = input( _DateVar , mmddyy10. ) ;
Run ;
LibName xlsLib clear ;
```


VIEWTABLE: Work.Sheet2

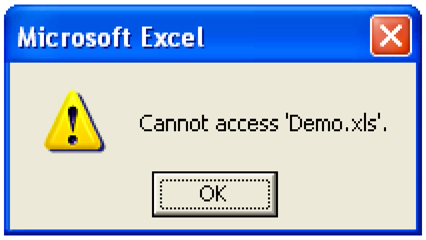
	DateVar	TimeVar	DateTimeVar
1	11FEB1732	22:28:00	20JUN69:22:28:00
2	30APR1951	8:45:00	11SEP01:08:45:00
3	24AUG1976	8:16:00	29JAN86:11:39:13
4	17MAR1982	21:01:00	01FEB03:09:00:00

Now, this looks much better. The dates are SAS dates, the times are SAS times, and the datetime values are SAS datetime values.

Valid data types for the **dbSasType=()** dataset option are **numeric**, **dateTime**, **date**, **time**, and **char1**, **char2**, **char3**, … where **1**, **2**, and **3** are the lengths of the character variables.

Access to the Excel Workbook

You cannot open an Excel workbook that is linked to SAS through an active libRef using the `excel` engine. SAS has exclusive rights to the file. If you try, you will get this message.



Cannot access 'Demo.xls'.

If you or someone else have the Excel workbook open when you access it with a `LibName` statement using the `excel` engine, the data in the workbook will be opened by SAS in read-only mode. You can close the workbook, but the access will still be read-only until you clear the libRef and rerun the `LibName` statement.

SAS caches the Excel workbook file when you connect to it through a `LibName` statement and the `excel` engine. If someone has the Excel file open when you connect to it, they can make changes and even save the changes, but those changes will not be visible to SAS because you are reading the cached file. If you clear the libref and then resubmit the `LibName` statement, the changes will then be visible in SAS.

Conclusions

The `excel` engine that became available with SAS 9 has improved the power and flexibility of `LibName` access to Microsoft Excel data. However, it will not work miracles. If your data in Excel are not in good order, you will still have a lot of handwork to do in SAS. Here are some of my ideas for best practices.

- Since SAS caches your Excel file if someone has it open, be diligent about clearing your libref as soon as it's not needed. If you need it again, resubmit the `LibName` statement and you will get the latest data.
- Always have your Windows Registry set to instruct Microsoft Jet to scan all of the rows when guessing the data type. It really doesn't take that long; remember that an Excel table contains no more than 65,536 rows.
- Use the `mixed=yes` option for the `LibName` statement as a standard practice. Sure, there may be times when you don't want that, but usually it will protect you.
- If your Excel file came from a customer, preserve the timestamp on the file by setting it to read only from Windows Explorer and using things like `validVarName=any` with named literals. That way, there can be no question about whether you changed something in the data. Otherwise, you can create named ranges in the Excel file and you can change the data types of problem cells to make your SAS code cleaner.
- Be especially wary of time and datetime values, they will not be converted properly without special attention.

Acknowledgments

Jennifer Bjurstrom of SAS Technical Support supplied the directions to modify the Windows Registry key that instructs Microsoft Jet on how many rows of Excel data to scan when determining the data type.

Mike Rhoads of Westat helped with editing, suggestions, critique, and encouragement.

Disclaimer

Contact Information

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

Edward Heaton
Westat
1650 Research Boulevard
Rockville, MD 20850-3195
Phone: (301) 610-4818
Email: EdHeaton@Westat.com
URL: http://www.westat.com