Paper 1752-2014

# A Note on Type Conversions and Numeric Precision in SAS®: Numeric to character and back again

Andrew Clapson, Statistics Canada

## ABSTRACT

One of the first lessons that SAS® programmers learn on the job is that numeric and character variables do not play well together, and that type mismatches are one of the more common source of errors in their–otherwise flawless–SAS programs. Luckily, converting variables from one type to another in SAS (that is, casting) is not difficult, requiring only the judicious use of either the input() or put() function. There remains, however, the danger of data being lost in the conversion process. This type of error is most likely to occur in cases of character-to-numeric variable conversion, most especially when the user does not fully understand the data contained in the data set. This paper will review the basics of data storage for character and numeric variables in SAS, the use of formats and informats for conversions, and how to ensure accurate type conversion of even high-precision numeric values.

## INTRODUCTION

Conversions and data type errors are an all-too-common situation for many SAS programmers. Understanding exactly how SAS stores different types of variables will make it easier to understand the nature of the underlying data being stored and calculated. In addition, understanding how variable storage affects precision will lead to safer and faster conversions of data from one type to another.

## BASICS OF DATA STORAGE

To start, let's get some basics out of the way: your computer - and, more specifically, SAS - stores numbers and letters (characters) in markedly different ways. Character strings are stored byte-by-byte, so the length of a character string corresponds exactly to the number of individual characters contained in an observation. Numbers, however, are stored much more compactly and as a result the storage required for a numeric value does *not* have a one-to-one correspondence with the length (in digits) of the number that is stored.

### CHARACTER VALUES

SAS stores character values as a sequence of individual characters: letters, numbers, spaces, accents, control characters, etc. If you are using Windows, SAS by default uses the ASCII codeset.[1] Each individual character is allocated one byte of storage - that is, eight bits of space. These eight bits can represent a range of $2^8$, or 256 different values (from 0-255) and each of these values represents a character from the particular encoding set being used. SAS itself doesn't particularly 'care' if the individual character itself is a number or a letter, as each requires an equal amount of space. For example, all of the values in the below datastep require four bytes of storage:

```
data WORK.CharValues;
       String1 = 'ABcd';
       String2 = '$# !';
       String3 = '2014';
run;
```

The maximum length allowed for character variables will depend on the system and computer used, but is generally a maximum of 2GB on a 32-bit platform and significantly higher on a 64-bit platform ($2^{46}$ bytes).

### NUMERIC VALUES

Computer scientists took a different tack for storing and calculating numeric values. Because numbers require the use of only a small range of individual digits (i.e. two for binary, eight for octal, ten for decimal and sixteen for hexadecimal), our primary concern is not the actual digits that must be displayed (0-9, for decimal), but accurately representing a large range of *values*, in terms of magnitude. Since the majority of SAS users work with decimal numbers, let's focus on those.

Default numeric storage in SAS is a 'double' (a 'long real' in Windows terms) which means that SAS sets aside eight bytes of storage for each numeric observation that you want to store, unless you specify otherwise. However, this does NOT mean that we only have room for eight digits - far from it!

[1] Of course, this is changeable, but of the purposes of this paper I will not look at multiple encodings.

The key to understanding numeric storage is to understand how memory is allocated for each value. Numeric storage can be broken down into three parts: there is a sign (positive/negative), a significand (essentially the integer portion of the number) and an exponent, all expressed for a given base (two, as all numbers are ultimately expressed and computed in binary). So for a default numeric variable of eight bytes (8*8 bits, or 64 bits of space), SAS will allocate space in the following way:

**Table 1. Storage breakdown for an 8 byte numeric observation:**

| Sign | Significand | Base | Exponent |
|---|---|---|---|
| +/- | 0 - 9,007,199,254,740,990 | 2^ | (-1,023) - 0 |
| *1 bit* | *53 bits* | - | *10 bits (bias)* |

The maximum value of the significand is the greatest integer that can be exactly represented in SAS, which amounts to a maximum of fifteen significant digits of precision. The exponent term simply scales the magnitude of a number in order to represent very large or very small numbers, but the precision, in terms of number of significant digits, will remain the same. Note: SAS follows the IEEE standard for floating point representation, with the exception of IBM mainframes (z/OS).

However, this degree of precision will change if the storage allocated to a numeric variable is decreased. The smallest numeric variable length that SAS allows is three bytes (3*8 bits, or 24 bits of space), which will result in the following allocation of space:

**Table 2. Storage breakdown for a 3 byte numeric observation:**

| Sign | Significand | Base | Exponent |
|---|---|---|---|
| +/- | 0 – 8,192 | 2^ | 1 – 1,024 |
| 1 bit | 13 bits | - | 10 bits |

Decreasing the variable length from eight bytes to three bytes resulted in a loss of eleven digits of precision, which is quite a decrease. All this to say, what you see is NOT what you get with numeric variables: a numeric value with a length of eight does not mean that it can only hold eight digits. Though SAS allows the length of numeric variables to range between three and eight bytes, the above length of three bytes should only be used to store small integer values, such as dummy or categorical variables, or for small loop counters. Any variables used in floating point calculations should be stored at full precision, as the default eight byte numeric variable.

## CONVERTING FROM ONE TYPE TO ANOTHER

How is all this detail relevant for type conversions? Conversion operations involve changing the way that the underlying data is stored, so it's important to understand what's going on behind the scenes in order to maintain precision and to ensure accuracy. All conversions will be carried out by using PUT or INPUT functions, or a combination of the two, along with the format or informat appropriate to the stored data.

### CHARACTER TO NUMERIC

The most common case - and arguably the more frustrating of the two - is the conversion of a character variable to a numeric value. Often when reading in data from outside SAS we end up with numbers stored as character strings. If we wish to perform calculations on these values a conversion will be required. As an example, consider:

```
data WORK.CharToNum_1;
       Num1 = input('342.354', 8.);
       Num2 = input('342342.354', 8.);
              put 'Num1 = ' Num1 'Num2 = ' Num2;
run;
```

```
Num1 = 342.354 Num2 = 342342.3
```

What happened? We just demonstrated above that the default numeric storage setting of eight bytes was enough to store fifteen digits, enough to accurately represent numbers as large as nine quadrillion! The '8.' specified in the above code should be more than enough storage, shouldn't it?

The key here is to remember that the INPUT function takes an *informat*, which tells SAS what to expect in terms of input data; it is not equivalent to a LENGTH statement, which simply defines the storage space required for a variable. Here, because we're reading in a character value, the width of informat that we choose will, in fact, be the number of digits that we expect to read in! Recall that character values are stored pretty much 'as you see them', meaning that four digits will require an informat of '$4.', and fourteen digits will require an informat of '$14.' or wider. So even though eight bytes is more than enough storage (length) for these values, an *informat* of '8.' will not be wide enough to capture all the digits.

Let's try and make the informat a bit wider, then:

```
data WORK.CharToNum_2;
       Num1 = input('342.354', 8.);
       Num2 = input('342342.354', 12.);
              put 'Num1 = ' Num1 'Num2 = ' Num2;
run;
```

```
Num1 = 342.354 Num2 = 342342.354
```

There we go! Now that the second INPUT function is expecting a width of up to twelve digits, it will accurately capture and convert the entire number. As well, to make sure that we aren't losing precision due to any leading blanks it's never a bad idea to include a STRIP or LEFT function on the character variable that we're converting.

### NUMERIC TO CHARACTER

In cases where we want a numeric value to be stored as a character variable, such as a numeric ID that we want to sort in a particular way, we instead use the PUT function, along with a numeric format, as in the below example:

```
data WORK.NumToChar;
       Char1 = put(10010221, 8.);
       Char2 = put(10010231, 10.);
              put 'Char1 = '  Char1 $QUOTE. ' Char2 = ' Char2 $QUOTE.;
run;
```

```
Char1 = "10010221" Char2 = "  10010231"
```

The main risk in the above operation is truncation; it's important to define a numeric format that is wide enough to contain the entire number, unless you are purposely attempting to truncate the value. However, if you do try and use a numeric format that isn't wide enough, SAS will generally enforce a 'BESTw.' type format and print a warning to the log to this effect. On the other hand, if the specified format is *wider* than the value to be converted, there will be leading blanks in the character value, as seen in the 'Char2' variable above. This occurs because all numeric values are right-aligned, so any 'leftover' width after converting the number to a string will be padded out with blanks; again, the inclusion of a LEFT function will remedy this.

In addition, because the PUT function returns values using formats, as opposed to informats, this function provides a lot of flexibility when it comes to variable conversions. Depending on the nature of the stored numeric variable, you can convert a numeric value to a string showing a percentage amount, a comma-separated monetary value or even the year, month or day of the week (if the variable stored is a SAS date value).

## CONCLUSION

Understanding the way that SAS stores values for different data types can help users avoid errors when converting variables from one type to another. Because the concept of 'length' means different things for character and numeric variables, it's vital to understand the difference between the space required for data storage and the width required to correctly read and convert the different data types.

Always remember that character variables are stored one byte per character, so when converting, a good rule of thumb to follow is to *at a minimum* match the informat with the maximum expected number width. Indeed, one way to convert variables when you are unsure of their maximum length is to use an informat with the same length as the target character variable or to simply use the maximum numeric informat of '32.' For cases when the character variable to be converted is longer than thirty-two bytes, some additional work will be required, such as splitting up the values using the SUBSTR function (or similar).

A similar concern is present when converting numeric variables to character values, so an adequately wide format must be specified in these cases. However, a loss of precision in this situation is generally a less serious problem, as no numeric calculations can be performed on character variables. As well, the use of certain formats with the PUT function provides the ability to convert numeric values into various types of formatted strings, if desired.

## ACKNOWLEDGMENTS

The author would like to thank John Ladds and Art Tabachneck for their valuable input and advice.

## CONTACT INFORMATION

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

Name: Andrew Clapson
Organization: Statistics Canada
Address: Jean Talon Building, 170 Tunney's Pasture
City, Province, Postal Code: Ottawa, ON | Canada | K1A 0T6
Work Phone: 1-613-951-4308
Email: andrew.clapson@statcan.gc.ca