SESUG Paper 102-2024

The Dreaded Data type Conversion, Character to Numeric, Numeric to Character, a deep dive.

Emily Frances Zucker, RTI International

ABSTRACT

Data type conversions are an inevitable part of the job of a SAS programmer. Statistical analysis and mathematical functions require the variables to be numeric, while readability require the variable to be character. Sometimes, you pull in data from a database or excel and it is not read in as the desired data type. You may need to change the type to perform a specific function and then change it back. Changing the data type seems like an easy, fast process initially, but when you can't remember the exact function and inputs for your desired outcome, it can be a frustrating pause to your flow. To reduce this confusion, a deep dive is given to understand how SAS stores character and numeric data, the methods of converting data types with a focus on PUT and INPUT functions, some examples where data conversions are useful for data manipulation, and some memory tricks are offered to hopefully eliminate that annoying google search once and for all.

INTRODUCTION

Changing the data type of a SAS variable from numeric to character or vise versa is recommended to be done using the PUT and INPUT functions. However, remembering which function to use for each case can be confusing, especially for those new to SAS or those who only need to make this conversion without other changes to the data. Even experienced SAS users find themselves researching which function to use, as these conversions are not intuitive. There are other conversion methods that seem easier to remember and do the job, so why are those not recommended? This paper will explain how SAS stores numeric and character data, what a format and an informat is, and what the intended use of PUT and INPUT functions were and why they are recommended for this purpose over other methods. Understanding these concepts can make the conversion process easier to grasp and remember next time you need to use it.

USES OF EACH DATA TYPE

The choice between storing a variable as character or numeric depends on your intended use.

Numeric data is suitable for quantifiable mathematical formulas and offers more statistical analysis capabilities than character data. Even for categorical data analysis, numeric data in SAS is easier to code because it eliminates the need to type quotes and strings when referencing categories. Examples of using numeric data include:

- Summing financial figures
- Calculating time intervals between dates
- Coding long string categories

Character data, on the other hand, can contain any character, making it easily readable and understandable. It preserves the data in its original form, including special characters, long decimal places, and leading zeros, which can be desirable in some cases. Examples of using character data include:

- Searching for specific words or characters within strings
- Adding or removing prefixes or suffixes from observations
- Displaying strings with alphabetic and special characters for clarity

HOW SAS STORES CHARACTER AND NUMERIC DATA

Like most data on our computers, data in SAS datasets is stored as binary code. Numeric data is 8 bytes long, with each byte consisting of 8 bits, resulting in 64 bits (0s and 1s) to represent all possible numbers. Character data, on the other hand, is 1 byte (8 bits) per character and uses a defined collating sequence to match the 8-bit string to a character. A common collating sequence for Windows machines is the ASCII collating sequence.

In the metadata of a SAS dataset, there is an indicator for each variable specifying whether it is character or numeric. This indicator directs the machine on how to interpret the binary code and display the data. Essentially, the storage format (character or numeric) dictates how SAS reads and processes the binary information.

If you want to change the data type of an existing variable from numeric to character or vice versa, this cannot be done directly in the core data storage. Instead, the values from the original variable are read into a new variable with the desired data type *(P. Shah, personal communication, August 21, 2024).*

INFORMATS AND FORMATS

As mentioned, the core of SAS data is a series of 0s and 1s, and the data type is stored in the metadata. So, where do informats and formats come into play? Understanding the history of SAS and its initial uses can help clarify this.

When SAS was first launched, it primarily worked with .txt or .dat files. Microsoft Office—including Excel, PowerPoint, Word, and PDFs—was not yet widely used. SAS users needed a way to read character data and assign it a data type of numeric or character. Informats served this purpose by reading in text data and acting as a “crosswalk” for SAS to interpret and display the text data correctly.

In the early days of SAS, users also needed a way to output their reports to .txt files, as Excel and other file types were not available. This is where formats came into play. Formats take data of any type and instruct SAS on how to display it in a desired character form.

Documentation of every built-in SAS format and informat can be found on the SAS 9.4 documentation webpage.

CONVERTING DATA TYPES

PUT/INPUT FUNCTIONS

The most accurate way to convert numeric to character or character to numeric in SAS is by using the PUT/INPUT functions:

Character to numeric: Numvar=input(Charvar,num_informat);

Numeric to character: CharVar=put(NumVar,num_format);

Let’s break down these formulas and explain how they work. Since variable types are stored in the metadata, you cannot directly change a variable’s type. Instead, you create a new variable of the desired type and assign it the original variable’s values.

Although using PUT and INPUT functions for this purpose is common today, these functions were not initially developed for this need. I mentioned when SAS was developed, most work involved importing character data from .txt files and exporting character data to .txt files. An INPUT statement takes data

from a .txt file and assigns it a data type using a specified informat. Similarly, a PUT statement takes SAS data of any type and outputs a text data type from it using a specified format.

The general pattern for PUT and INPUT statements are:

- **IN**PUT-> **In**formats -> Reading **in** character data
- **PUT** -> Formats-> Out**put**ting data in character displays

Eventually, SAS developed functions from the PUT and INPUT statements to perform these processes on variables. The INPUT function takes a character variable and converts it to any specified variable type using an informat. The PUT function can start with a variable of any type and convert it to a character variable using a specified format.

One important thing to note is the informat used in the input statement should match the appearance of the source character data, while the format used in the put statement should be a format that can be applied to the numeric variable you are converting, the results of that application being what you would like to see in character data form. For the purposes of converting character to numeric and vice versa, both the format and the informat will be numeric.

Here is a breakdown of the functions I mention above:

Character to numeric

- **Numvar=input(Charvar,num_informat);**
  - NumVar->name of new numeric variable
  - CharVar->original variable
  - Num_informat-> numeric informat that matches the data of CharVar

Numeric to character

- **CharVar=put(NumVar,num_format);**
  - CharVar-> name of new character variable
  - NumVar->original variable
  - Num_format-> numeric format that is applied to NumVar before retrieving character output

Examples

For demonstration purposes, I will use a small example dataset, as shown in Display 1.

ID	Date	Name	Cost	Plan Type
00010145	January 10,2023	Mary	$70.19	Home
00205962	Feburary 03,2024	Joe	82.23	Car
00321470	June 29,2023	Frank	$55.79	Home
04444409	December 24,2023	Johnnie	58.90	Car
00057897	May 18,2024	Barbara	$91.21	Boat

Display 1. Starting Dataset in Excel

Let's assume that, to maintain their desired values, these numbers are initially stored as text in Excel. Due to their text format in Excel, these values are read and stored as character variables in SAS, as seen in Output 1.

	ID	Date	Name	Cost	Plan Type
1	00010145	January 10,2023	Mary	$70.19	Home
2	00205962	Feburary 03,2024	Joe	82.23	Car
3	00321470	June 29,2023	Frank	$55.79	Home
4	04444409	December 24,2023	Johnnie	58.90	Car
5	00057897	May 18,2024	Barbara	$91.21	Boat

Output 1. Initial SAS dataset

However, since the IDs are numbers, we want them to be stored as a numeric data type. We use the following INPUT statement to achieve this:

```
data demo1;
 set demo(rename=(ID=ID_orig));
 ID=input(ID_old,8.);
run;
```

Remember, whenever you want to change the data type in SAS, you need to create a new variable with the desired type and assign it the values from the old variable. In this instance, we rename ID to ID_orig when it is being read in. Then, with the INPUT function, we create a new numeric variable called ID and assign it the values from ID_orig, ID_orig being read in using the 8. Informat.

This results in Output 2:

	ID_orig	Date	Name	Cost	Plan Type	ID
1	00010145	January 10,2023	Mary	$70.19	Home	10145
2	00205962	Feburary 03,2024	Joe	82.23	Car	205962
3	00321470	June 29,2023	Frank	$55.79	Home	321470
4	04444409	December 24,2023	Johnnie	58.90	Car	4444409
5	00057897	May 18,2024	Barbara	$91.21	Boat	57897

Output 2.

As you can see, ID was renamed to ID_orig, and a new numeric variable named ID was created. However, ID is missing its leading zeros. Losing leading zeros is a common issue when reading numeric data from other sources. Let's assume we don't have ID_orig and we want to restore the leading zeros in our ID variable. This requires converting the ID variable back to a character type and filling in the leading zeros to a total length of 8.

```
data demo2;
 set demo1(rename=ID=ID_num);
 ID=put(ID_num,z8.);
run;
```

This results in Output 3:

	ID_orig	Date	Name	Cost	Plan Type	ID_num	ID
1	00010145	January 10,2023	Mary	$70.19	Home	10145	00010145
2	00205962	Feburary 03,2024	Joe	82.23	Car	205962	00205962
3	00321470	June 29,2023	Frank	$55.79	Home	321470	00321470
4	04444409	December 24,2023	Johnnie	58.90	Car	4444409	04444409
5	00057897	May 18,2024	Barbara	$91.21	Boat	57897	00057897

Output 3

The Zw. format adds leading zeros up to the specified length. We use a PUT function to read in numeric ID_num, assign it the format z8. and save those values in our new ID variable as a character variable. To finish, we can drop the old ID variables. Dropping these will be included in my code examples going forward.

```
data demo;
 set demo1(rename=ID=ID_num);
 ID=put(ID_num,z8.);
 drop ID_orig ID_num;
run;
```

As an addendum, if you want your ID variable to be numeric and display leading zeros, you can achieve that by assigning the numeric value the z8. format.

```
data demo_ad;
 set demo(rename=(ID=ID_orig));
 ID=input(ID_orig, 8.);
 format ID z8.;
 drop ID_orig;
run;
```

This addendum output is shown in Output 4.

	Date	Name	Cost	Plan Type	ID
1	January 10,2023	Mary	$70.19	Home	00010145
2	Feburary 03,2024	Joe	82.23	Car	00205962
3	June 29,2023	Frank	$55.79	Home	00321470
4	December 24,2023	Johnnie	58.90	Car	04444409
5	May 18,2024	Barbara	$91.21	Boat	00057897

Output 4

IMPLICIT METHODS

There are other ways to convert the data type of a variable using implicit methods. These methods allow SAS to “guess” the format or informat to use, which can sometimes lead to truncation or loss of data without your knowledge.

Example 1: Using a LENGTH Statement

One implicit method is using a LENGTH statement to set the data type and then assigning the new variable to equal the original variable. Let’s convert our cost variable to a numeric data type using this method:

```
data demo1;
 set demo(rename=(cost=cost_orig));
 length cost 8.;
 cost=cost_orig;
 drop cost_orig;
run;
```

	Date	Name	Plan Type	ID	cost
1	January 10,2023	Mary	Home	00010145	.
2	Feburary 03,2024	Joe	Car	00205962	82.23
3	June 29,2023	Frank	Home	00321470	.
4	December 24,2023	Johnnie	Car	04444409	58.9
5	May 18,2024	Barbara	Boat	00057897	.

Output 5. Converting cost to numeric using a LENGTH statement

Because we used an implicit method, SAS chose an informat to read in cost that was not compatible with the variable. Any value containing a $ was dropped from the data.

This issue can also occur when converting character to numeric using the LENGTH statement. For example, let’s start with the demo_ad dataset and use this method to convert numeric ID to a character variable:

```
data demo3;
 set demo_ad(rename=(ID=ID_orig));
 length ID $8.;
 ID=ID_orig;
 drop ID_orig;
run;
```

	Date	Name	Cost	Plan Type	ID
1	January 10,2023	Mary	$70.19	Home	10145
2	Feburary 03,2024	Joe	82.23	Car	205962
3	June 29,2023	Frank	$55.79	Home	321470
4	December 24,2023	Johnnie	58.90	Car	4444409
5	May 18,2024	Barbara	$91.21	Boat	57897

Output 6. Converting ID to character using a LENGTH statement

We lose our leading zeros because ID was originally saved as numeric, thus no leading zeros, and then SAS chose its own format to represent the variables rather than the format being chosen by us.

Example 2: Multiplying by 1

Another implicit method of conversion is to multiply your character variable by 1 to convert it to numeric. This produces the same results as using a length statement in our example:

```
data demo4;
 set demo (rename=(Cost=cost_orig ID=ID_orig));
 Cost=cost_orig*1;
 ID=ID_orig*1;
 drop ID_orig cost_orig;
run;
```

	Date	Name	Plan Type	Cost	ID
1	January 10,2023	Mary	Home	.	10145
2	Feburary 03,2024	Joe	Car	82.23	205962
3	June 29,2023	Frank	Home	.	321470
4	December 24,2023	Johnnie	Car	58.9	4444409
5	May 18,2024	Barbara	Boat	.	57897

Output 7. Converting cost and ID to numeric by multiplying by 1

It is best practice to use PUT or INPUT functions to do your data type conversions. This allows you to define the formats and informats SAS should use, preventing data loss or truncation.

PUT/INPUT FURTHER EXAMPLES

Now that we've demonstrated the weaknesses of using implicit methods, let's use PUT/INPUT functions to correctly convert our cost variable into a numeric variable.

```
data demo5;
 set demo(rename=(cost=cost_orig));
 cost=input(cost_orig,DOLLAR8.2);
 format cost DOLLAR8.2;
 drop cost_orig;
```

```
run;
```

In this example, DOLLAR8.2 is used as an informat in the INPUT function, and then as a format in the FORMAT statement. The INPUT function tells SAS to read cost_orig as a DOLLAR8.2 formatted variable and store the results as a numeric variable since DOLLAR8.2 is a numeric informat. The FORMAT statement tells SAS to display the numeric variable with DOLLAR8.2 formatting. This illustrates how a format and an informat can share the same name, which is common with many built-in SAS informats and formats. It's important to remember that informats and formats serve different purposes, as previously discussed and demonstrated.

	Date	Name	Plan Type	ID	cost
1	January 10,2023	Mary	Home	00010145	$70.19
2	Feburary 03,2024	Joe	Car	00205962	$82.23
3	June 29,2023	Frank	Home	00321470	$55.79
4	December 24,2023	Johnnie	Car	04444409	$58.90
5	May 18,2024	Barbara	Boat	00057897	$91.21

Output 8. Converting cost to numeric using INPUT function

Another variable we may want to convert in this example dataset is date. Having date as a numeric data type allows you to calculate time elapsed from one date to another, which can be useful for many statistical applications such as: the time it took a customer to sign up for something, calculating age from birthdays, how long a business process takes to complete, etc.

```
data demo6;
 set demo5(rename=(Date=Date_orig));
 Date_num=input(date_orig, ANYDTDTE16.);
 drop date_orig;
run;
```

	Name	Plan Type	ID	cost	Date_num
1	Mary	Home	00010145	$70.19	23020
2	Joe	Car	00205962	$82.23	23409
3	Frank	Home	00321470	$55.79	23190
4	Johnnie	Car	04444409	$58.90	23368
5	Barbara	Boat	00057897	$91.21	23514

Output 9. Converting date to numeric using INPUT function

Once we have the numeric date value, we can save a character string as well for readability in any format we'd like. Let's do a different format than the one we started with:

```
data demo7;
 set demo6;
 date=put(date_num,MMDDYY9.);
run;
```

	Name	Plan Type	ID	cost	Date_num	date
1	Mary	Home	00010145	$70.19	23020	01/10/23
2	Joe	Car	00205962	$82.23	23409	02/03/24
3	Frank	Home	00321470	$55.79	23190	06/29/23
4	Johnnie	Car	04444409	$58.90	23368	12/24/23
5	Barbara	Boat	00057897	$91.21	23514	05/18/24

Output 10. Converting date_num to a character string using PUT

Now we have a dataset with variables formatted for their intended use. One last thing you may add to this dataset is a numerical category for plan type. This will make programming it easier when referencing it in analysis. You may also want to re-order the variables as well.

```
data demo8;
 retain ID Name Date Date_num Plan 'Plan Type'n Cost;
 set demo7;
 if 'Plan Type'n="Home" then Plan=1;
 if 'Plan Type'n="Car" then Plan=2;
 if 'Plan Type'n="Boat" then Plan=3;
run;
```

	ID	Name	Date	Date_num	Plan	Plan Type	Cost
1	00010145	Mary	01/10/23	23020	1	Home	$70.19
2	00205962	Joe	02/03/24	23409	2	Car	$82.23
3	00321470	Frank	06/29/23	23190	1	Home	$55.79
4	04444409	Johnnie	12/24/23	23368	2	Car	$58.90
5	00057897	Barbara	05/18/24	23514	3	Boat	$91.21

Output 11. Final Result

Voilà!

MEMORY TRICKS

Given that PUT and INPUT functions are the most reliable ways to convert a variable's data type, it is helpful to easily remember when to use each function and how. As mentioned, it can be tricky to remember which one to use in each case. Let's discuss some ways to help remember.

LINKING THE "IN"S

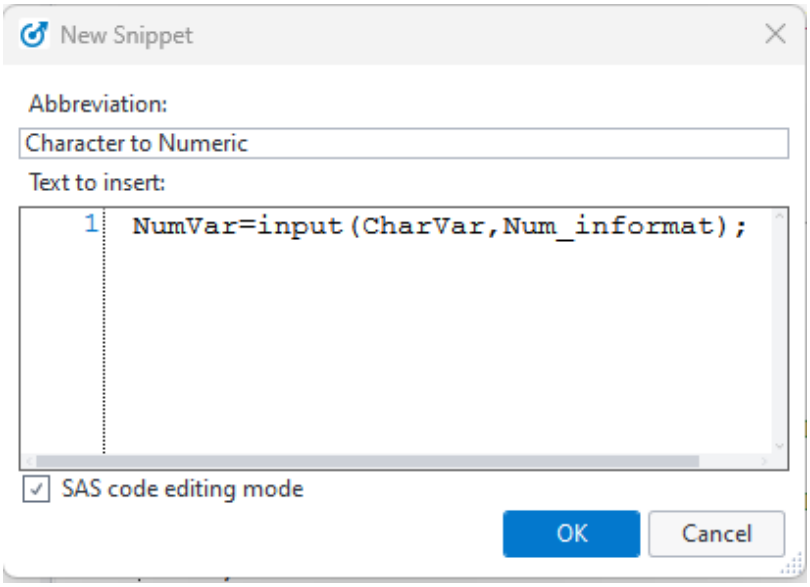
One way to remember is to go back to what these functions were originally used for: reading in and outputting character data. Group everything containing "IN" together in your mind, including their use case. So, **IN**PUT goes with **IN**format and they are used to read **IN** character data. Then, you can deduce what is left without "IN" goes together: **PUT** goes with FORMAT, and they are used to out**PUT** data in a character format. Therefore, you can ask yourself, "Am I reading character data **IN** (starting with a character type variable) or wanting to **PUT** character data out (ending with a character type variable)?" If you are reading it **IN**, use **IN**PUT; if you are **PUT**ting it out, use **PUT**.

CODE SNIPPETS IN SAS ENTERPRISE GUIDE

Another way to make this an easily memorable experience, if you use SAS Enterprise Guide, is to use code snippets.

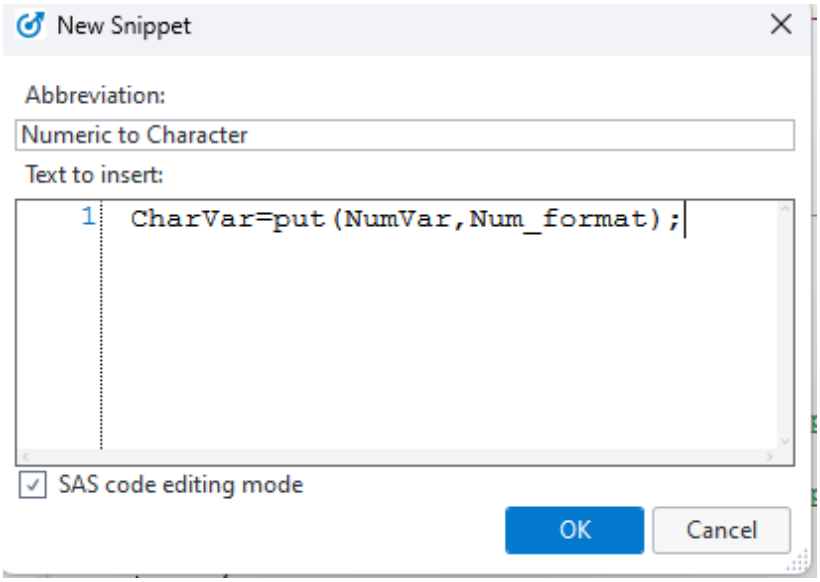
To make a new code snippet in SAS Enterprise Guide,

1. Go to program -> New Snippet.
2. Create a snippet for Character to Numeric conversions, see Display 2:



Display 2. Making a Code Snippet for Character to Numeric conversion

Similarly, you can make a code snippet for Numeric to Character conversions, see Display 3:



Display 3. Making a Code Snippet for Numeric to Character conversion

To insert the code snippet you've saved into your code:

1. Go to Program-> Manage micros and snippets
2. Click on the snippet you want
3. Click Run

You can also type Alt + F8 to pull up the code snippet menu. There are many interesting things you can do with code snippets! If you are interested in learning more, I recommend reading my colleague Mark McLean's paper, *Creating and using code snippets with SAS Enterprise Guide.*

CONCLUSION

Converting data from character to numeric or vice versa may initially seem complex. However, with SAS's extensive range of formats and informats, understanding this process leads to numerous possibilities for data manipulation. When you're new to SAS, terms like format, informat, PUT, and INPUT might seem hard to keep track of. However, learning their history clarifies the conversion process. Ask yourself if you are bringing character data **IN** or **PUT**ting character data out. This will guide you to the appropriate **PUT** or **IN**PUT function you use with a numeric format or informat. This method allows for various data re-formatting tasks, such as adding leading zeros, inserting dollar signs for monetary values, and formatting dates in different ways. Even if you are an experienced programmer, this knowledge can save an annoying google search and keep you in the coding flow we all enjoy.

A POEM

Every story starts with a Character--
INPUT takes it IN

PUT sends it out

No mix or matching

INPUT only wants that text

And PUT will only write it

The definition of compatibility

REFERENCES

McLean, M. (2023). *Creating and using code snippets with SAS Enterprise Guide*. In *Proceedings of the SouthEast SAS Users Group (SESUG) Conference*. SESUG. https://sesug.org/proceedings/sesug_2023_final_papers/Learning_SAS_I/SESUG2023_Paper_159_Final_PDF.pdf

SAS Institute Inc. (n.d.). *SAS® 9.4 Formats and Informats: Reference*. Retrieved August 20, 2024, from https://support.sas.com/documentation/cdl/en/leforinforref/64790/HTML/default/viewer.htm#n0cq8eha2o93mdn1lg8n5ursmkxm.htm

SAS Institute Inc. (n.d.). *SAS® 9.4 Formats and Informats: Reference*. Retrieved August 20, 2024, from https://support.sas.com/documentation/cdl/en/leforinforref/64790/HTML/default/viewer.htm#n11m54nggvjybhn1w2a8mbczw04q.htm

SAS Institute Inc. (n.d.). *SAS® 9.4 Functions and CALL Routines: Reference*. Retrieved August 20, 2024, from https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lefunctionsref/n0mlfb88dkhbmun1x08qbh5xbs7e.htm

CONTACT INFORMATION

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

Emily Zucker
RTI International
ezucker@rti.org