An Animated Guide: Using the Put and INput functions

Russ Lavery Independent Contractor (Numeric Resources)
Alejandro Jaramillo (DataMeans)

ABSTRACT

This paper explores features and complexity in the use of Put and INput functions. Put and INput are useful for converting variables from character to numeric and the reverse. This paper starts with simple examples and builds to more complex examples showing dates and nested use [e.g. Put(INput(cust_id,6.0),Z6.0)].

The Put function converts from a numeric or character input/variable/target to a character output, applying a format as part of the conversion. **The Put function can be applied to a character or numeric input, but the output of the Put function is always character**. INput converts only from a character input/variable/target to either numeric or character output/variables, applying an INformat as part of the conversion. **The INput function can only be applied to a character target, but the output can be character or numeric.** As a quick summary, Put can take two types of input and produces one type of output (character). INput takes one type of input (character) and can produce two types of output.

INTRODUCTION

The use of Put and INput can be easily remembered if one recalls the original use of INformats and formats. "In the old, old days" data sources and destinations were usually text. An illustration of reading from/writing to text is shown below. The put and INput functions have the same logic as put and INput statements.

INput statements, and INformats, convert external "text files" into either SAS® character or numeric variables in a SAS data set– converting from "text only" to two types of variables.

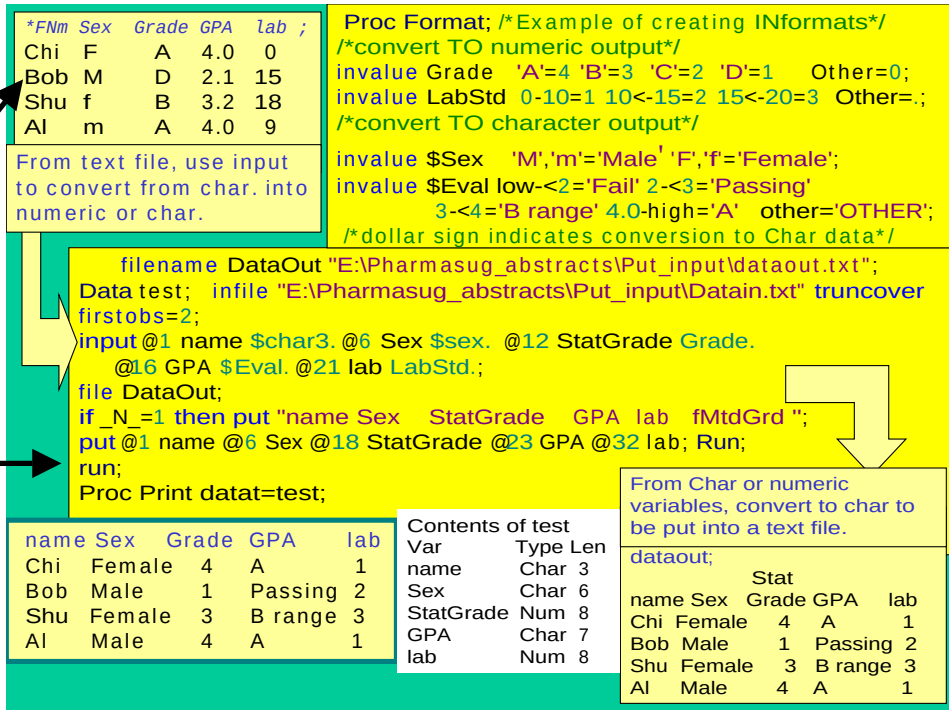

Figure 1

Put statements, and formats, are used to convert SAS character and numeric variables "into character" – two types into one.

COMBINATIONS OF SITUATIONS INVOLVING USING PUT AND INPUT FUNCTIONS

Put and INput functions are difficult to understand because several factors are associated with their use and the factors *combine* to create a large number of situations we must code, organize, study and then master. The creation of examples was approached as problem in statistical combinatorics. Some combinations of factors were *expected* to throw errors.

As we organized the research for this paper, we discovered four *possible* ways to code the *creation* of a format statement and four ways to code an INformat statement. Accordingly we coded a Proc Format that created four formats and four INformats. Some of the syntax that our combinatorial analysis created were, in fact, invalid SAS syntax and threw errors.

ABSTRACTED RULES AND EXAMPLES

FORMAT CREATION AND USE

As review, if a format is applied to a variable, it changes how the SAS variable is displayed – but does not change the stored value of the variable. As examples, think of a numeric variable X that has a value of 1.
Here is how formats change how that value of 1 is displayed, but not how it is stored.

```
X=1;
Put @10 X weekday.   ---> The 1 displays as a 7
     @20 X 5.3       ---> The 1 displays as a 1.000
     @25 X 5.1       ---> The 1 displays as a 1.0
     @30 X z5.2      ---> The 1 displays as a 01.00
     @40 x worddate20. -> The 1 displays as January 2, 1960
     @70 X weekdate17. -> The 1 displays as Sat, Jan 2, 1960
```

X has the value of 1 but is displayed in many ways.

We explored more format rules using the code below. Below we investigate format creation.

```
Proc format;/*Create 4 char and 4 num formats - with odd coding*/
/* 1*/ value $FCCC  "123" ="500";*->NOTE: Format $FCCC has been output.;
/* 2*/ value $FCNC   123  ="500";*->NOTE: Format $FCNC has been output.;
/* 3*/ value $FCCN  "123" = 500 ;*->NOTE: Format $FCCN has been output.;
/* 4*/ value $FCNN   123  = 500 ;*->NOTE: Format $FCNN has been output.;

/* 5*/*value FNCC "123"="500"; *ERROR:string '123' not acceptable to numeric format/INformat;
/* 6*/ value FNNC  123 ="500";*->NOTE: Format FNNC has been output. ;
/* 7*/*value FNF_CN "123"=500; *ERROR:string '123' not acceptable to numeric format/INformat;
/* 8*/ value FNNN  123  = 500 ;*->NOTE: Format FNNN has been output. ;run;
```

We suggest that the reader note examples /* 4*/ and /* 6*/. They seem as if they might be mis-coded. /* 4*/ is a character format (the format name starts with $) but the values in the value statement are both *numeric*. /* 6*/ creates a numeric format but the value on the right hand side (RHS) of the equal sign is character.

We conclude that SAS character formats are forgiving, *on creation*. When creating a character format, SAS does not care if the values on the right *or* left hand side of the equal sign are numeric or character. When creating a numeric format, the value to the left of the equal sign *must be* numeric. The value to the right of the equal sign can be character or numeric. We abstract a rule that says a character format is applied to a character target and a numeric format is applied to a numeric target.

The Put function uses a format to convert from a numeric variable or a character variable/constant and returns a character value. The Put function is most often used for converting a numeric value to a character value (without generating a note in the log). We will use the formats we created above to show details of Put function use. The Put statement has two parameters (target and format) separated by a comma.
An example is: `newvar = Put (target , format.).`

We were particularly interested in *applying* formats /* 4*/ and /* 6*/ (created above) because they seemed to be constructed wrongly, or at least oddly. /* 4*/ is a character format but the values on the left and the right of the equal sign are numeric. In /* 6*/, we created a numeric format, but coded a character value on the right of the equal sign. While SAS created these formats, we wondered if these oddly coded formats could cause problems in *use*.

Rule 1: On format creation:
Character formats can have numeric values on LHS and/or RHS of the = in the value statement, but Numeric formats must have a numeric value on LHS of the = in the value statement. /* 5*/ and /* 7*/ throw errors.

```
Proc format;
Proc format;/*Create 4 char and 4 num formats - with odd coding*/
/* 1*/ value $FCCC  "123" ="500";*->NOTE: Format $FCCC has been output.;
/* 2*/ value $FCNC   123  ="500";*->NOTE: Format $FCNC has been output.;
/* 3*/ value $FCCN  "123" = 500 ;*->NOTE: Format $FCCN has been output.;
/* 4*/ value $FCNN   123  = 500 ;*->NOTE: Format $FCNN has been output.;

/* 5*/*value FNCC "123"="500"; *ERROR:string '123' not acceptable to numeric format/INformat;
/* 6*/ value FNNC  123 ="500";*->NOTE: Format FNNC has been output. ;
/* 7*/*value FNF_CN "123"=500; *ERROR:string '123' not acceptable to numeric format/INformat;
/* 8*/ value FNNN  123  = 500 ;*->NOTE: Format FNNN has been output. ;run;
```

Rule 2: The Put statement accepts numeric or character targets, and numeric or character formats, ***but always produces a character value.*** FCCC_1_C123 and FNNC_6_N123 are both valued at 500 and are char 3.

```
Proc format; /* 1*/ value $FCCC "123"="500";*->NOTE: Format $FCCC has been output.;
             /* 6*/ value FNNC  123  ="500";*->NOTE: Format  FNNC has been output.;

Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
FCCC_1_C123          =Put(CV123,$FCCC.);  *produces a var. with VALUE=500 & TYPE=CHAR3;
FNNC_6_N123  =Put(NV123,FNNC.);           *produces a var. with VALUE=500 & TYPE=CHAR3;
Run;
```

Rule 3: If the target is character we must apply a character format to it and the most interesting example is /*4*/. As shown in /*4*/, a character format can be created with numeric values on the left and right sides (LHS RHS) of the equal sign, but the format must be defined as character (have a $ as the first character in the format name).

```
Proc format;/* 4*/ value $FCNN  123 = 500 ;*->NOTE: Format $FCNN has been output.;

Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
FCNN_4_C123          =Put(CV123,$FCNN.); *produces a var. with VALUE=500 & TYPE=CHAR3;
```

Rule 4: If the target is numeric **we must apply a numeric format but we will get a character output**. The interesting examples are /* 6*/ and /* 8*/. The name of a numeric format can NOT start with a $. When a numeric format is created there must be a numeric value on the left hand side (LHS) of the equal sign. When the format is created, the type (character/numeric) value on the right hand side (RHS) of the equal sign is irrelevant.

```
Proc format; /* 6*/ value FNNC  123  ="500";*->NOTE: Format FNNC has been output. ;
             /* 8*/ value FNNN  123  = 500 ;*->NOTE: Format FNNN has been output. ;

Data FormatUse;
CV123="123";      NV123=123;       Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
FNNC_6_N123   =Put(NV123,FNNC.); *produces a var. with VALUE=500 & TYPE=CHAR3;
FNNN_8_N123   =Put(NV123,FNNN.); *produces a var. with VALUE=500 & TYPE=CHAR3;
```

Rule 5: The output of a Put function will be character. In /* 1*/ the RHS of the equal sign is character and the result of the Put is character. In /* 3*/ the RHS of the equal sign is numeric and the result of the Put is still character.

```
Proc format; /* 1*/ value $FCCC  "123" ="500";*->NOTE: Format $FCCC has been output.;
             /* 3*/ value $FCCN  "123" = 500 ;*->NOTE: Format $FCCN has been output.;
```

It seems that SAS knows that formats are used to produce character and converts the RHS of a value statement to character.

```
Data FormatUse;
CV123="123";   NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
FCCC_1_C123          =Put(CV123,$FCCC.);    *VALUE= 500 - TYPE=CHAR3;
FCCN_3_C123          =Put(CV123,$FCCN.);    *VALUE= 500 - TYPE=CHAR3;
```

Rule 6: If we apply a character format to a numeric variable, SAS sends us a message that the variable has been defined as numeric. If we apply a numeric format to a character variable, SAS sends a message that the format could not be found. While the error messages are not very informative, the fact that SAS sends an error grabs our attention and is helpful.

```
Data FormatUse;
CV123="123";   NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
*FCCN_3_N123  =Put(NV123,$FCCN.);  *Variable NV123 has already been defined as numeric.;
*FNNC_6_C123  =Put(CV123,FNNC.);   *ERROR 48-59:The format $FNNC not found/could not be loaded
```

RULE 7 – FOR PUT STATEMENT - TREATMENT OF TARGETS OUT OF "MAPPING RANGE"

The "out of mapping range" issue needs separate development and different examples from the code shown above.

The treatment of values not in the "target range" can cause problems and the major issue is truncation. The default behavior of a format is to "show" any "non-mapped" values as they are in the data. This is good, if one is formatting values in a data set and less good if one is using formats to recode values. There is a danger when we pass values that are outside the mapping range to the Put. Good programming practice, when creating formats, demands the coding of an OTHER category to address this problem

CHARACTER FORMATS AND THE PUT -TREATMENT OF TARGETS OUT OF "MAPPING RANGE"

The length of the output variable is determined by the length of the longest variable on the RHS of the equal sign ("xxxxYYY"). Notice, in the example below, that HowLongA ends up as seven characters, not as long as the target string we fed to the Put. Only a truncated version of the target was passed through to the output variable.

```
*CHARACTER FORMAT AND PUT;
Proc format ; value $testB "10" = "xxxx"
                          "210"= "xxxxYYY"
 "A longlong LHS"= "xxxxYYY";run; /*Will the put display 14 or 7 chars*/

data TestingLength;
HowLongA=put("123456789",$testB.);
HowLongB=put("10",$testB.);
HowLongC=put("210",$testB.);    run;
```

7 char wide

14 char wide

```
Proc print; run;
Obs   HowLongA  HowLongB   HowLongC
 1     1234567     xxxx      xxxxYYY
```

If a target is not in the "mapping range" and is shorter than the longest variable on the RHS of the equals, there is no problem. If the target is longer than the longest variable on the RHS of the equals, it is truncated. Here are examples of applying a Put and formats to variables not in the "mapping range" for our formats.

In the example below, applying a character format works for the short variable but truncates the longer variable.

```
Proc format;/*Create 4 char and 4 num formats - with odd coding*/
/* 4*/ value $FCNN   123  = 500 ;*->NOTE: Format $FCNN has been output.;

Data FormatUse;
CV123="123";   NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";
 FCNN_4_COutShort   =Put(CShortOut,$FCNN.);  *VALUE= 9 - TYPE=CHAR3 -passed through;
 FCNN_4_COutLong    =Put(ClongOut,$FCNN.);   *VALUE= 333 - TYPE=CHAR3 -truncated;
```

NUMERIC FORMATS AND THE PUT-TREATMENT OF TARGETS OUT OF "MAPPING RANGE"

Numeric formats also take the length of the output variable from the length of the longest string on the RHS of the equals in the value statement. Out of range numbers, shorter than or equal to that length, are passed through unchanged. Numbers longer than that length are converted to scientific notation and then passed thorough.

```
*NUMERIC FORMAT AND PUT;
Proc format ;
value  testNB 1 = "xx"   210= "xxxx"  99999= "xxxxYY"; run;
data TestingLengthN;
HowLongA=put(1,testNB.);
HowLongB=put(210,testNB.);
HowLongC=put(99999,testNB.);
HowLongD=put(777777,testNB.);
HowLongE=put(7777777,testNB.);
HowLongF=put(55,testNB.);run;
```

Max Length =6

How LongA	How LongB	How LongC	How LongD	How LongE	How LongF
xx	xxxx	xxxxYY	777777	7.78E6	55

All of these are character

INFORMAT CREATION AND USE

INFORMAT CREATION

INformats change how a value is stored. This implies that INformats are used in the movement or transformation of data starting in one "place" (maybe an external file or a SAS data set) and becoming data inside a SAS file.

For example, If 01/01/60 is read with an INformat of mmddyy8. SAS stores a zero (A SAS date is the number of days since Jan, 1 1960). If the value (500) is read with an INformat of comma10., SAS stores –500. If the value 1,000,000 is read with an INformat of comma10. SAS stores 1000000. INformats change how a value is stored.

An INput function uses an INformat to convert to either character or numeric but requires a character target. The INput function has two parameters (a character target and an INformat) separated by a comma.

An example is `newvar = INput (CHARtarget , INformat.);`

Below, we investigate INformat creation.

```
Proc format;/*Create 4 char and 4 num INformats - with odd coding*/
*sample character INformats;
/*note  INValue is used to make an INformat*/
/* 1*/ INvalue $ICCC "123"="100";*  ->NOTE: INformat $ICCC has been output.;
/* 2*/ INvalue $ICNC  123 ="100";*  ->NOTE: INformat $ICNC has been output.;
/* 3*/ INvalue $ICCN "123"= 100 ;*  ->NOTE: INformat $ICCN has been output.;
/* 4*/ INvalue $ICNN  123 = 100 ;*  ->NOTE: INformat $ICNN has been output.;

*sample numeric INformats;
/* 5*/ *INvalue INCC "123"="100"; *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 6*/ *INvalue INNC  123 ="100    *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 7*/  INvalue INCN "123"= 100 ;* ->NOTE: INformat INCN has been output.;
/* 8*/  INvalue INNN  123 = 100;*  ->NOTE: INformat INNN has been output.;  run;
```

The Input function uses an INformat to convert a ONLY A character target to either a character or numeric output. We were particularly interested in applying several of the above INformats because they seemed to be constructed wrongly, or at least oddly, since the target of an INput function must be character.

It seems that the INformats /* 2*/, /* 4*/ and /* 7*/ will likely teach us most about the process.
/*2*/ is a character INformat but the LHS of the equal sign is numeric.
/*4*/ is a character INformat but the LHS and RHS of the equal sign are both numeric.
/*7*/ is a numeric INformat but the LHS of the equal sign is character.

Rule 1:
We see that, like character formats, character INformats are forgiving. When creating a character INformat, SAS does not care if the values on the right or left hand side of the equal sign are character. When creating a numeric format, the value to the right of the equal sign must be numeric. The value to the left of the equal sign can be character or numeric.

```
/* 4*/ INvalue $ICNN  123 = 100 ;*  ->NOTE: INformat $ICNN has been output.;

/* 5*/ *INvalue INCC "123"="100"; *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 6*/ *INvalue INNC  123 ="100"; *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 7*/  INvalue INCN "123"= 100 ;*  ->NOTE: INformat INCN has been output.;
/* 8*/  INvalue INNN  123 = 100 ;*  ->NOTE: INformat INNN has been output.;
```

Rule 2: The INput function wants character targets, and either numeric or character INformats. INput produces either character or numeric values. A numeric INformat applied to a character target produces numeric output.

```
Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";

ICCC_1_C123          =INput(CV123,$ICCC.); *produces a var. with VALUE = 100 - TYPE=CHAR3;
INCN_7_C123          =INput(CV123,INCN.); *produces a var. with VALUE = 100 - TYPE=Num 8;
```

Rule 3: If we pass a numeric target to an INput function, SAS will convert the target to character and write a note in the log. It will create the variable in the PDV and send it to the output data set, ***but the variable will not be valued.*** It "appears" as if SAS helped us by converting and that the INput were successful. We consider this a subtle, and dangerous, error.

```
Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";

ICCC_1_N123     =INput(NV123,$ICCC.); *Not valued-TYPE=CHAR3- Num-to-char note in log;
ICNC_2_NoutSame =INput(Nout,$ICNC.); *Not valued-TYPE=CHAR3- Num-to-char note in log;
```

The note in the log is placed after the data step and looks like this.

```
NOTE: Numeric values have been converted to character values at the places given by:
(Line):(Column).
      1048:28   1049:23   1052:28   1053:23   1061:32   1062:22   1067:28   1068:23   1071:28
      1072:23   1077:22   1085:23   1088:28   1089:23   1092:28   1098:22   1101:32   1104:23
```

Rule 4: If the INformat applied is character (the Informat name starts with $), INput returns a character.
The target must always be character.

```
Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";

ICNN_4_C123   =INput(CV123,$ICNN.); *produces a var. with VALUE = 100 - TYPE=CHAR3;
```

Rule 5: If the INformat applied is numeric, the function returns a numeric value. The target must always be character.

```
Data FormatUse;
CV123="123";    NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";

INNN_8_C123   =INput(CV123,INNN.); *produces a var. with VALUE =  100 - TYPE=Num 8;
```

RULE 6 –INPUT - TREATMENT OF TARGETS OUT OF "MAPPING RANGE"

The "out of mapping range" issue needs separate development and examples different from the ones used above.

The way INput and Informats treat values not in the "target range" can cause problems. Good programming practice, when creating INformats, demands the coding of an OTHER category to handle this situation.

The default behavior of an INformat is to let values that are not in the "mapping range" flow, *as they are,* into the new variable. A major issue is truncation and determining how many positions will be passed through to the output variable. This is complicated by the fact that INput and INformats create both character and numeric variables.

INPUT - CHARACTER TO CHARACTER CONVERSION - TARGETS OUT OF CHARACTER "MAPPING RANGE"

Truncation is the major issue here. In our investigation, the first step is to create a character INformat that will let us determine how the length of the "output" string is determined. Remember for a character INformat, we can have character values on both the LHS and the RHS of the equals. Values outside the mapping range will be bolded.

```
*INPut - the longest string is on the RHS of the =;
Proc format ; /* $ in name and the conversion is Char -> Char */
invalue $CTestCN "12"  ="joy12"      /*2 char on left and 5 char on right*/
                 "123" ="GLad123"    /*3 char on left and 7 char on right*/
                 "1234"="happy1234"; /*4 char on left and 9 char on right*/
```

Max Length =9

```
data TestingLengthV2; *will create a Char. variable;
HowLongA=INput("123",$CTestCN.); /*Mapped*/
HowLongB=INput("1234",$CTestCN.); /*Mapped*/
HowLongC=INput("12345",$CTestCN.);/*Will not be truncated*/
HowLongD=INput("9876543210",$CTestCN.); /*truncated*/
HowLongE=INput("1234987654",$CTestCN.); /*truncated*/
run;
proc contents data=TestingLengthV2;run;
Proc print data=TestingLengthV2; run;
```

Obs	HowLongA	HowLongB	HowLongC	HowLongD	HowLongE
1	GLad123	happy1234	12345	987654321	123498765

Data Set Name WORK.TESTINGLENGTHV2
Observations 1
Variables 5
All vars are character 9

#	Variable	Type	Len
1	HowLongA	Char	9
2	HowLongB	Char	9
3	HowLongC	Char	9
4	HowLongD	Char	9
5	HowLongE	Char	9

In the above example, the longest string in the value statement was 9 characters and was on the RHS of the equal sign. SAS passed **ONLY** nine characters, from values that were outside of the "mapping range" to "character type" created variables.

Lets see if SAS does something different if the longest string is on the LHS of the equal sign, as is shown below.

```
*INPut - the longest string is on the LHS of the = ;
Proc format ; /* $ in name and Char=Char */
invalue $CT_CN "12"  ="joy12"       /*2 char on left and 5 char on right*/
               "123" ="GLad123"     /*3 char on left and 7 char on right*/
               "123happy"="happyR"; /*8 char on left and 6 char on right*/
```

Max Length =8

```
data TestingLengthV3;
HowLongA=INput("123",$CT_CN.); *will create a Char. variable;
HowLongB=INput("1234",$CT_CN.); /*Not Mapped Not Truncated*/
HowLongC=INput("12345",$CT_CN.); /* Not Mapped Not Truncated*/
HowLongD=INput("9876543210",$CT_CN.); /*truncated*/
HowLongE=INput("1234987654",$CT_CN.); /*truncated*/
run;
proc contents data=TestingLengthV3; run;
Proc print data=TestingLengthV3;run;
```

obs	HowLongA	HowLongB	HowLongC	HowLongD	HowLongE
1	GLad123	1234	12345	98765432	12349876

Data Set Name WORK.TESTINGLENGTHV3
Observations 1
Variables 5
All vars are character 8

#	Variable	Type	Len
	HowLongA	Char	8
	HowLongB	Char	8
	HowLongC	Char	8
	HowLongD	Char	8
	HowLongE	Char	8

In the above example, the longest string in the value statement was 8 characters and was on the LHS of the equal sign. SAS passed **ONLY** eight characters, from values outside the "mapping range" to created character variables.

The above example allows us to abstract a rule" For user-created **character** INformats, SAS looks at the longest string, on ***either*** the RHS or LHS of the equals, and uses that to determine how many characters it should take from values outside the "mapping range". With user-created INformats, character values that are out of "mapping range" are truncated to the length of the longest value in the INformat.

INPUT - CHARACTER TO NUMERIC CONVERSION - TARGETS OUT OF "MAPPING RANGE"

This situation initially appears to be simpler than the one above since SAS throws an error if we try to create a numeric INformat with a character value to the right of the equal sign. Additionally, an INput, with a numeric INformat creates a numeric variable and we need not worry about the length of a created numeric variable. However, numeric out of "mapping ranges" can have their own subtle problems. Please focus your attention on `HowLongD and HowLongE.`

Let's examine a character to numeric conversion.

```
*Numeric INformat;
Proc format ; /*no $ in name: Do a Char->Numeric conversion */
invalue NT_CN "1"  =1000    /*1 char on left of = */
              "12" =10000   /*2 char on left of = */
              "123"=100000; /*3 char on left of = */
data TestingLengthV4;
HowLongA=INput("1",NT_CN.); *will create a numeric variable;
HowLongB=INput("12",NT_CN.);
HowLongC=INput("123",NT_CN.);
HowLongD=INput("9876543210",NT_CN.);
HowLongE=INput("1234987654",NT_CN.); run;
```

Data Set Name WORK.TESTINGLENGTHV4
Observations 1
Variables 5

Variable	Type	Len
HowLongA	Num	8
HowLongB	Num	8
HowLongC	Num	8
HowLongD	Num	8
HowLongE	Num	8

The effect of out of "mapping range" is complicated and is best understood by reading the comments below.

1 was in "mapping range"	12 was in "mapping range"	123 was in mapping range"	Target was truncated to 987	Target was truncated to 123
HowLongA	HowLongB	HowLongC	HowLongD	HowLongE
1000	10000	100000	987	100000
and mapped to numeric 1000	and was mapped to numeric 10000	and was mapped to numeric 100000	987 is not in the "mapping range" and passed through as num. 987	The truncated value of 123 is in "mapping range" **& mapped to num. 100000.**

COMBINING PUT AND INPUT AND TRICKS FOR UNUSUAL SITUATIONS

Often SAS programmers are more interested in tricks they can use in their programs than in tedious explanations of SAS internal processes. In an attempt to please those readers we offer a few examples of how to use put and INput.

One example is the commonly encountered need to zero fill character fields (like zip codes that had been converted to numeric at some time, losing their leading zeros in the process). The second example is converting text representations of dates into SAS dates.

"ZERO FILLING" EXAMPLE / TRICK

It is common to get data from clients in a variety of formats and with a variety of data quality issues.
A simple version of this "zero filling" problem is zip codes being delivered as numeric, without leading zeros.
To convert numeric zips to zero filled character, use a Put. `CharZip=Put(NumZip , Z5.);`
If the zip code is character, without leading zeros, the problem is more complex. The solution is illustrated below.

In a more complex example, character product codes have been entered oddly- with the main issues being
1) a lack of leading zeros for *some* of the product codes
2) alignment.

The business situation is:
Product codes are all six place character information. Product codes can start with a letter or number.
For a valid product code: a letter is allowed only in position one. All other places should be zeros or digits.
Data entry was not very disciplined, but the only error in our example is a lack of leading zeros.
Product codes starting with letters (e.g. A12345) were all entered correctly.
The code below nests Put and INput to fix this problem. The trick is shown below in blue.

Knowledge of SAS Sorting order helps understanding the GE, LE logic in code below. Any number sorts lower than any character. Capital letters sort lower than lower case letters

```
/*  Background on show SAS sorts character data.
The ASCII English-language ASCII sequence sorts from
 the smallest to the largest in the order shown Below,

blank ! " # $ % & ' ( ) * + , - . /0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z[ \] ^_
a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~                */

data combo;
infile datalines truncover;
input @1 odd_prod $char6.;
if upcase(substr(odd_prod,1,1)) GE "A" and /* GE, LE logic used here*/
   upcase(substr(odd_prod,1,1)) LE "z"
    then NewProd=odd_prod; /*this obs started with a letter - do not fix this obs*/
    else NewProd= Put(INput(odd_prod,6.0),Z6.0);  * there is no Zn.n Informat;
datalines;
A12345 <-OK as it is
B54321 <-OK as it is
     1 <-want to 0 fill
    21 <-want to 0 fill
   321 <-want to 0 fill
  4321 <-want to 0 fill
 54321 <-want to 0 fill
654321 <-OK as it is
;
run;


proc print ;run;
```

Logic

Business rule: If the string starts with a letter, use that string. If not, we must "zero fill"/"zero pad".

As we convert we will want to use the Z. NUMERIC FORMAT to zero pad, and to create a character output. We must do this in two steps, because there is no Zn.n INformat.

INput takes either character input and produces character, or numeric, depending on the type of INformat applied. Use input to convert `odd_prod` to numeric and to convert it as if the `odd_prod` had been 6.0.

Put takes numeric or character input and always produces character. The look of the resulting charter variable depends on the format applied. Use Put to convert from numeric to character as if the numeric value had been associated with a Z6.0 format

Results

From →	To New Prod
odd_prod	
A12345	A12345
B54321	B54321
1	000001
21	000021
321	000321
4321	004321
54321	054321
654321	654321

If the programmer has access to V9, s/he could use the following code, and save a bit of typing.

```
data Combo2;
infile datalines truncover firstobs=2;
input @1 odd_name $char6.;
if anyalpha(odd_name,1) = 1 then Newvar=odd_name;  /* anyalpha ia a sas 9 function*/
else Newvar= Put(INput(odd_name,6.0),Z6.0);
datalines;
```

The logic for the nesting of put and INput follows. What happens is that the original string, odd_prod, starts as character string with the problems shown above.

We want character output, zero filled and the Z W.w numeric format *looks* like a good tool to use in this task. The problem is that, we want to apply the Z **format** (***there is no Z INformat in SAS***) and the Z format must be applied to a numeric variable. Since our starting point is a character variable, we must first convert the character value to numeric and then convert it back to character with the Z format. Lets step through the process with the value " 54321". We must nest the use of Put and INput to solve this problem.

INput(odd_prod , 6.0) will take the character value in the variable odd_prod and convert it to numeric, as if it were six digits long with zero digits to the right of the decimal. After this inner function executes, SAS is storing a "working numeric value" of 54321.

Put("working numeric value" , Z6.0) makes a character string out of the numeric 54321 and creates the character string as if the numeric value had been associated with a Z6.0 format – a zero filled format. The result is "054321".

CONVERTING A CHARACTER DATE TO A SAS DATE EXAMPLE / TRICK

If one is fortunate enough to be sent character dates that happen to have the same structure as an INformat, the character value can be easily converted into a SAS date using the INput function and an INformat. Below are examples of this technique using several commonly encountered date INformats.

```
Data makedate;
good1 =INput("03/16/99"   ,MMDDYY8.);
good2A=INput("mar99"      ,monyy5.);
good2B=INput("mar1999"    ,monyy7.);/*SAS returns the first day of month*/
good3A=INput("99/03/16"  ,yymmdd10.);
good3B=INput("1999-03-16",yymmdd10.);
good4 =INput("199903"    ,yymmn6.);
run;
```

The N, in the INformat used to make the variable called good4, must be used and indicates that you did not separate the year and month values by blanks or by special characters. SAS automatically adds a day value of 01 to the value to make a valid SAS date variable.

The data set MakeDate is shown below.

	good1	good2A	good2B	good3A	good3B	good4
1	14319	14304	14304	14319	14319	14304

FIGURE 6

CONCLUSION

There are a bewildering number of combinations one can create when coding formats and Infomrats, Put and INput.
We can abstract the following rules describing how Put and INput behave.

Rules for PUT function and formats:

Rule 1: On format creation:
Character formats can have numeric values on LHS **and** RHS of the = in the value statement. Numeric formats must have a numeric value on LHS of the = in the value statement.
Rule 2: The Put function accepts numeric or character targets, and numeric or character formats, but always produces a character value.
Rule 3: If the target in the put function is character, we must apply a character format. The most interesting example is /* 4*/. As shown in /* 4*/, a format can be created with numeric values on the left and right sides (LHS & RHS) of the equal sign, but to be character, the format must have a $ as the first character in the format name.
Rule 4: If the target in the put function is numeric we must apply a numeric format. The most interesting examples are /* 6*/ and /* 8*/. A numeric format name must not start with a $. When a numeric format is created there must be a numeric value on the left hand side of the equal sign. When a numeric format is created, the type (character/numeric) value on the right hand side (RHS) of the equal sign is irrelevant.
Rule 5: The output of a Put will be character. In /* 1*/ the RHS of the equal is character and the result of the Put is character. In /* 3*/ the RHS of the equal is numeric and the result of the Put is still character.
Rule 6: Put does NOT help us by performing variable type conversions if we miscode.
Rule 7: Avoid passing "outside of mapping range" values to a format. The rules are complex.

Rules for INPUT function and INformats:

Rule 1:
Like character formats, character INformats are forgiving. When creating a character INformat, SAS does not care if the values on the right **or** left hand side of the equal sign are character. When creating a numeric INformat, the value to the right of the equal sign must be numeric but the value to the left of the equal sign can be character or numeric.
Rule 2: The INput function wants character targets, and either numeric or character INformats. The INput function produces either character or numeric values.
Rule 3: If we pass a numeric target to an INput function, SAS will convert the target to character and write a note in the log. It will create the variable in the PDV and send it to the output data set, **but the variable will not be valued.** This is a subtle error;
Rule 4: If the INformat applied is character (the name starts with a $), the function returns a character value.
Rule 5: If the INformat applied is numeric, the function returns a numeric value.
RULE 6: Avoid passing values that are "outside the mapping range" to an INformat. The rules are complex.

CONTACT

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

Russell Lavery Contractor for Numeric Resources
Ardmore, PA 19003,
Email: russ.lavery@verizon.net

Alejandro Jaramillo
Principal Datameans
www.datameans.com