

Paper 243-31

Getting in to the Picture (Format)

Andrew H. Karp, Sierra Information Services, Sonoma, CA USA

ABSTRACT

Although part of the SAS Format facility for well over two decades, PICTURE Formats are often not well understood, and consequently tend to be underutilized by even experienced SAS users. Yet, they provide a wealth of tools to effectively portray the values of numeric variables and often avoid the need for either tedious data step coding or to create new variables in your data sets. This tutorial describes how PICTURE Formats “work,” how to write them in PROC FORMAT statements and how to apply them in both Data and Procedure Steps. By mastering the concepts and techniques shown in this paper you will be able to apply the power of PICTURE formats in your SAS programs and in so doing often complete your data management, reporting and analysis tasks faster and easier than resorting to other potentially more tedious and time-intensive methods in the SAS System.

INTRODUCTION

A Format contains the instructions used by the SAS System to display, or portray the values of variables. More formally, formats control the “external representation” of a variable’s values. There are two broad classes of SAS Formats: VALUE Formats and PICTURE Formats. VALUE Formats are either “supplied” or “internal” to the SAS System (that is, they ‘come with’ our installation of SAS System Software) or you can create your own using PROC FORMAT and the VALUE Statement. VALUE Formats can be used to externally represent the values of either character or numeric variables.

Experienced SAS users are already aware of the broad array of tools, as well as the flexibility offered by appropriate use of both SAS-supplied and user-created VALUE Formats. Using them enhances the appearance of our output, reduces ambiguity about the definitions of values of variables appearing in our report, and can be used as a resource-saving alternative to aggregating (that is, “rolling up”) data from one unit of analysis to another. They are also exceptionally well suited for tasks such as “recoding” or “bucketing” the values of a variable in to a smaller number of discrete groups. For more information about Value Formats, please see my paper, “My Friend the SAS Format,” in the SUGI 30 Proceedings.

Picture Formats are different from Value Formats in two critical respects. First, Picture Formats can be used ONLY with numeric variables. Second, a Picture Format creates a template that is used to display the values of numeric variables. (Note: A “Picture Format Template” is NOT the same thing as a Style or Table Template in the Output Delivery System.). Also, there are no “SAS-supplied” Picture Formats; rather, you create them using the PICTURE Statement in PROC Format. Picture Formats are stored in a FORMATS Catalog in a SAS Library, just like Value Formats. And, you can use PROC FORMAT tools such as the FMTLIB, CNTLIN, CNTLOUT and MULTILABEL Options with Picture Formats as well as Value Formats.

So, what is a Picture Format Template? It is a series of commands that control how the values of numeric variables are displayed in your SAS-generated output. Once you master the core concepts and functionalities of Picture Formats you will find them a powerful and flexible tool with which to enhance the quality of your reports and analyses.

GETTING STARTED WITH PICTURE FORMATS: A BASIC EXAMPLE AND ESSENTIAL CONCEPTS**PICTURE FORMAT VS. THE DATA STEP...THE WINNER IS?**

To fix ideas, suppose we have a numeric variable in a data set representing a series of telephone numbers in the United States. The first three digits are the area code, the next three are the “exchange” and the last four are the number itself. What we want to do is insert a hyphen (dash) between the third and fourth digit and between the sixth and seventh digit. Here’s an example data set, where the “raw data” are included within the Data Step creating the data set using a “Datalines” statement.

```
* example 1;
data phones1;
input phone_number;
datalines;
2022933923
4154410702
9083038859
7079351413
;
```

```
run;

options nonumber nocenter nodate;
proc print data=phones1;
title 'Getting in to the Picture Format';
title2 'Phone Number Data Set 1';
run;
```

The PROC PRINT-generated output is:

Now, what the boss really wants is to have the output displayed with hyphens between the area code and exchange and between the exchange and the remaining four digits of the telephone number. Since we don't have a SAS-supplied "telephone number format," there's no help there. And, it's pretty clear that a VALUE format isn't going to help either, since it will control the display (that is, the representation) of the values of the variable.

<i>Obs</i>	<i>phone_number</i>
1	2022933923
2	4154410702
3	9083038859
4	7079351413

One approach might be to turn the telephone number in to a character, rather than a numeric, variable and then use a combination of the SUBSTR (substring) SAS Programming Language Function and the concatenation operator in a data step to "break apart" the "pieces" of the telephone number and then essentially reassemble them in a new variable with the required hyphens. A Data Step implementing this approach might look like this (with a PROC PRINT to display the results):

```
21 * example 2;
22 data phones2(drop=char_phone);
23 length new_number1 $ 12;
24 input phone_number;
25 char_phone = put(phone_number,$10.);
26 new_number1 = substr(char_phone,1,3)||'-'||substr(char_phone,4,3)||'-'||substr(char_phone,7,4);
27 datalines;
28 2022933923
29 4154410702
30 9083038859
31 7079351413
32 ;
33 run;
34
35 proc print data=phones2;
36 title2 'Phone Number Data Set 2';
37 run;
```

The output is shown on the next page:

Getting in to the Picture Format Phone Number Data Set 2

<i>Obs</i>	<i>new_number1</i>	<i>phone_number</i>
1	202-293-3923	2022933923
2	415-441-0702	4154410702
3	908-303-8859	9083038859
4	707-935-1413	7079351413

Well, we got what we wanted, but at the expense of doing a lot of work in the Data Step (i.e., numeric-to-character conversion of the value of the original telephone number variable, then a fairly complex assignment statement to display the values of telephone number in the desired way. While there are other ways this Data Step could have been written, the “take-home message” here is that because of the tools available in the Picture Format facility, there’s no need to apply a Data Step approach to this problem in the first place.

EXAMPLE 1: USING A SIMPLE PICTURE FORMAT

Here’s how a Picture Format is created, and then applied to the telephone number data set. After seeing what it does, we’ll go over the syntax and options of this initial, and very basic Picture Format,

and use it to set the stage for identifying more of the Picture Format’s functionalities.

```

60 proc format;
61 picture phone_a
62     low-high = '999-999-9999';
63 run;
64
65 proc print data=phones1;
66 title2 'Using a Picture Format';
67 format phone_number phone_a.;
68 run;

```

Name of Picture Format

The template, showing a series of digit selectors

Range of values to which the Picture Format will be applied

Associating the Picture Format to a Variable

Let’s go over the syntax step by step. First, we’re calling, or starting the Format Procedure with the PROC FORMAT statement, just like we would to create a Value Format. But, the PICTURE Statement tells the PROC that we’re about to create a Picture Format, the name of which is given immediately to the right of “PICTURE.” So, far, we know we’re creating a Picture Format called “phone_a.” Next, we’re supplying the range of values to which the Picture is going to be applied. In this example, we’re using the ‘low’ and ‘high’ keywords, separated by a dash, to instruct PROC FORMAT that the Picture Format we’re creating will be used to display all (non-missing) values of the variable to which it will be associated in either a subsequent Data or Procedure Step. To the right of the equals sign is the template, or instructions as to how the value of the variable is to be displayed. The strings of “9’s” are called *digit selectors*, and will be explained in more detail later in this paper.

Once the Phone_a Picture Format is created, we can associate it to the values of a variable in, say a PROC PRINT Step, and see the results. Here they are:

Getting in to the Picture Format Using a Picture Format

<i>Obs</i>	<i>phone_number</i>
1	202-293-3923
2	415-441-0702
3	908-303-8859
4	707-935-1413

Just from this basic example we can see that the Picture Format gave us exactly what we needed without a lot of tedious, potentially error-prone and inefficient Data Step programming. With the Picture Format, we did NOT need to do anything other than create the format and then associate it to a variable to obtain the data display we need.

Now, let's identify some other basic, but very useful aspects of the Picture Format facility before delving its details and advanced capabilities. Suppose, having seen the fine job you've done on the previous task, the boss changes her mind and asks that the area code be enclosed in parentheses, with one space between the right parenthesis and the exchange, and she still wants a hyphen between the exchange and the rest of the telephone number. Most of you know this as a typical "boss question" that usually starts with something like "Well, this is nice, but how hard would it be to make a couple of tiny changes..."

Fortunately, the Picture Format's PREFIX option will give us just what we want. While we'll get in to the details of digit selectors and other details of the Picture Format facility shortly, one core rule for Picture Formats is that if you use digit selectors as the template for displaying your data, the first position of the template must be a digit selector. That means we can't make the right parentheses we need part of the template itself, since it is obviously not a digit. Instead, we will instruct PROC FORMAT to make the left parentheses the prefix to the template, which will start with the (required) digit selector. Here we go:

```

74
75 proc format;
76   picture phone_b (default = 16)
77     low-high = '999) 999-9999'
78     ( prefix = '(' ) ;
79   run;
80
81 proc print data=phones1;
82   title2 'Using a Picture Format with a PREFIX';
83   format phone_number phone_b.;
84   run;

```

This PROC FORMAT task shows you two important options to the PICTURE statement. First, we're making the default length of the Picture Format 16 characters, which is wide enough to accommodate BOTH the template AND the specified prefix. The prefix we want is given in the PREFIX option, which, like the DEFAULT option is enclosed in parentheses. Be careful: The PREFIX *option* is in the parentheses, and the *value* of the option is a *left parentheses symbol enclosed in single quotes*. The PROC PRINT Output is:

So, with just a small amount of additional work in the PROC FORMAT step, we have exactly what the boss wants (until, of course, she changes her mind again) without resort to complex and tedious Data Step coding.

Getting in to the Picture Format Using a Picture Format with a PREFIX

Obs	phone_number
1	(202) 293-3923
2	(415) 441-0702
3	(908) 303-8859
4	(707) 935-1413

Getting in to the Picture Format Data Set Phones 3 (Internal Values)

Obs	number
1	7079967380
2	2024561414
3	9351413
4	4154410702
5	1005551212
6	1008484
7	5551212
8	9967380
9	2338786
10	9876

One more fairly basic example and then we'll take a more detailed look at PICTURE Statement syntax, options, rules and more advanced capabilities. Suppose we have a data set that looks like the one shown to the left.

In the United States, some of the values of Number in this data set correspond to local numbers (seven digits, three for exchange and four for the number) and others to long distance number (ten digits, with the first three representing the area code. We also have three "bad" observations in the data set. A valid local number (again, in the USA), has to be at least seven digits long and must start

with the number two. Valid area codes in long distance numbers start with the number two. Under these “rules,” observations 5, 6 and 10 have invalid values of a telephone number. What we now need to do is create a picture format that has three “rules” or value ranges to it: 1) if the length of the telephone number is ten digits and starts with the number two, then we want the area code enclosed in parentheses and a hyphen separating the exchange from the number or, 2) if the length of the telephone number is seven digits and the first digit is a two, then we want a hyphen separating the exchange from the number or, 3) first digit of a ten or seven digit number is a 1, or the length of the phone number is LESS than seven digits then we want “Invalid Number” displayed in our output. Needless to say, coding these rules in a Data Step, especially when we are “starting” with a numeric variable, could be very tedious.

Fortunately, the Picture Format facility lets avoid a lot of unnecessary programming and still get what we need, quickly and easily. Here’s the solution:

```

218
219 proc format;
220   picture phone_c (default = 35)
221     low - 1999999 = 'Invalid Local Number'
222     2000000 - 9999999 = '999-9999'
223     1000000000 - 1999999999 = 'Invalid Long Distance Number'
224     2000000000 - high = '999) 999-9999' (prefix = '(' );
225   run;
226
227 proc print data=phones3;
228   format number phone_c.;
229   title 'Conditional Picture Format';
230   run;

```

In this example, we can see how a series of value ranges were supplied, each with a different picture template. So, the variable’s display is controlled by its internal value. The results shown below and to the left:

Getting in to the Picture Format
Conditional Picture Format

Obs	number
1	(707) 996-7380
2	(202) 456-1414
3	935-1413
4	(415) 441-0702
5	Invalid Long Distance Number
6	Invalid Local Number
7	555-1212
8	996-7380
9	233-8786
10	Invalid Local Number

The three examples of Picture Formats we’ve seen so far should be enough to convince you that they offer a power range of tools to display or portray the values of numeric variables without extensive, tedious, data step coding. So, having seen these examples, I hope you’ll want to continue reading this paper to see even more tools and capabilities of Picture Formats and how you can apply them in your work.

PICTURE FORMAT DETAILS AND SYNTAX

RULES FOR PICTURE FORMATS

Picture Format names can be up to 32 characters in length starting with the release of SAS 9.1 Software. The name you give to a Picture Format cannot be the name of a SAS-Supplied format, nor may it end in a number. Picture Formats are used to display the values of numeric variables. Like Value Formats, Picture Formats can be stored in either a temporary or permanent Formats Catalog. Although not discussed in this paper, the new (to SAS 8, and enhanced in SAS 9) MULTILABEL option can be used to create a Picture Format with overlapping value ranges. (For more information, please see the PROC FORMAT documentation and/or my paper "Using Multilabel Formats, available for download at www.SierraInformation.com)

A Picture Format may be up to 40 characters in length. A 'picture' or 'template' is a series of characters in single quote marks. The characters forming the 'template' or 'picture' can be one of three types:

- *Digit selectors*, which numeric characters ranging from zero to nine defining positions in which the values of numbers in the variable will be displayed. As we will soon see, there is a critical difference in results when you apply a "zero digit selector" versus a "non-zero digit selector."
- *Message characters*, which are non-numeric characters that will be printed in the picture. For example of a message character, see the Phone_C Picture format above where some values of telephone number were displayed as either "Invalid Long Distance Number" or "Invalid Local Number."
- *Directives*, which control the display of date, time or datetime variables. These special characters require specification of the DATATYPE= option in the PICTURE Statement, and will be discussed in detail below.

UNDERSTANDING DIGIT SELECTORS

Perhaps one of the most confusing aspects of Picture Formats is the "digit selector." But, by working through a few examples, we'll see how digit selectors work and how to specify them correctly for your particular data presentation needs. First, let's review some core concepts: 1) a Picture Format is creating a template (or 'picture') that will display the values of a numeric variable in your SAS-generated output; 2) if you are using a Picture Format to display numeric values (as opposed to message characters, which we will discuss next), you'll need to tell PROC FORMAT how many "slots" or "spaces" in the Picture Format are needed to display the values of the numeric variable *and* what to display if there's no value to display in a "slot." That's that the digit selector does.

A digit selector is either a zero (0) or the numbers one (1) through nine (9). Most SAS users, and the PROC FORMAT documentation, use *either* a zero *or* a nine as digit selectors, so that's the same convention I'll apply in this paper. When you specify zero as the digit selector, any leading zeros in the number to be displayed are shown as blanks. When nine is specified as the digit selector, the leading zeros are displayed in the output. Perhaps the easiest way to remember how digit selectors work is the saying taught to me by Pete Lund of Looking Glass Analytics, who has also written extensively about the SAS Formats (see below.) The saying is: "Nines print zeros, and zeros print blanks."

Let's take a look at how digit selectors are used in a Picture Format, and what happens when you use either a zero or a nine for a digit selector. The example data set below shows some made-up values for sales of parts in an automobile store.

First, a Picture Format with a string of nines as digit selectors is created and applied the variable SALES.

*Getting in to the Picture Format
Understanding Digit Selectors*

<i>Obs</i>	<i>item</i>	<i>sales</i>
1	tires	154000.89
2	batteries	891094.50
3	windshields	34305.08
4	sparkplugs	189041.03
5	polish	56012.25
6	gas_additives	435.87
7	engines	5689.00
8	distributor_caps	0.00

```

108 proc format;
109   picture part_fmt_a
110     low - high = '999,999,999.99' (prefix = '$');
111   run;
112
113 proc print data=parts;
114   format sales part_fmt_a.;
115   title3 'Nines Print Zeros';
116   run;

```

The output looks like this:

Getting in to the Picture Format
Understanding Digit Selectors
Nines Print Zeros

Obs	item	sales
1	tires	000,154,000.89
2	batteries	000,891,094.50
3	windshields	000,034,305.08
4	sparkplugs	000,189,041.03
5	polish	000,056,012.25
6	gas_additives	000,000,435.87
7	engines	000,005,689.00
8	distributor_caps	000,000,000.00

Using a series of nines as the digit selectors results in having zeros displayed in the output for every “position” in the picture template for which there was no value of the variable to which it was applied. Remembering that “nines print zeros and zeros print blanks,” one potential approach to a better-looking result might be to replace all the nines with zeros. What happens when we do that?

```

118 proc format;
119   picture part_fmt_b
120     low - high = '000,000,000.00' (prefix = '$ ');
121   run;
122
123 proc print data=parts;
124   format sales part_fmt_b.;
125   title3 'Zeros Print Blanks';
126   run;

```

The output now looks like this:

Getting in to the Picture Format
Understanding Digit Selectors
Zeros Print Blanks


Obs	item	sales
1	tires	\$ 154,000.89
2	batteries	\$ 891,094.50
3	windshields	\$ 34,305.08
4	sparkplugs	\$ 189,041.03
5	polish	\$ 56,012.25
6	gas_additives	\$ 435.87
7	engines	\$ 5,689.00
8	distributor_caps	

Well, by using nines as our digit selectors we've managed to address the problem with leading zeros. But, remember, since "nines print blanks," if the value of the variable to which the Picture Format is applied is a zero, then the formatted value that appears in our output is a blank! In many situations, that can lead to some confusion...in our example, having a 'blank' value of sales for distributor caps is misleading. Does it mean we sold no distributor caps, or does it mean we are missing data in the source data set for this value of the parts variable? Since we don't want to give our clients/customers confusing reports, one way to solve this problem is to use both zeros and nines as digit selectors in the same Picture template. Here's an example that will give us the solution we need:

```

298 proc format;
299   picture part_fmt_c
300     low - high = '000,000,009.99' (prefix = '$ ');
301   run;
302
303 proc print data=parts;
304   format sales part_fmt_c.;
305   sum sales;
306   title3 'Using Both Zeros and Nines in the Same Picture';
307   run;

```



In this Picture Format I've combined both zero and nine digit selectors in one template. The result, shown below, gives us exactly what we need. If you're creating a picture format to display numeric variables that may have values of zeros, I'd recommend your using an appropriate combination of zero and nine digit selectors so that you have zero values displayed in your output.

Getting in to the Picture Format
Understanding Digit Selectors
Using Both Zeros and Nines in the Same Picture

Obs	item	sales
1	tires	\$ 154,000.89
2	batteries	\$ 891,094.50
3	windshields	\$ 34,305.08
4	sparkplugs	\$ 189,041.03
5	polish	\$ 56,012.25
6	gas_additives	\$ 435.87
7	engines	\$ 5,689.00
8	distributor_caps	\$ 0.00
		\$ 1,330,578.62



EVEN MORE PICTURE FORMAT TOOLS: THE MULTIPLIER AND ROUND OPTIONS

In my opinion, two of the most useful and powerful tools in the Picture Format "arsenal" are the MULTIPLIER (or MULT) and ROUND options. With these, we can usually avoid "pre-processing" observations via a Data Step before obtaining the output we need without a potentially tedious, time-consuming or resource-intensive Data Step.

Let's first take a look at the MULT option, and then the ROUND Option, and then we'll use both in one Picture Format.

THE MULT OPTION

This option allows you to provide a constant by which the values of the number is to be multiplied before it is formatted. With it, you can easy carry out tasks such as "round up" financial data to the nearest thousand (or some other appropriate value) convert values from one unit of measurement to another (e.g., from inches to centimeters or from US dollars to another currency, Using the MULT option is not only easy, but it avoids unnecessary Data Step processing and allows you to easily change the value of the multiplier, if, for example you are using it to calculate currency exchange rates that change between each "run" of a report.

Here is a PROC REPORT task that generates a report from some (simulated) credit card transaction data. This very

powerful PROC is used to group and sum credit card charges by year and credit card used. This data set has over 265,000 observations in it, so processing time may be something to keep in mind as we consider requests to change the report.

```

159 proc report nowindows data=picture.card_data1(where=(year(trans_date) > 1999))
160     headline headskip split='*';
161 column cardtype trans_date=year trans_date charge_amount;
162 define cardtype/group 'Card Group' format=$cardf.;
163 define year/group 'Year' format = year.;
164 define trans_date/group format=yyq. 'Year*and*Quarter' width=15;
165 define charge_amount/sum format=dollar16.2 'Total Charges';
166 break after cardtype /summarize skip ol;
167 rbreak after/summarize skip dol;
168 title2 'Credit Card Charges Data Set';
169 run;

```

The output is:

Getting in to the Picture Format
Credit Card Charges Data Set

Card Group	Year	Year and Quarter	Total Charges
All MasterCard Products	2000	2000 Q1	\$11,695,978.60
		2000 Q2	\$11,849,094.11
		2000 Q3	\$11,986,563.95
		2000 Q4	\$12,000,030.20
	2001	2001 Q1	\$6,479,911.84
		2001 Q2	\$6,821,829.31
		2001 Q3	\$4,913,234.70
		2001 Q4	\$1,962,720.18
<i>All MasterCard Products</i>			\$67,509,362.89
All Visa Products	2000	2000 Q1	\$5,776,005.52
		2000 Q2	\$5,749,991.18
		2000 Q3	\$5,947,971.57
		2000 Q4	\$5,782,536.44
	2001	2001 Q1	\$3,180,145.36
		2001 Q2	\$3,237,830.61
		2001 Q3	\$2,348,017.60
		2001 Q4	\$976,658.05
<i>All Visa Products</i>			\$32,999,156.33
			\$100,508,519.22

So far, so good. But, what do we do when the boss asks her typical “how hard would it be” question: can we display the credit card charges rounded to the nearest thousand dollars? Some might want to create a new data set, and create a new variable in that data set, where each of the more than quarter-million individual records had their values of the variable charge_amount rounded to the nearest thousand, and then have PROC REPORT re-generate the analysis we need.

USING THE ROUND OPTION

We can avoid the Data Step with the MULT Option, and avoid any potential truncation problems by also specifying the ROUND Option. Without the ROUND Option, PROC FORMAT would *automatically truncate* any decimal portion of the variable's value and then display the result according to the defined template. When you specify the ROUND Option and the MULT Option, PROC FORMAT *first* multiplies the variable's value by the supplied multiplier, and *then* rounds the results to the nearest integer and *then* formats the value according to the template. According to the PROC FORMAT documentation, a value of exactly .5 is rounded *up* to the next highest integer.

For complete details on how the steps PROC FORMAT follows to build Picture Formats, see Chapter 23 of the BASE SAS Procedures Documentation, and in particular, Table 23.1 ("Building a Picture Format").

The code sample below shows how the MULT Option is added to the Picture Format, and then how the Format is applied in a PROC REPORT Define Statement. Then, the output generated by the PROC REPORT task is displayed.

```

198 proc format;
199   picture card_b_fmt (round)
200     low-high = '000,000,009' (mult=.001) ;
201   run;
202
203 proc report nowindows data=picture.card_data1(where=(year(trans_date) > 1999))
204   headline headskip split='*';
205 column cardtype trans_date=year trans_date charge_amount;
206 define cardtype/group 'Card Group' format=$cardf.;
207 define year/group 'Year' format = year.;
208 define trans_date/group format=yyq. 'Year*and*Quarter' width=15;
209 define charge_amount/sum format=card_b_fmt. 'Total*Charges*(000)';
210 break after cardtype /summarize skip ol;
211 rbreak after/summarize skip dol;
212 title2 'Credit Card Charges Data Set';
213 title3 'Using the Card_B_Fmt Picture Format';
214 run;

```

The resulting output looks like this:

Getting in to the Picture Format
Credit Card Charges Data Set
Using the Card_B_Fmt Picture Format

Card Group	Year	Year and Quarter	Total Charges (000)
All MasterCard Products	2000	2000Q1	11,696
		2000Q2	11,849
		2000Q3	11,987
		2000Q4	12,000
	2001	2001Q1	6,480
		2001Q2	6,622
		2001Q3	4,913
		2001Q4	1,963
<i>All MasterCard Products</i>			67,509
All Visa Products	2000	2000Q1	5,776
		2000Q2	5,750
		2000Q3	5,948
		2000Q4	5,783
	2001	2001Q1	3,180
		2001Q2	3,238
		2001Q3	2,348
		2001Q4	977
<i>All Visa Products</i>			32,999
			100,509

USING PICTURE FORMATS TO CONVERT VALUES OF VARIABLES

In my opinion, the most powerful application of Picture Formats is to carry out conversions of variable values from one unit of measurement to another without having to use a Data Step to operate on every observation in the source data set. Instead, a Picture Format can give us what we want and affords greater flexibility in our programming.

Perhaps the best example of how Picture Formats can help us to convert values of one variable to another is a conversion of numeric variables from one currency to another. This is a fairly common requirement for SAS users working with data that are drawn from financial/accounting systems in many countries, each with its own currency. And, since the rate at which currencies are exchanged changes frequently, a SAS program that takes data stored in one currency and displays it in another currency needs to be updated with the latest exchange rate prior its execution.

Here is an example of how a Picture Format is used to display values that are stored in US dollars in the credit card transaction file seen above as the equivalent value in Japanese Yen. The Picture Format code below shows how the value supplied to the MULT option is the exchange rate on the date I first generated this example. Obviously, if this is something you need to do at your job on an ongoing basis, its very easy to update the value in the MULT option with the latest exchange rate just before you run your program again. Here are the PROC FORMAT and PROC REPORT steps that generate the desired report: Notice that I've modified the PROC REPORT task so that the output it generates clearly indicates that the values displayed are the Japanese Yen.

```

220 proc format;
221   picture jpy_fmt (round)
222     /* exchange rate is 1 US Dollar = 118.29 JPY on 11/05/05 */
223     low-high = '000,000,000,000' (mult=118.29);
224   run;
225
226 proc report nowindows data=picture.card_data1(where=(year > 1999))
227   headline headskip split='*';
228 column cardtype trans_date=year trans_date charge_amount;
229 define cardtype/group 'Card Group' format=$cardf.;
230 define year/group 'Year' format = year.;
231 define trans_date/group format=yyq. 'Year*and*Quarter' width=15;
232 define charge_amount/sum format=jpy_fmt. 'Total*Charges*JPY';
233 break after cardtype /summarize skip ol;
234 rbreak after/summarize skip dol;
235 title2 'Credit Card Charges Data Set';
236 title3 'Converting Dollars to Yen';
237 run;

```

The resulting output is shown on the next page:

*Getting in to the Picture Format
Credit Card Charges Data Set
Converting Dollars to Yen*

<i>Card Group</i>	<i>Year</i>	<i>Year and Quarter</i>	<i>Total Charges JPY</i>
MasterCard	2000	2000Q1	1,383,517,309
		2000Q2	1,401,629,342
		2000Q3	1,417,890,650
		2000Q4	1,419,483,572
	2001	2001Q1	766,508,772
		2001Q2	783,296,189
		2001Q3	581,186,533
		2001Q4	232,170,170
<i>MasterCard</i>			<i>7,985,682,536</i>
Visa	2000	2000Q1	683,243,693
		2000Q2	680,166,457
		2000Q3	703,585,557
		2000Q4	684,016,235
	2001	2001Q1	376,179,395
		2001Q2	383,002,983
		2001Q3	277,747,002
		2001Q4	115,528,881
<i>Visa</i>			<i>3,903,470,202</i>
			<i>11,889,152,739</i>

USING PICTURE FORMATS IN VARIABLE ASSIGNMENT STATEMENTS

Picture Formats can also be used to assign the values of new variables in Data Steps. While I've generally want to avoid creating "extra" variables in data sets, there may be times when we need to assign values to a new variable based on the commands placed in a Picture Format. Here is an example: suppose the boss asks: "How hard would it be to give me an Excel™ spreadsheet that shows the amount of credit card transactions by credit card and year, with separate columns showing total charge amounts in US Dollars, Japanese Yen and British Pounds?" Here is what we will do:

- 1) Create Picture Formats for Great Britain Pounds (GBP) and Japanese Yen (JPY). The variable charge_amount already "holds" the data in US Dollars.
- 2) Use the PUT Function in a Data Step to create two new variables, each of which applies the previous-created Picture Format to the values of charge_amount.
- 3) Before exporting the results an Excel workbook, we will look at the results of steps 1 and 2 using PROC PRINT. If we are happy with them, then a tool such as PROC EXPORT or the Export Wizard can be employed to create an Excel workbook from the data set.

First, let's create the appropriate Picture Formats:

```

255 proc format;
256   picture jpy_fmt (round)
257     low-high = '000,000,000,000' (mult=118.29 prefix='JPY ');
258   picture gbp_fmt (round)
259     low-high = '000,000,000,000' (prefix = 'GBP ' mult=.5658);
260   run;

```

Now, well, use these Picture Formats to assign values to new variables in a Data Step and then look at the results:

```

282 data picture.card_data3;
283 set picture.card_data2;
284 * note: new vars are character;
285 JPY = put(charge_amount,JPY_FMT.);
286 GBP = put(charge_amount,GBP_FMT.);
287 label
288   charge_amount = 'US Dollars'
289   JPY           = 'Japanese Yen'
290   GBP           = 'British Pounds'
291   transactions  = 'Transaction Count';
292 format charge_amount dollar18.2;
293 run;
294
295 proc print data=picture.card_data3 label noobs;
296 sum transactions;
297 format transactions comma12.;
298 label trans_date = 'Year';
299 title2 'Assigning Values to Variables Using Picture Formats';
300 run;

```

The results are:

Getting in to the Picture Format
Assigning Values to Variables Using Picture Formats

<i>cardtype</i>	<i>Year</i>	<i>Transaction Count</i>	<i>US Dollars</i>	<i>Japanese Yen</i>	<i>British Pounds</i>
MC	1999	348	\$6,769,432.54	JPY 800,756,175	GBP 3,830,145
MC	2000	366	\$14,394,637.67	JPY 1,702,741,690	GBP 8,144,486
MC	2001	363	\$6,042,819.90	JPY 714,805,166	GBP 3,419,027
MG	1999	348	\$11,108,410.17	JPY 1,314,013,839	GBP 6,285,138
MG	2000	366	\$23,603,545.86	JPY 2,792,063,440	GBP 13,354,886
MG	2001	362	\$9,897,392.75	JPY 1,170,762,588	GBP 5,599,945
MP	1999	347	\$4,593,576.64	JPY 543,374,181	GBP 2,599,046
MP	2000	366	\$9,533,483.33	JPY 1,127,715,743	GBP 5,394,045
MP	2001	363	\$4,037,483.38	JPY 477,593,909	GBP 2,284,408
VC	1999	348	\$6,697,977.84	JPY 792,303,799	GBP 3,789,716
VC	2000	366	\$14,322,044.34	JPY 1,694,154,625	GBP 8,103,413
VC	2001	363	\$6,048,885.86	JPY 715,522,708	GBP 3,422,460
VG	1999	337	\$1,917,230.26	JPY 226,789,167	GBP 1,084,769
VG	2000	366	\$3,890,527.69	JPY 460,210,520	GBP 2,201,261
VG	2001	360	\$1,667,274.92	JPY 197,221,950	GBP 943,344
VP	1999	342	\$2,361,984.97	JPY 279,399,202	GBP 1,336,411
VP	2000	366	\$5,043,932.68	JPY 596,646,797	GBP 2,853,857
VP	2001	358	\$2,026,490.84	JPY 239,713,601	GBP 1,146,589
		6,435			

USING DIRECTIVES WITH DATE, TIME OR DATETIME VARIABLES

The last section of this paper discusses the use of *directives* used to format the values of date, time or datetime variables in Picture Formats. Even though we have a wide range of useful SAS-supplied Value Formats to associate to variables representing dates and times, these Picture Format directives give us an even broader array of ways to display this type of data. In order to apply directives in a Picture Format, you need to include the DATATYPE= option in the Picture Statement. The valid values of this option are DATE, TIME or DATETIME.

This table shows some of the directives you can use in a Picture Format to customize the display of values of your date, time or datetime variables. Please consult the PROC FORMAT documentation for a complete list.

%a Weekday abbreviation	%m Month number
%A Weekday name	%U Week of the year
%B Month name	%W Weekday number
%b Month abbreviation	%y Two-digit year (no century)
%d Day of month number	%Y Four-digit year (with century)
%j Day of year number	

To fix ideas about directives might be used to create a customized Picture Format, below is an excerpt from the credit card transaction file that's been used for several other examples in this paper. Notice that the variable `trans_date` is a SAS Date variable (that is, an integer representing the number of days from January 1, 1960.)

Getting in to the Picture Format
Card Sample Data Set

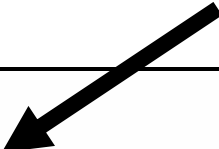
<i>Obs</i>	<i>cardnumber</i>	<i>charge_amount</i>	<i>trans_date</i>	<i>cardtype</i>
1	9631-1882-1895-6743	333.35	14359	MC
2	9631-1977-2464-8560	678.22	14735	VC
3	9631-2252-3309-0946	299.97	14638	VC
4	9631-2252-3309-0946	222.33	14596	VC
5	9631-3412-0432-0156	674.68	15133	VP
6	9631-5398-4757-5161	457.11	14827	MG
7	9631-5780-2098-6164	77.45	14478	MC
8	9631-7259-2547-2879	300.66	14923	VC
9	9631-8657-5136-8015	398.07	14598	VC
10	9632-1451-0411-9691	855.36	14397	MC
11	9632-1695-0593-7743	933.11	14586	MC
12	9632-2487-4712-5614	381.06	15253	MP
13	9632-3438-1227-7448	491.07	14714	VG
14	9632-4638-1238-1596	891.74	15010	MP
15	9632-4865-3422-3280	499.86	14968	MG
16	9632-4891-5920-6873	479.29	14990	MG
17	9632-6379-2827-9543	723.92	15051	MC
18	9632-8045-2962-5084	852.73	14839	MG
19	9632-8492-3685-4607	795.40	14931	VC
20	9632-9273-3755-6854	66.83	14900	MG
21	9632-9901-0316-3175	417.11	14780	MG

Here is an example of a PROC FORMAT task where a Picture Format is created using directives to display the values of the SAS date variable `trans_date`.

```

357 proc format;
358     picture date_b_fmt (default = 45)
359     low-high = 'Trans. Date Was: %B %d, %Y (%A)'
360     (datatype = date);
361     run;

```

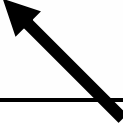


Next, the date_b_fmt Picture Format is associated to the variable trans_date in the following PROC PRINT step:

```

366
367 proc print data=picture.card_sample;
368     format trans_date date_b_fmt.;
369     title2 'Using Date_b_fmt';
370     run;

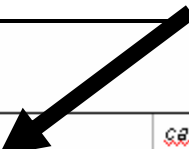
```



The output is:

Getting in to the Picture Format
Using Date_b_fmt

<u>Obs</u>	<u>cardnumber</u>	<u>charge_amount</u>	<u>trans_date</u>	<u>cardtype</u>
1	9631-1882-1895-6743	333.35	Trans. Date Was: April 25, 1999 (Sunday)	MC
2	9631-1977-2464-8560	678.22	Trans. Date Was: May 5, 2000 (Friday)	VC
3	9631-2252-3309-0946	299.97	Trans. Date Was: January 29, 2000 (Saturday)	VC
4	9631-2252-3309-0946	222.33	Trans. Date Was: December 18, 1999 (Saturday)	VC
5	9631-3412-0432-0156	674.68	Trans. Date Was: June 7, 2001 (Thursday)	VP
6	9631-5398-4757-5161	457.11	Trans. Date Was: August 5, 2000 (Saturday)	MG
7	9631-5780-2098-6164	77.45	Trans. Date Was: August 22, 1999 (Sunday)	MC
8	9631-7259-2547-2879	300.66	Trans. Date Was: November 9, 2000 (Thursday)	VC
9	9631-8657-5136-8015	398.07	Trans. Date Was: December 20, 1999 (Monday)	VC
10	9632-1451-0411-9691	855.36	Trans. Date Was: June 2, 1999 (Wednesday)	MC
11	9632-1695-0593-7743	933.11	Trans. Date Was: December 8, 1999 (Wednesday)	MC
12	9632-2487-4712-5614	381.06	Trans. Date Was: October 5, 2001 (Friday)	MP
13	9632-3438-1227-7448	491.07	Trans. Date Was: April 14, 2000 (Friday)	VG
14	9632-4638-1238-1596	891.74	Trans. Date Was: February 4, 2001 (Sunday)	MP
15	9632-4865-3422-3280	499.86	Trans. Date Was: December 24, 2000 (Sunday)	MG
16	9632-4891-5920-6873	479.29	Trans. Date Was: January 15, 2001 (Monday)	MG
17	9632-6379-2827-9543	723.92	Trans. Date Was: March 17, 2001 (Saturday)	MC
18	9632-8045-2962-5084	852.73	Trans. Date Was: August 17, 2000 (Thursday)	MG
19	9632-8492-3685-4607	795.40	Trans. Date Was: November 17, 2000 (Friday)	VC
20	9632-9273-3755-6854	66.83	Trans. Date Was: October 17, 2000 (Tuesday)	MG
21	9632-9901-0316-3175	417.11	Trans. Date Was: June 19, 2000 (Monday)	MG



CONCLUSION

The Picture Format facility contains a wealth of tools you can use to work effectively—and efficiently—with your data. Taking the time to master the concepts and capabilities of this aspect of the SAS System can provide you with a range of enhanced capabilities to portray, manage and analyze your data.

REFERENCES

Karp, Andrew, *My Friend the SAS Format*, Proceedings of the Thirtieth Annual SAS Users Group International Conference, Cary: NC SAS Institute, Inc., 2005

URL: <http://www2.sas.com/proceedings/sugi30/253-30.pdf>

Lund, Peter, *More Than Just VALUE: A Look Into the Depths of PROC FORMAT*, Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference, Cary, NC: SAS Institute, Inc, 2002

URL: <http://www2.sas.com/proceedings/sugi27/p004-27.pdf>

SAS Institute, Inc., *BASE SAS 9.1.3 Procedures Guide*, Cary, NC: SAS Institute, Inc, 2004

ACKNOWLEDGMENTS

Thanks to Rick Langston of SAS Institute's Research and Development unit for his insights in to the SAS Format facility and for answering several questions I had while preparing this paper. I'd also like to thank Pete Lund of Looking Glass Analytics, Olympia, Washington, for his advice and insights during the development of this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact me at:

Andrew H. Karp
Sierra Information Services
19229 Sonoma Hwy PMB 264
Sonoma, CA 95476 USA
707 996 7380 voice
sierrainfo@aol.com
www.sierrainformation.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.