SAS® software allows us to interpret data values as they are read with an INPUT statement using "built-in" INFORMATS. For example, we could read a date value in the form MM/DD/YY using an MMDDYY8. INFORMAT. We could accomplish this with an INFORMAT statement like this:

```plaintext
DATA DATEEX;
  INFORMAT DATE MMDDYY8. ;
  INPUT @1 DATE ETC.;
DATALINES;
```

We could also choose to supply the INFORMAT directly in the INPUT statement:

```plaintext
DATA DATEEX;
  INPUT @1 DATE MMDDYY8. ETC.;
DATALINES;
```

You may not have been aware that you can create your own INFORMATS using PROC FORMAT. This paper will show you how to create your own INFORMATS and some useful (and perhaps clever) things you can do with them.

For our first example, we will show you the "old-fashioned" way to do what we call "data cleaning." In this example, we will be converting all values for SEX other than 'M' or 'F' to a character missing value (blank). Here is the code:

```plaintext
*==================================*
* Example 1                        *
* Data cleaning the old-fashioned  *
* way (on character data)          *
*==================================*

DATA EX 01;
  INPUT @1 ID 3. @5 SEX $SEXMF .;
  IF SEX NOT IN ('M', 'F') THEN SEX = '';
DATALINES;
```

If you are unfamiliar with the IN statement, it's equivalent to multiple OR's. Thus, the IF statement could also be written:

```plaintext
IF NOT (SEX = 'M' OR SEX = 'F') THEN SEX = '';
```

Using either logical statement, any values for SEX other than 'M' or 'F' will be converted to a missing value. What, you've never seen a DATALINES statement! It's the modern equivalent of a CARDS statement. Since many of the college-aged programmers have never seen (or heard of) a computer card, the SAS language decided to use a more up to date term, DATALINES as an alternative to CARDS. You are free to use either term in any post 6.07 version of SAS software.

Now, getting back to the subject at hand, let's accomplish the same data cleaning task using a user defined INFORMAT. Here is the code:

```plaintext
*==================================*
* Example 2                        *
* Using a user defined INFORMAT to  *
* do the data cleaning (char data) *
*==================================*

PROC FORMAT;
  INVALUE $SEXMF 'M', 'F' = SAME OTHER = '';
RUN;

DATA EX 02;
  INPUT @1 ID 3. @5 SEX $SEXMF1. ;
DATALINES;
```

OK, What have we done? First, we used PROC FORMAT with an INVALUE statement to create our INFORMAT. This is easy to remember since it's just like using a VALUE statement to create a FORMAT. Just as with a VALUE statement, we first give the INFORMAT a name, beginning with a $ for a character INFORMAT. However, a user defined INFORMAT can only be seven characters in length,
including the $ if used. Also, be sure not to use an INFORMAT name that is the same as a standard built-in SAS INFORMAT name. Next we can specify a range of values and the corresponding informatted value. In the example above, the first range was the individual character values 'M' and 'F'. The informatted value was the special value _SAME_ that will leave these two values unchanged. The range OTHER (in this case all characters other than 'M' or 'F') will be converted to a blank which is a missing value for a character variable. Notice that when we used our user-defined INFORMAT in the INPUT statement, we wrote it as SEXMF1. Just as we do with "built-in" INFORMATS, we can append a number to our user defined INFORMAT to specify how many characters to read from the input record. The default length for an INFORMAT is the longest INFORMATTED value (in this case it is equal to 1). We think it is an excellent idea to follow all the user defined INFORMATS with a number to ensure that you read the number of columns from the input record that you intend to. When defining an INFORMAT for a numeric variable, it is necessary to supply the trailing number unless you are using "list-directed" input.

Let's look at another data-cleaning application. For this example, we will perform range checking on some numeric variables. As before, we will present the "traditional" approach first. Here is the program:

```
P*==================================*
P* Example 3
P* Range checking the old-fashioned way
P*==================================*
DATA EX 03;
INPUT @1 ID $3.
@4 SBP 3.
@7 DBP 3.;
    IF SBP LT 40 OR SBP GT 300 THEN
        SBP = .;
    IF DBP LT 10 OR DBP GT 150 THEN
        DBP = .;
DATALINES;
    001160090
    002310220
    003020008
    004 080
    005150070
RUN;
```

As you can see, we are setting values of SBP (systolic blood pressure - the larger of the two numbers in a blood pressure reading - i.e. the 120 in 120/80) to missing if a data value is below 40 or greater than 300. For DBP (diastolic blood pressure - the lower of the two numbers) we are doing the same for values below 10 or greater than 150.

Let's see how a user-defined INFORMAT can accomplish the same goal:

```
P*==================================*
P* Example 4
P* Range checking using a user defined INFORMAT
P*==================================*
PROC FORMAT;
    INVALUE SBP 40 - 300 = SAME
    OTHER = .;
    INVALUE DBP 10 - 150 = SAME
    OTHER = .;
RUN;
DATA EX 04;
INPUT @1 ID $3.
    @4 SBP SBP3.
    @7 DBP DBP3.;
DATALINES;
    001160090
    002310220
    003020008
    004 080
    005150070
RUN;
```

As you can see, we have created two INFORMATS, one for SBP and one for DBP. Notice that you can give the INFORMAT the same name as a variable in your data set if you desire. We can specify ranges and/or individual values in the INVALUE statement. In this example we are leaving all values of SBP between 40 and 300 and all values of DBP between 10 and 150 unchanged. Values outside these ranges are to be set equal to a missing value. If you are using a version of SAS software prior to version 6.07, range specifications are only treated as character strings and the program in example 4 will not run correctly. If you are using a version of SAS software prior to version 6.07, you will need to write your FORMAT statement as shown below:

```
PROC FORMAT;
    INVALUE SBP '040' - '300' = SAME
    OTHER = .;
```
INVALUE DBP '010' - '150' = SAME;
OTHER = .;
RUN;

With SAS versions 6.07 and later, the way that INFORMAT ranges are treated is different from the pre-6.07 versions. The INFORMAT first checks if the inputted value is numeric. If so, it then checks for any unquoted ranges to see if the numeric value falls within any range. If the value read in is not a numeric, the INFORMAT will search for any quoted ranges.

We can extend example 4 to provide us with additional information concerning out-of-range values. To do this, we will use two alternate missing values, namely, .H and .L. The use of numeric missing values other than a single period (.) may be new to you. Besides the single period missing value, there are 27 additional missing value designations: _ (that's an underscore) and .A through .Z. In the example to follow, we will use a separate missing value to store high and low out-of-range values. We will then be able to produce counts of high and low out-of-range values and, at the same time, be able to compute statistics on the two variables without including these out-of-range values. Here is the program:

```sas
* Example 5
* More range checking using an IN-FORMAT and alternate missing values.
*
* ====
* PROC FORMAT;
INVALUE SBP LOW - <40 = .L
40 - 300 = SAME
301 - HIGH = .H;
INVALUE DBP LOW - <10 = .L
10 - 150 = SAME
151 - HIGH = .H;
VALUE CHECK .H = 'High'
.L = 'Low'
. = 'Missing'
OTHER = 'Valid';
RUN;

DATA EX 05;
  INPUT @1 ID @4 SBP @7 DBP;
DATALINES;
  001 160 90
  002 H H
  003 L L
  004 . 80
  005 150 70;
RUN;

PROC PRINT NOOBS;
  TITLE 'Listing from PROC PRINT';
RUN;

PROC FREQ;
  TITLE 'Listing from PROC FREQ';
  TABLES SBP DBP / MISSING NOCUM;
RUN;

PROC MEANS N MEAN MAXDEC=1;
  TITLE 'Listing from PROC MEANS';
RUN;
```

First, the output from this program is shown below. Look at it for a moment and we will then explain how things worked.

**Listing from PROC PRINT**

<table>
<thead>
<tr>
<th>ID</th>
<th>SBP</th>
<th>DBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>160</td>
<td>90</td>
</tr>
<tr>
<td>002</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>003</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>004</td>
<td>.</td>
<td>80</td>
</tr>
<tr>
<td>005</td>
<td>150</td>
<td>70</td>
</tr>
</tbody>
</table>

**Listing from PROC FREQ**

<table>
<thead>
<tr>
<th>SBP</th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>Low</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>Missing</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>Valid</td>
<td>2</td>
<td>40.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DBP</th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>Low</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td>Valid</td>
<td>3</td>
<td>60.0</td>
</tr>
</tbody>
</table>

**Listing from PROC MEANS**

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBP</td>
<td>2</td>
<td>155.0</td>
</tr>
<tr>
<td>DBP</td>
<td>3</td>
<td>80.0</td>
</tr>
</tbody>
</table>

We define separate missing values for high and low data values using the special missing values .L and .H. We chose .L and .H for convenience (it is easy to remember that .L is for Low and .H is for High). We could have alternatively chosen any of the 27 available missing values such as .A and .B to represent the low and high data values. Values in the valid ranges are untouched because we specified
SAME for our informatted value. We also used a VALUE statement to create a FORMAT so that we can label the appropriate missing values. We did not format the variables SBP and DBP when we created the PROC PRINT listing so that you can see the actual "internal" values stored in the SAS data set. Note that we used a MISSING option on the TABLES statement of PROC FREQ. This was necessary to show us frequencies for the various missing values. Without this option, the missing values would be noted at the bottom of the table and would not have been counted in the body of the table itself. PROC MEANS was run to demonstrate that no distinction will be made between the three different missing values when statistical calculations are made. We see in the output from PROC MEANS that there were 2 nonmissing values of SBP and 3 nonmissing values of DBP.

You may choose to use a user defined INFORMAT in a similar manner the next time you have raw data that needs to be checked for out-of-range values.

For our next trick, we will read a combination of character and numeric values for a single variable. In this example, we will be reading a list of temperatures. In place of 98.6 (normal body temperature) the researcher coded an 'N' (for Normal). First, here is a "traditional" approach to the problem:

```
DATA EX06;
INPUT DUMMY $ @@;
IF DUMMY = 'N' THEN TEMP = 98.6;
ELSE TEMP = INPUT (DUMMY, B.);
DROP DUMMY;
DATALINES;
98.9 101.2 N 99.7
RUN;
```

The "classic" approach used here is to read all data values as character. We have to do this because had we read the data values as numeric, the N's would have turned into missing values. Once we have read in our character variable (DUMMY), we then test if it is an 'N' or not. If so, we set our temperature variable (TEMP) to 98.6; if not, we use the INPUT function to perform a character to numeric conversion.

Starting with release 6.07, Enhanced Numeric INFORMATS became available. These allow us to read a combination of character and numeric data in one step. We can define both numeric and character ranges for the same INFORMAT. Here is a solution to the problem above using an enhanced numeric format:

```
DATA EX07;
INFORMAT TEMP TEMPER.;
INPUT TEMP @@;
DATALINES;
98.9 101.2 N 99.7
RUN;
```

Notice what an elegant solution this enhanced numeric INFORMAT gives us. We don't have to read in a DUMMY character variable, we omit the testing and the character to numeric conversion. This is clearly a good way to read a mixture of character and numeric data values.

We will now demonstrate a case conversion from lower to upper case. It is a common problem to have a mixture of upper and lower case character values in a raw data file. Using the UPPCASE string function provides us with the traditional approach:

```
DATA EX08;
INPUT @1 ID $3.
@4 (Q1-Q5) ($1.);  
ARRAY Q[5];
```

382
DO I = 1 TO 5;
Q[I] = UPCASE (Q[I]);
IF Q[I] NOT IN ('Y', 'N') THEN
Q[I] = ' ';
END;
DROP I;
DATALINES;
001YNYNN
002yyynn
RUN;

An UPCASE option on the INFORMAT statement provides an alternative:

*========================================================================*
* Example 9                                           *
* Reading a mixture of character and numeric data using INFORMATS *
*========================================================================*
PROC FORMAT;
INVALUE $CONV (UPCASE) 'Y' = 'Y'
'N' = 'N'
OTHER = ' ';
RUN;
DATALINES;
001YNYNN
002yyynn
RUN;

The UPCASE option which is placed in parentheses following the INVALUE name, instructs the program to convert the value to upper case before testing it against the range of values. Thus, if we read in a lower case 'y', it would be converted to upper case, match the value of 'Y' causing the informat ted value of 'Y' to be added to the data set.

What else can we do with INFORMATS? How about recoding data values? Here is an example using a traditional approach to recoding a variable. In this example, we will assign a value of 'HEART' for one list of diagnosis codes and a value of 'LUNGS' for another list of codes. For all codes not in either list, we will assign the category of 'OTHER'. Here is the traditional, data step approach:

*========================================================================*
* Example 10                                           *
* Recoding the old-fashioned way                        *
*========================================================================*
DATA EX_09;
LENGTH DX $5;
INPUT @1 ID $3.
@4 (Q1-Q5) ($CONV1.);
DATALINES;
00121
00205
00399
00424
RUN;

As you can see in this example, we can provide a list of specific codes or a range of codes and recode these values to a new category.

The program above needs little explanation other than pointing out the IN statement is much more convenient to write than a series of logical OR's when testing a value against a long list of alternatives, although the logical OR's are more efficient in versions of SAS from 6.08 and above. Now for the INFORMAT version:

*========================================================================*
* Example 11                                           *
* Recoding using INFORMATS                                  *
*========================================================================*
PROC FORMAT;
INVALUE $DXCODE '01', 'V5', '21', '23', '24' = 'HEART'
'02', '03', '05', '11', '13', '15' = 'LUNGS'
OTHER = 'OTHER';
RUN;
DATALINES;
00121
00205
00399
00424
RUN;

Here is one more recoding example where we want to group data values alphabetically. We want to test the first letter of a subject's name and assign a group depending on which alphabetical range the letter belongs. Again, we start with the traditional approach:

*========================================================================*
* Example 12                                           *
* Grouping subjects alphabetically                       *
* the old-fashioned way                                 *
*========================================================================*
DATA EX_10;
LENGTH DX $5;
INPUT @1 ID $3.
@4 DX $DXCODE2.;
DATALINES;
00121
00205
00399
00424
RUN;
DATA EX 12;
	LENGTH INIT $ 1;
	INPUT NAME : $12. SCORE;
	INIT = UPCASE(SUBSTR(NAME,1,1));
	IF 'A' LE INIT LE 'F' THEN
		GROUP = '1';
	ELSE IF 'G' LE INIT LE 'L' THEN
		GROUP = '2';
	ELSE IF 'M' LE INIT LE 'R' THEN
		GROUP = '3';
	ELSE IF 'S' LE INIT LE 'Z' THEN
		GROUP = '4';
	ELSE GROUP = ' '; 
DATALINES;
 CODY 100
 SMITH 40
 PASS 95
 pear 55
 GOULD 58
 8ZEBRA 70
 RUN;

When we run this program, CODY will be in GROUP 1, SMITH in GROUP 4, pear in GROUP 3 (his initial was converted to upper case before the comparison), GOULD in GROUP 2, and 8ZEBRA will be converted to a missing value.

Using a user-defined INFORMAT makes this process a bit simpler:

*========================================================================*
* Example 13 *
* Grouping subjects alphabetically with a user-defined INFORMAT *
*========================================================================*

PROC FORMAT;
	INVALUE $INIT (UPCASE)
	'A' - 'F' = '1'
	'G' - 'L' = '2'
	'M' - 'R' = '3'
	'S' - 'Z' = '4'
	OTHER = ' ';
RUN;

DATA EX 13;
	INPUT @1 GROUP $INIT1.
	@1 NAME : $12. SCORE;
DATALINES;
 CODY 100
 SMITH 40
 PASS 95
 pear 55
 GOULD 58
 8ZEBRA 70
 RUN;

We first read a single character in column one according to our user-defined INFORMAT ($INIT). This will automatically convert the value to upper case (because of the UPCASE option) and then determine in which alphabetic range the value lies. The pointer is then moved back to column one (@1) and the NAME is read using a standard character INFORMAT.

We hope that these examples of user-defined INFORMATS will whet your appetite and encourage you to try using them in your next program. As you can see from these examples, creating and using an INFORMAT can sometimes be easier and require less programming than the traditional data step approach. Finally, you may want to save some of your INFORMATS in a private library, just as you have done with FORMATS in the past. It's always fun to find a new way to solve an old problem and SAS Software always provides alternative approaches.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.

My sincere thanks to Warren Repole Jr. for giving me the idea of using alternate missing values in his NY Area SAS Users Group talk in October, 1993.