Biostatistics 600
SAS Lab
Supplement 1 Fall 2012

How to Enter Data
in the Program Editor Window: Instream Data
(commands=instream.sas)

If you are planning to enter a very small amount of data, it will often be convenient to type the data in the SAS program rather than reading it from another file. This is known as **instream** data. It is a quick and easy way to enter data into SAS for an analysis.

You will need 4 basic types of statements to enter data in this fashion:

- Data
- Input
- Cards or datalines
- A semicolon on a line by itself to end the data

Note: You must have at least one blank between each data value. More than one blank is OK. It is important to have something as a placeholder for each variable, even when the value is missing. A period will serve to indicate a missing value for both numeric and character variables entered in this way. The data do not need to be lined up exactly in columns. For example, if you wanted to enter data from a medical exam for 5 people, you could do it as shown below.

```
data medexam;
   input lname $ id sbp age;
   cards;
Smith     1028  135  .
Williams  1337  126  49
Brun      1829  148  56
Agassi    1553  118  65
Vernon    1626  129  60
  ;
proc print data=medexam;
run;
```

Entering Data for More than 1 case on the same line:

If you want to enter data on the same line for several cases, you can use the **@@ symbol**:

```
data test;
   input x y group $ @@;
   cards;
1  2 A   3 12 A   15 22 B   17 29 B   11 44 C   13 29 C
7 21 D  11 29 D   16 19 E   25 27 E   41 12 F   17 19 F
;
proc print data=test;
run;
```

This results in the following output, which shows that data have been entered for 12 cases:

OBS	X	Y	GROUP
1	1	2	A
2	3	12	A
3	15	22	B
4	17	29	B
5	11	44	C
6	13	29	C
7	7	21	D
8	11	29	D
9	16	19	E
10	25	27	E
11	41	12	F
12	17	19	F

Entering Data for a Table:

This is a very handy way to enter data from a table that you wish to analyze. Because weights are used in this analysis, it is not necessary to enter the values for each respondent individually. For example, if the following information were reported in a newspaper article in which the same respondents were asked to rate President Bush's job performance before and after September 11, and you wished to carry out a brief analysis, you could use the SAS commands below to create the table.

Is President Bush doing a good job in office?

	AFTER SEPT 11	
BEFORE SEPT 11	NO	YES
NO	5	80
YES	3	82

The following commands could be used to enter the data and carry out a simple analysis:

```
data opinion;
  input BEFORE $ AFTER $ count;
  cards;
No   No    5
No   Yes  80
Yes  No    3
Yes  Yes  82
;

proc freq;
  weight count;
  tables BEFORE * AFTER;
run;
```

The output from these commands is shown below:

Obs	BEFORE	AFTER	count
1	No	No	5
2	No	Yes	80
3	Yes	No	3
4	Yes	Yes	82

The FREQ Procedure
Table of BEFORE by AFTER

BEFORE / AFTER	No	Yes	Total
Frequency			
Percent			
Row Pct			
Col Pct			
No	5	80	85
	2.94	47.06	50.00
	5.88	94.12	
	62.50	49.38	
Yes	3	82	85
	1.76	48.24	50.00
	3.53	96.47	
	37.50	50.62	
Total	8	162	170
	4.71	95.29	100.0000

How to Create a SAS Data Set from Raw Data Files
(commands=readdata.sas)

Introduction:

This handout discusses how to set up a SAS command file to create a temporary SAS data set from a number of different raw data file types. Because the data set that is being created is temporary, it will be stored in the **WORK library**, and will be erased when the current SAS run is completed. The commands that generate the data set must be resubmitted to SAS each time SAS is started to recreate the data. However, all of the information on how to read the different types of raw data files is equally applicable to both temporary and permanent SAS data sets.

Raw data files (sometimes called ascii files, flat files, text files or unformatted files) can come from many different sources: from a database program, such as Access, from a spreadsheet program, such as Excel, or from a raw data file on a CD from a government or private agency. The first step is to be sure you know the characteristics of the raw data file. You can check the raw data by using a text editor or word processing program. For small files you can use Windows Notepad, for larger files you can use Microsoft Word or Word Perfect (be sure if you open your raw data file with a word processing program, that you save it as text only or unformatted text when you quit). To be able to read a raw data file, you will need a codebook that gives information about the data contained in the file.

The types of raw data files discussed in this handout are:

a) Blank separated values (list form)
b) Comma separated values (.csv files--these typically have been saved from Excel)
c) Tab separated values (.txt files--these can come from a number of different applications, including Excel)
d) Fixed-column data (often the form of data from government agencies, or research groups, such as ICPSR--the Inter University Consortium for Political and Social Research)

Once you have identified the type of raw data that is to be read, you can customize your command file to read the data into SAS. The command files that read in these types of data can be very simple or very long and complex, depending on the number and types of variables to be read.

The part of SAS that creates a new data set is the **data step**. The data step for reading raw data from a file has 3 essential statements:

- Data
- Infile
- Input

Other statements may be added to the data step to create new variables, carry out data transformations, or recode variables.

Reading blank separated values (list or free form data):

Raw data that are separated by blanks are often called list or free form data. When this type of data is used, each value is separated from the next by one or more blanks. If there are any missing values, they must be indicated by a placeholder, such as a period. Note that a period can be used to indicate a missing value for either character or numeric variables. Missing values can also be denoted by a missing value code, such as 99 or 999. The data do not need to be lined up in columns, so lines can be of unequal length, and can appear "ragged".

Here is an excerpt of a raw data file that is separated by blanks. Notice that the values in the file are not lined up in columns. The name of the raw data file is class.dat. Missing values are indicated by a period (.), with a blank between periods for contiguous missing values.

```
Warren F 29 68 139
Kalbfleisch F 35 64 120
Pierce M . . 112
Walker F 22 56 133
Rogers M 45 68 145
Baldwin M 47 72 128
Mims F 48 67 152
Lambini F 36 . 120
Gossert M . 73 139
```

The SAS data step to read this type of raw data is very simple. The data statement names the data set to be created, and the infile statement indicates the raw data file to be read. The input statement lists the variables to be read in the order in which they appear in the raw data file. No variables can be skipped at the beginning of the variable list, but you may stop reading variables before reaching the end of the list. Here are the SAS commands that were used to read in this data:

```
data class;
     infile "class.dat";
     input lname $ sex $ age height sbp;
run;
```

Note that character variable names are followed by a $. Without a $ after a variable name, SAS assumes that the variable is numeric (the default).

Length statement:

Sometimes it is necessary to include a **length** statement to allow character variables to be longer than the default length of 8 characters. Character variables can be from 1 to 32,767 characters long. We recommend limiting the lengths of character variables to 16 characters or less, if possible, because many procedures in SAS will display a maximum of 16 characters in their output. However, this rule need not apply to variables containing information such as names or addresses. Note that the length statement

comes *before* the input statement, so the length of the variable is set up before the variable is read. Because LNAME is the first variable mentioned, it will be the first variable in the data set.

```
data class;
      infile "class.dat";
       length lname $ 12;
      input lname $ sex $ age height sbp;
run;
```

Reading raw data separated by commas (.csv files):

Often raw data files will be in the form of **CSV (Comma Separated Values) files**. These files can be created by Excel (by going to **File>Save as>** and choosing the **file type: .csv**), and are very easy for SAS to read. An excerpt of a csv file called PULSE.CSV is shown below. Note that the first line of data contains the variable names.

```
pulse1,pulse2,ran,smokes,sex,height,weight,activity
64,88,1,2,1,66,140,2
58,70,1,2,1,72,145,2
62,76,1,1,1,73,160,3
66,78,1,1,1,73,190,1
```

SAS commands to read in this raw data file are shown below.

```
data pulse;
   infile "pulse.csv" firstobs=2 delimiter = ","  dsd;
   input pulse1 pulse2 ran smokes sex height weight activity;
run;
```

There are several modifications to the infile statement in the previous example:

a) **`delimiter = "," or dlm=","`** tells SAS that commas are used to separate the values in the raw data file, not the default, which is a blank.
b) **`firstobs = 2`** tells SAS to begin reading the file at line 2, which is where the data values begin.
c) **`dsd`** allows SAS to read consecutive commas as an indication of missing values.

Note: this data set may also be imported directly into SAS by using the SAS Import Wizard, and selecting the file type as commas separated values.

Reading in raw data separated by tabs (.txt files):

Raw data separated by tabs may be created by Excel (saving a file with the text option) or by other applications. You can determine if your data are separated by tabs by viewing the file in a word processing program, such a Microsoft Word, and having the program display all formatting characters. The example below shows how tab-separated data appear when viewed without the tabs visible. This raw data file is called clinic.txt:

```
id    group date  sbp   wt    sideffct
131   1     4/2/95      129   150   1
131   1     5/5/95      118   154   1
131   1     6/1/95      119   152   0
131   1     7/10/95     116   151   1
131   1     8/14/95     111   153   0
131   1     10/12/95    109   148   1
105   2     7/15/95     145   188   0
105   2     8/22/95     147   185   1
105   2     11/28/95    133   184   0
105   2     12/20/95    129   185   0
222   1     3/14/95     159   201   0
```

The **infile** statement must be modified to tell SAS that the delimiters are tabs. Since there is no character equivalent of tab, the hexadecimal equivalent of tab is indicated in the **dlm =** option, as shown below:

```
data clinic;
  infile "clinic.txt" firstobs=2 dsd dlm="09"X;
  input id group date $ sbp wt sideffct;
run;
```

Note that DATE has been read as a character variable, which does not allow us to do date math using this variable. The example below shows how to read in DATE using an **informat**, and display it as a date, using a **format** statement.

```
data clinic;
  infile "clinic.txt"  dsd missover firstobs=2 dlm="09"X ;
  input id group date :mmddyy8. sbp wt sideffct;
  format date mmddyy10.;
run;
proc print data=clinic;
run;
```

Partial output from these commands is shown below:

```
Obs   id    group        date      sbp     wt    sideffct
 1    131      1      04/02/1995    129    150        1
 2    131      1      05/05/1995    118    154        1
 3    131      1      06/01/1995    119    152        0
```

Reading raw data that are aligned in columns:

Raw data may be aligned in columns, with each variable always in the same location. There may or may not be blanks between the values for given variables. An example is shown below. This is an excerpt from the raw data file: marflt.dat:

```
182030190 8:21LGAYYZ  366 458 390104 16  3123178
114030190 7:10LGALAX2,475 357 390172 18  6196210
20203019010:43LGAORD  740 369 244151 11  5157210
```

```
219030190 9:31LGALON3,442 412 334198 17  7222250
43903019012:16LGALAX2,475 422 267167 13  5185210
```

Because there are not blanks separating values in this raw data file, the data must read into SAS in a manner that identifies the column location of each variable.

Column-style input:

To read data that are lined up in columns, the input statement is set up by listing each variable followed by the column-range in which it can be found. Character variables should be followed by a $, and then the column-range. It is possible when using this type of input to go to any column in the raw data file, and to skip columns. A code sheet will tell you which columns should to be read for each variable.. Many large data sets that are distributed by the government are documented in this manner.

Here is an example of a command file to read in raw data from marflt.dat. Notice that not all values are read in this example. Proc print is also used to print out the first 10 cases of the marflt data set.

```
data marflt;
     infile "marflt.dat" ;
     input flight 1-3 depart $ 15-17 dest $ 18-20 boarded 34-36;
run;
proc print data=marflt(obs=10);
run;
```

The output from these commands is shown below:

Obs	flight	depart	dest	boarded
1	182	LGA	YYZ	104
2	114	LGA	LAX	172
3	202	LGA	ORD	151
4	219	LGA	LON	198
5	439	LGA	LAX	167
6	387	LGA	CPH	152
7	290	LGA	WAS	96
8	523	LGA	ORD	177
9	982	LGA	DFW	49
10	622	LGA	FRA	207

Reading column data that is on more than one line:

Sometimes the raw data for a single case are included on more than one line. An example of this is shown in the excerpt from the file afifi.dat shown below.

```
 340  70 160  23   4  62  38  53  29 100 187  90 190 390   0 394 241 131 400   1
 340  70 160  23   4 129  74  72  53 190 187 120 130 300  15 394 241 112 365   2
 412  56 173  11   4  83  66 110  60  10 182 126 221 407 110 362 240 166 500   1
 412  56 173  11   4 102  75 108  63  90 182 281 100 206  50 564 266 154 330   2
```

This data represents information on patients measured at 2 time points. First, measurements were made for each patient when they came in to the emergency room, and then these same measurements were made either just before discharge, or if the patient died, just before death. The first part of the information for a given patient is the same on both lines of raw data, the remainder of the data is different.

Here are SAS commands to read in this raw data file and to create a SAS data set called AFIFI. In this command file, a new line is indicated by a # sign, followed by the line number. In addition, there is a number after the column-range for the variables HGB1 and HGB2. This number tells SAS how many decimal places should be inserted in the values of these 2 variables. (There are no decimals in the original raw data file.) Thus, the value of HGB1 for the first patient is 13.1, rather than 131 as it appears in the raw data, and the value for HGB2 is 11.2. If there is an actual decimal point in the raw data, its placement will take precedence over what is specified in the input statement.

```
data afifi;
  infile "afifi.dat";
  input
  #1 idnum 1-4  age 5-8 sex 13-15 surv 16 shoktype 17-20 sbp1 21-24
     hgb1 69-72 1
  #2 sbp2 21-24 hgb2 69-72 1;
run;
```

An alternative way to read in raw data from two lines is shown below. Here the slash means to skip to the next line. You can use as many slashes as necessary to tell SAS how many lines to skip, and which lines to read.

```
data afifi;
  infile "afifi.dat";
  input
  idnum 1-4  age 5-8 sex 13-15 surv 16 shoktype 17-20 sbp1 21-24
     hgb1 69-72 1
     /sbp2 21-24 hgb2 69-72 1;
run;
```

Formatted-style input:

Raw data that are aligned in columns can also be read with formatted style input. The input statement must first indicate the column in which to begin reading with an @ sign, e.g. @46 to start reading at column 46 (by default, the first variable will be read, starting @1). Then the variable name is followed by the **format of the variable in the form w.d (where w indicates the total width of the variable, including any signs and decimal points, and d indicates the number of places after the decimal).** Note that explicit decimals in the data will override a decimal specification given in the input statement. The @ can be used to move around to different places in the data. The @ sign may point to any column that you wish and you may go back to previous columns if desired, or portions of the data may be skipped.

```
data afifi;
  infile "afifi.dat";
  input
  #1 @1  idnum 4.0  @5  age 4.0 @13 sex 3. @16 surv 1.
     @17  shoktype 4. @21 sbp1 4. @69  hgb1 4.1
  #2 @21 sbp2 4. @69  hgb2 4.1;
run;
```

Note that the format **4.0 is equivalent to the format 4. It is critical that the format be given with a period after it (e.g. 4. rather than 4)**, because that allows SAS to distinguish between a format and a column location.

Mixed style input is also allowed. The example below shows how to read the marflt.dat raw data into SAS using column-style input for some variables, and formatted-style input for others. The commands below show how to read the variable DATE using the **mmddyy6. informat**, so you can do math with this variable later. The **format** statement after the input statement tells SAS to display the date using the **mmddyy10. format**, which will insert slashes between the month, day and year values, and display a four-digit year. The informat must match the way the raw data are set up, but the format statement can use any valid SAS date format to display the date. The date itself will be stored internally in SAS as the number of days from Jan. 1, 1960 to the date of the flight. Again, note the use of the period at the end of the **informat mmddyy6.** and the **mmddyy10. format.**

The variable MILES is read with a **comma5. informat**, because the value of miles contains a comma in the raw data. We display MILES with a comma5. format, by using the format statement.

```
data marflt2;
  infile "marflt.dat";
  input flight 1-3
            @4 date mmddyy6.
            @10 time time5.
            orig $ 15-17
            dest $ 18-20
            @21 miles comma5.
            mail 26-29
            freight 30-33
            boarded 34-36
            transfer 37-39
            nonrev 40-42
            deplane 43-45
            capacity 46-48;
format date mmddyy10. time time5. miles comma5.; run;
```

The results of the above commands are shown below:

Obs	flight	date	time	orig	dest	miles	mail	freight	boarded	transfer	nonrev	deplane	capacity
1	182	03/01/1990	8:21	LGA	YYZ	366	458	390	104	16	3	123	178
2	114	03/01/1990	7:10	LGA	LAX	2,475	357	390	172	18	6	196	210
3	202	03/01/1990	10:43	LGA	ORD	740	369	244	151	11	5	157	210
4	219	03/01/1990	9:31	LGA	LON	3,442	412	334	198	17	7	222	250
5	439	03/01/1990	12:16	LGA	LAX	2,475	422	267	167	13	5	185	210
6	387	03/01/1990	11:40	LGA	CPH	3,856	423	398	152	8	3	163	250
7	290	03/01/1990	6:56	LGA	WAS	229	327	253	96	16	7	117	180

Infile Options for Special Situations:

Sometimes your data will require special options for it to be read correctly into SAS. The infile statement allows a number of options to be specified. These infile options may appear in any order in the infile statement, after the raw data file is specified.

1. The missover option:

The missover option is used to prevent SAS from going to the next line to complete a case if it did not find enough values on a given line of raw data. The missover option will often correct problems in reading raw data that are separated by blanks, when the number of cases reported by SAS to be in your data set is less than expected.

In the example below, the raw data file "huge.dat" has 400 lines in it, but SAS creates a dataset with only 200 observations, as shown in the SAS NOTE from the SAS Log below.

```
data huge;
      infile "huge.dat";
      input v1-v100;
run;
```

The above commands result in the following note in the SAS log:

```
NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: SAS went to a new line when INPUT statement reached past the end of a
      line.
NOTE: The data set WORK.HUGE has 200 observations and 100 variables.
```

The addition of the missover option on the infile line corrects this problem.

```
data huge;
      infile "huge.dat" missover;
      input v1-v100;
run;
```

```
NOTE: The infile "huge.dat" is:
      FILENAME=C:\kwelch\workshop\data\huge.dat,
      RECFM=V,LRECL=256
NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: The data set WORK.HUGE has 400 observations and 100 variables.
NOTE: The DATA statement used 0.59 seconds.
```

However, there is still a problem in the data, as can be seen in the output from proc means (there are zero cases for the variables v86 to v100.

Variable	N	Mean	Std Dev	Minimum	Maximum
V70	400	0.4775000	0.5001190	0	1.0000000
V71	400	0.4825000	0.5003194	0	1.0000000
V72	400	0.5125000	0.5004697	0	1.0000000
V73	400	0.5050000	0.5006011	0	1.0000000
V74	400	0.5025000	0.5006199	0	1.0000000
V75	400	0.5150000	0.5004008	0	1.0000000
V76	400	0.4850000	0.5004008	0	1.0000000
V77	400	0.4600000	0.4990216	0	1.0000000
V78	400	0.4925000	0.5005699	0	1.0000000
V79	400	0.5175000	0.5003194	0	1.0000000
V80	400	0.5450000	0.4985945	0	1.0000000
V81	400	0.5000000	0.5006262	0	1.0000000
V82	400	0.5275000	0.4998684	0	1.0000000
V83	400	0.4925000	0.5005699	0	1.0000000
V84	400	0.4800000	0.5002255	0	1.0000000
V85	400	0.5050000	0.5006011	0	1.0000000
V86	0	.	.	.	.
V87	0	.	.	.	.
V88	0	.	.	.	.
V89	0	.	.	.	.
V90	0	.	.	.	.
V91	0	.	.	.	.
V92	0	.	.	.	.
V93	0	.	.	.	.
V94	0	.	.	.	.
V95	0	.	.	.	.
V96	0	.	.	.	.
V97	0	.	.	.	.
V98	0	.	.	.	.
V99	0	.	.	.	.
V100	0	.	.	.	.

2. Using LRECL for very long lines of raw data:

If your raw data file has very long lines, you will need to use the **lrecl** option on the infile statement. The lrecl (logical record length) option tells SAS the longest length (the longest number of characters) that any line in the raw data could possibly have. The default length used by SAS for Windows is 256, so if your data file has more than 256 *characters* (count characters by counting each letter, number, space, period or blank in your data line) you will need to give an lrecl statement. (Note: the default lrecl differs for different operating systems). You cannnot go wrong by giving an lrecl value that is too large. If you don"t know the exact length, guess, and guess at a large value. Here is an example of reading in a raw data file that has a logical record length that is set at 2000.

```
data huge;
      infile "huge.dat" missover lrecl=2000;
      input v1-v100;
run;
```

Portion of the SAS Log:

```
NOTE: The infile "huge.dat" is:
      FILENAME=C:\kwelch\workshop\data\huge.dat,
      RECFM=V,LRECL=2000

NOTE: 400 records were read from the infile "huge.dat".
      The minimum record length was 300.
      The maximum record length was 300.
NOTE: The data set WORK.HUGE has 400 observations and 100 variables.
NOTE: The DATA statement used 0.48 seconds.
```

Now, the data set now has the required 400 observations, and that all variables have values, as shown in the output from proc means below:

Variable	N	Mean	Std Dev	Minimum	Maximum
V81	400	0.5000000	0.5006262	0	1.0000000
V82	400	0.5275000	0.4998684	0	1.0000000
V83	400	0.4925000	0.5005699	0	1.0000000
V84	400	0.4800000	0.5002255	0	1.0000000
V85	400	0.5050000	0.5006011	0	1.0000000
V86	400	0.5000000	0.5006262	0	1.0000000
V87	400	0.5000000	0.5006262	0	1.0000000
V88	400	0.5350000	0.4993981	0	1.0000000
V89	400	0.4875000	0.5004697	0	1.0000000
V90	400	0.5250000	0.5000000	0	1.0000000
V91	400	0.4850000	0.5004008	0	1.0000000
V92	400	0.4700000	0.4997242	0	1.0000000
V93	400	0.4875000	0.5004697	0	1.0000000
V94	400	0.5025000	0.5006199	0	1.0000000
V95	400	0.5050000	0.5006011	0	1.0000000
V96	400	0.4425000	0.4973048	0	1.0000000
V97	400	0.4975000	0.5006199	0	1.0000000
V98	400	0.5175000	0.5003194	0	1.0000000
V99	400	0.4875000	0.5004697	0	1.0000000
V100	400	0.5025000	0.5006199	0	1.0000000

Checking your data after it has been read into SAS:

It is critically important to check the values in your SAS data set before proceeding with your analysis! Just because the data were read into SAS does not guarantee that they were read correctly. **Data checking should be the first step before moving on to any statistical analyses.**

1. **Check the log:** After reading raw data into SAS, check the log to verify that the number of cases that were read matches what it should be, and that the data set has the number of cases that you expect. If you have fewer cases than you expect, check your infile statement, you might want to add a missover option. Check the input statement also, to be sure that it is correct. The log will also alert you to any problems that SAS encountered in reading the data. SAS will print warnings (a limited number of them) indicating if there are problems in the data that you have read in. Save the log if

you are having trouble reading your data. It is the best way to figure out how to remedy any problems!

2. **Run descriptive statistics using proc means to check the data:** Simple descriptive statistics are very easy to produce using proc means. The output from this procedure will give you several very important pieces of information. First, the minimum and maximum can be checked to see if they conform to the values that make sense for the variables that you are reading. Second, check the n (i.e., sample size) for each variable. The n will tell you if there are many missing values for a particular variable and may alert you to possible problems with the data that should be addressed.

3. **Check the distributions of continuous variables with a histogram or box and whiskers plot:** This can be done using SAS Proc Univariate, or Proc Boxplot or Proc Sgplot. The histogram and box and whiskers plot will give you an idea if there are outliers that should be checked, if the distribution of a variable is symmetric, and the general shape of the distribution.(We'll talk about graphs in SAS later).

4. **Check the values of categorical variables with proc freq:** This is a useful way to check categorical variables that can have a limited number of values. Knowing the values that occur can help to determine if there were any errors in reading the data, and knowing the number of cases in each category can help to understand the data.

Using Dates in SAS
(commands=date.sas)

Introduction:

A date value is stored in SAS as the number of days from January 1, 1960 to the given date. If the date is before January 1, 1960, it will have a negative value, if it is after this date, it will have a positive value. SAS dates can be subtracted, to get the number of days between two dates, or manipulated in any way that normal numeric values can be. Dates can be displayed using a SAS date format, or simply as a numeric value (with no format). There are many SAS formats for dates, a few of which are listed in the table below. Note that all SAS date formats end in a period, to distinguish them from SAS variable names.

Selected SAS Date Formats

SAS Date Format	Example
date7.	12SEP06
date9.	12SEP2006
datetime10.	12SEP06:03
datetime13.	12SEP06:03:19
datetime16.	12SEP06:03:19:42
ddmmyy10.	23/09/2006
mmddyy10.	09/232006
monyy7.	JUN2006
yymmdd8.	06-06-15

Example of Reading in Raw Data Using a Date Format:

Here is an example of reading in a date value from a raw data file using a SAS date format. Note that the width of the date variable may not always be the same in the raw data file, due to different number of integers in the month and day that are coded. This is not a problem for SAS, when the colon format modifier is used in front of the mmddyy8. **date format**, as in the commands shown below. A portion of the raw data is shown here:

Data Excerpt from SURVEY.DAT

```
1 10/4/93 1 1 1 1 2 2 . 1 1.5 1 . . .
2 10/13/93 2 1 3 2 3 3 2 2 3 3 3 3 3
3 10/13/93 1 1 1 1 1 1 3 2 1 1 1 1 1
4 10/21/93 1 1 1 1 1 2 . 2 1 1 1 . 1
5 10/21/93 1 2 1 1 2 3 3 2 2 2 1 4 3
6 11/19/93 1 4 1 1 4 4 3 1 4 . . . .
```

```
7 11/29/93 1 2 2 1 1 1 1 1 1 2 2 1 1
```

The SAS commands to read in this raw data are shown below:

```
data survey;
  infile "survey.dat";

  input Pt_num  DateRec :mmddyy8. Phone FstAppt ConvApp Staff Confer
  Txhelp AddSvc Tx_Loc FeelTx Wait ConTime RxExpl Confcare;

  lastdate="01FEB1997"D;

  today = date( );

  days  = lastdate - daterec;
  years = (lastdate - daterec)/365.25;

  format daterec today mmddyy10. lastdate date9.;
run;

title "Printout Showing Dates with Date Formats";
proc print data=survey(obs=10);
  var pt_num daterec lastdate today days years;
run;

title "Contents Showing Formats for Date Variables";
proc contents data=survey;
run;
```

Notice that the variable DATEREC is read in using the mmddyy8. **informat**, but it is displayed using the **mmdyy10.** format.

The new variable LASTDATE is entered using a **date constant**. The date constant is listed in quotes, and gives the value of the date as a two-digit day, followed the first three letters of the month, followed by a two- or four-digit year, followed by a D to tell SAS that this is a date constant, and should be treated as a date (which is numeric) and not a character value.

The variable TODAY is created using the DATE ( ) function, which automatically returns today's date, as set in your computer. The new variables DAYS, and YEARS are calculated using mathematical functions to calculate the time between two dates.

The format statement tells SAS to *display* the two date variables, DATEREC and TODAY, using the SAS date format **mmddyy10.,** while the variable LASTDATE will be displayed using the DATE9. format. Any other valid date format could have been chosen to display the values of these variables, or they could have been left as the number of days from the reference date of January 1, 1960. You do not need to display dates using the same format in which they were originally read into SAS.

The output from these commands is shown below:

Printout Showing Dates with Date Formats

Obs	Pt_num	DateRec	lastdate	today	days	years
1	1	10/04/1993	01FEB1997	08/16/2006	1216	3.32923
2	2	10/13/1993	01FEB1997	08/16/2006	1207	3.30459
3	3	10/13/1993	01FEB1997	08/16/2006	1207	3.30459
4	4	10/21/1993	01FEB1997	08/16/2006	1199	3.28268
5	5	10/21/1993	01FEB1997	08/16/2006	1199	3.28268
6	6	11/19/1993	01FEB1997	08/16/2006	1170	3.20329
7	7	11/29/1993	01FEB1997	08/16/2006	1160	3.17591
8	8	12/02/1993	01FEB1997	08/16/2006	1157	3.16769
9	9	12/09/1993	01FEB1997	08/16/2006	1150	3.14853
10	10	12/13/1993	01FEB1997	08/16/2006	1146	3.13758

Contents Showing Formats for Date Variables

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
9	AddSvc	Num	8	
13	ConTime	Num	8	
15	Confcare	Num	8	
7	Confer	Num	8	
5	ConvApp	Num	8	
2	DateRec	Num	8	MMDDYY10.
11	FeelTx	Num	8	
4	FstAppt	Num	8	
3	Phone	Num	8	
1	Pt_num	Num	8	
14	RxExpl	Num	8	
6	Staff	Num	8	
10	Tx_Loc	Num	8	
8	Txhelp	Num	8	
12	Wait	Num	8	
18	days	Num	8	
16	lastdate	Num	8	DATE9.
17	today	Num	8	MMDDYY10.
19	years	Num	8	

Example of Using the MDY Function to Read a Date:

Date values are sometimes entered as separate variables representing month, day and year. The following example illustrates how these values can be used with the **mdy** function in SAS to create date variables.

```
data dates;

  length name $12;
  input name $ bmon bday byr intmon intday intyr;

  if bday = . then bday = 15;
  if intday = . then intday = 15;

  birdate = mdy(bmon,bday,byr);
  intdate = mdy(intmon,intday,intyr);
```

```
  intage = int((intdate-birdate)/365);

  format birdate intdate date9.;

  cards;
  Roger    12 12 84  9 3  94
  Samantha 1  20 85  9 15 94
  Henry    10 6  83  10 2 94
  William  4  17 82  10 5 94
  Petra    6  .  83  9 14 94
  ;

proc print data=dates;
  title 'Printing Dates Using SAS Date Formats';
run;
```

The output from this program is shown below:

```
                        Printing Dates Using SAS Date Formats
                                                                  B           I
                                    I    I                        I           N    I
                                    N    N    I                   R           T    N
       N        B    B             T    T    N                   D           D    T
O      A        M    D    B        M    D    T                   A           A    A
B      M        O    A    Y        O    A    Y                   T           T    G
S      E        N    Y    R        N    Y    R                   E           E    E

1   Roger      12   12   84    9    3   94   12DEC1984   03SEP1994    9
2   Samantha    1   20   85    9   15   94   20JAN1985   15SEP1994    9
3   Henry      10    6   83   10    2   94   06OCT1983   02OCT1994   10
4   William     4   17   82   10    5   94   17APR1982   05OCT1994   12
5   Petra       6   15   83    9   14   94   15JUN1983   14SEP1994   11
```

You can temporarily remove date formats from SAS variables by using a format statement with Proc Print. This does not change the formats that are saved in the SAS dataset, but simply changes the way the variables are displayed for this Proc.

```
proc print data=dates;
  format birdate intdate;
  title "Printing Dates as Ordinary Numeric Values";
run;
```

The output from this program is shown below:

```
              Printing Dates as Ordinary Numeric Values

 OBS NAME     BMON BDAY BYR INTMON INTDAY INTYR BIRDATE INTDATE INTAGE

  1  Roger     12   12   84    9      3     94    9112   12664     9
  2  Samantha   1   20   85    9     15     94    9151   12676     9
  3  Henry     10    6   83   10      2     94    8679   12693    10
  4  William    4   17   82   10      5     94    8142   12696    12
  5  Petra      6   15   83    9     14     94    8566   12675    11
```

Using the Yearcutoff option to set a 100-year window for two digit years:

You can use the **yearcutoff** option to set a 100-year window to determine how SAS will interpret dates that are only 2 digits long. The default yearcutoff for SAS 9 is 1920, so a 2 - digit year that is 00 will be read as 2000. However, if you wish to change that, you can change the yearcutoff option to be a different year, say 1900. Then, the year 00 will be read as 1900.

```
options yearcutoff = 1900;

data testdate;
  input chkdate :MMDDYY8.;
  format chkdate mmddyy10.;
  cards;
  01/01/50
  01/01/49
  01/01/01
  01/01/98
  01/01/00
  ;
proc print data=testdate;
  title "Printout of Dates with yearcutoff at 1900";
run;
```

The output from these commands is shown below:

Printout of Dates with yearcutoff at 1900

Obs	chkdate
1	01/01/1950
2	01/01/1949
3	01/01/1901
4	01/01/1998
5	01/01/1900