SAS Programming (BIOL-4V190)

Chapter 3

Reading Raw Data from External Files

The first thing that you usually need to do in any SAS program is to access your data and read it into the program.

A SAS program can be used to read data that is stored in many different file formats including text or flat files, SAS data sets, Excel spreadsheets, Oracle Tables, etc.

In this course we will cover accessing data from text files and from SAS data sets.

Text files are files that can be opened up and read with a simple text editor such as Notepad.

Data stored in this type of file can be read into a SAS program using:

list input method

column input method

formatted input method

Data can also be included directly in your SAS program rather than stored in an external file and read into the program using the same methods listed above.

3.7 Placing Data Lines Directly in Your Program (the DATALINES Statement)

The simplest of the three methods is list input method.

To use this method, your data must have at least one blank space between the data columns.

The blank space serves as a delimiter between data values. So when SAS encounters a blank, it “understands” that the next character encountered is the beginning of the next variable.

The syntax for using list input with data included in the data step is:

```
data name;
  input variablename1 variablename2 $ variablename3...variablename'n';
datalines;
a line of data
another line of data
even more data
;
run;
```

```
*Program 3-5 Demonstrating the DATALINES statement - page 36;
data demographic;
  input Gender $ Age Height Weight;
datalines;
M 50 68 155
F 23 60 101
M 65  72 220
F 35 65 133
M 15 71 166
;
run;
```

This program reads data into a data set called demographic. The input statement lists the variable names in the order in which these are located in the data. So Gender corresponds to the first data field, Age to the second data field, etc. The $ after the variable name Gender tells SAS that Gender will be a character variable.

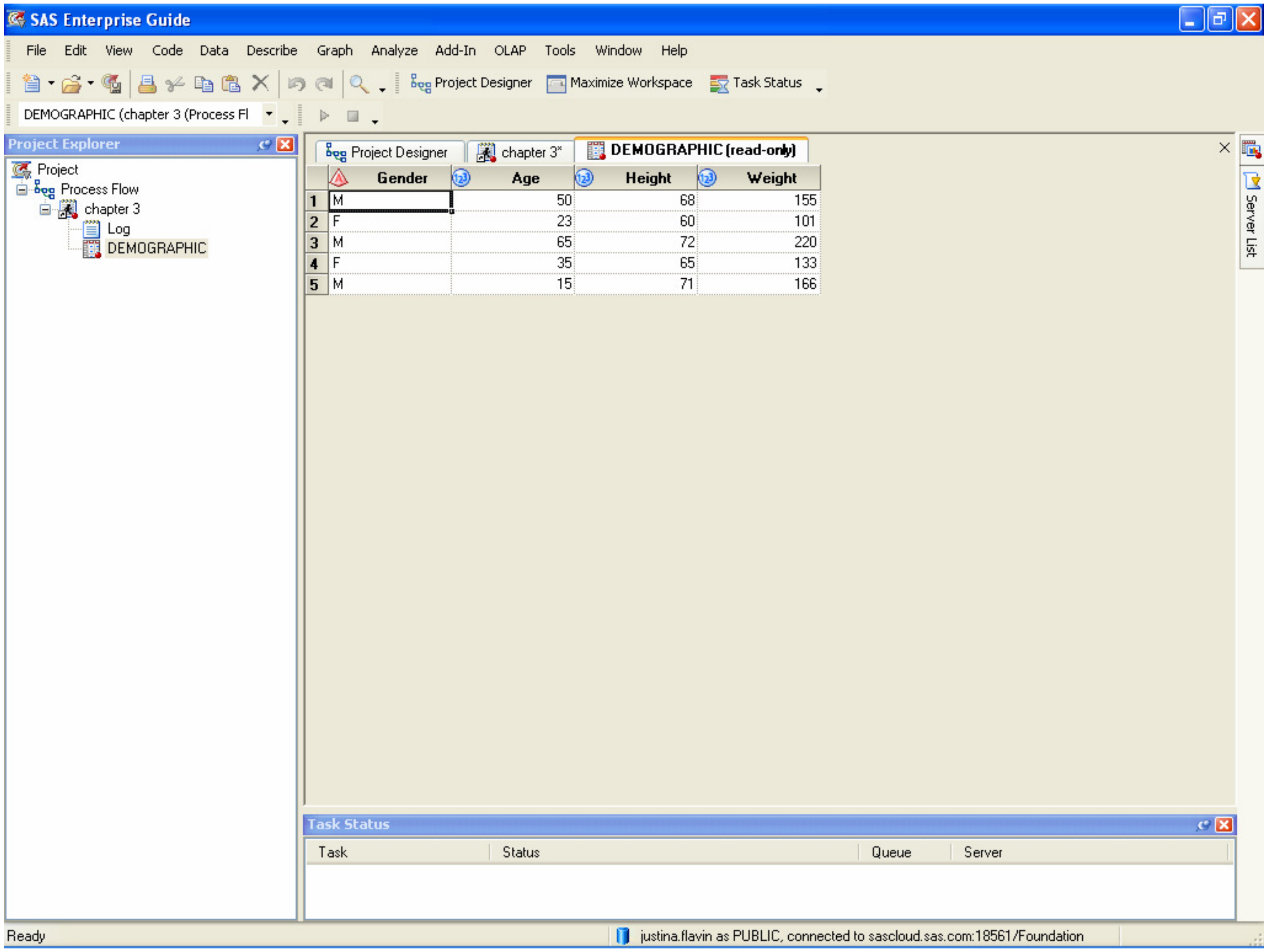
To include the data in the data step, use the keyword datalines (or cards) before your lines of data. The datalines statement and the lines of data must always be the last lines in your data step code.

Notice that on the third line of data there is an extra blank space between the second and third values. This extra blank space does not cause any problems in reading in the data.

Select and run this code and we will take a look at the results.

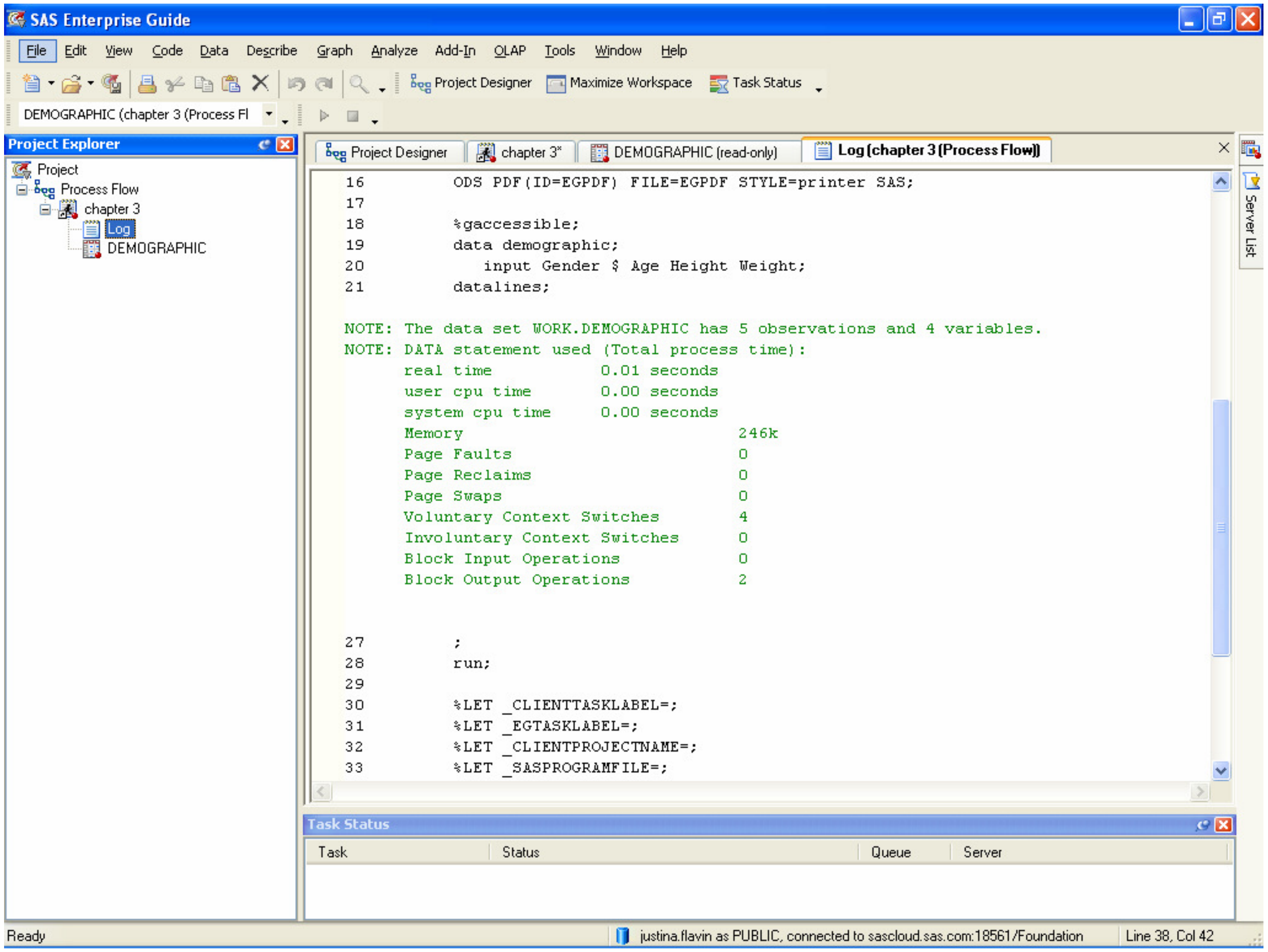
In general, when you execute code, SAS will automatically make a non-code window the active window so you can view the results of the code that you have submitted. In this case, we submitted code to create a data set, so the active window is now a window which displays the contents of the data set.

The red icon to the left of Gender indicates a character variable while the blue icons to the left of the other variables are indicative of numeric variables.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
DEMOGRAPHIC (chapter 3 (Process Fl
Project Explorer
Project
Process Flow
chapter 3
Log
DEMOGRAPHIC
Project Designer
chapter 3*
DEMOGRAPHIC (read-only)
Server List
Task Status
Task Status Queue Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Gender	Age	Height	Weight
1	M	50	68	155
2	F	23	60	101
3	M	65	72	220
4	F	35	65	133
5	M	15	71	166

Now let's look at the contents of the log window.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
DEMOGRAPHIC (chapter 3 (Process Fl
Project Explorer
Project
Process Flow
chapter 3
Log
DEMOGRAPHIC
Project Designer
chapter 3*
DEMOGRAPHIC (read-only)
Log (chapter 3 (Process Flow))
Server List
Task Status
Task
Status
Queue
Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation
Line 38, Col 42

```
          ODS PDF(ID=EGPDF) FILE=EGPDF STYLE=printer SAS;

          %gaccessible;
          data demographic;
             input Gender $ Age Height Weight;
          datalines;

NOTE: The data set WORK.DEMOGRAPHIC has 5 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      Memory                            246k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        4
      Involuntary Context Switches      0
      Block Input Operations            0
      Block Output Operations           2


          ;
          run;

          %LET _CLIENTTASKLABEL=;
          %LET _EGTASKLABEL=;
          %LET _CLIENTPROJECTNAME=;
          %LET _SASPROGRAMFILE=;
```

Scrolling through the log window, you can view the code that was submitted followed by some processing information.

It is always a good idea to review the log after submitting code to check for errors or unexpected results.

Like the code editor, SAS color codes the lines generated in the log. NOTES are informational messages and are in GREEN while ERROR messages are in RED.

When SAS encounters an error in your program, the default action is to stop processing the code. The text of the message should give you some idea as to what is causing the error.

While notes are informational in nature, these should also be reviewed for unexpected results.

A third type of message that is not illustrated here is a WARNING message. Warning messages may or may not be indicative of a problem but can and should be eliminated from your program by rewriting some of the code.

Examining our program log, we can see that a data set called DEMOGRAPHIC was created in the WORK library. Note: libraries will be discussed in Chapter 4.

The data set has 5 observations (or rows) and 4 variables (or columns).

Reviewing the data in your program, is that what you expected?

Sometimes, code can be written in such a way that it does not generate an error message because it is syntactically correct – but it is not generating the results that you expected.

For this reason, it is important to get into the habit of reviewing the results and to use the information in the log to check your work.

You now have created your first data set in SAS.

To celebrate this momentous occasion, you may want to print out the data so that you can frame it.

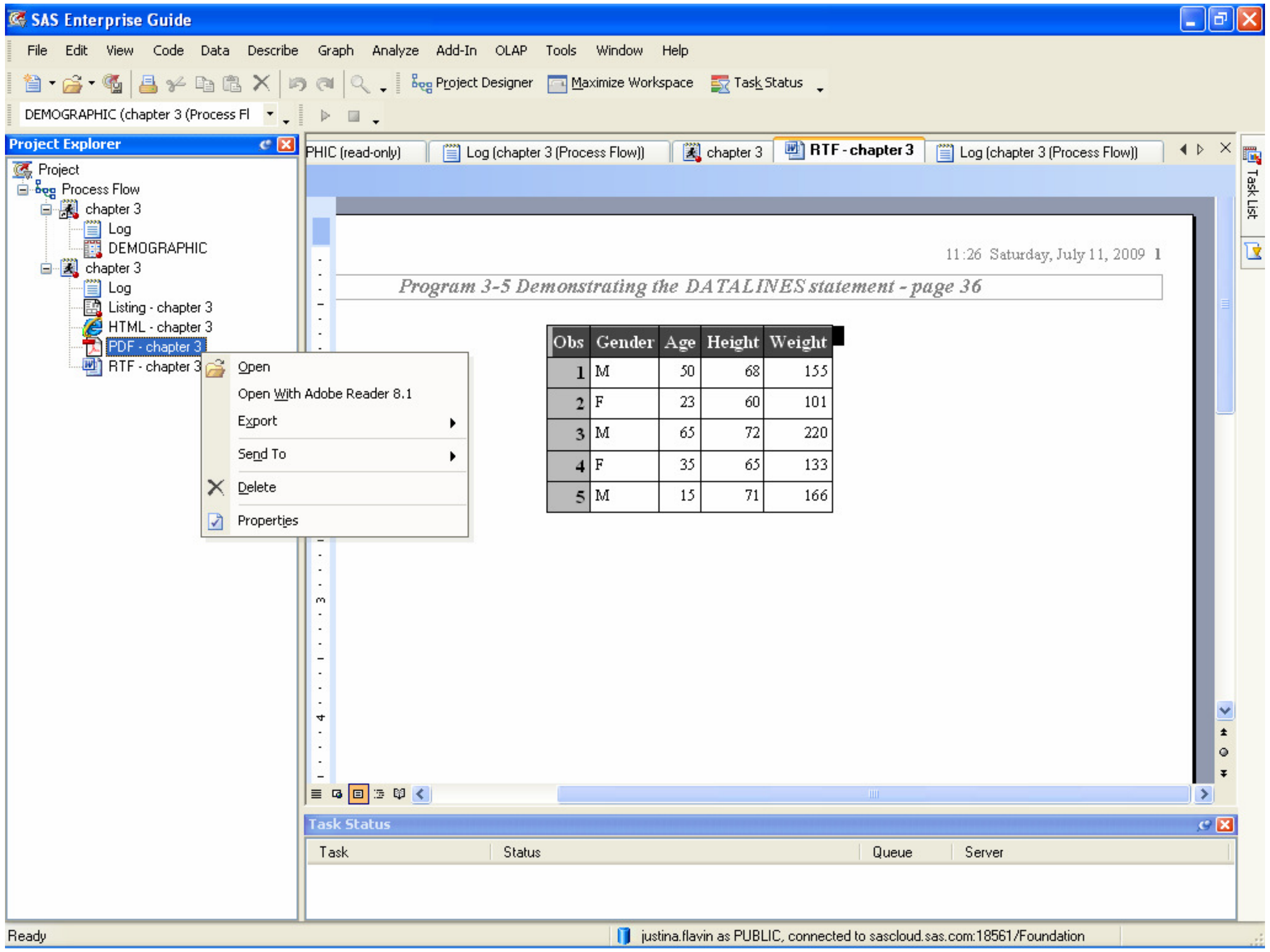
How might you do that?

By using a procedure called PROC PRINT.

The basic syntax is:

```
proc print;
run;
```

Executing the PROC PRINT statement in the code, generates the output in several formats, any of which can be managed (i.e. saved to a file or printed) by selecting the item on the left hand side and right clicking with the mouse to bring up a menu box.



11:26 Saturday, July 11, 2009 1

Program 3-5 Demonstrating the DATALINES statement - page 36

Obs	Gender	Age	Height	Weight
1	M	50	68	155
2	F	23	60	101
3	M	65	72	220
4	F	35	65	133
5	M	15	71	166

justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

Suppose that you wish to create a printout of your data that prints the variables is a specific order and omits printing of the variable weight.

This can be accomplished by the addition of the VAR statement.

The VAR statement is not unique to PROC PRINT. It is used in almost all procedures to select variables.

The syntax is:

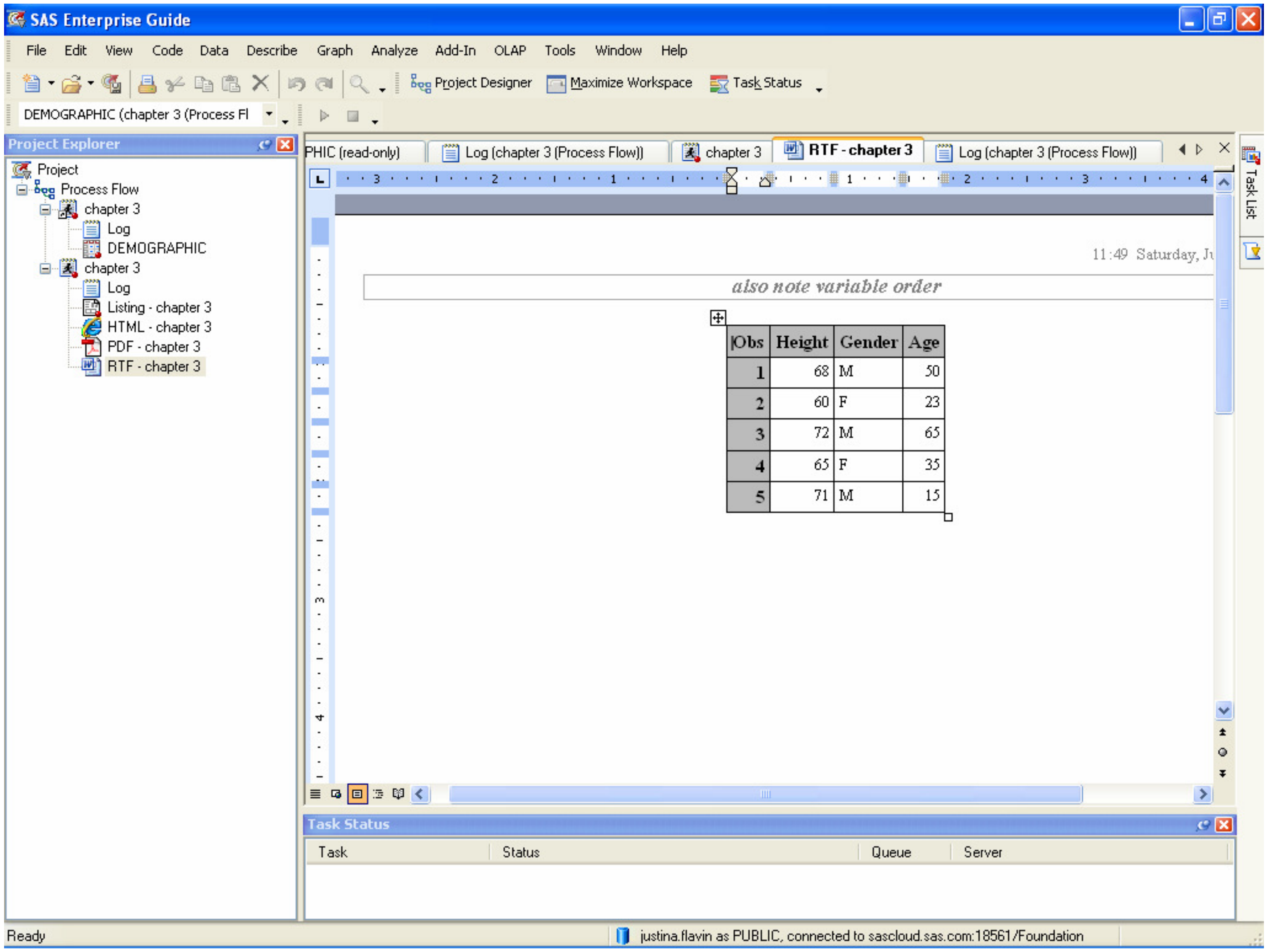
```
proc print;
  var variablename1 variablename2...variablename'n';
run;
```

Example:

```
/* The VAR statement is used in many procedures to select variables. */
proc print data=demographic;
  title also note variable order;
  var height gender age;
run;
title;
```

Notice that now the variables are printed out in the order specified on the VAR statement.

In the previous example, the variables printed out in the order in which they were stored within the data set.



11:49 Saturday, Ju

also note variable order

Obs	Height	Gender	Age
1	68	M	50
2	60	F	23
3	72	M	65
4	65	F	35
5	71	M	15

In this example, notice the addition of **data=demographic** on the PROC PRINT statement.

The addition of data=*datasetname* in any procedure specifically provides SAS with the name of the data set to be used.

If data=*datasetname* is omitted, then by default, SAS will use the last data set that was created in your SAS session.

In the previous example, this was omitted, so SAS used the last data set created which was demographic.

In general, it is a good idea to get into the habit of adding the data=*datasetname* to your procedure code.

Also notice the addition of the title statement within the PROC PRINT.

Because the title statement is a global statement, it can be placed outside of the procedure as was illustrated in the previous example, or within the procedure code as shown in this example.

Notice too that because it is numbered as title and not title2 or title3, etc. it overwrites the previous title statement that was submitted with the previous PROC PRINT.

The null title statement at the end of this code segment has the effect of turning off or cancelling all previous title statements.

So unless you execute some new title statements, any future output in this session will have no title text.

3.2 Reading Data Values Separated by Blanks

Now we will look at a similar example using list input method.

The only difference in this example is that the data resides in an external text file rather than as data lines in the data step.

Syntax:

```
data name;
  infile 'directory name and path followed by name of file';
  input variablename1 variablename2 $ variablename3...variablename'n';
run;
```

Notice that there is now an **infile** statement which replaces the datalines statement (and the lines of data).

When using an infile statement, you must tell SAS where to find the file by placing the directory path and the file name in single or double quotes.

Additionally, the infile statement must be placed in your code before the input statement.

Logically this makes sense since the infile statement tells SAS where to find the file and then the input statement provides SAS with the instructions for reading those data.

In this course, all data files that you will need are stored on the SAS server and must be accessed from the server.

You cannot download these files onto your computer and try to access them by specifying a path to a local directory because the SAS server cannot communicate with your local drive.

```
*Program 3-1 Demonstrating list input with blanks as delimiters - page 31;
data demographics;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.txt";
  input Gender $ Age Height Weight;
run;
```

The data file that is used is mydata.txt

The directory where it resides is: /courses/u_ucsd.edu1/i_536036/c_629

If you view the mydata.txt file, you will find that it contains the same data values as were used in the previous example (Program 3-5) where the data lines were included in the data step.

If you run this code, the data set generated, demographics, is the same as demographic, the data set generated in the previous example.

Most SAS statements allow for options on the statement line.

Options are optional items that are not required for statement execution but may be necessary for the statement to execute properly.

An example of an option that we have already seen is the data=datasetname option on the PROC PRINT statement.

There are times when you may need to use options on your infile statement .

The generic syntax is:

```
data name;
  infile 'directory name and path followed by name of file' option1 option2...option'n';
  input variablename1 variablename2 $ variablename3...variablename'n';
run;
```

Some SAS statements have a few options while others have quite a few.

The INFILE statement has many options and we will now look at a few of these.

3.4 Reading Data Values Separated by Commas (CSV Files)

In the previous examples, blanks were used as a delimiter between data values.

But you may have data files that use another character as a delimiter.

An example would be csv files that are created by Excel.

For SAS to be able to read in these data properly, we must tell it to use another character as the delimiter by adding an OPTION onto the INFILE statement.

When the delimiter is a comma, we add the letters ‘dsd’ onto the INFILE statement.
dsd means ‘delimiter sensitive data’

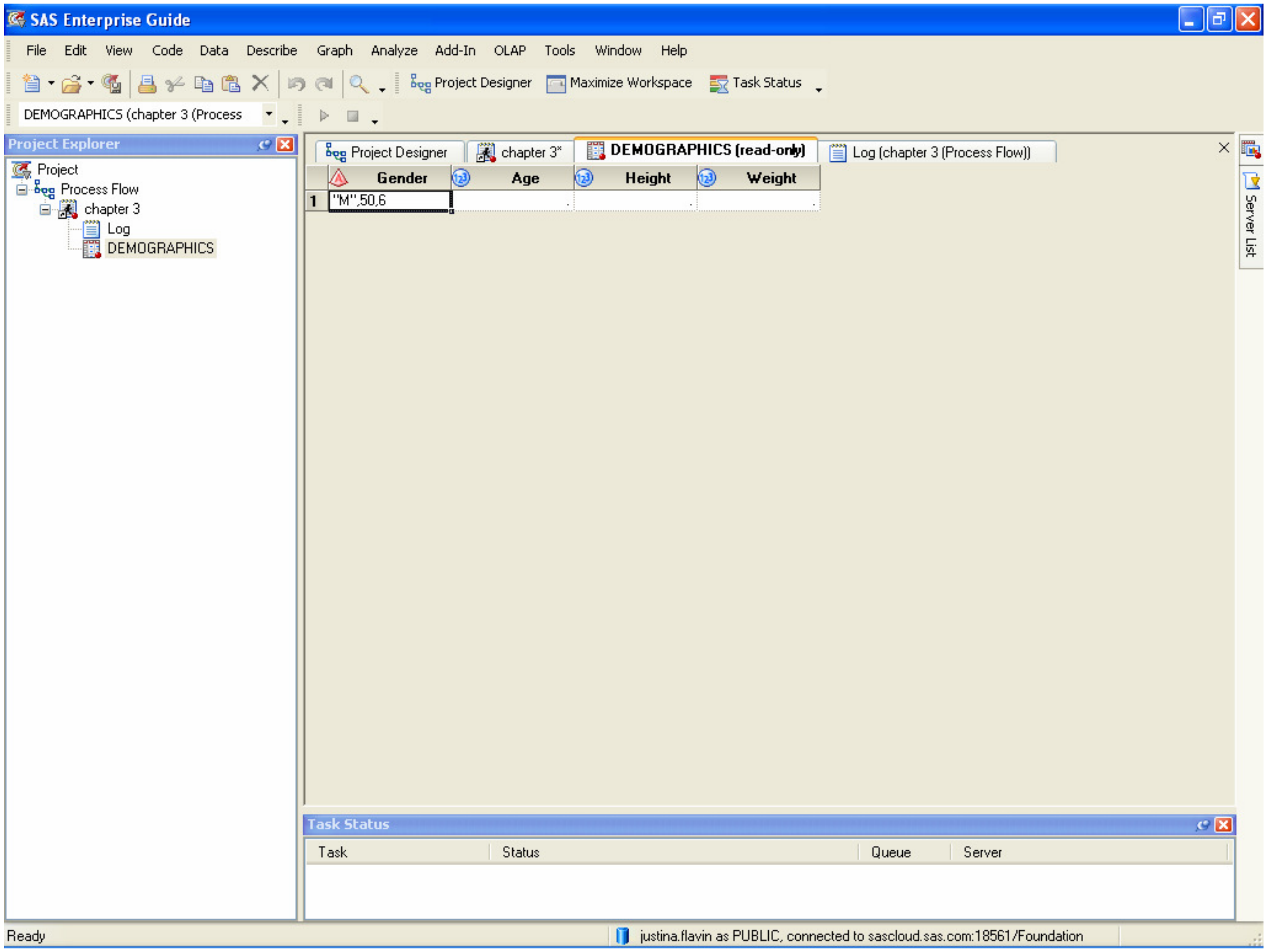
Here is an example program that illustrates the use of the dsd option with a csv file:

```
*Program 3-3 Reading data from a comma-separated variables (csv) file - page 33;
data demographics;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv" dsd;
  input Gender $ Age Height Weight;
run;
```

The data set generated is again identical to the previous DEMOGRAPHICS data sets.

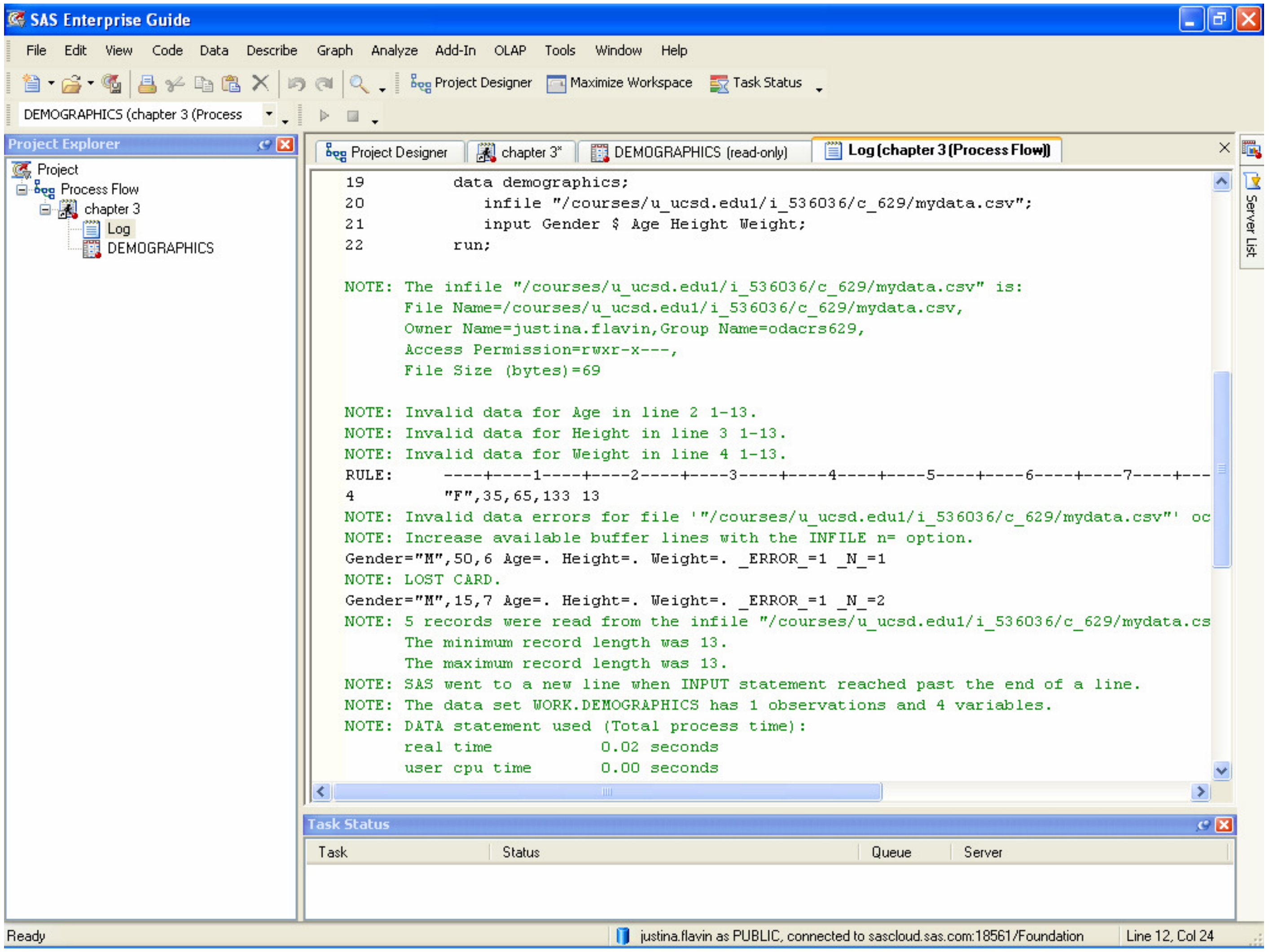
In this example, we created a data set called demographics. In the previous example, we also created a data set called demographics.
When you submitted the code for this example, SAS created a new demographics data set by overwriting the previous demographics data set.
The upshot here is that if you wish to replace a SAS data set with new data, you simply need to reuse the data set name.

Now let’s look at what happens if we rerun this code without the dsd option. Go into the code window and delete dsd from the line. SAS creates a data set, but it doesn’t look correct, does it?



	Gender	Age	Height	Weight
1	"M",50,6	.	.	.

Let's take a look at the log.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
DEMOGRAPHICS (chapter 3 (Process
Project Explorer
Project
Process Flow
chapter 3
Log
DEMOGRAPHICS
Project Designer
chapter 3*
DEMOGRAPHICS (read-only)
Log (chapter 3 (Process Flow))
Server List
Task Status
Task
Status
Queue
Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation
Line 12, Col 24

```
         data demographics;
            infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv";
            input Gender $ Age Height Weight;
         run;

NOTE: The infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv" is:
      File Name=/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv,
      Owner Name=justina.flavin,Group Name=odacrs629,
      Access Permission=rwxr-x---,
      File Size (bytes)=69

NOTE: Invalid data for Age in line 2 1-13.
NOTE: Invalid data for Height in line 3 1-13.
NOTE: Invalid data for Weight in line 4 1-13.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---
4          "F",35,65,133 13
NOTE: Invalid data errors for file '"/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv"' oc
NOTE: Increase available buffer lines with the INFILE n= option.
Gender="M",50,6 Age=. Height=. Weight=. _ERROR_=1 _N_=1
NOTE: LOST CARD.
Gender="M",15,7 Age=. Height=. Weight=. _ERROR_=1 _N_=2
NOTE: 5 records were read from the infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.cs
      The minimum record length was 13.
      The maximum record length was 13.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.DEMOGRAPHICS has 1 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      user cpu time       0.00 seconds
```

First, notice that there are no ERROR messages in the log. But SAS has provided a number of NOTES regarding what it was finding as it read in the lines of data.

The first NOTE: Invalid data for Age in line 2 1-13. tells us that SAS was trying to read in a value for Age from the second data line in columns 1-13. Taking a look at the data file (it is shown on page 33), we can see that that is clearly incorrect, since the variable Age resides in columns 5-6. In fact all 4 of our variables can be found in columns 1-13, but the log is telling us that SAS is attempting to read in Age using all 13 columns.

Then in the second Note statement we can see that SAS has now moved on to the third data line and is attempting to read in a value for Height from columns 1-13. The NOTE: Increase available buffer lines with the INFILE n= option. is meaningless. When SAS encounters an error or problem in your program, it attempts to generate statements that may be helpful to you in diagnosing the problem. Many times these statements ARE helpful, but sometimes they are not – this note about the buffer lines is an example of that. A good rule of thumb is to try to understand the first message (ERROR, WARNING, or NOTE) that indicates that something is wrong in your program because often, the subsequent “problem” messages resolve themselves once you fix the initial problem.

The **NOTE: SAS went to a new line when INPUT statement reached past the end of a line.** message is one that you may often see when reading in data from text files. This is usually indicative of a problem with missing values in the data and can usually be solved by adding the proper option onto your infile statement – this will be covered a bit later in the course. So if you ever get this statement in your log, there is probably a problem in your code and your data set is most likely incorrect!

This data step code provides an excellent example of why it is important to review your results (in this case the data set that was created) and the log carefully.

Don’t be the kind of programmer who says “there are no errors in my log so my code must be correct”.

Here is an explanation of what SAS is doing with the code that we submitted.

Without the dsd option on the INFILE line, SAS uses a blank space as the default delimiter. Since the dsd option is missing from the input statement, SAS is expecting to find 4 values separated by blank spaces on the data line.

The input statement instructs SAS to create 4 variables – Gender, Age, Height, and Weight – which it does.

When SAS reads the first line of data, it assigns the character string “M”,50,6 to the variable Gender, but because no blanks are encountered on the line of data, nothing is assigned to the rest of the variables.

If you are wondering why SAS did not put the entire data line into the variable Gender, it is because by default, variables are assigned a length of 8 – the length can be changed, and we will learn how to do this later on in the course. Thus SAS read in the first 8 characters on the line of data.

Trying to be helpful, SAS moves on to the second line of data and attempts to read in a value for Age. This action generates the **NOTE: SAS went to a new line when INPUT statement reached past the end of a line.** as well as **NOTE: Invalid data for Age in line 2 1-13.** SAS is unable to assign a value to Age because the first character it encounters on the second data line is the “ character. Since Age was specified as a numeric variable on the input line, SAS doesn’t expect to encounter character values when reading values for Age, so ignores that line of data.

Based on the next note statement, SAS tries to assign the 3rd line of data to the variable Height and encounters the same issue as it found for Age (a character value on the data line), so no value is assigned to Height.

The same action occurs for Weight.

Thus we end up with a data set that has one observation.

As mentioned previously, the INFILE statement has many options that can be used.

As you just saw, while dsd is an optional statement, it is REQUIRED when trying to read in comma delimited data.

Suppose I had not provided an explanation of the dsd option, but you saw it in the code and wanted to know why it was there? Or if perhaps you are wondering if there is an option that allows you to read data with another type of delimiter such as a semi-colon (;)?

Let’s use the online documentation to find out a bit more.

Using the instructions provided in the Introduction slides, go to

http://support.sas.com/documentation/onlinedoc/91pdf/index.html

and select SAS OnlineDoc 9.1 for the Web.

Then navigate to Base SAS -> SAS Language Reference: Dictionary -> Dictionary of Language Elements -> Statements -> INFILE Statement -> Syntax.

Scroll down the page until you find DSD under the Options section.

Were you also able to find the option which allows you to specify a different delimiter character?

You may wish to read up on that as we will be covering that a bit later in this chapter.

You will find in SAS that there are often multiple ways of doing the same thing.

This next section illustrates this.

3.5 Using an Alternative Method to Specify an External File

When working with external files, the location and name of a file can be specified on a filename statement instead of on the infile statement in the data step.

The filename statement is another example of a global statement.

Syntax:

```
filename fileref ‘directory name and path followed by name of file’;

data name;
   infile fileref;
   input variablename1 variablename2 $ variablename3...variablename’n’;
run;
```

Fileref is a user supplied character string that is specified on the filename statement.

Then on the INFILE statement, it is used like an alias or short name for the location and file name.

```
*Program 3-4 Using a FILENAME statement to identify an external file - page 34;
filename preston  "/courses/u_ucsd.edu1/i_536036/c_629/mydata.csv";

data demographics;
  infile preston dsd;
  input Gender $ Age Height Weight;
run;
```

In this example, *fileref* is assigned to **preston** on the FILENAME statement. Then on the INFILE statement, **preston** is used in place of the location and file name information.

Using a filename statement may look like it is more work since it requires an additional line of code.
But there are times when it is much more advantageous to use filenames.

For instance, suppose that you have multiple external files that you read into a SAS program to generate a weekly report about product production at your company. Every week you receive files that are formatted the same way as the previous week's files, but each week the file names are different. So each week you open up your SAS code, update the filenames and run the report.

If the filenames are scattered throughout the code in INFILE statements, you have to search through the code to find and replace these. But if you create a series of FILENAME statements and place these at the beginning of your program, then updating the filenames each week becomes a much easier task.

3.6 Reading Data Values Separated by Delimiters Other Than Blanks or Commas

As you may have already discovered, the option that can be added to the infile statement to explicity specify the delimiter characters is the DLM option.

The syntax is:
dlm=‘*delimiter character(s)*’

Here is an example of using the dlm option to read in a file that uses colons (:) as the delimiter

```
data info;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/colon_delim.txt" dlm=':';
  input Gender $ Age Height Weight;
run;
```

Again, if you were to remove the dlm option in this program, your colon delimited data would not be read in properly.

the dlm option can always be used on any infile statement.
For instance in Program 3-1, the infile statement could be changed to:
infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.txt“ dlm=‘ ‘;

This would explicity specify the delimiter character, a blank space. It is not necessary though because a blank space is the default delimiter character. Likewise, when using the DSD option, it is not necessary to specify that the delimiter is a comma, because a comma is the default delimiter in that case.

3.8 Specifying INFILE Options with the DATALINES Statement

Our first example illustrated reading in data with the DATALINES statement. Since the data used blanks as the delimiters and were included as part of the data step and not stored in an external file, we did not have to use an infile statement.

But what if we had a non-blank delimiter character? Per our previous examples we know that we have to include an option on the infile statement so that SAS knows what the delimiter character is. So we do the same thing here and include an infile statement with the proper option.

```
*Program 3-6 Using INFILE options with DATALINES - page 37;
data demographics;
  infile datalines dsd;
  input Gender $ Age Height Weight;
datalines;
"M",50,68,155
"F",23,60,101
"M",65,72,220
"F",35,65,133
"M",15,71,166
;
run;
```

Notice that on the infile statement, **datalines** replaces the path & filename specification that we saw before.

As you were reading Chapters 1 & 2, you encountered some concepts and procedures that were presented without much explanation.

Now that you know how to create a SAS data set, it is meaningful to go back and look at some of these items.

Adding New Variables to a Data Set**

Assignment statements are used in a data set to create new variables.

An assignment statement is an instruction to perform the computation on the right hand side of the equal sign and assign the resulting value to the variable named on the left.

There is a table of arithmetic operators on page 19 which you may find helpful.

Here are some examples of assignment statements:

```
amt_due = purchases – returns;
```

This creates a variable called amt_due by subtracting the value of the variable **returns** from the variable **purchases.**

```
x=(y+z)/2;
```

This creates a new variable called x by adding the variables y and z and then diving that sum by 2.

```
data demographic;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/mydata.txt";
  input Gender $ Age Height Weight;
  *compute BMI;
  BMI=(weight/2.2)/(height*.0254)**2;
run;
```

If you run this code you will find that the data set demographic will now contain the 4 variables Gender, Age, Height, and Weight as well as the new calculated variable BMI.

Also notice the addition of a comment line prior to the BMI calculation.

A comment line within a data step is perfectly acceptable and can be used to provide comments or explanations about the code.

With complicated data step code, adding comments can greatly enhance the readability and maintainability of the code.

*** Basic Statistical Procedures***

Two useful statistical procedures are PROC FREQ and PROC MEANS.

These will be covered in more detail later in the course, but the basic syntax will be shown here.

PROC FREQ provides frequency counts of variable values. It is very useful for obtaining summary information about the values of variables in large data sets.

Syntax:

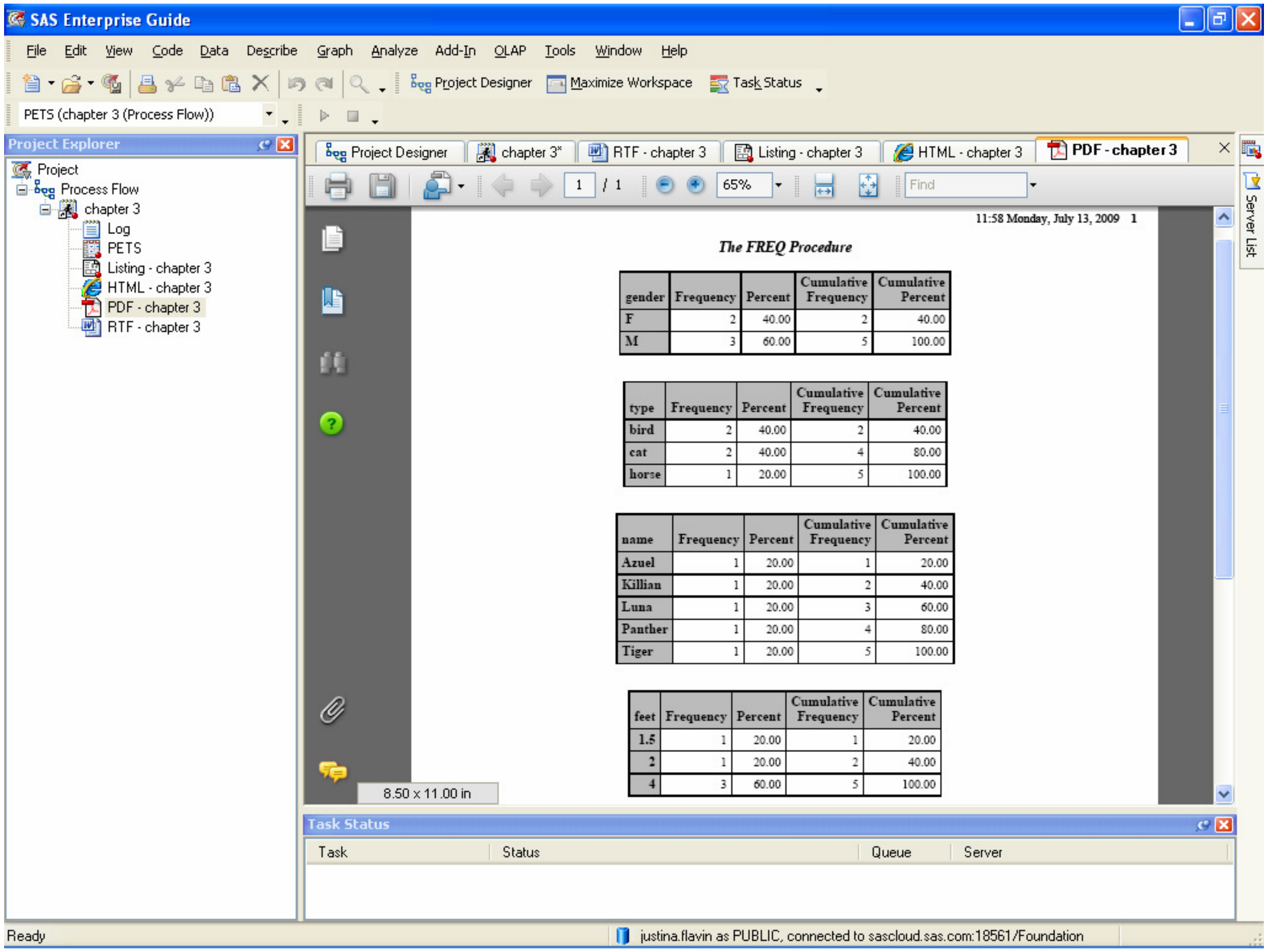
```
/* Generate frequency counts of all variables */
proc freq;
run;
```

Example:

```
data pets;
  input gender $ type $ name $ feet;
datalines;
F horse Luna 4
M bird Azuel 2
M bird Killian 1.5
F cat Panther 4
M cat Tiger 4
;
run;

* frequency counts with PROC FREQ ;
proc freq;
run;
```

Here is the PROC FREQ output.



11:58 Monday, July 13, 2009 1

The FREQ Procedure

gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	2	40.00	2	40.00
M	3	60.00	5	100.00

type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
bird	2	40.00	2	40.00
cat	2	40.00	4	80.00
horse	1	20.00	5	100.00

name	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Azuel	1	20.00	1	20.00
Killian	1	20.00	2	40.00
Luna	1	20.00	3	60.00
Panther	1	20.00	4	80.00
Tiger	1	20.00	5	100.00

feet	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1.5	1	20.00	1	20.00
2	1	20.00	2	40.00
4	3	60.00	5	100.00

PROC FREQ generates summary information about each variable in the data set.

The first column provides the name of the variable and all values of that variable. By default, values are listed in ascending order. The second column provides the frequency count of each variable value.

In this example the variable sex has 2 values of ‘F’ and 3 values of ‘M’.

The Percent Column shows the percentage of the total number of observations having that value. For sex, there are 2 values of F and a total of 5 observations, so 2/5=40%.

The Cumulative Frequency column provides a running total of the frequency count on that row plus all other rows above it. The Cumulative Percent column shows the value of the cumulative frequency column as a percentage of the total number of observations.

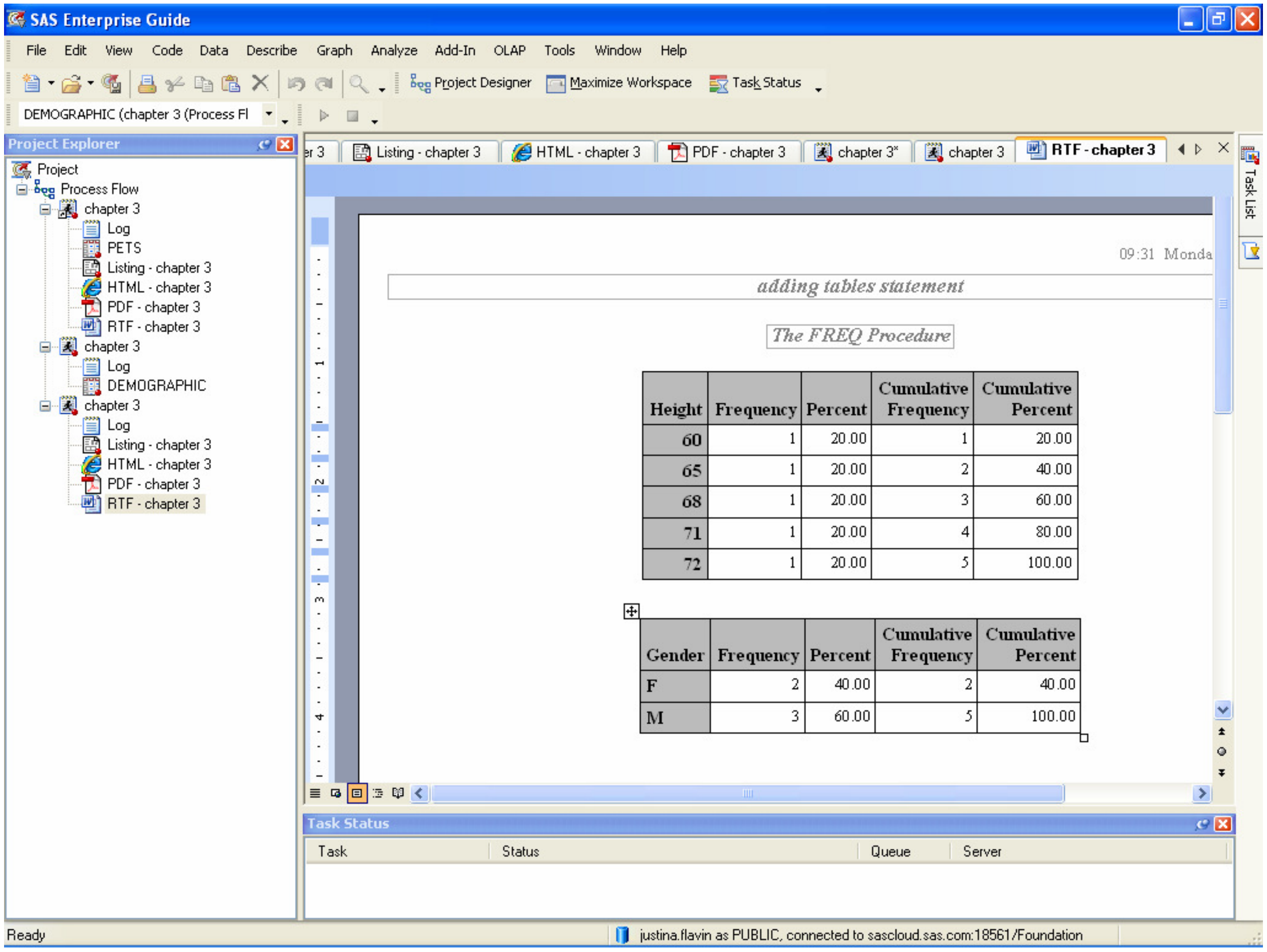
The TABLES statement can be added to select or limit the variables to be included in the output.

```
/* TABLES statement generates frequency counts of some variables */
proc freq;
  tables variablename1 variablename2...variablename'n';
run;

/* TABLES statement generates frequency counts of some variables */
proc freq data=demographic;
  title adding tables statement;
  tables height gender;
run;
```

Notice the addition of the data=demographic statement on the PROC FREQ statement. This specifically tells SAS which data set to use. Had we not included it, SAS would have used PETS, the last data set created.

By adding the tables statement, the output is limited to just the variables listed. Also, notice that the output for height is listed first followed by gender. Had we omitted the tables statement, the variables would be listed on the output in the order in which they occur in the data set and Gender would have been the first variable listed followed by Age, Height, Weight, and BMI.



09:31 Monda

adding tables statement

The FREQ Procedure

Height	Frequency	Percent	Cumulative Frequency	Cumulative Percent
60	1	20.00	1	20.00
65	1	20.00	2	40.00
68	1	20.00	3	60.00
71	1	20.00	4	80.00
72	1	20.00	5	100.00

Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	2	40.00	2	40.00
M	3	60.00	5	100.00

PROC MEANS provides some simple descriptive statistics on the numeric variables in a data set.

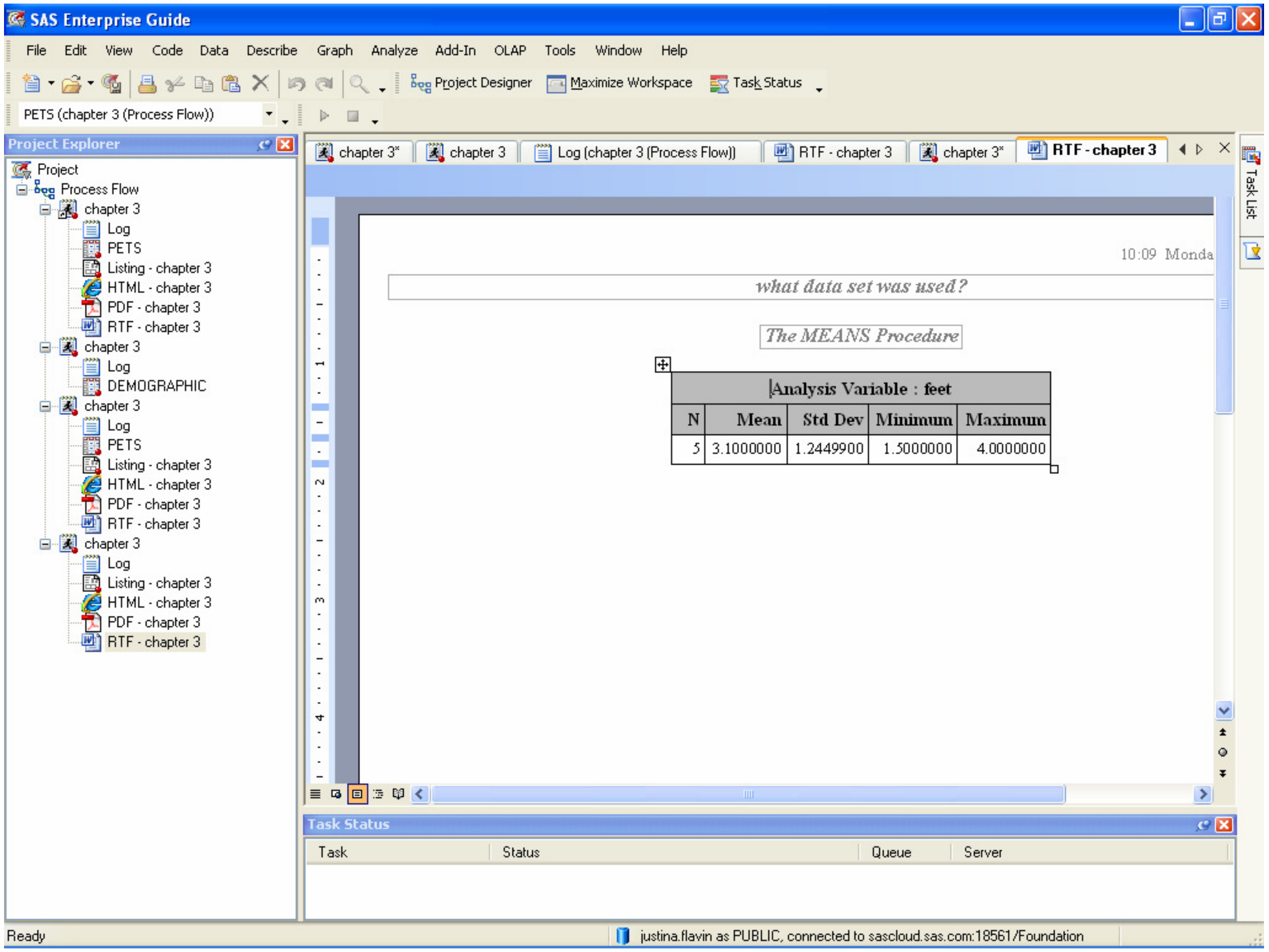
Syntax:

```
proc means;
run;
```

Output is shown on the next slide.

Notice that in the code, no data set was specified, so as mentioned previously, SAS uses the last data set created which was PETS. So it is good programming practice to always add the data=*datasetname* option on your PROC statements.

The PETS data set contains 1 numeric variable (feet). The N column provides the number of nonmissing observations used to calculate the statistics. Std Dev is the sample standard deviation. The Minimum and Maximum columns provide the range of the data values and can be verified by looking at the data.



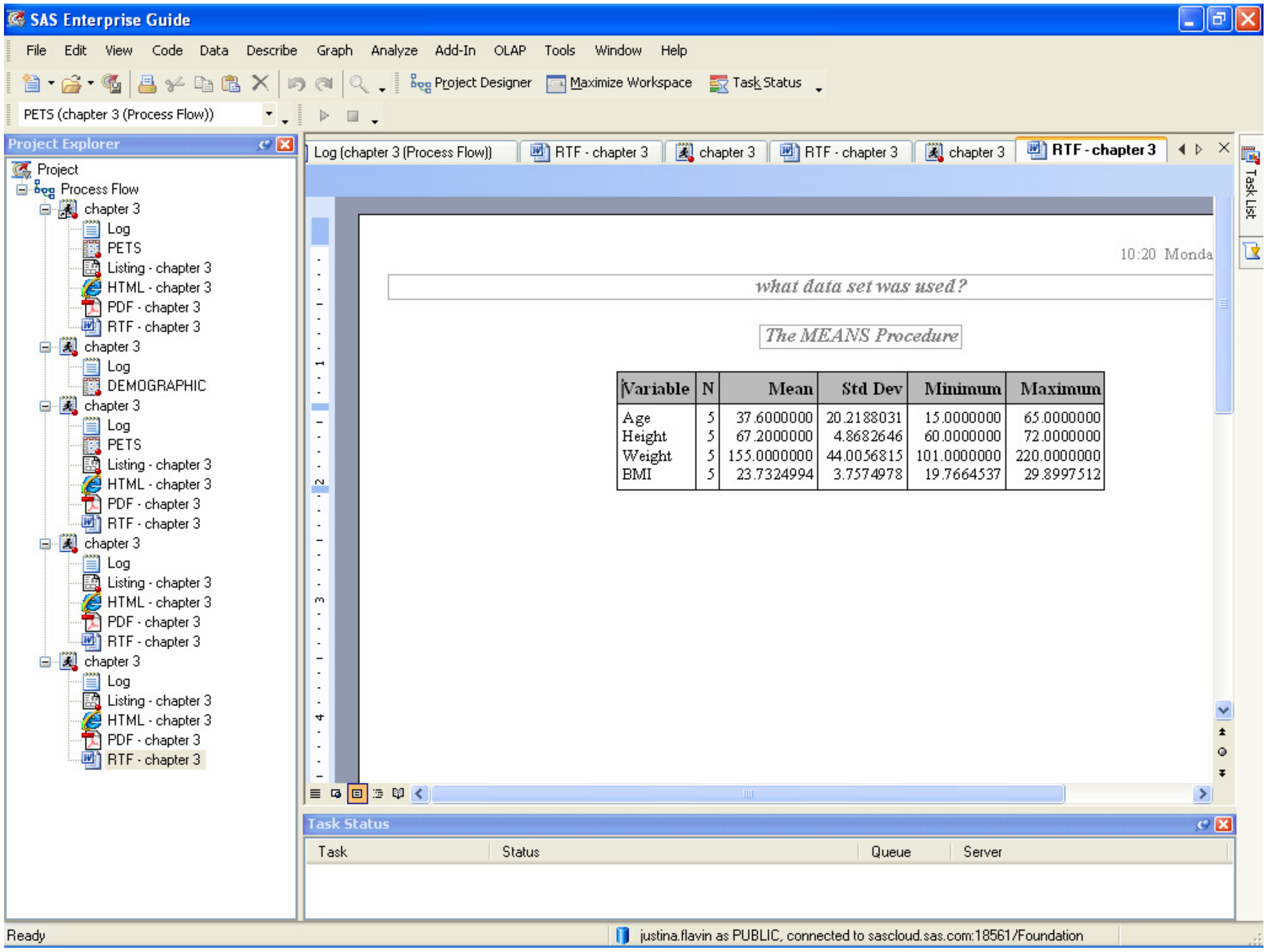
10:09 Monda

what data set was used?

The MEANS Procedure

Analysis Variable : feet				
N	Mean	Std Dev	Minimum	Maximum
5	3.1000000	1.2449900	1.5000000	4.0000000

Running PROC MEANS on the DEMOGRAPHIC data set provides the same summary information for all the numeric variables. Like PROC FREQ, the variables are listed in the order in which they are stored in the data set.



10:20 Monda

what data set was used?

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	5	37.6000000	20.2188031	15.0000000	65.0000000
Height	5	67.2000000	4.8682646	60.0000000	72.0000000
Weight	5	155.0000000	44.0056815	101.0000000	220.0000000
BMI	5	23.7324994	3.7574978	19.7664537	29.8997512

Like PROC FREQ, the variables to be included on the output can be controlled, using the VAR statement.

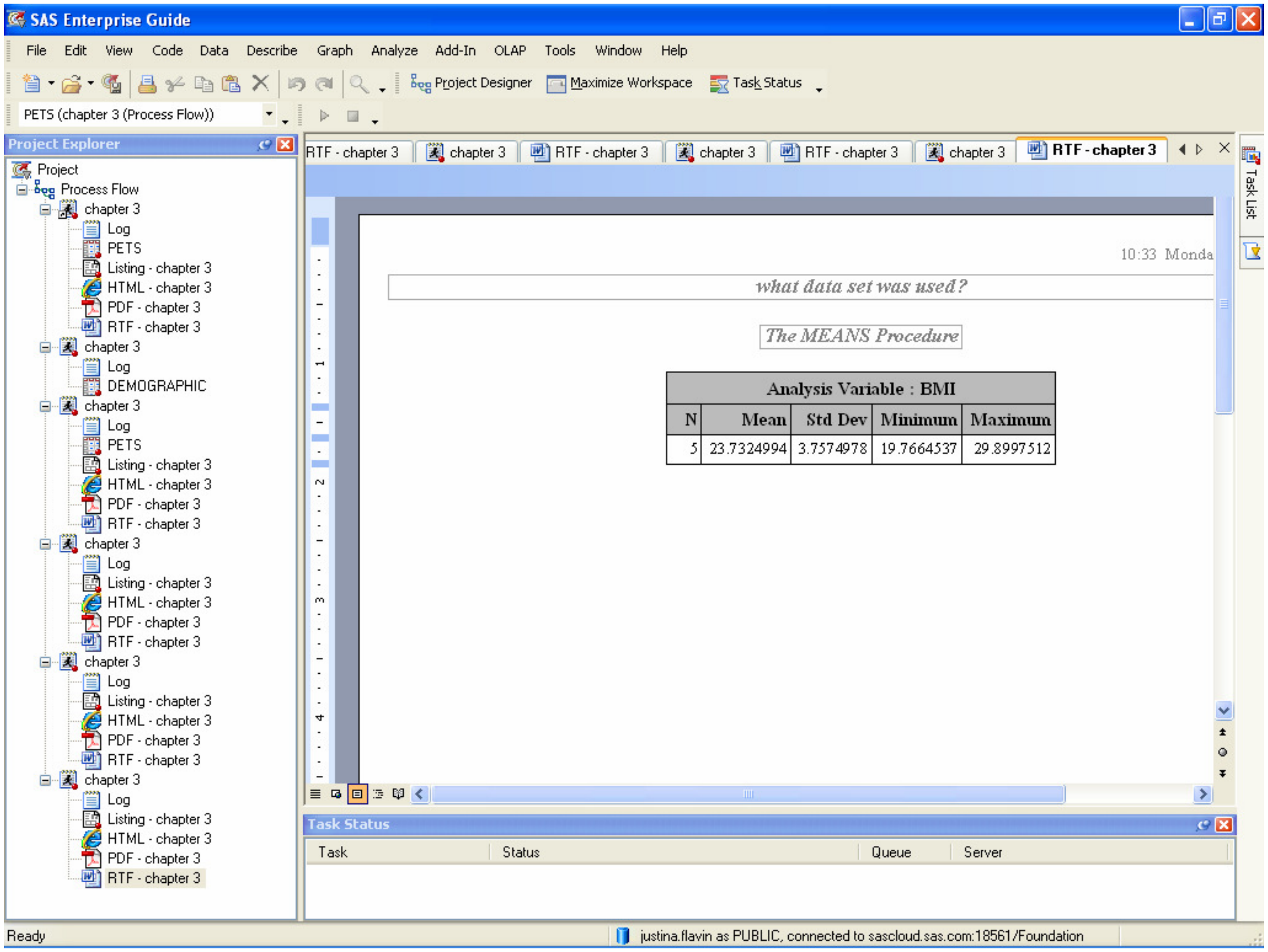
Syntax:

```
proc means;
   var variablename1 variablename2...variablename'n';
run;
```

Example:

```
proc means data=demographic;
   var bmi;
run;
```

Now the output is limited to the variable BMI.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer
Maximize Workspace
Task Status
PETS (chapter 3 (Process Flow))
Project Explorer
Task List
Task Status
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

10:33 Monda

what data set was used?

The MEANS Procedure

Analysis Variable : BMI				
N	Mean	Std Dev	Minimum	Maximum
5	23.7324994	3.7574978	19.7664537	29.8997512

For now, this is as much as we will cover on PROC MEANS and PROC FREQ. These procedures will be covered in more detail in later lectures.

If you are interested in reading more about these procedures, you can go to the online documentation:

http://support.sas.com/documentation/onlinedoc/91pdf/index.html and select SAS OnlineDoc 9.1 for the Web.

Then navigate to Base SAS -> Base SAS Procedures Guide -> Procedures and then scroll down to the FREQ Procedure and the MEANS Procedure

3.9 Reading Raw Data from Fixed Columns—Method 1: Column Input

The next method we will look at for reading in data from fixed columns is column input method.

With column input method, you specify the exact starting and ending columns for each variable and SAS then reads in everything between those two columns.

Column input method can read data with missing values and character data with embedded blanks – these would be problematic if trying to use list input method.

Syntax:

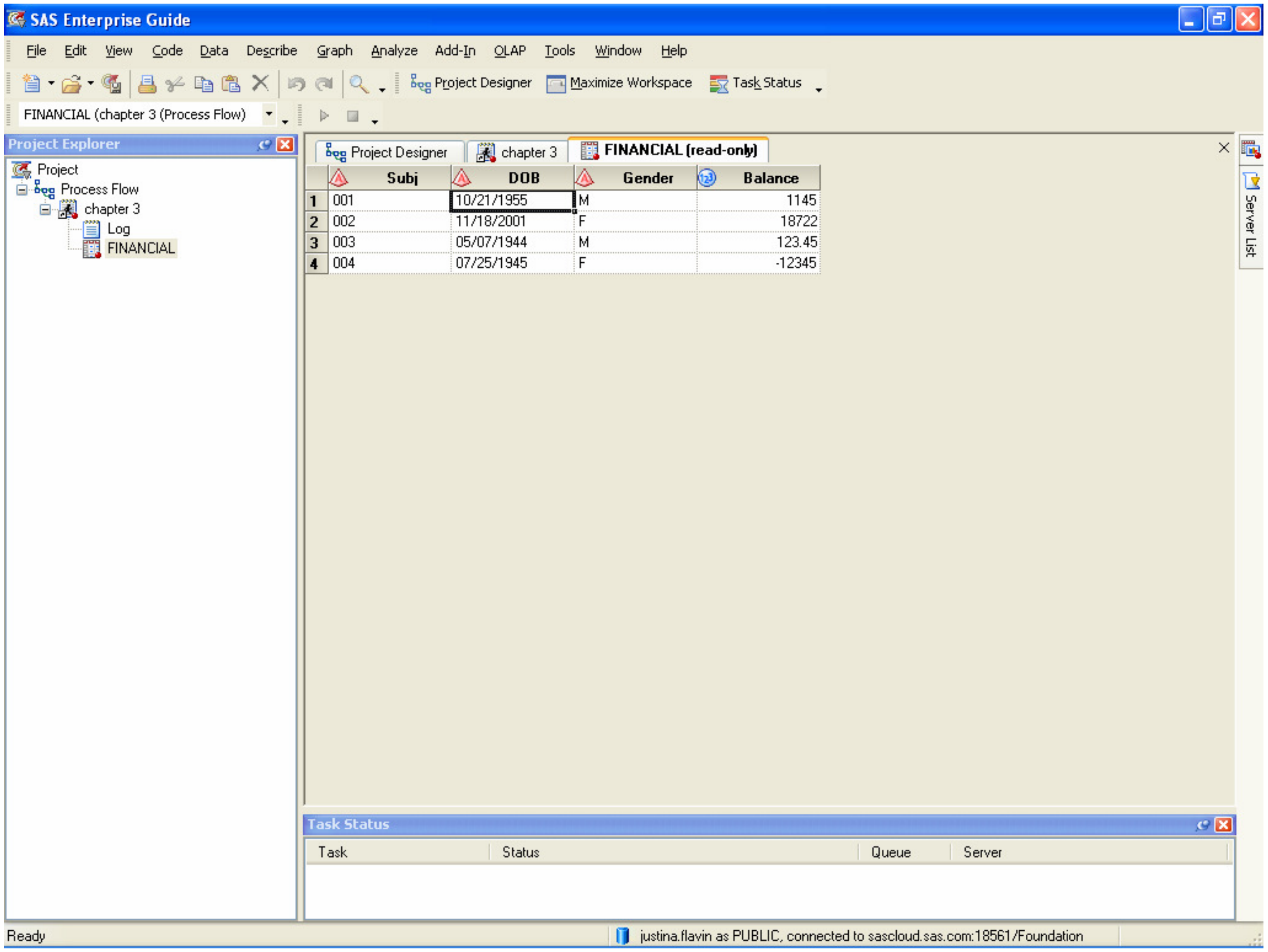
```
data name;
infile ‘directory name and path followed by name of file’;
  input variablename1 col1-col2
        variablename2 $ col3-col4
        variablename3 col5;
run;
```

```
*Program 3-7 Demonstrating column input - page 38;
data financial;
infile "/courses/u_ucsd.edu1/i_536036/c_629/bank.txt";
input Subj     $   1-3
      DOB      $  4-13
      Gender   $    14
      Balance    15-21;
run;
```

In this example, the data for the variable Subj is in columns 1-3, DOB is in 4-13, Gender is in column 14, and Balance in columns 15-21.

The contents of the data file bank.txt is shown on page 38.

Notice in the dataset that DOB looks like a date. Because of the presence of the "/" character in the data, we had to read it into the data set as a character variable. In the next section we will see how to (properly) read dates into a data set, so that SAS recognises them as valid date values.



	Subj	DOB	Gender	Balance
1	001	10/21/1955	M	1145
2	002	11/18/2001	F	18722
3	003	05/07/1944	M	123.45
4	004	07/25/1945	F	-12345

The third and most powerful method for reading in data from a text file is formatted input method. This method requires an understanding of the way SAS stores dates and times and familiarity with the concepts of informats and formats.

We will cover Chapter 9 in a future lecture, but please refer to Section 9.2 now.

9.2 How SAS Stores Dates

In a SAS data set, dates are stored as the elapsed number of days between 01 January 1960 and the date itself.

Dates prior to 01 January 1960 are negative values, dates after 01 January 1960 are positive values.

Times are stored as the elapsed number of seconds between midnight and that time of day.

In the examples shown in Section 9.2, we see that the date June 15, 2006 is stored as the number 16967 which is the number of days between January 1, 1960 and June 15, 2006.

The date October 21, 1950 is a negative value since it is a date prior to January 1, 1960.

To properly read and display dates and times in SAS, **informats** and **formats** must be used.

SAS contains many standard informats and formats that are built in and are part of the programming language.

In addition, you can create your own formats using the PROC FORMAT procedure which we will cover in a future lecture.

Informats are instructions that tell SAS how to read the data.
For any variable, there is usually only 1 informat that can be used to properly read in the data.
Which informat you should be using depends upon what the data looks like.
When informats are used on an **input** statement, the keyword **informat** is not used. However, an **informat** statement can also be used within a data step.

To use an informat on an **input** statement, a column pointer "@" or an informat modifier ":" must be used.

A column pointer "points" to the specified column in the data.

Syntax:

```
input @columnnumber1 variablename1 informat. @columnnumber2 variablename2 $informat.;
```

Formats are instructions that tell SAS how to display the data.

Unlike informats, there are often many different formats which can be used to display the data.
Which format you use depends upon how you want your data to look in your output.

Formats are specified on a **format** statement in a DATA or PROC step.

Syntax:

```
format variablename1 format. variablename2 $format.;
```

Both informats and formats specify width of the data values being read in or displayed.

Here are some commonly used informats and formats:

Informats:

w.d informat reads numeric data with implied decimal points
w specifies the width of the data value
d optionally specifies the number of digits after the decimal point

$w informat reads character data
w specifies the width of the data value

mmddyy*w*. informat reads date values in the form *mmddyy* or *mmddyyyy*

Formats:

mmddyyw. format displays dates in the form mm/dd/yy

dollarw.d format writes numeric values with a leading dollar sign, a comma that separates every three digits, and a period that separates the decimal fraction

There are many, many more informats and formats available and no course or book could possibly cover them all.

Again, you need to refer to the online documentation to see the complete list of informats and formats that are available.

As you start writing your own programs in SAS, you will probably find yourself referring to the informats and formats sections in the online documentation quite a bit.

Information on informats and formats may be found at:

http://support.sas.com/documentation/onlinedoc/91pdf/index.html and select SAS OnlineDoc 9.1 for the Web.

Then navigate to Base SAS ->SAS Language Reference: Dictionary ->Dictionary of Language Elements and navigate to Informats or Formats.

Within the informats and formats sections there are subsections that list the informats/formats by category.

These subsections are very helpful when browsing or looking for the correct informat to use.

In this next example, we will use the formatted input method to read the same data file (bank.txt) that was used in the column input method example.

3.10 Reading Raw Data from Fixed Columns—Method 2: Formatted Input

```
*Program 3-8 Demonstrating formatted input - page 40;
data financial2;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/bank.txt";
  input @1  Subj        $3.
        @4  DOB    mmddyy10.
        @14 Gender      $1.
        @15 Balance      7.;
run;
```

The input statement can be read and interpreted as follows:

At column 1, read the variable Subj using the $w informat for a character variable of length 3

At column 4, read the variable DOB using the mmddyy informat for a date value of length 10

At column 14, read the variable Gender using the $w informat for a character variable of length 1

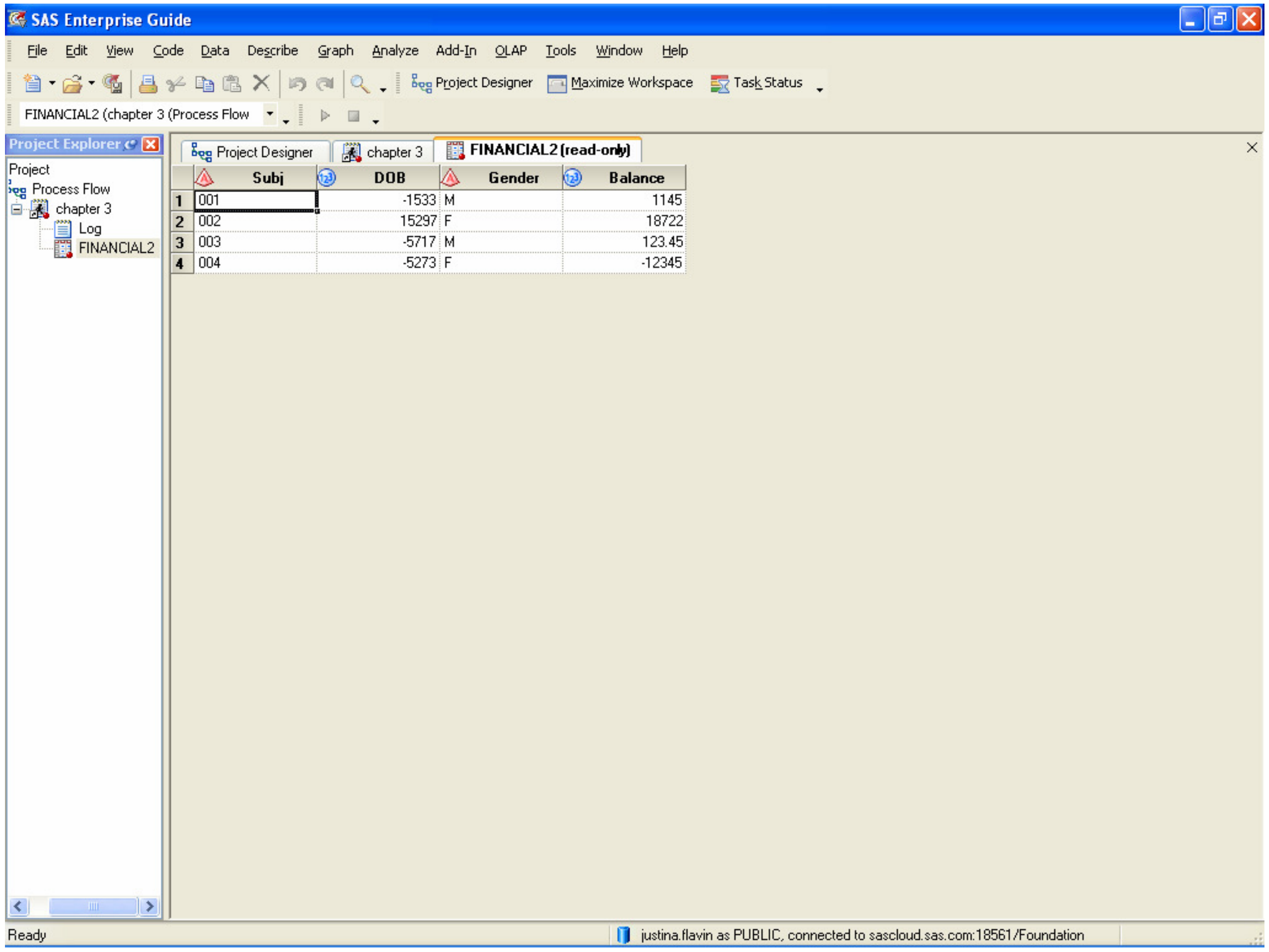
At column 15, read the variable Balance using the w.d informat for an integer that may have up to 7 digits

Now you may be wondering how to know what the correct informat is for a given variable. You need to visually inspect the data and then determine which informat will read it properly. For instance, the DOB field as listed on page 38 is of the form: mm/dd/yyyy. You know that you need to use an informat to read this date properly, so you could refer to the informats section on the online documentation and use the informats by category section to browse all the date and time informats. The details of each informat provide the specification of what the data must look like. Browse through the informats and you will find that the correct one to use for DOB is the mmddyy informat.

Notice that DOB looks different but the other variables have not changed from the previous example.

DOB no longer “looks” like a date. Notice also that it is now stored as a numeric rather than a character variable.

Because we used the proper informat with this date variable, SAS calculates the number of days between each date and January 1, 1960 and stores that value instead. In the next example we will see how to apply formats to make this value look like a date again.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
FINANCIAL2 (chapter 3 (Process Flow
Project Explorer
Project
Process Flow
chapter 3
Log
FINANCIAL2
Project Designer
chapter 3
FINANCIAL2 (read-only)
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Subj	DOB	Gender	Balance
1	001	-1533	M	1145
2	002	15297	F	18722
3	003	-5717	M	123.45
4	004	-5273	F	-12345

Now we will see how to add a format to DOB to display the variable in a more meaningful way.
As mentioned previously, formats can be used to display the data.
Unlike informats, there many be several formats that you can use to display your data.
Which format you use depends upon how you want your data to look in your output.

The format statement can be used in both the data step and in procedures.

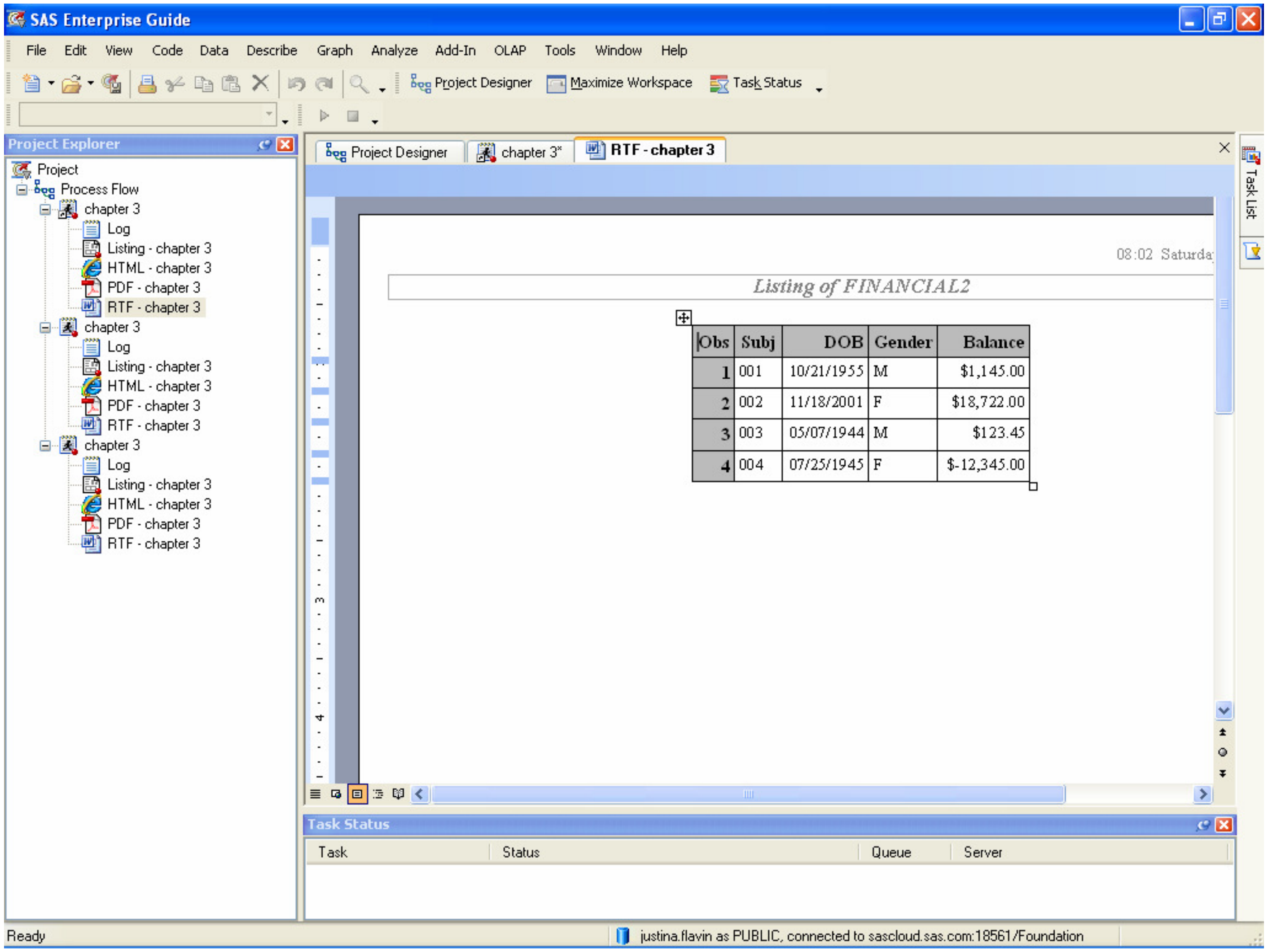
Here is an example of using a format statement in PROC PRINT:

```
*Program 3-9 Demonstrating a format statement - page 42;
title "Listing of FINANCIAL";
proc print data=financial2;
  format DOB     mmddyy10.
         Balance dollar11.2;
run;
```

We have now applied formats to the variables DOB and Balance.

Notice that DOB looks like a date again and balance is now displayed as currency with the addition of the $.

It is important to remember that in applying these formats in the PROC PRINT procedure, we have not “changed” the underlying data, we have simply changed the way the data are displayed in the output.



08:02 Saturda

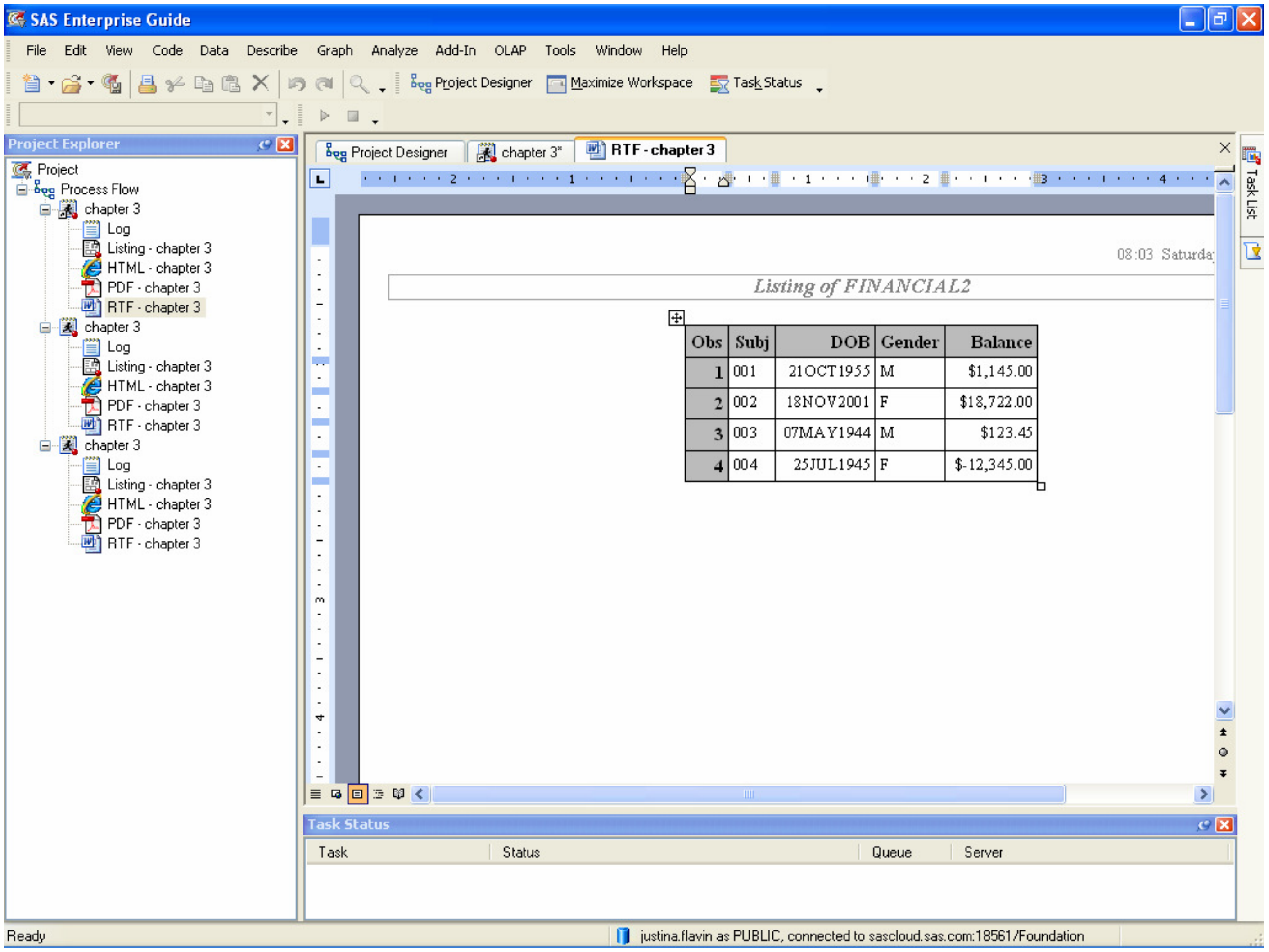
Listing of FINANCIAL2

Obs	Subj	DOB	Gender	Balance
1	001	10/21/1955	M	$1,145.00
2	002	11/18/2001	F	$18,722.00
3	003	05/07/1944	M	$123.45
4	004	07/25/1945	F	$-12,345.00

Now let’s rerun the PROC PRINT again, this time applying a different format to DOB.

```
*Program 3-10 Rerunning program Program 3-9 with a different format - page 42;
title "Listing of FINANCIAL2";
proc print data=financial2;
  format DOB     date9.
         Balance dollar11.2;
run;
```

Looking at the output, we now see that DOB is still displayed as a date, but in a different way. Again, this has not affected the underlying data, only the display of the data.



Listing of FINANCIAL2

Obs	Subj	DOB	Gender	Balance
1	001	21OCT1955	M	$1,145.00
2	002	18NOV2001	F	$18,722.00
3	003	07MAY1944	M	$123.45
4	004	25JUL1945	F	$-12,345.00

3.11 Using a FORMAT Statement in a DATA Step versus in a Procedure

As mentioned previously, a format statement can be used in both a data step and a procedure.

When used in a procedure, the format associated with a variable is said to be a temporary format. This is because the format is applied to the variable only in that procedure.

When a format statement is applied in a data step, the format associated with a variable is said to be a permanent format. This is because in a data set, the format is stored with the variable.

A variable with a format attached to it is said to be a formatted variable. When viewing the data in the data set, the variable will be displayed with the format applied to it. Further, when a formatted variable is used in a procedure, the variable will be displayed per the format in any output produced.

However, if you wish to display a formatted variable using a different format in a procedure, a format statement can be used to apply a different format to the variable.

Finally, to remove a permanent format from a variable, use a null format statement which lists the variable or variable(s) without any format names.

Syntax:

```
format variablename1 variablename2 variablenamen;
```

The next example will illustrate the use of informats along with an informat modifier.

3.12 Using Informats with List Input

To use an informat with list input, a colon “:” (informat modifier) precedes the informat name.
The informat modifier instructs SAS to stop inputting when a delimiter is found.

Syntax:

```
input variablename1 : informat. variablename2 : $informat.;
```

```
*Program 3-11 Using INFORMATS with List Input - page 44;
data list_example;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/list.csv" dsd;
  input Subj   :       $3.
        Name   :      $20.
        DOB    : mmddyy10.
        Salary :  dollar8.;
  format DOB date9. Salary dollar8.;
run;
```

The data for this example are shown on page 44.

By adding the colon after the variable names, we are instructing SAS to read in up to 3 characters for Subj, up to 20 characters for Name, up to 10 digits for DOB and up to 8 digits for Salary.

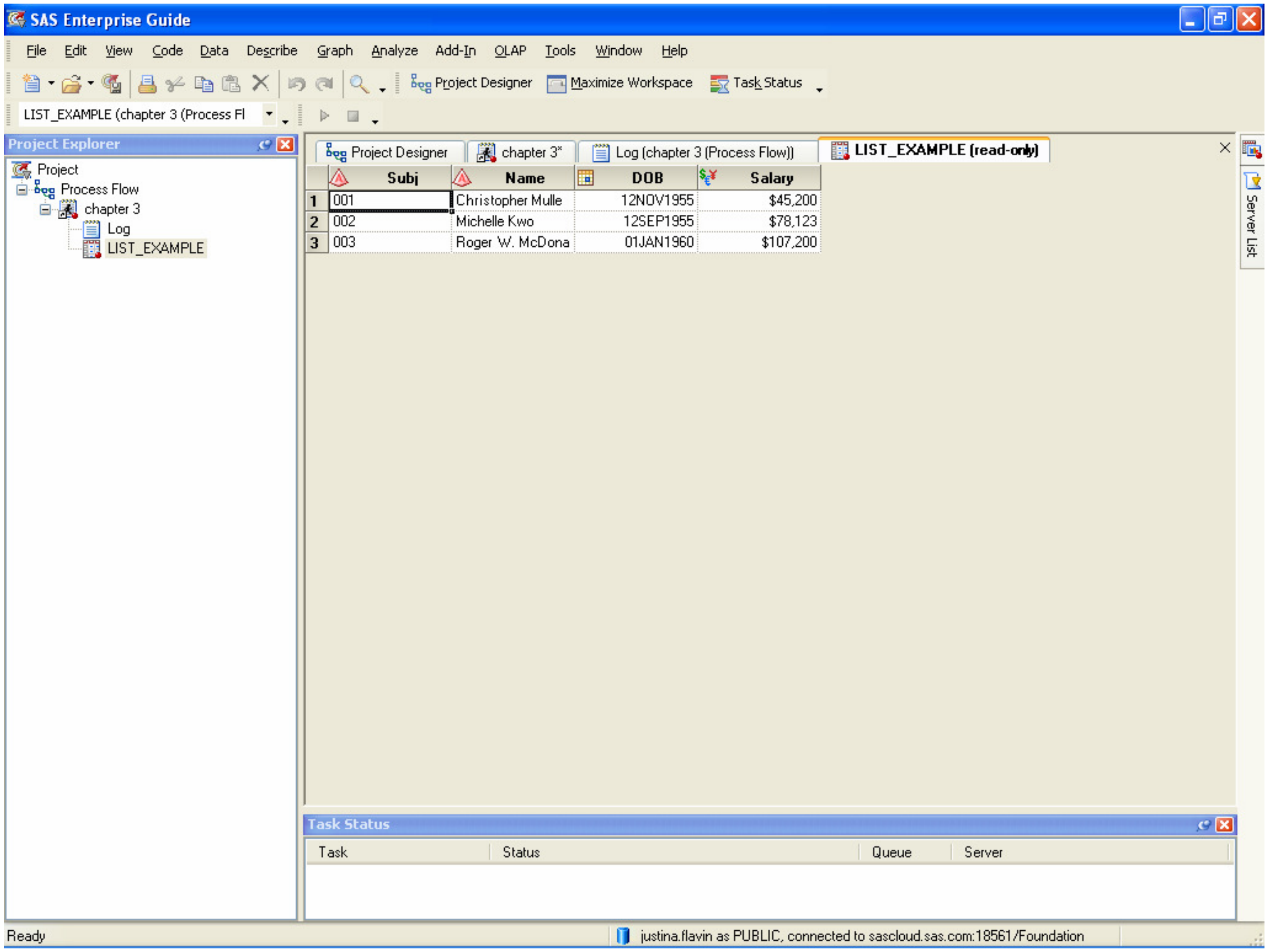
The format statement instructs SAS to store DOB with the date. format and Salary with the dollar. format in the data set. So these formats are permanent formats.

Notice the new icons on the DOB and Salary columns.

DOB has a calendar icon which indices that this variable is stored with a date format, while Salary has a financial icon to indicate that it is stored with a currency format.

Wouldn't it be nice to know what formats those are?

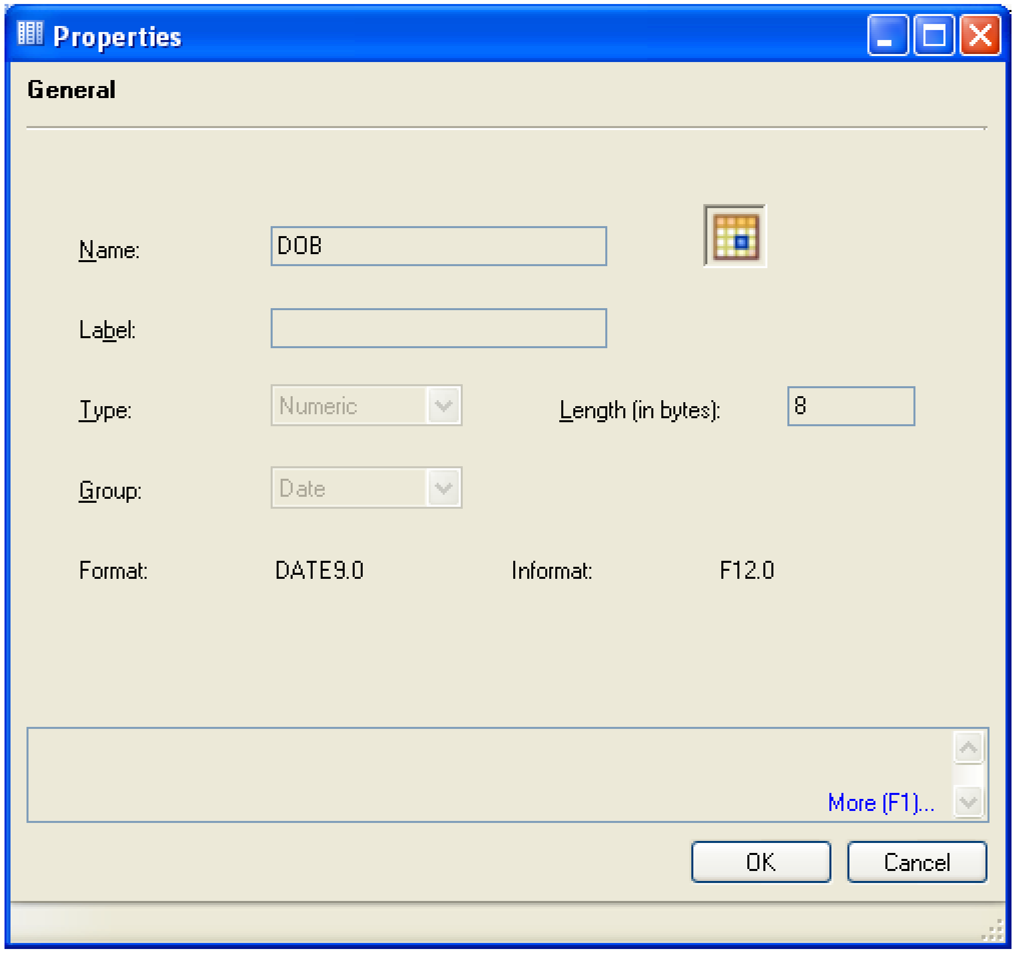
In the next slide we will learn how to do this.



	Subj	Name	DOB	Salary
1	001	Christopher Mulle	12NOV1955	$45,200
2	002	Michelle Kwo	12SEP1955	$78,123
3	003	Roger W. McDona	01JAN1960	$107,200

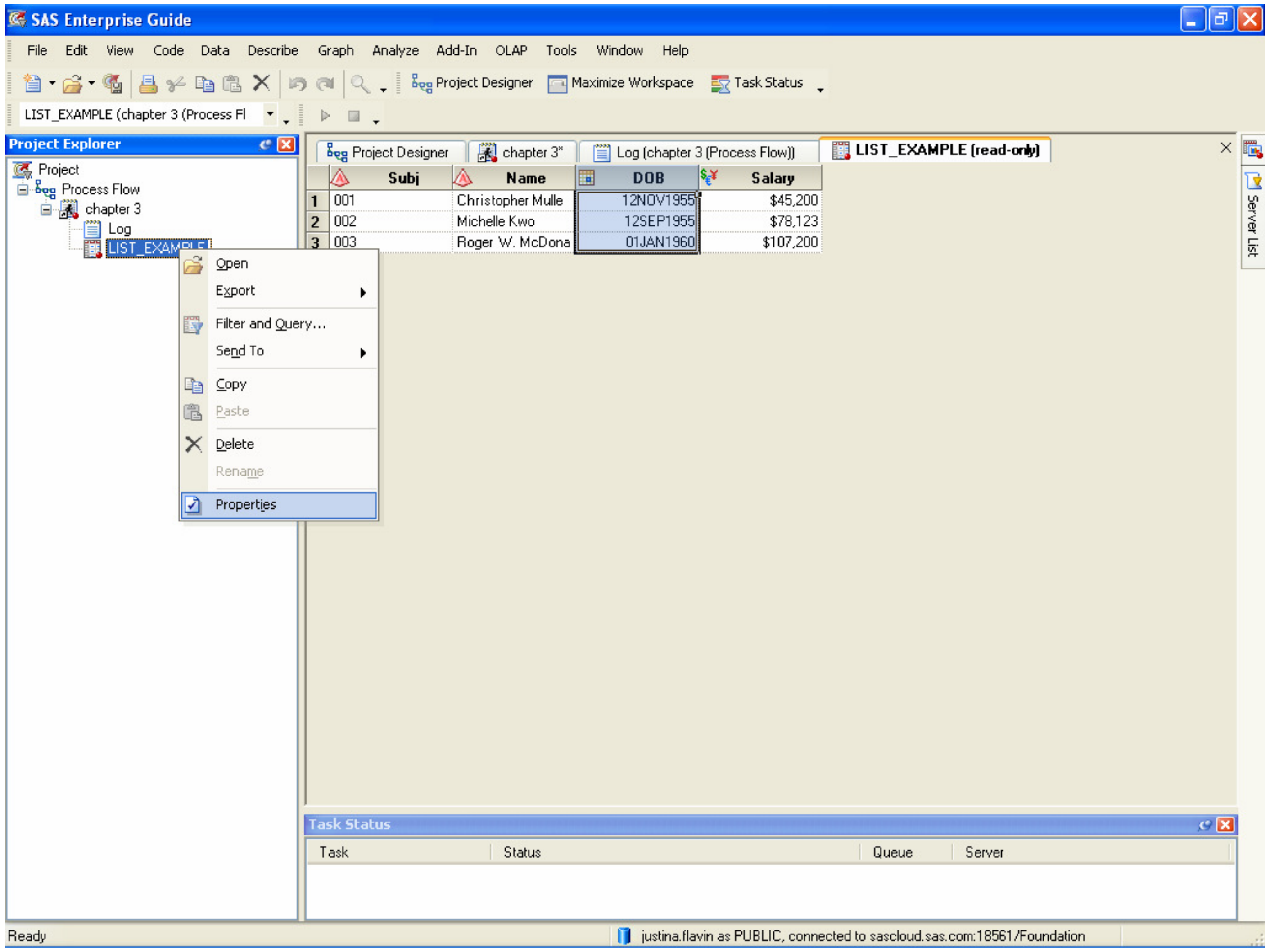
There are two ways to view the format information.

One way is to hover over the variable name, right click and then select Properties on the floating menu. This brings up a floating Properties box. Now we can see that DOB is stored with the DATE9. format.



Name:	DOB		
Label:			
Type:	Numeric	Length (in bytes):	8
Group:	Date		
Format:	DATE9.0	Informat:	F12.0

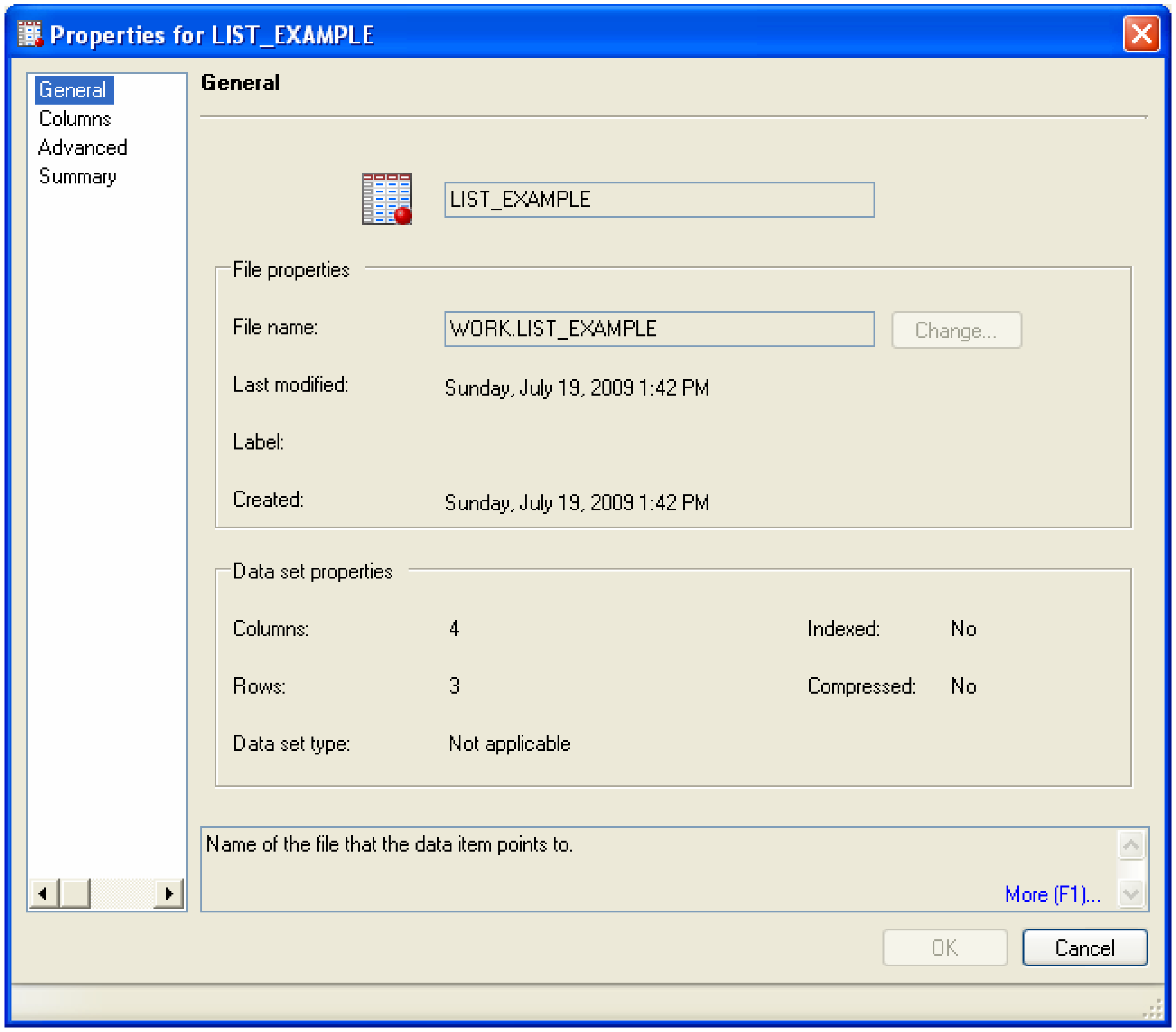
Another way to view formats is to highlight the data set name in the Project Explorer window, right click and then select Properties on the floating menu.



	Subj	Name	DOB	Salary
1	001	Christopher Mulle	12NOV1955	$45,200
2	002	Michelle Kwo	12SEP1955	$78,123
3	003	Roger W. McDona	01JAN1960	$107,200

Task	Status	Queue	Server

This brings up a floating box containing information about the entire data set.



General

LIST_EXAMPLE

File properties

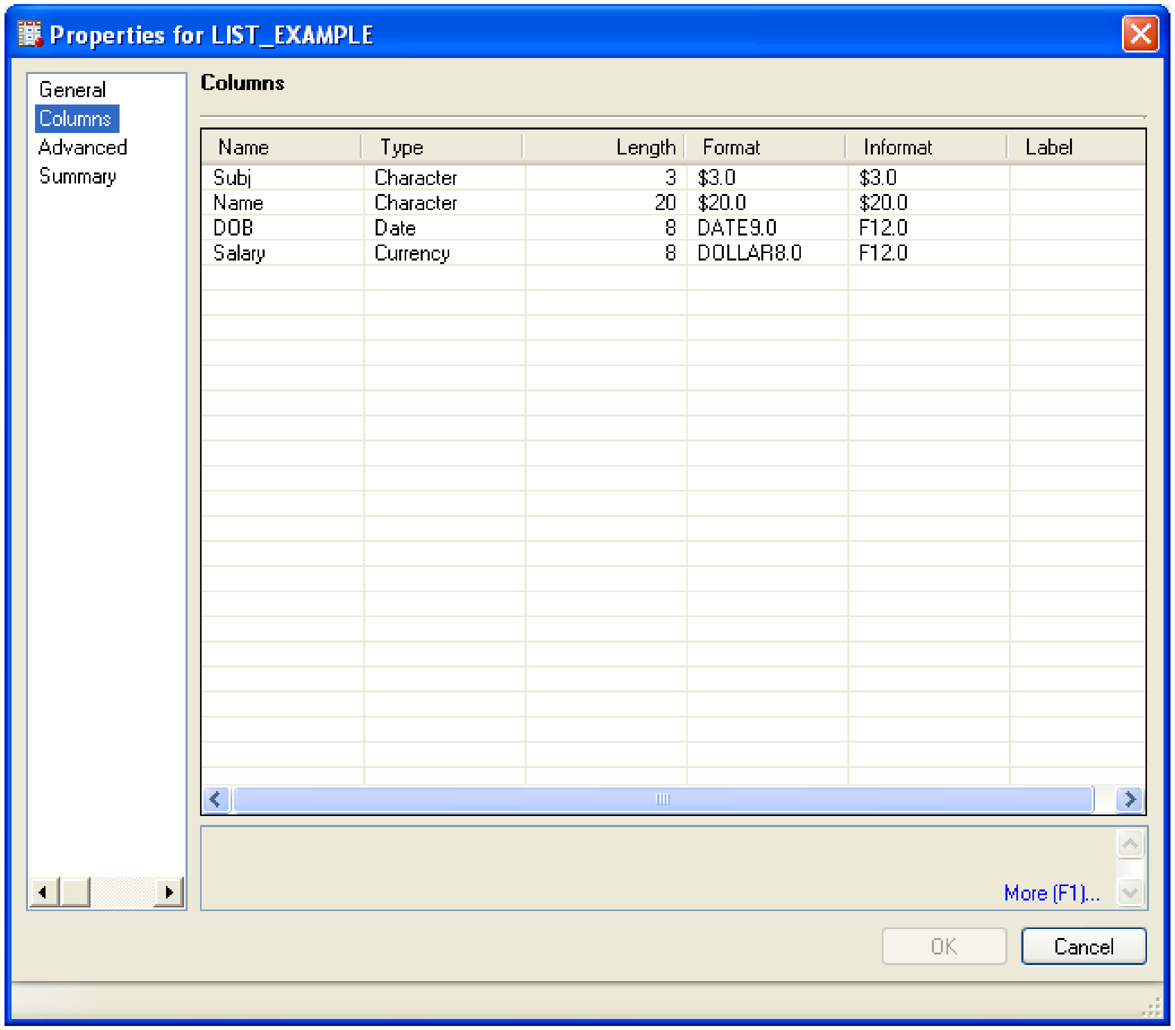
File name:	WORK.LIST_EXAMPLE
Last modified:	Sunday, July 19, 2009 1:42 PM
Label:	
Created:	Sunday, July 19, 2009 1:42 PM

Data set properties

Columns:	4	Indexed:	No
Rows:	3	Compressed:	No
Data set type:	Not applicable		

Name of the file that the data item points to.

Select **Columns** on the left hand side to view detail information about the variables in the data set.



Columns

Name	Type	Length	Format	Informat	Label
Subj	Character	3	$3.0	$3.0	
Name	Character	20	$20.0	$20.0	
DOB	Date	8	DATE9.0	F12.0	
Salary	Currency	8	DOLLAR8.0	F12.0	

The next example illustrates reading the same data file (list.csv) by using an informat statement.

3.13 Supplying an INFORMAT Statement with List Input

The general syntax of an informat statement is:

```
informat variablename1 informatname1 variablename2 $informatname2;
```

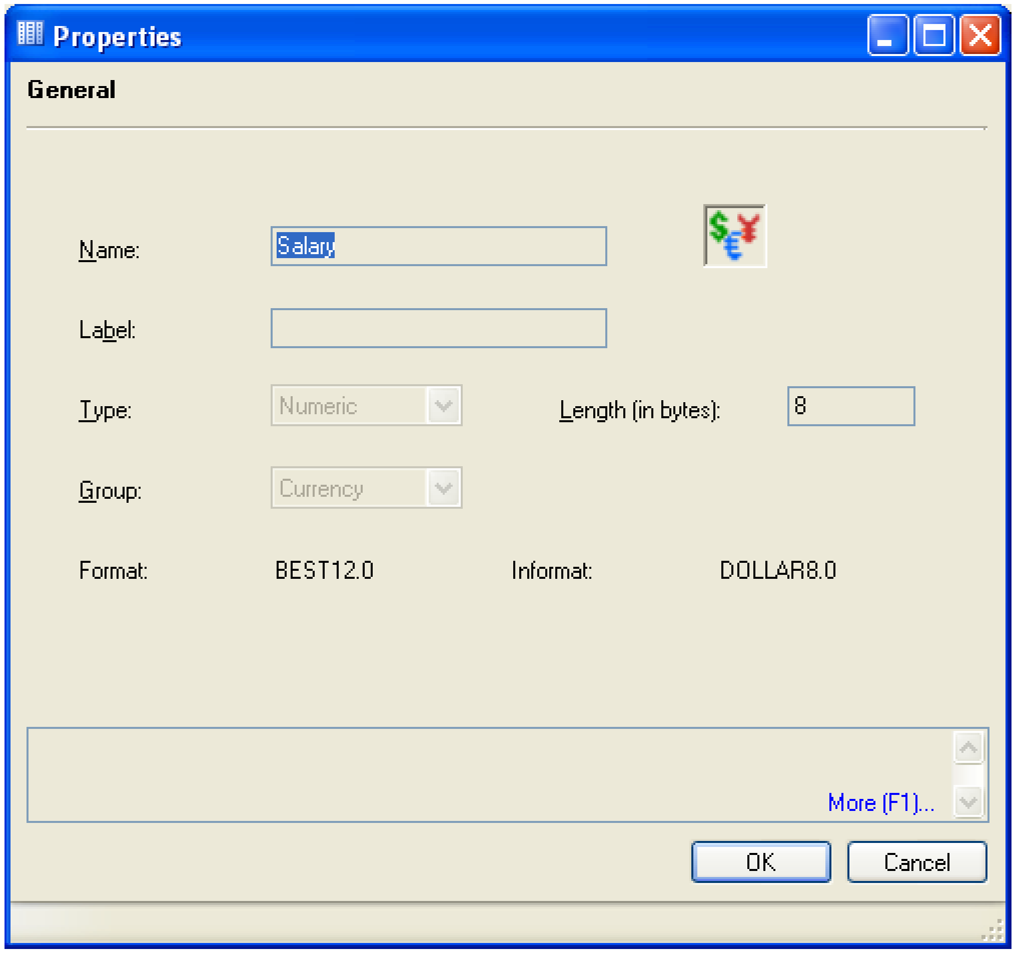
The informat statement transfers the informat information from the input statement, thus simplifying the input statement.

The informat statement MUST precede an input statement in the data step, since SAS processes each line of code sequentially and uses that information to set the variable names and variable attributes prior reading in the data.

To illustrate this concept, run the following code which will create a (valid) SAS data set having no observations but having variables with attributes as defined on the informat statement:

```
data informat_only;
  informat Subj        $3.
           Name       $20.
           DOB   mmddyy10.
           Salary dollar8.;
run;
```

Looking at the detail information for Salary, we can see that SAS has defined a variable with the DOLLAR8. informat as specified on the informat statement. There is no data in the variable at the moment, but when SAS reaches the input statement in the data step, it knows to use the DOLLAR8. informat with that data. Another way to think of this is that when SAS reads the informat statement, it sets up the variable "bins" for collecting the data that will be read in on the input statement. So for this reason, the informat statement MUST precede an input statement.



Name: Salary

Label:

Type: Numeric

Length (in bytes): 8

Group: Currency

Format: BEST12.0

Informat: DOLLAR8.0

Here is the complete example that includes the input statement.

```
*Program 3-12 Supplying an INFORMAT with List Input - page 45;
data list_example;
  informat Subj       $3.
           Name      $20.
           DOB   mmddyy10.
           Salary dollar8.
   infile "/courses/u_ucsd.edu1/i_536036/c_629/list.csv" dsd;
   input Subj
         Name
         DOB
         Salary;
   format DOB date9. Salary dollar8.;
run;
```

Notice that with the addition of the informat statement the input statement simply lists the variable names on it.

Again, the data set generated should be identical to the one in the previous example.

So as you can see, there is more than one way to write the code to read in the same data file.

Either way is acceptable, and one is not better than the other.

3.14 Using List Input with Embedded Delimiters

To use an informat with list input on data having embedded blanks, an ampersand “&” (also know as an informat modifier) precedes the informat name.

The informat modifier tells SAS that it should only treat two (or more) consecutive blanks as a delimiter.

This also means that any variables for which you are using the “&” input modifier must have at least two blanks following the variable data field.

Syntax:

```
input variablename1 & informat. variablename2 : $informat. Variablename3 & $informat.;
```

Here is the example. The data are illustrated on page 46.

```
*Program 3-13 Demonstrating the ampersand modifier for list input - page 46;
data list_example;
  infile "/courses/u_ucsd.edu1/i_536036/c_629/list.txt";
  input Subj   :       $3.
        Name   &      $20.
         DOB    : mmddyy10.
       Salary :  dollar8.;
  format DOB date9. Salary dollar8.;
run;
```

Notice that “&” is used in place of “:” for the variable Name which has a blank in the data between the first and last name.

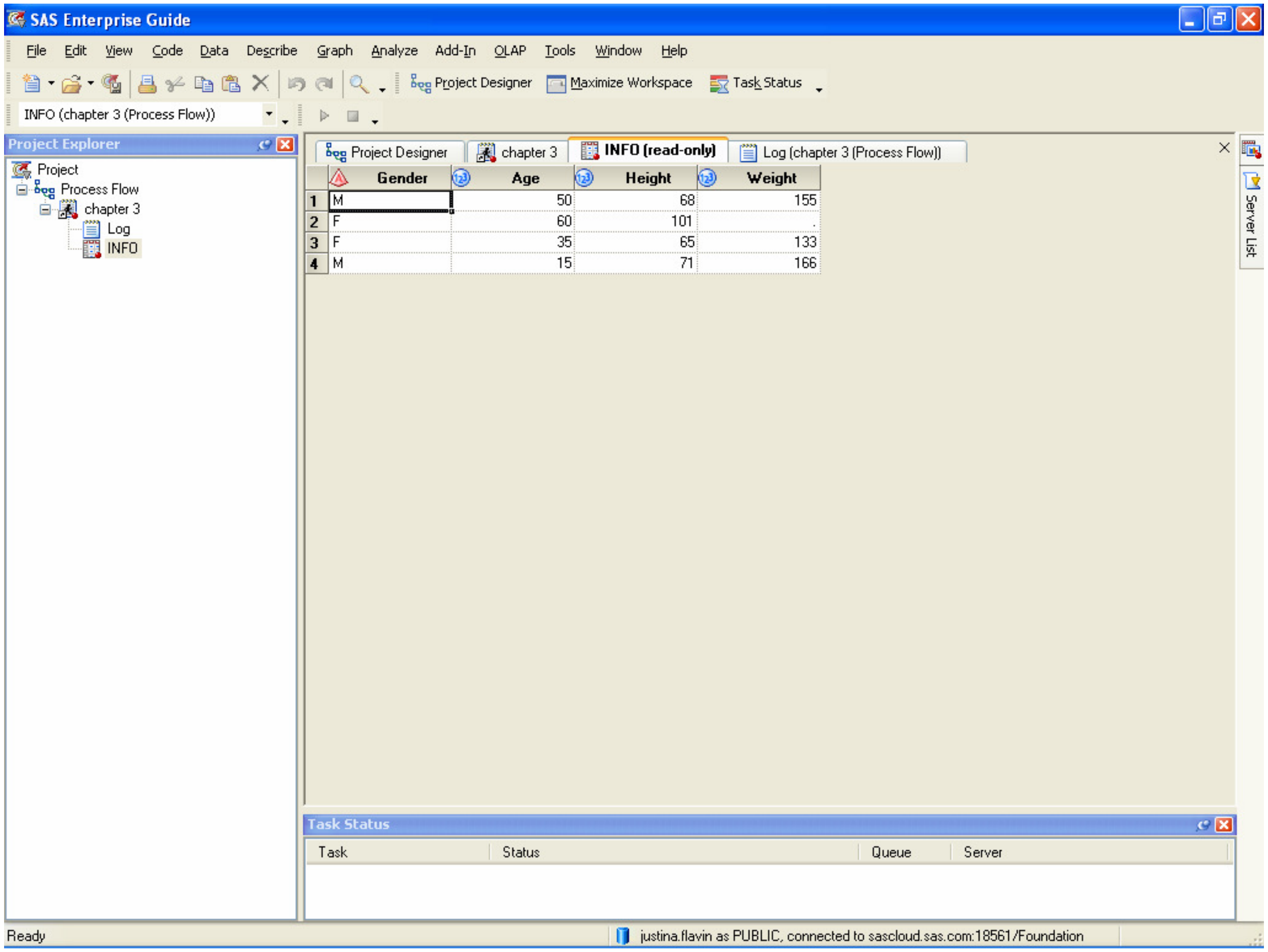
Now we will address the problem of missing values in data. Data having missing values is a common problem and the missing values can cause your data to be read into your data set incorrectly. In general, missing values cause the most problems with list input method, so a good rule of thumb is that if you have data with missing values, use the column input or formatted input methods.

3.3 Specifying Missing Values with List Input

Let's start with an example of trying to use list input to read in some data with a missing value. There is a missing value on the second line of data.

```
* example 1 - missing value in 2nd data line;
data info;
  input Gender $ Age Height Weight;
datalines;
M 50 68 155
F 60 101
M  65  72 220
F 35 65 133
M 15 71 166
;
run;
```

Notice that SAS created a data set, but it only contains 4 observations, not 5.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer
Maximize Workspace
Task Status
INFO (chapter 3 (Process Flow))
Project Explorer
Project
Process Flow
chapter 3
Log
INFO
Project Designer
chapter 3
INFO (read-only)
Log (chapter 3 (Process Flow))
Server List
Task Status
Task
Status
Queue
Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Gender	Age	Height	Weight
1	M	50	68	155
2	F	60	101	.
3	F	35	65	133
4	M	15	71	166

Looking at the log, there are no red ERROR lines, but we do see:

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

From a previous example, we know that this is indicative of a problem.

In this example, SAS is expecting to find 4 data values on each data line, since there are 4 variables specified on the input statement.

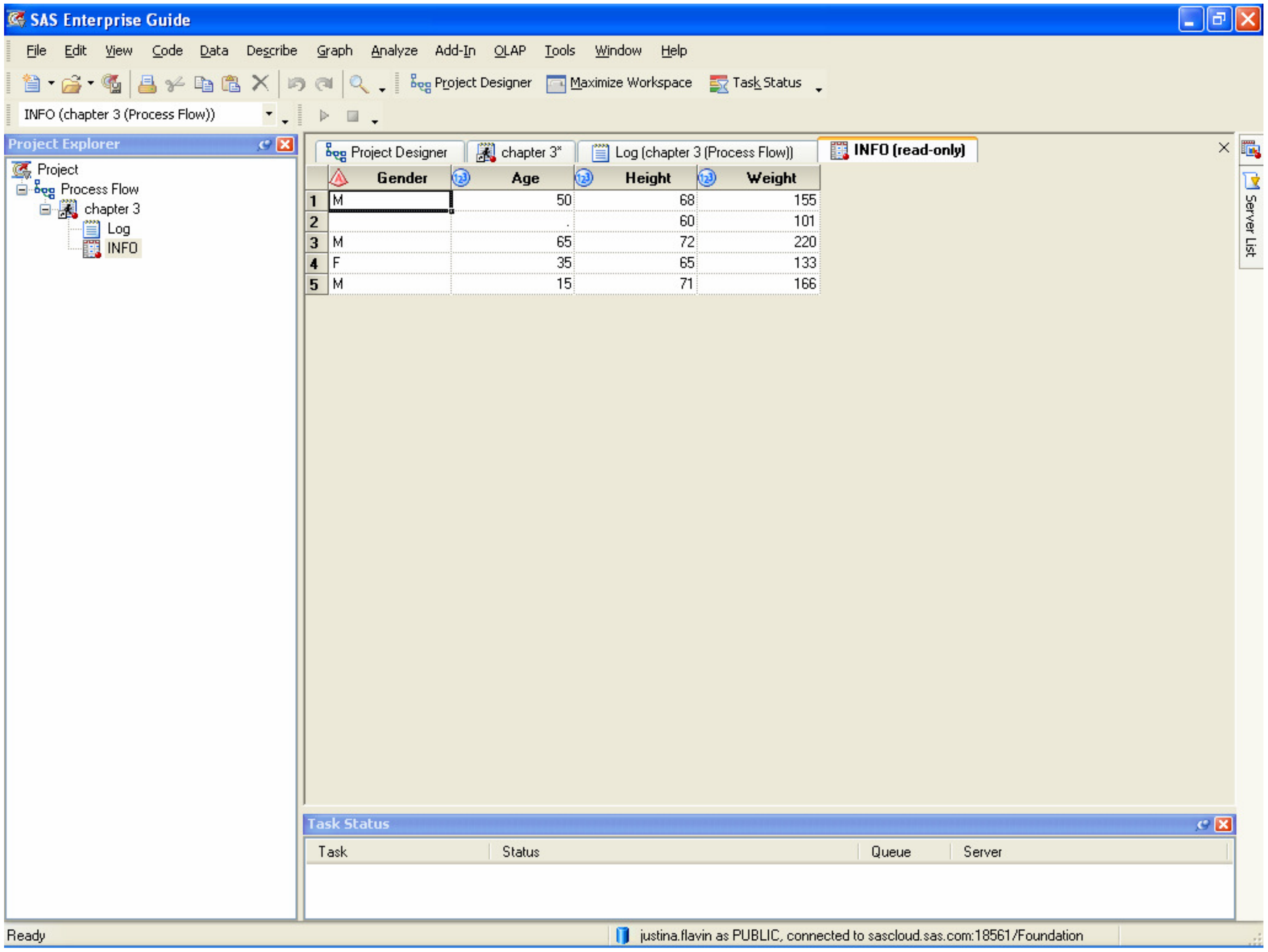
When SAS gets to the second data line, it only finds 3 values, which it assigns to the first three variables. Not finding a value for the last variable, Weight, it moves to the third data line. The first variable it encounters is the character ‘M’. Since Weight is a numeric value, it cannot assign that value to Weight. It ignores the rest of the values on the third data line and moves on to the fourth and fifth data lines which are correctly read into the data set.

The solution to this problem, is to add a period or dot as a placeholder for the missing value in the data.

By default, SAS recognises “.” as a missing value.

```
* example 1 solution - adding dots for the missing characters in 2nd data line;
data info;
  input Gender $ Age Height Weight;
datalines;
M 50 68 155
. . 60 101
M  65  72 220
F 35 65 133
M 15 71 166
;
run;
```

The resulting data set INFO is now correct.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer
Maximize Workspace
Task Status
INFO (chapter 3 (Process Flow))
Project Explorer
Project
Process Flow
chapter 3
Log
INFO
Project Designer
chapter 3*
Log (chapter 3 (Process Flow))
INFO (read-only)
Server List
Task Status
Task
Status
Queue
Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Gender	Age	Height	Weight
1	M	50	68	155
2		.	60	101
3	M	65	72	220
4	F	35	65	133
5	M	15	71	166

In this next example, we will address the issue of missing values at the end of a line of data.

We could add the ‘.’ character to every line of data, but suppose we had a few hundred data lines – this would be tedious.

An easier solution is to add an infile cards statement with the missover option. MISSOVER prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement. When an INPUT statement reaches the end of the current input data record, variables without any values assigned are set to missing.

Also note the use of the word “cards” on both the infile statement and before the data lines. “Cards” and “Datalines” are interchangable. In older versions of SAS, “datalines” did not exist, only “cards” did.

```
* example 2 solution - use the missover option;
data info;
  infile cards missover;
  input Gender $ Age Height Wt1 Wt2 Wt3;
cards;
M 50 68 155 153 154
F 23 60 101 95
M 65 72 220 210 215
F 35 65 133 132 134
M 15 71 166 166
;
run;
```

As mentioned previously, when working with data having missing values, a better solution is to use the column input or formatted input methods. This is because neither method uses a blank as a delimiter and both methods read whatever characters may be in the specified columns, so a blanks are read in as valid characters.

This also prevents you from having to go through your data and populate it with dots where there are missing values.

In this example, also note the use of both column input and formatted input. An input statement does not have to use one or the other exclusively, it is acceptable to use both.

```
* example 3 - use column or formatted input;
data info;
  infile cards missover;
  input Gender $1 Age 3-4 @6 Height 2. Wt1 9-11 Wt2 13-15 @17 Wt3 3.;
cards;
M 50 68 155 153 154
F    60 101  95
M    72 220 210 215
F 35 65 133 132 134
M 15 71 166 166
;
run;
```

Finally, here is some additional useful information when using input statements to create data sets.

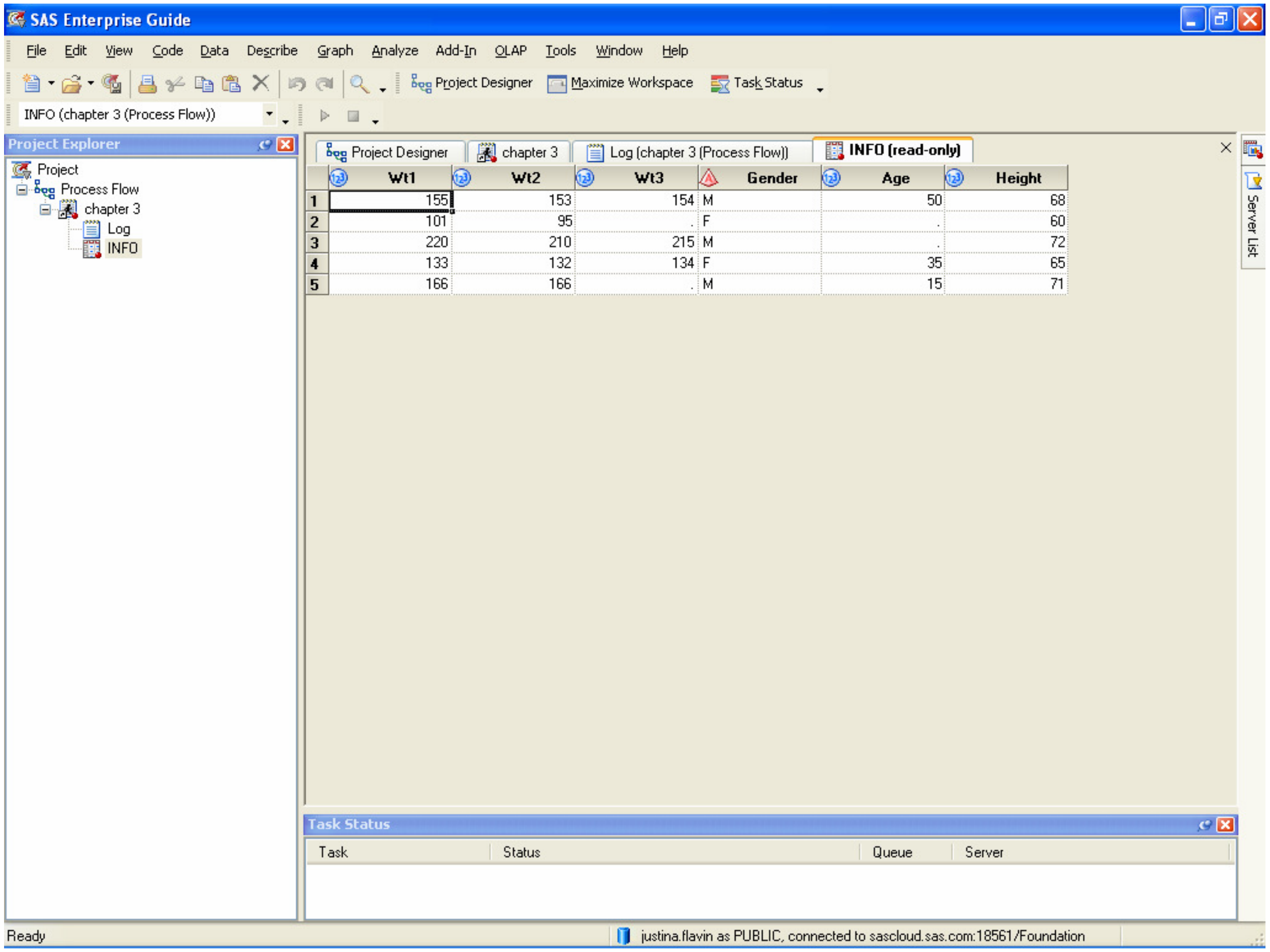
(1) You don't have to read in the data in the same order as it occurs in the raw file.

In this example, we read in the weight variables at the end of the data line before we read in Gender, Age, and Height.

```
data info;
  infile cards missover;
  input Wt1 9-11 Wt2 13-15 @17 Wt3 3. Gender $1 Age 3-4 @6 Height 2.;
cards;
M 50 68 155 153 154
F    60 101  95
M    72 220 210 215
F 35 65 133 132 134
M 15 71 166 166
;
run;
```

Notice that in the data set that was created, the variables are stored in the same order in which they were listed on the input statement, not the order in which they occurred in the data.

Generally, it is better to read in your variables in the same order in which they occur in the raw data, but there may be times when you wish to reorder the data in your data set, so this is a way to do so.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
INFO (chapter 3 (Process Flow))
Project Explorer
Project
Process Flow
chapter 3
Log
INFO
Project Designer chapter 3 Log (chapter 3 (Process Flow)) INFO (read-only)
Server List
Task Status
Task Status Queue Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Wt1	Wt2	Wt3	Gender	Age	Height
1	155	153	154	M	50	68
2	101	95	.	F	.	60
3	220	210	215	M	.	72
4	133	132	134	F	35	65
5	166	166	.	M	15	71

(2) You can read and re-read data columns.

In this example, we read all of the variables in (as expected), but then go back through the data lines and reread some of the columns to create some additional (and admittedly, strange) new variables.

```
data info;
  infile cards missover;
  input Gender $1 Age 3-4 @6 Height 2. Wt1 9-11 Wt2 13-15 @17 Wt3 3. sexage $ 1-4 @13 wt26 3.;
cards;
M 50 68 155 153 154
F    60 101  95
M    72 220 210 215
F 35 65 133 132 134
M 15 71 166 166
;
run;
```

This technique could be useful if, for instance, you had a a character string that contained a “long identifier” for an individual, but a portion of that string could also be used as a “short identifier”.

So you would read the entire value into your long identifier variable, and then re-read only the columns needed for your short identifier variable.

An example of this would be a person’s social security number.

(3) Not all data columns need to be read into the data set

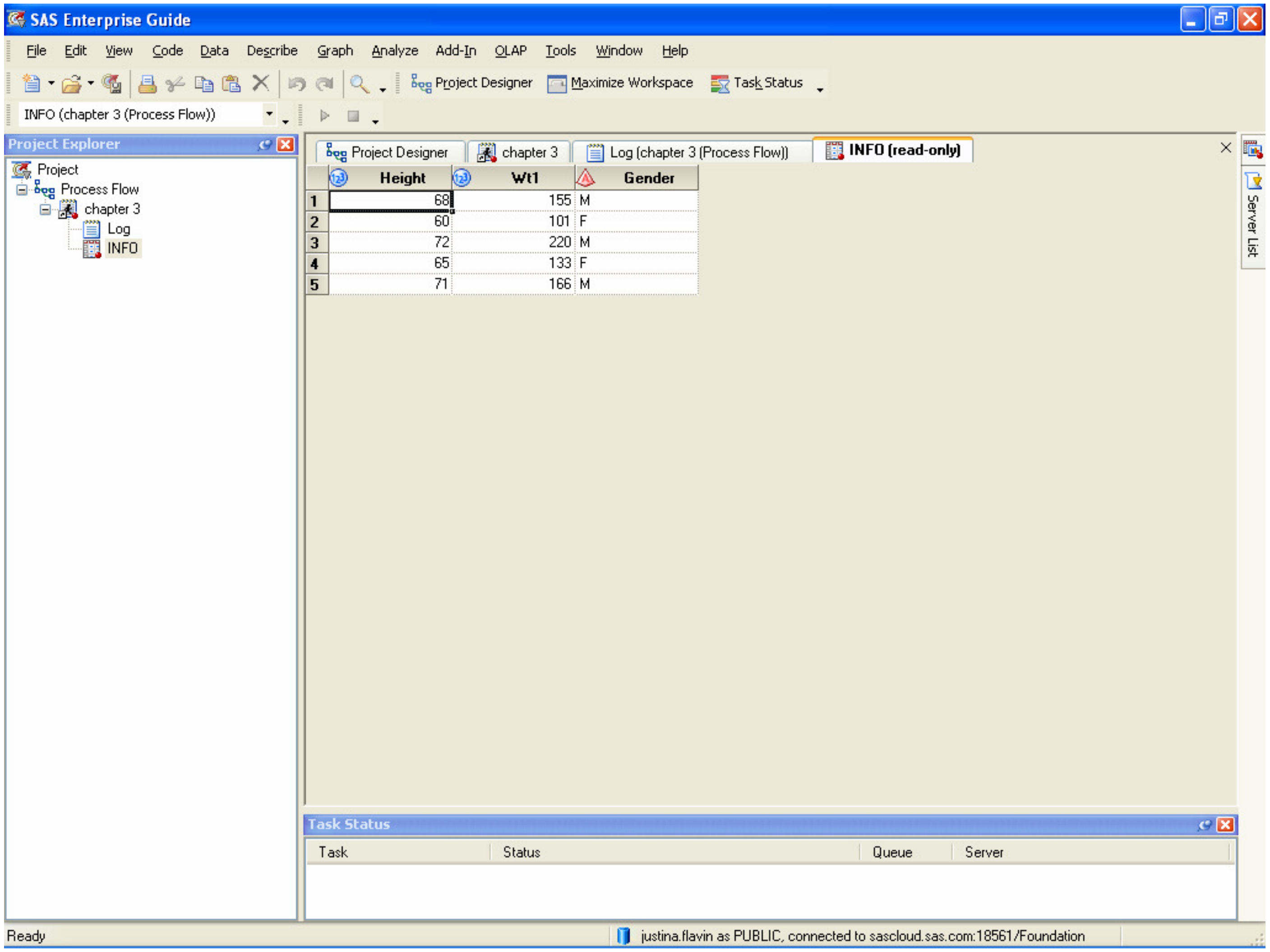
This example illustrates reading in only 3 of the variables on the data lines – Gender, Height, and Wt1.

The data in all other columns is ignored.

If you have a data file with variables that you do not need, there is no need to waste time writing the code to read these into your data set. This will also free up computational and data set resources.

```
data info;
  infile cards missover;
  input @6 Height 2. Wt1 9-11 Gender $1 ;
cards;
M 50 68 155 153 154
F    60 101  95
M    72 220 210 215
F 35 65 133 132 134
M 15 71 166 166
;
run;
```

Here is the resulting data set.


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer
Maximize Workspace
Task Status
INFO (chapter 3 (Process Flow))
Project Explorer
Project
Process Flow
chapter 3
Log
INFO
Project Designer
chapter 3
Log (chapter 3 (Process Flow))
INFO (read-only)
Server List
Task Status
Task
Status
Queue
Server
Ready
justina.flavin as PUBLIC, connected to sascloud.sas.com:18561/Foundation

	Height	Wt1	Gender
1	68	155	M
2	60	101	F
3	72	220	M
4	65	133	F
5	71	166	M