Writing Observations Explicitly

To override the default way in which the DATA step writes observations to output, you can use an OUTPUT statement in the DATA step. Placing an explicit OUTPUT statement in a DATA step overrides the implicit output at the end of the DATA step. The observations are added to a data set only when the explicit OUTPUT statement is executed.

Syntax, OUTPUT statement:

OUTPUT <*SAS-data-set(s)*>**;**

SAS-data-set(s) names the data set or data sets to which the observation is written. All data set names that are specified in the OUTPUT statement must also appear in the DATA statement.

Using an OUTPUT statement without a following data set name causes the current observation to be written to all data sets that are specified in the DATA statement.

With an OUTPUT statement, your program now writes a single observation to output—observation 5. For more information on subsetting IF statements, see "Using a Subsetting IF Statement" on page 146.

```
data work.usa5;
  set cert.usa(keep=manager wagerate);
  if _n_=5 then output;
run;
proc print data=work.usa5;
run;
```

Figure 4.5 *Single Observation*

Obs	Dept	WageCat	WageRate	Manager	JobType
1	ADM20	S	4522.5	Coxe	240

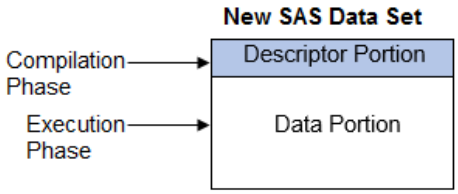
Suppose your DATA statement contains two data set names, and you include an OUTPUT statement that references only one of the data sets. The DATA step creates both data sets, but only the data set that is specified in the OUTPUT statement contains output. For example, the program below creates two temporary data sets, Empty and Full. The result of this DATA step is that the data set Empty is created but contains no observations, and the data set Full contains all of the observations from Cert.Usa.

```
data empty full;
  set cert.usa;
  output full;
run;
```

How SAS Processes Programs

When you submit a DATA step, SAS processes the DATA step and creates a new SAS data set. A SAS DATA step is processed in two phases:

Figure 7.1 *DATA Step Process*



When you submit a DATA step for execution, SAS checks the syntax of the SAS statements and compiles them. In this phase, SAS identifies the type and length of each new variable, and determines whether a variable type conversion is necessary for each subsequent reference to a variable. During the compilation phase, SAS creates the following items:

- program data vector (PDV)
- descriptor information

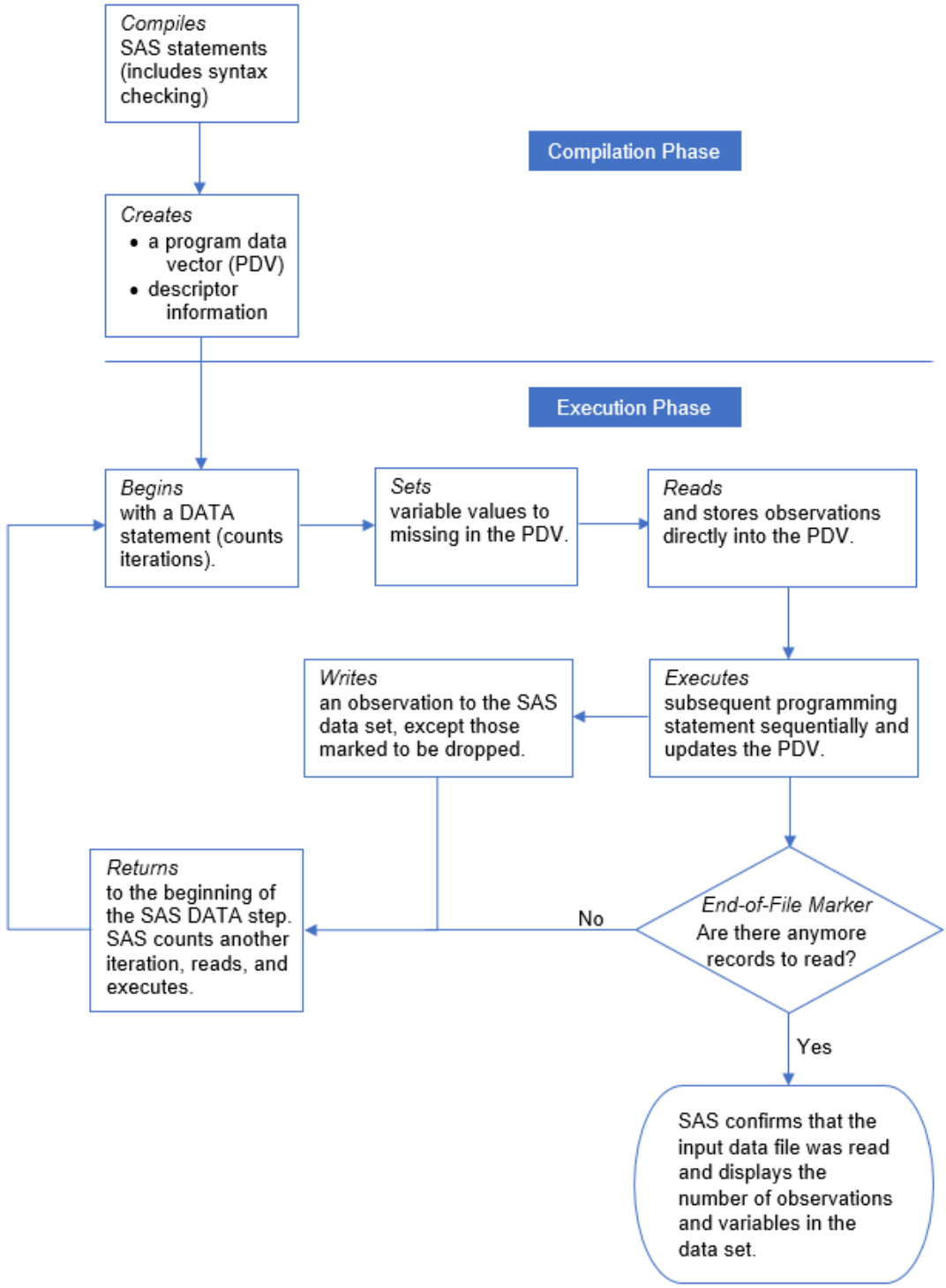
When the compilation phase is complete, the descriptor portion of the new data set is created.

By default, a simple DATA step iterates once for each observation that is being created. The flow of action in the execution phase of a simple DATA step is described as follows:

1. The DATA step begins with a DATA statement. Each time the DATA statement executes, a new iteration of the DATA step begins, and the _N_ automatic variable is incremented by 1. The _N_ automatic variable represents the number of times the DATA step has iterated.
2. SAS sets the newly created program variables to missing in the program data vector (PDV).
3. SAS reads an observation from a SAS data set directly into the PDV. You can use MERGE, SET, MODIFY, or UPDATE statement to read a record.
4. SAS executes any subsequent programming statements sequentially and updates the PDV.
5. When SAS executes the last statement in the DATA step, all values (except temporary variables and those marked to be dropped) are written as a single observation to the data set. Note that variables that you read with a SET, MERGE, MODIFY, or UPDATE statement are not reset to missing here.
6. SAS counts another iteration, reads the next observation, and executes the subsequent programming statements for the current observation.
7. The DATA step terminates when SAS encounters the end-of-file in a SAS data set.

Figure 7.2 shows the general flow of DATA step processing for reading raw data.

Figure 7.2 *Compilation and Execution Phases of DATA Step Processing*



Compilation Phase

Program Data Vector (PDV)

The PDV is a logical area in memory where SAS builds a data set, one observation at a time. When a program executes, SAS reads data values or creates them by executing SAS language statements. The data values are assigned to the appropriate variables in the PDV. From here, SAS writes the values to a SAS data set as a single observation.

Along with data set variables and computed variables, the PDV contains these automatic variables:

- the _N_ variable, which counts the number of times the DATA step iterates.
- the _ERROR_ variable, which signals the occurrence of an error caused by the data during execution. The value of _ERROR_ is 0 when there are no errors. When an error occurs, whether one error or multiple errors, the value is set to 1. The default value is 0.

Note: SAS does not write these variables to the output data set.

Syntax Checking

During the compilation phase, SAS scans each statement in the DATA step, looking for syntax errors. Here are examples:

- missing or misspelled keywords
- invalid variable names
- missing or invalid punctuation
- invalid options

Data Set Variables

As the SET statement compiles, a slot is added to the PDV for each variable in the new data set. Generally, variable attributes such as length and type are determined the first time a variable is encountered.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.3 *Program Data Vector*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit			_N_	_ERROR_

Any variables that are created with an assignment statement in the DATA step are also added to the PDV. For example, the assignment statement below creates two variables,

Total and SalePrice. As the statement is compiled, the variable is added to the PDV. The attributes of the variable are determined by the expression in the statement. Because the expression contains an arithmetic operator and produces a numeric value, Total and SalePrice are defined as numeric variables and are assigned the default length of 8.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.4 *Program Data Vector*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_

Descriptor Portion of the SAS Data Set

The descriptor portion is information that SAS creates and maintains about each SAS data set, including data set attributes and variable attributes. Here are examples:

- the name of the data set and its member type
- the date and time that the data set was created
- the names, data types (character or numeric), and lengths of the variables

Extended attribute descriptor information is defined by the user and includes the name of the attribute, the name of the variable, and the value of the attribute. The descriptor information also contains information about extended attributes (if defined in a data set). You can use the CONTENTS procedure to display descriptor information.

```
proc contents data=work.update;
run;
```

Figure 7.5 *CONTENTS Procedure Output: Data Set Descriptor Specifics*

Data Set Name	WORK.UPDATE	**Observations**	9
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	07/25/2018 15:13:34	**Observation Length**	64
Last Modified	07/25/2018 15:13:34	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1021
Obs in First Data Page	9
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\Users\Student1\SAS Temporary Files_TD18132_D7C049_\update.sas7bdat
Release Created	9.0401M4
Host Created	X64_10PRO
Owner Name	Student1
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes				
#	**Variable**	**Type**	**Len**	**Format**
4	BackOrd	Num	8	
5	CostPerUnit	Num	8	DOLLAR6.2
2	IDnum	Char	5	
3	InStock	Num	8	
1	Item	Char	13	
7	SalePrice	Num	8	DOLLAR6.2
6	Total	Num	8	

At this point, the data set contains the seven variables that are defined in the input data set and in the assignment statement. _N_ and _ERROR_ are not written to the data set. There are no observations because the DATA step has not yet executed. During execution, each raw data record is processed and is then written to the data set as an observation.

Execution Phase

Initializing Variables

At the beginning of the execution phase, the value of _N_ is 1. Because there are no data errors, the value of _ERROR_ is 0.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.6 *Program Data Vector: Initializing Variables*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
		.	.	.	.	.	1	0

The remaining variables are initialized to missing. Missing numeric values are represented by periods, and missing character values are represented by blanks.

SET Statement

The SET statement identifies the location of the input data set. Columns are added to the PDV in the order in which they appear in the input table. Attributes are inherited from the input table.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Sequentially Process Statements

After the SET statement, SAS executes the remaining statements sequentially and updates the values in the PDV.

```
data work.update;
  set cert.invent;
  Total=instock+backord;                    /*#1*/
  SalePrice=(CostPerUnit*0.65)+CostPerUnit; /*#2*/
  format CostPerUnit SalePrice dollar6.2;
run;
```

1 SAS processes the first assignment statement to create the new variable, Total. The values of InStock and BackOrd are added together to create a value for Total. See Figure 7.7 below for a visual representation of how the PDV processes the first assignment statement.

2 SAS processes the second assignment statement to create the new variable, SalePrice. The value of CostPerUnit is multiplied by 0.65, and the resulting value is added to the value of CostPerUnit to create a value for SalePrice. See Figure 7.8 below for a visual representation of how the PDV processes the second assignment statement.

Figure 7.7 *PDV: Create a New Variable, Total*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
Bird Feeder	LG088	3	20	$5.00	23	.	1	0

Figure 7.8 *PDV: Create a New Variable, SalePrice*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
Bird Feeder	LG088	3	20	$5.00	23	$8.25	1	0

The formats for each variable are applied before SAS adds the values to the PDV.

End of the DATA Step

At the end of the DATA step, several actions occur. First, the values in the PDV are written to the output data set as the first observation.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.9 *Program Data Vector and Output Data Set*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
Bird Feeder	LG088	3	20	$5.00	23	$8.25	1	0

SAS Data Set Work.Update Output

Obs	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
1	Bird Feeder	LG088	3	20	$5.00	23	$8.25

Next, control returns to the top of the DATA step, and the value of _N_ increments from 1 to 2. Finally, the variable values in the PDV are reset to missing. Notice that the automatic variable _ERROR_ is reset to 0 if necessary.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.10 *Program Data Vector and Output Data Set*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
Bird Feeder	LG088	3	20	$5.00	.	.	2	0

SAS Data Set Work.Update Output

Obs	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
1	Bird Feeder	LG088	3	20	$5.00	23	$8.25

Iterations of the DATA Step

You can see that the DATA step works like a loop, repetitively executing statements to read data values and create observations one by one. At the beginning of the second iteration, the value of _N_ is 2, and _ERROR_ is still 0. Each loop (or cycle of execution) is called an *iteration*.

Figure 7.11 *Iterations of the DATA Step*

```
data work.update;
     set cert.invent;
     Total=instock+backord;
     SalePrice=(CostPerUnit*0.65)+CostPerUnit;
     format CostPerUnit SalePrice dollar6.2;
run;
```

As the SET statement executes for the second time, the values from the second record are read from the input table into the PDV.

Figure 7.12 *Program Data Vector and Output Data Set*

Program Data Vector

Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice	_N_	_ERROR_
6 Glass Mugs	SB082	6	12	$1.50	.	.	2	0

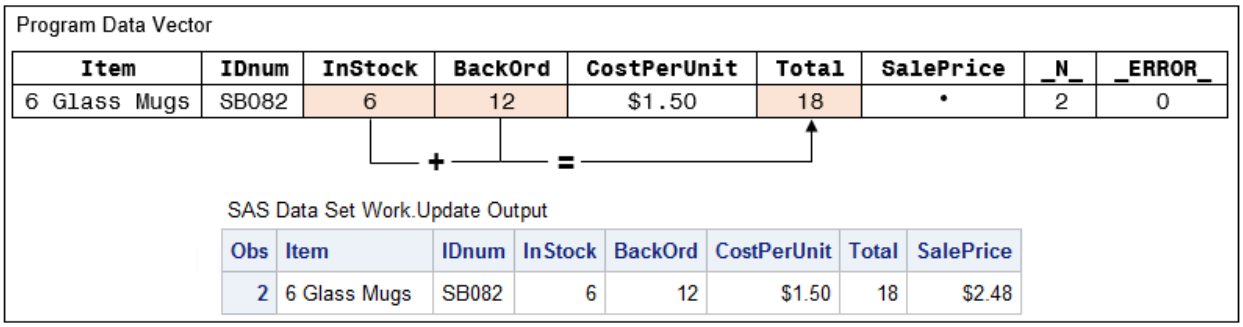
SAS Data Set Work.Update Output

Obs	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
2	6 Glass Mugs	SB082	6	12	$1.50	18	$2.48

Next, the value for Total is calculated based on the current values for InStock and BackOrd.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

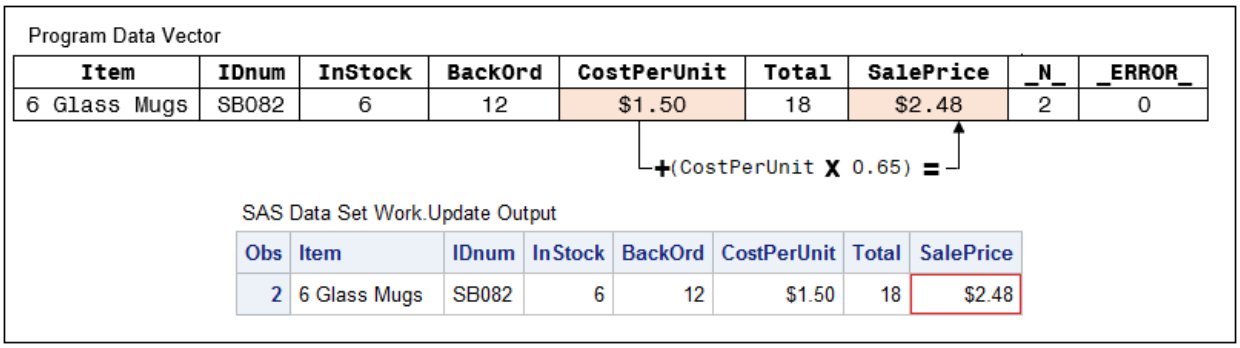
Figure 7.13 *Program Data Vector and Output Data Set*



Next, the value for SalePrice is calculated based on the values for CostPerUnit, multiplied by 0.65, and added to the value of CostPerUnit.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.14 *Program Data Vector and Output Data Set*



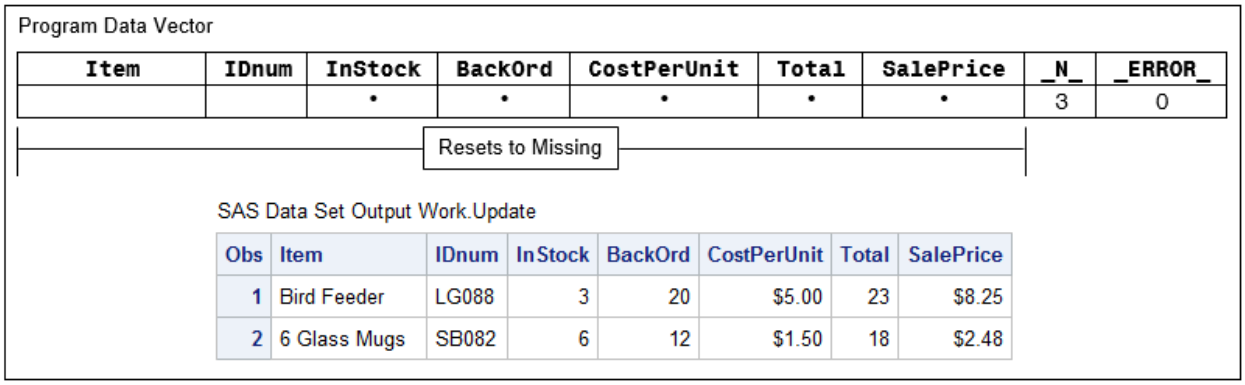
The RUN statement indicates the end of the DATA step loop. At the bottom of the DATA step, the values in the PDV are written to the data set as the second observation.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Next, the value of _N_ increments from 2 to 3, control returns to the top of the DATA step, and the values for Item, IDnum, InStock, BackOrd, CostPerUnit, Total, and SalePrice are reset to missing.

```
data work.update;
  set cert.invent;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar6.2;
run;
```

Figure 7.15 *Program Data Vector and Output Data*



Obs	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
1	Bird Feeder	LG088	3	20	$5.00	23	$8.25
2	6 Glass Mugs	SB082	6	12	$1.50	18	$2.48

When PROC IMPORT reads raw data, SAS sets the value of each variable in the DATA step to missing at the beginning of each cycle of execution, with these exceptions:

- variables that are named in a RETAIN statement
- variables that are created in a sum statement
- automatic variables

In contrast, when reading variables from a SAS data set, SAS sets the values to missing only before the first cycle of execution of the DATA step. Therefore, the variables retain their values until new values become available (for example, through an assignment statement or through the next execution of a SET or MERGE statement). Variables that are created with options in a SET or MERGE statement also retain their values from one cycle of execution to the next.

End-of-File Marker

The execution phase continues in this manner until the end-of-file marker is reached in the input data file. When there are no more records in the input data file to be read, the data portion of the new data set is complete and the DATA step stops.

This is the output data set that SAS creates:

Figure 7.16 *SAS Data Set Work.Update*

	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
1	Bird Feeder	LG088	3	20	$5.00	23	$8.25
2	6 Glass Mugs	SB082	6	12	$1.50	18	$2.48
3	Glass Tray	BQ049	12	6	$2.50	18	$4.13
4	Padded Hangrs	MN256	15	6	$2.00	21	$3.30
5	Jewelry Box	AJ498	23	0	$6.50	23	$10.73
6	Red Apron	AQ072	9	12	$1.00	21	$1.65
7	Crystal Vase	AQ672	27	0	$7.00	27	$11.55
8	Picnic Basket	LS930	21	0	$3.50	21	$5.78
9	Brass Clock	AN910	2	10	$11.50	12	$18.98

End of the Execution Phase

At the end of the execution phase, the SAS log confirms that the input data file was read, and it displays the number of observations and variables in the data set.

Log 7.1 *SAS Log*

```
NOTE: There were 9 observations read from the data set
      CERT.INVENT.
NOTE: The data set WORK.UPDATE has 9 observations and 7
      variables.
```

Recall that you can display the data set with the PRINT procedure.

```
proc print data=work.update;
run;
```

Output 7.1 *Output from the PRINT Procedure*

Obs	Item	IDnum	InStock	BackOrd	CostPerUnit	Total	SalePrice
1	Bird Feeder	LG088	3	20	$5.00	23	$8.25
2	6 Glass Mugs	SB082	6	12	$1.50	18	$2.48
3	Glass Tray	BQ049	12	6	$2.50	18	$4.13
4	Padded Hangrs	MN256	15	6	$2.00	21	$3.30
5	Jewelry Box	AJ498	23	0	$6.50	23	$10.73
6	Red Apron	AQ072	9	12	$1.00	21	$1.65
7	Crystal Vase	AQ672	27	0	$7.00	27	$11.55
8	Picnic Basket	LS930	21	0	$3.50	21	$5.78
9	Brass Clock	AN910	2	10	$11.50	12	$18.98

Debugging a DATA Step

Diagnosing Errors in the Compilation Phase

Errors that are detected during the compilation phase include these:

- misspelled keywords and data set names
- unbalanced quotation marks
- invalid options

During the compilation phase, SAS can interpret some syntax errors (such as the keyword DATA misspelled as DAAT). If it cannot interpret the error, SAS does the following:

- prints the word ERROR followed by an error message in the SAS log
- compiles but does not execute the step where the error occurred, and prints the following message:

  ```
  NOTE: The SAS System stopped processing this step because of errors.
  ```

Some errors are explained fully by the message that SAS prints; other error messages are not as easy to interpret. For example, because SAS statements are free-format, when you fail to end a SAS statement with a semicolon, SAS cannot detect the error.

Diagnosing Errors in the Execution Phase

When SAS detects an error in the execution phase, the following can occur, depending on the type of error:

- A note, warning, or error message is displayed in the SAS log.
- The values that are stored in the PDV are displayed in the SAS log.
- The processing of the step either continues or stops.

Debugging Data Errors

Recall that data errors occur when data values are not appropriate for the SAS statements that are specified in a program. SAS detects data errors during program execution. When a data error is detected, SAS continues to execute the program.

In general, SAS procedures analyze data, produce output, or manage SAS files. In addition, SAS procedures can be used to detect invalid data. In addition to the PRINT procedure showing missing values, the following procedures can be used to detect invalid data:

- PROC FREQ
- PROC MEANS

The FREQ procedure detects invalid character and numeric values by looking at distinct values. You can use PROC FREQ to identify any variables that were not given an expected value.

Syntax, FREQ procedure:

PROC FREQ DATA=*SAS-data-set* <*NLEVELS*>;
  TABLES *variable(s);*
RUN;

- The NLEVELS option displays a table that provides the number of distinct values for each variable that is named in the TABLES statement.
- The TABLES statement specifies the frequency tables to produce based on the number of variables that are specified.

In the following example, the data set contains invalid characters for the variables Gender and Age. PROC FREQ displays the distinct values of variables and is therefore useful for finding invalid values in data. You can use PROC FREQ with the TABLES statement to produce a frequency table for specific variables.

```
proc freq data=cert.pats;
      tables Gender Age;
run;
```

In the following figures, notice the valid (M and F) and invalid (G) values for Gender, and the valid and invalid (202) values for Age. In both the Gender and Age FREQ tables, data in one observation needs to be cleaned.

Output 7.2 *FREQ Procedure Output*

Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	10	66.67	10	66.67
G	2	13.33	12	80.00
M	3	20.00	15	100.00

Age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
16	1	6.67	1	6.67
18	1	6.67	2	13.33
39	2	13.33	4	26.67
40	1	6.67	5	33.33
42	1	6.67	6	40.00
48	1	6.67	7	46.67
56	1	6.67	8	53.33
57	1	6.67	9	60.00
59	1	6.67	10	66.67
60	1	6.67	11	73.33
63	1	6.67	12	80.00
64	1	6.67	13	86.67
116	1	6.67	14	93.33
202	1	6.67	15	100.00

The MEANS procedure can also be used to validate data because it produces summary reports that display descriptive statistics. For example, PROC MEANS can show whether the values for a particular variable are within their expected range.

Syntax, MEANS procedure:

PROC MEANS DATA=*SAS-data-set <statistics>*;
    VAR *variable(s)*;
RUN;

- The statistics to display can be specified as an option in the PROC MEANS statement.
- The VAR statement specifies the analysis variables and their order in the results.

Using the same data set as in the previous example, you can submit PROC MEANS to determine whether the age of all test subjects is within a reasonable range. Notice that the VAR statement is specified with that particular variable (Age) to get the statistical information, or range, of the data values.

```
proc means data=cert.pats;
  var Age;
run;
```

The following figure shows the output for the MEANS procedure. It displays a range of 16 to 202, which clearly indicates that there is invalid data somewhere in the Age column.

Output 7.3 *MEANS Procedure Output*

Analysis Variable : Age				
N	Mean	Std Dev	Minimum	Maximum
15	61.2666667	45.3375698	16.0000000	202.0000000

Using an Assignment Statement to Clean Invalid Data

You can use an assignment statement or a conditional clause to programmatically clean invalid data when it is identified.

For example, if your input data contains a field and that field contains an invalid value, you can use an assignment statement to clean your data. To avoid overwriting your original data set, you can use the DATA statement to create a new data set. The new data set contains all of the data from your original data set, along with the correct values for invalid data.

The following example assumes that Gender has an invalid value of **G** in the input data. This error might be the result of a data entry error. If **G** should actually be **M**, it is possible to correct the invalid data for Gender by using an assignment statement along with an IF-THEN statement:

```
data work.pats_clean;
  set cert.pats;
  gender=upcase(Gender);
  if Gender='G' then Gender='M';
run;
proc print data=work.pats_clean;
run;
```

Notice that two observations contain invalid values for Age. These values exceed a maximum value of 100. It is possible to uniquely identify each of the observations by specifying the variable ID. After checking the date of birth in each of the observations and determining the correct value for Age, you can change the data by inserting an IF-THEN-ELSE statement:

```
data work.clean_data;
  set cert.pats;
  gender=upcase(Gender);
  if Gender='G' then Gender='M';
  if id=1147 then age=65;
  else if id=5277 then age=75;
run;
proc print data=work.clean_data;
run;
```

Output 7.4 *PROC PRINT Output of Work.Clean_Data Data Set*

Obs	ID	Gender	Age
1	1129	F	48
2	1147	M	65
3	1387	F	57
4	2304	F	16
5	2486	F	63
6	4759	F	60
7	5277	F	75
8	5438	F	42
9	6745	M	18
10	6488	F	59
11	8045	M	40
12	8125	M	39
13	9012	F	39
14	9125	F	56
15	9968	M	64

Another way of ensuring that your output data set contains valid data is to programmatically identify invalid data and delete the associated observations from your output data set:

```
data work.clean_data;
  set cert.pats;
  gender=upcase(Gender);
  if Gender='G' then Gender='M';
  if Age>110 then delete;
run;
proc print data=work.clean_data;
run;
```

Output 7.5 *PROC PRINT Output of Work.Clean_Data Data Set with Deleted Observations*

Obs	ID	Gender	Age
1	1129	F	48
2	1387	F	57
3	2304	F	16
4	2486	F	63
5	4759	F	60
6	5438	F	42
7	6745	M	18
8	6488	F	59
9	8045	M	40
10	8125	M	39
11	9012	F	39
12	9125	F	56
13	9968	M	64

Testing Your Programs

Limiting Observations

Remember that you can use the OBS= option in the SET statement to limit the number of observations that are read or created during the execution of the DATA step.

```
data work.limitobs;
  set cert.invent (obs=10);
  total=instock+backord;
run;
```

When processed, this DATA step creates the Work.LimitObs data set with variables but with only 10 observations.

Example: Viewing Execution in the SAS Log

You can view the execution process in the SAS log. Use the PUTLOG statement to print the PDV in the SAS log. This enables you to view the execution process as the control goes from one record to the next. You can also place the PUTLOG statement before the FORMAT statement to see how the variables are being populated.

```
data work.update;
  set cert.invent;
  putlog 'PDV After SET Statement';
  putlog _all_;
  Total=instock+backord;
  SalePrice=(CostPerUnit*0.65)+CostPerUnit;
  format CostPerUnit SalePrice dollar 6.2;
run;
```

Log 7.2 *SAS Log*

```
PDV After SET Statement
Item=Bird Feeder IDnum=LG088 InStock=3 BackOrd=20
CostPerUnit=$5.00 Total=. SalePrice=. _ERROR_=0 _N_=1

PDV After SET Statement
Item=6 Glass Mugs IDnum=SB082 InStock=6 BackOrd=12
CostPerUnit=$1.50 Total=. SalePrice=. _ERROR_=0 _N_=2

PDV After SET Statement
Item=Glass Tray IDnum=BQ049 InStock=12 BackOrd=6
CostPerUnit=$2.50 Total=. SalePrice=. _ERROR_=0 _N_=3
PDV After SET Statement
Item=Padded Hangrs IDnum=MN256 InStock=15 BackOrd=6
CostPerUnit=$2.00 Total=. SalePrice=. _ERROR_=0 _N_=4

PDV After SET Statement
Item=Jewelry Box IDnum=AJ498 InStock=23 BackOrd=0
CostPerUnit=$6.50 Total=. SalePrice=. _ERROR_=0 _N_=5

PDV After SET Statement
Item=Red Apron IDnum=AQ072 InStock=9 BackOrd=12
CostPerUnit=$1.00 Total=. SalePrice=. _ERROR_=0 _N_=6

PDV After SET Statement
Item=Crystal Vase IDnum=AQ672 InStock=27 BackOrd=0
CostPerUnit=$7.00 Total=. SalePrice=. _ERROR_=0 _N_=7

PDV After SET Statement
Item=Picnic Basket IDnum=LS930 InStock=21 BackOrd=0
CostPerUnit=$3.50 Total=. SalePrice=. _ERROR_=0 _N_=8

PDV After SET Statement
Item=Brass Clock IDnum=AN910 InStock=2 BackOrd=10
CostPerUnit=$11.50 Total=. SalePrice=. _ERROR_=0 _N_=9
```