SAS Programming (BIOL-4V190)

Chapter 12

Working with Character Functions

12.1 Introduction

In Chapter 11, we saw how numeric functions may be used.

In this chapter we will learn about character functions which can be used to manipulate character data.

Not all available character functions will be presented in this chapter and many will be mentioned but not shown in examples, so please refer to the online documentation for additional information.

12.2 Determining the Length of a Character Value

There are three length functions:

LENGTH - returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string

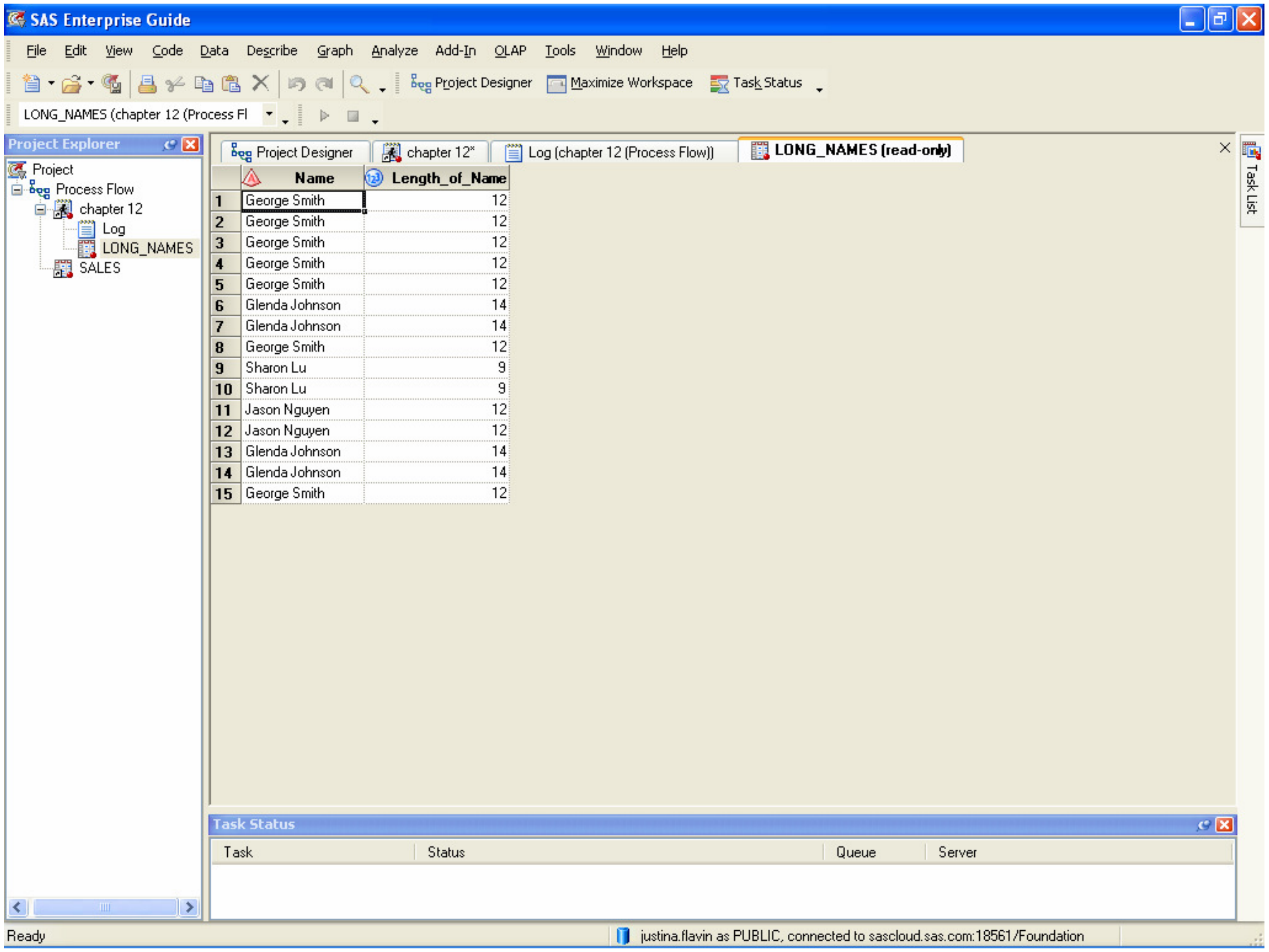
LENGTHN - returns the length of a non-blank character string, excluding trailing blanks, and returns 0 for a blank character string

LENGTHC - returns the length of a character string, including trailing blanks

In this example, we create a new (numeric) variable called Length_of_Name which contains the length of the character string value in Name.

```
*Program 12-1 Determining the length of a character value - page 213;
data long_names;
   set learn.sales;
   Length_of_Name=lengthn(Name);
   keep Name Length_of_Name;
run;
```

Notice that the value in Length_of_Name includes the blank between the first and last name.


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

	Name	Length_of_Name
1	George Smith	12
2	George Smith	12
3	George Smith	12
4	George Smith	12
5	George Smith	12
6	Glenda Johnson	14
7	Glenda Johnson	14
8	George Smith	12
9	Sharon Lu	9
10	Sharon Lu	9
11	Jason Nguyen	12
12	Jason Nguyen	12
13	Glenda Johnson	14
14	Glenda Johnson	14
15	George Smith	12

12.3 Changing the Case of Characters

There are three functions that can be used to change the case of character strings:

UPCASE – converts all characters to upper case
LOWCASE – converts all characters to lower case
PROPCASE – converts all words in an argument to proper case

In this example, the variable NAME in LEARN.MIXED has proper case character values. The same variable in LEARN.UPPER has upper case character values. The variables are shown on page 213. We wish to merge the two data sets using the variable NAME. The UPCASE function can be used to convert the values in LEARN.MIXED to upper case before merging the data.

```
*Program 12-2 Changing values to uppercase - page 214;
data mixed;
   set learn.mixed;
   Name = upcase(Name);
run;

data both;
   merge mixed learn.upper;
   by Name;
run;
```

Alternatively, the PROPCASE function could be applied to Name in LEARN.UPPER to convert those values to Proper case.

12.4 Removing Characters from Strings

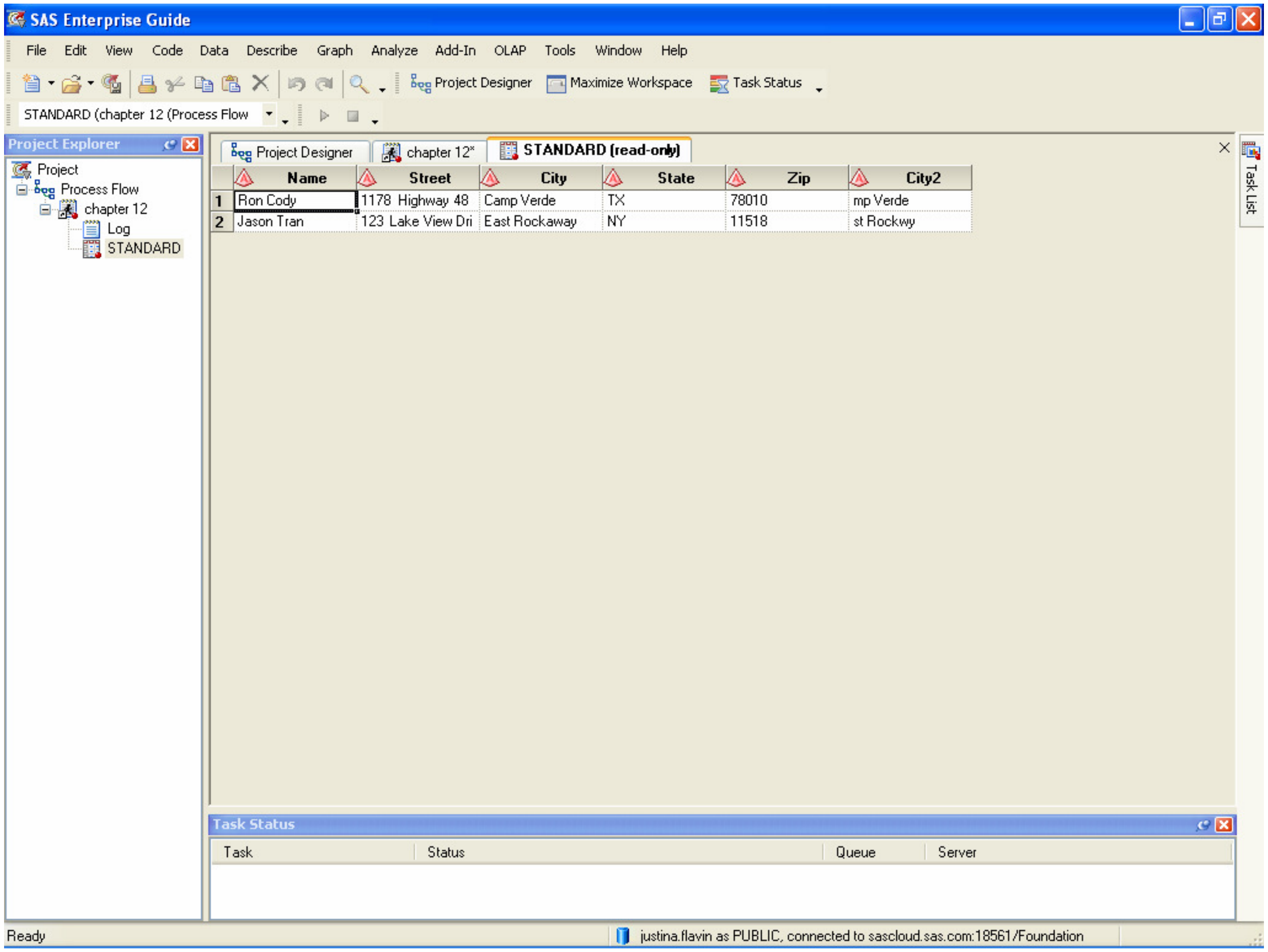
COMPBL – converts two or more blanks to a single blank
COMPRESS – removes blanks or specified characters from a character string

The data set LEARN.ADDRESS has several character variables. The contents of the data set is shown on page 214. The PROPCASE function is used to convert the values to proper case, then the COMPBL function is used to compress multiple blanks within the character strings. The same result could be achieved by reversing the order of the functions – the extra blank characters could be removed first and then the strings could be converted to proper case. For example: `Name=propcase(compbl(Name));`

City2 is a variable added to the example code to illustrate the use of the COMPRESS function. The character(s) to be removed are placed in quotes. In this example, the letters v,a, E, and C will be removed from the variable CITY. The specification is case sensitive, so ‘E’ will be removed while ‘e’ will not be removed.

```
*Program 12-3 Converting multiple blanks to a single blank
 and demonstrating the PROPCASE function - page 215;
data standard;
   set learn.address;
   Name = propcase(compbl(Name));
   Street = compbl(propcase(Street));
   City = compbl(propcase(City));
   State = upcase(State);
   City2 = compress(City, 'vaEC');
run;
```

Here is the data set STANDARD



	Name	Street	City	State	Zip	City2
1	Ron Cody	1178 Highway 48	Camp Verde	TX	78010	mp Verde
2	Jason Tran	123 Lake View Dri	East Rockaway	NY	11518	st Rockwy

12.5 Joining Two or More Strings Together

Concatenation is used to combine character strings.

The concatenation operator is the “double pipe” or ||

The “CAT” functions are:
CAT – concatenates character strings without removing leading and trailing blanks
CATS – concatenates character strings and removes leading and trailing blanks
CATT – concatenates character strings and removes trailing blanks
CATX – concatenates character strings, removes leading and trailing blanks, and inserts separators

This example has been simplified slightly from what is shown in the text.

```
*Program 12-4 Demonstrating the concatenation functions - page 216;
title "Demonstrating the Concatenation Functions";
* modified from text example;

data concat;
   Length Join Name1-Name4 $ 15;
   First = 'Ron   ';
   Last = 'Cody   ';
   Join = ':' || First || ':';
   Name1 = First || Last;
   Name2 = cat(First,Last);
   Name3 = cats(First,Last);
   Name4 = catx(' ',First,Last);
run;
```

The output is shown on page 216.

The variables First and Last have three trailing blanks.

Join is created by using the concatenation operator to append two character strings “:” to the variable First.

The three trailing blanks are not removed, hence there are three spaces before the second “:”.

Name1-Name4 also illustrate concatenating First and Last.

Name1 and Name2 use the concatenation operator and the cat function respectively to obtain the same result.

Name3 uses the cats function which removes all leading and trailing blanks.

Name4 uses the catx function which removes all leading and trailing blanks and places a separator, in this case, a blank space, between the arguments.

The separator can be any character string placed within the quotes.

12.6 Removing Leading or Trailing Blanks

There are several functions that can be used to achieve the same results as with the CAT functions:

TRIM - removes trailing blanks from character expressions
LEFT - left aligns a character expression
RIGHT – right aligns a character expression
STRIP - returns a character string with all leading and trailing blanks removed

```
*Program 12-5 Demonstrating the TRIM, LEFT, and STRIP functions - page 217;
data blanks;
   String = '   ABC  ';
   ***There are 3 leading and 2 trailing blanks in String;
   JoinLeft = ':' || left(String) || ':';
   JoinTrim = ':' || trim(String) || ':';
   JoinStrip = ':' || strip(String) || ':';
run;
```

The output for this example is shown on page 217, but the Listing output may be helpful too.

JoinLeft left justifies String, so the three leading blanks are moved to the end of the string.
There are now five trailing blanks before the colon is appended.

In JoinTrim, the trim function removes the trailing blanks from String, but leaves the three leading blanks, so after the first ‘:’ there are 3 blanks before ‘ABC:’

The strip function is similar to the CATS function and removes all leading and trailing blanks.
JoinStrip could also be created using the cats function: `cats(‘:’,String,’:’)`

12.7 Using the COMPRESS Function to Remove Characters from a String

Sometimes it is necessary to “clean up” your data by removing characters from text strings. The COMPRESS function allows you to do this.

Syntax:

`COMPRESS(<`*`source`*`><, `*`chars`*`><, `*`modifiers`*`>)`

where *source* specifies a source string that contains characters to remove
chars specifies a list of characters to be removed
modifiers is used to specify parameters to modify the behavior of the function

The first example will illustrate the use of the COMPRESS function without using the modifier option.

The data set PHONE is shown on page 218. We wish to standardise the variable PhoneNumber, by removing all characters that are not digits. Looking at the data, the non-character digits are blank, (,), -, . So these are the values that are specified for removal in the COMPRESS function.

```
*Program 12-6 Using the COMPRESS function to remove characters from a string - page 219;
data phone;
   length PhoneNumber $ 10;
   set learn.phone;
   PhoneNumber = compress(Phone,' ()-.');
   drop Phone;
run;
```

The results are shown on Page 219.

This example shows how to obtain the same result using modifiers.

A list of some of the modifiers is shown at the bottom of page 219.

A complete list may be found in the online documentation.

To use modifiers to obtain the same result, a list of characters to be removed is not supplied.

Since this second parameter is null, the function has two commas before the modifiers option.

The ‘k’ modifier instructs SAS to keep the listed characters and the ‘d’ modifier specifies digits, so ‘kd’ instructs SAS to keep digits and remove all other characters.

```
*Program 12-7 Demonstrating the COMPRESS modifiers - page 220;
data phone;
   length PhoneNumber $ 10;
   set learn.phone;
   PhoneNumber = compress(Phone,,'kd');
   *Keep only digits;
   drop Phone;
run;
```

12.8 Searching for Characters

Sometimes you may find it necessary to perform a “search and replace” operation on character strings.
The FIND function is function can help with this task.
FIND - searches for a specific substring of characters within a character string that you specify

Syntax:

```
FIND(string,substring<,modifiers><,startpos>)
FIND(string,substring<,startpos><,modifiers>)
```

string specifies a character constant, variable, or expression that will be searched for substrings

substring is a character constant, variable, or expression that specifies the substring of characters to search for in *string*

modifiers
“I” ignores character case during the search. If this modifier is not specified, FIND only searches for character substrings with the same case as the characters in *substring*.
“T” trims trailing blanks from *string* and *substring*.

startpos is an integer that specifies the position at which the search should start and the direction of the search

In this example, we have a data set MIXED_NUTS which is shown on page 221. We wish to remove the text strings so we can to convert the character variables into numeric variables and also standardise all the results to English units.

To convert the character values to numeric values, the method shown in a previous chapter of renaming and dropping variables is used.

Analysing the first if statement:

```
if find(Char_Weight,'lb','i') then  Weight = input(compress(Char_Weight,,'kd'),8.);
```

SAS evaluates Char_Weight and if the character string ‘lb’ is located, then Weight is created by first compressing all characters out of Char_Weight and then using the input function to convert this value to a numeric variable.

Otherwise if SAS finds the character string ‘kg’ in Char_Weight, then Weight is created similarly, but the value is multiplied by 2.2 to convert kgs to lbs.

The same process is used for the Height.

```
*Program 12-8 Demonstrating the FIND and COMPRESS functions - page 221;
data English;
   set learn.mixed_nuts(rename=
                     (Weight = Char_Weight
                      Height = Char_Height));
   if find(Char_Weight,'lb','i') then
      Weight = input(compress(Char_Weight,,'kd'),8.);
   else if find(Char_Weight,'kg','i') then
      Weight = 2.2*input(compress(Char_Weight,,'kd'),8.);
   if find(Char_Height,'in','i') then
      Height = input(compress(Char_Height,,'kd'),8.);
   else if find(Char_Height,'cm','i') then
      Height = input(compress(Char_Height,,'kd'),8.)/2.54;
   drop Char_:;
run;
```

The resulting data set is shown on page 222.

12.9 Searching for Individual Characters

A function similar to FIND is FINDC.

FIND searches for an exact character string, while FINDC searches for any of the characters specified.

FINDC - searches for specific characters that either appear or do not appear within a character string that you specify

```
data pie;
  food='Julian apple pie';
  pie_loc=find(food,'pie');
  char_loc=findc(food,'pie');
run;
```

In this example, the variable FOOD is searched twice.

With the find function, SAS is attempting to locate the starting position of the string ‘pie’.

‘pie’ is found and starts at the 14th character in FOOD, so pie_loc=14.

With the findc function, SAS searches the string for the first occurrence of any of the characters p, i, or e.

The first occurrence of any of those variables is the 4th character, ‘i’ in Julian, so char_loc=4.

12.10 Searching for Words in a String

To search for words in a character string, either the FINDW or INDEXW functions may be used.

The FINDW function is only available in SAS versions 9.2 and higher.

Syntax:

```
INDEXW(source, excerpt<,delimiter>)
```

source specifies the character expression to search

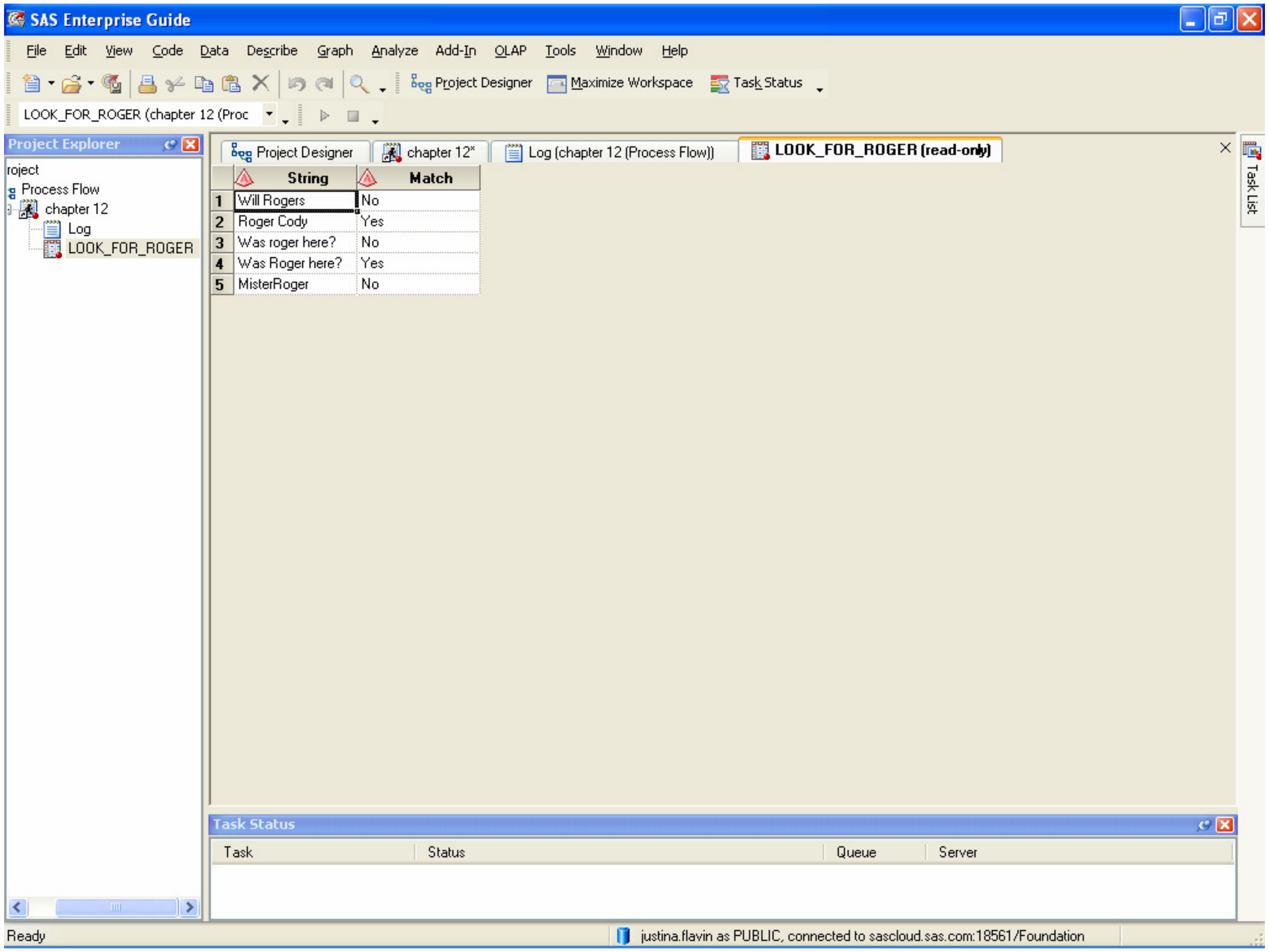
excerpt specifies the string of characters to search for in the character expression. SAS removes the leading and trailing delimiters from *excerpt*.

delimiter specifies a character expression that you want INDEXW to use as a word separator in the character string. The default delimiter is the blank character.

In this example, the variable Match records whether or not the INDEXW function successfully located the word ‘Roger’.

```
*Program 12-9 Demonstrating the FINDW functions - page 224;
*Note: the findw function is only available in SAS 9.2 and later;
data look_for_roger;
   input String $40.;
  *if findw(string,'Roger') then Match = 'Yes';
   if indexw(string,'Roger') then Match = 'Yes';
   else Match = 'No';
datalines;
Will Rogers
Roger Cody
Was roger here?
Was Roger here?
MisterRoger
;
run;
```

Observation 1 is not a match because of the ‘s’ at the end of ‘Rogers’
Observation 3 is not a match because ‘roger’ does not begin with a capital ‘R’
Observation 5 is not a match because ‘MisterRoger’ is a single word.


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

	String	Match
1	Will Rogers	No
2	Roger Cody	Yes
3	Was roger here?	No
4	Was Roger here?	Yes
5	MisterRoger	No

12.11 Searching for Character Classes

There are a series of functions that are similar to the FIND functions, but rather than searching for a particular character string, these search for groups or classes of characters.

These functions are listed below:

ANYALNUM - searches a character string for an alphanumeric character and returns the first position at which it is found

ANYALPHA - searches a character string for an alphabetic character and returns the first position at which it is found

ANYDIGIT - searches a character string for a digit and returns the first position at which it is found

ANYLOWER - searches a character string for a lowercase letter and returns the first position at which it is found

ANYPUNCT - searches a character string for a punctuation character and returns the first position at which it is found

ANYSPACE - searches a character string for a white-space character (blank, horizontal and vertical tab, carriage return, line feed, form feed) and returns the first position at which it is found

ANYUPPER - searches a character string for an uppercase letter and returns the first position at which it is found

The data in ID.TXT are shown on page 225.

We wish to partition the observations into two data sets, one that contains all values that have digits, the other that contains values that only have characters.

This example also illustrates creating multiple output data sets.

Multiple output data sets can be created from one data step by specifying more than one data set name on the data line.

Multiple output statements within the data step code are used to control which observations and variables are written to each output data set.

If the value of ID contains a digit, then the observation is written to MIXED.

Otherwise the observation is written to ONLY-ALPHA.

```
*Program 12-10 Demonstrating the ANYDIGIT function - page 225;
data only_alpha mixed;
   infile "/courses/u_ucsd.edu1/i_536036/c_629/id.txt" truncover;
   input ID $10.;
   if anydigit(ID) then output mixed;
   else output only_alpha;
run;
```

The resulting data sets are shown on page 226.

12.12 Using the NOT Functions for Data Cleaning

Similar to the ANY functions, there are a series of NOT functions which can be used to identify the position of any characters in the string which are not of the character class identified by the function.

NOTALNUM - searches a character string for a non-alphanumeric character and returns the first position at which it is found

NOTALPHA - searches a character string for a non-alphabetic character and returns the first position at which it is found

NOTDIGIT - searches a character string for any character that is not a digit and returns the first position at which that character is found

NOTLOWER - searches a character string for a character that is not a lowercase letter and returns the first position at which that character is found

NOTPUNCT - searches a character string for a character that is not a punctuation character and returns the first position at which it is found

NOTUPPER - sarches a character string for a character that is not an uppercase letter and returns the first position at which that character is found

```
*Program 12-11 Demonstrating the "NOT" functions for data cleaning - page 227;
title "Data Cleaning Application";
* modified from text example;
data _null_;
   set learn.cleaning;
   if notalpha(trim(Letters))  then put Subject= Letters=;
   if notdigit(trim(Numerals)) then put Subject= Numerals=;
   if notalnum(trim(Both))     then put Subject= Both=;
run;
```

This example illustrates the use of the special data set name _null_ which causes no output data set to be created.

The contents of CLEANING are shown on page 227.

We wish to identify any observations in the variable Letters that contain digits, any observations in the variable Numerals that contain non-alphabetic characters, and any observations in the variable Both that contain non-alphanumeric characters.

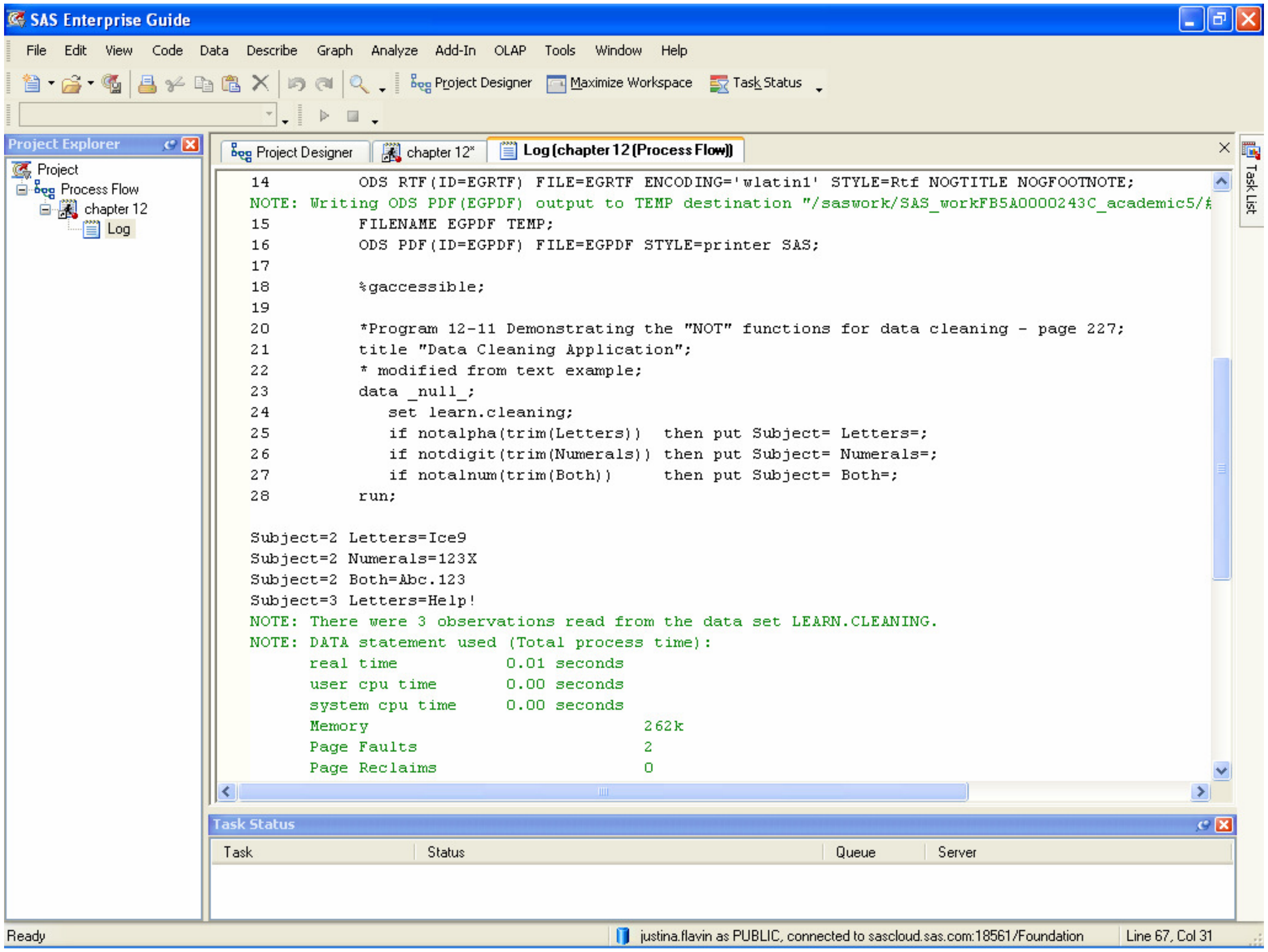
This is accomplished using the NOT functions.

The PUT Statement is used to write lines to the SAS log.

When a value is encountered that meets the criteria, then SAS writes a line to the log that contains the text: variable name=*variable value*.

This is known as named output.

The LOG contains the results of the put statements.



```
14         ODS RTF(ID=EGRTF) FILE=EGRTF ENCODING='wlatin1' STYLE=Rtf NOGTITLE NOGFOOTNOTE;
NOTE: Writing ODS PDF(EGPDF) output to TEMP destination "/saswork/SAS_workFB5A0000243C_academic5/
15         FILENAME EGPDF TEMP;
16         ODS PDF(ID=EGPDF) FILE=EGPDF STYLE=printer SAS;
17
18         %gaccessible;
19
20         *Program 12-11 Demonstrating the "NOT" functions for data cleaning - page 227;
21         title "Data Cleaning Application";
22         * modified from text example;
23         data _null_;
24            set learn.cleaning;
25            if notalpha(trim(Letters))  then put Subject= Letters=;
26            if notdigit(trim(Numerals)) then put Subject= Numerals=;
27            if notalnum(trim(Both))     then put Subject= Both=;
28         run;

Subject=2 Letters=Ice9
Subject=2 Numerals=123X
Subject=2 Both=Abc.123
Subject=3 Letters=Help!
NOTE: There were 3 observations read from the data set LEARN.CLEANING.
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                            262k
      Page Faults                       2
      Page Reclaims                     0
```

12.13 Describing a Real Blockbuster Data Cleaning Function

The VERIFY function returns the position of the first character that is unique to an expression

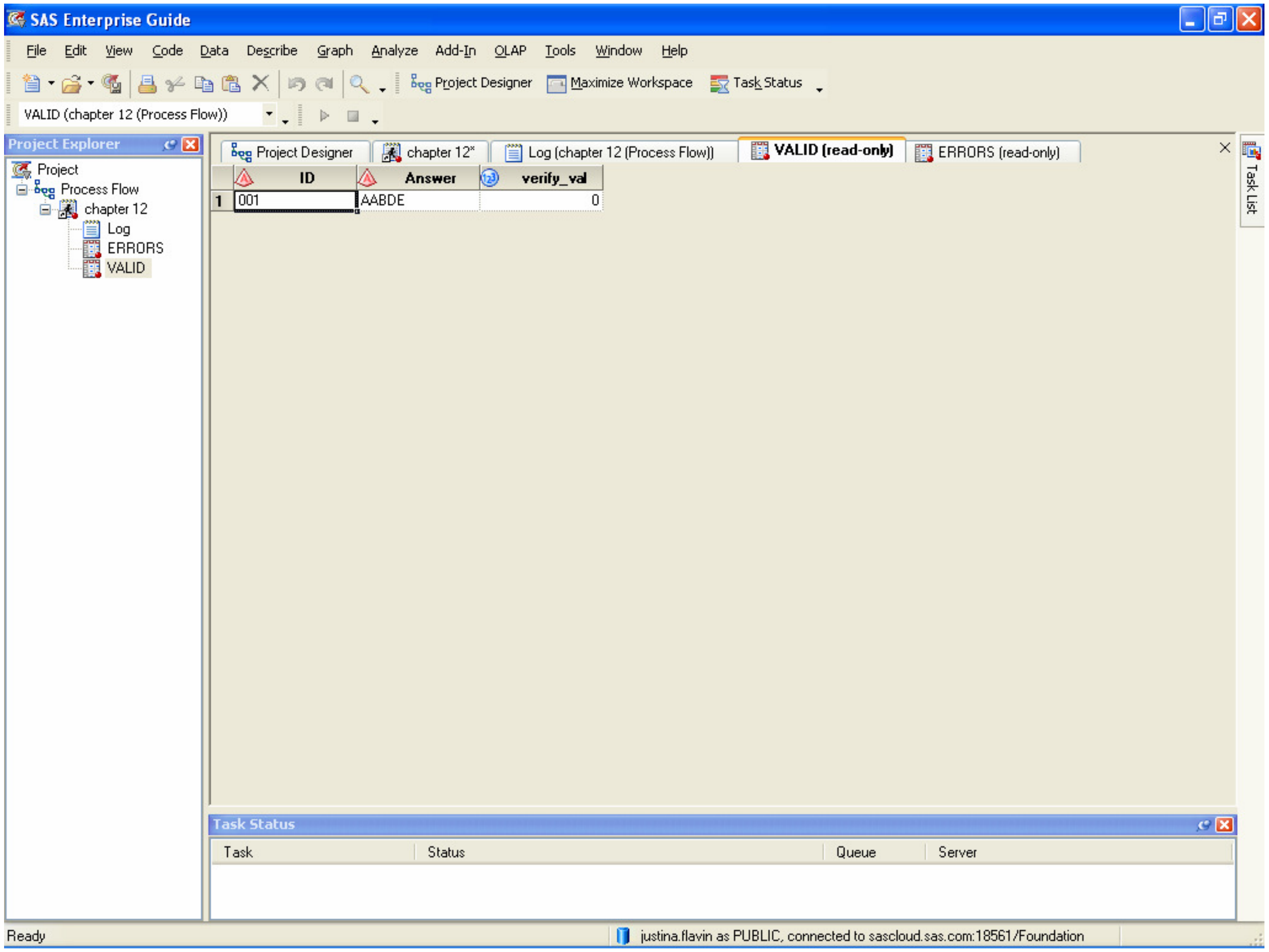
In this example, if SAS finds any characters other than ABCDE in the variable Answer, then the VERIFY function returns the position where it first encountered such a character.

Otherwise, the VERIFY function returns the value 0.

So we should expect to find that when verify_val=0, the observations will go into VALID, while any other values of verify_val will be in ERRORS.

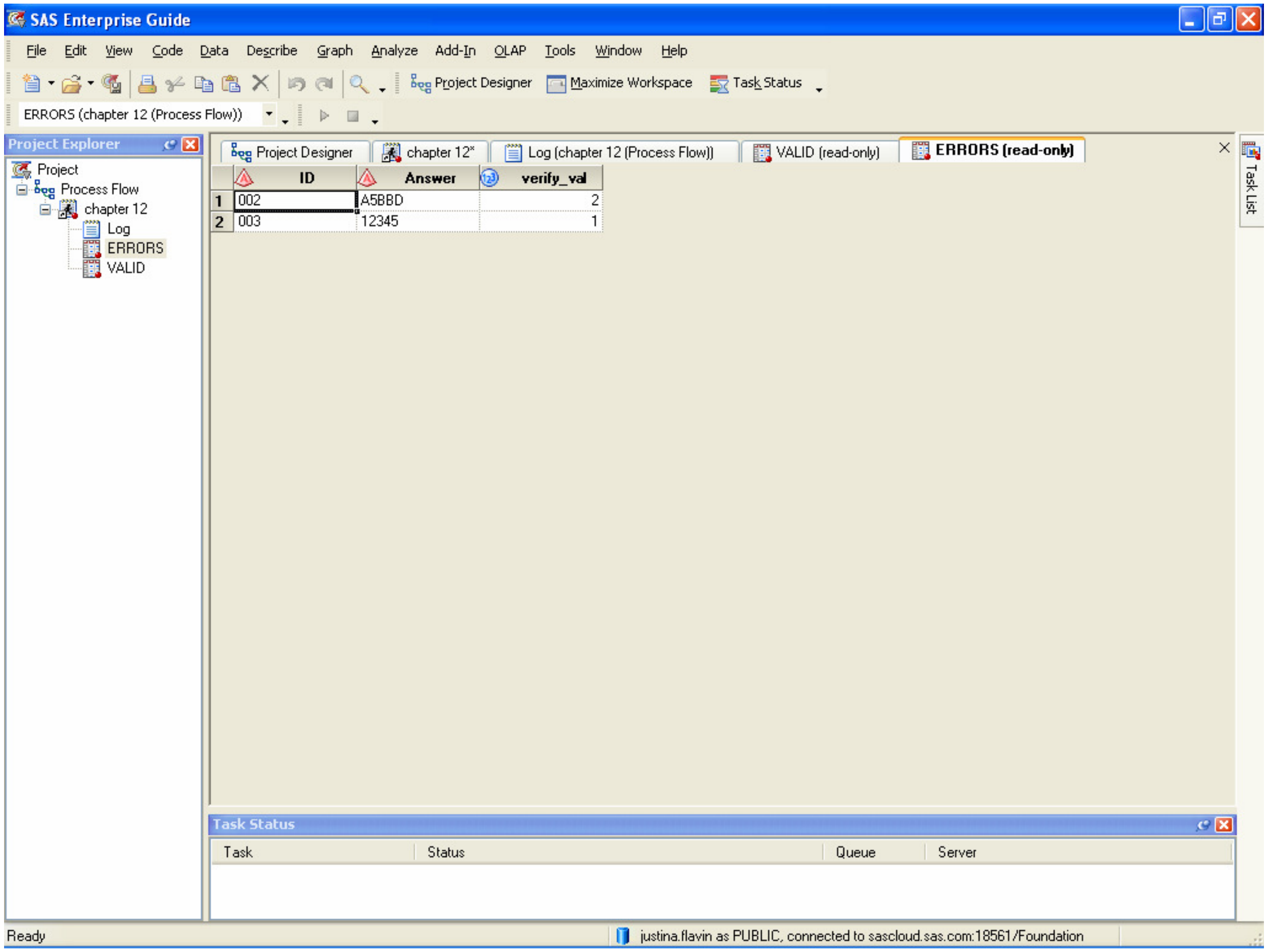
```
*Program 12-12 Using the VERIFY function for data cleaning - page 228;
data errors valid;
   input ID $  Answer : $5.;
   verify_val=verify(Answer,'ABCDE');
   if verify(Answer,'ABCDE') then output errors;
   else output valid;
datalines;
001 AABDE
002 A5BBD
003 12345
;
run;
```

Here is the data set VALID.


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

	ID	Answer	verify_val
1	001	AABDE	0

Here is the data set ERRORS.



	ID	Answer	verify_val
1	002	A5BBD	2
2	003	12345	1

12.14 Extracting Part of a String

To extract parts of a character string, the substring function is used.

Syntax:

SUBSTR(*string, position*<,*length*>)

variable specifies a valid SAS variable name

string specifies any SAS character expression

position specifies a numeric expression that is the beginning character position

length specifies a numeric expression that is the length of the substring to extract

The substring function operates in two different ways, depending upon whether it is on the left or the right side of the =

SUBSTR (to the left of =) - replaces character value contents

SUBSTR (to the right of =) - extracts a substring from an argument

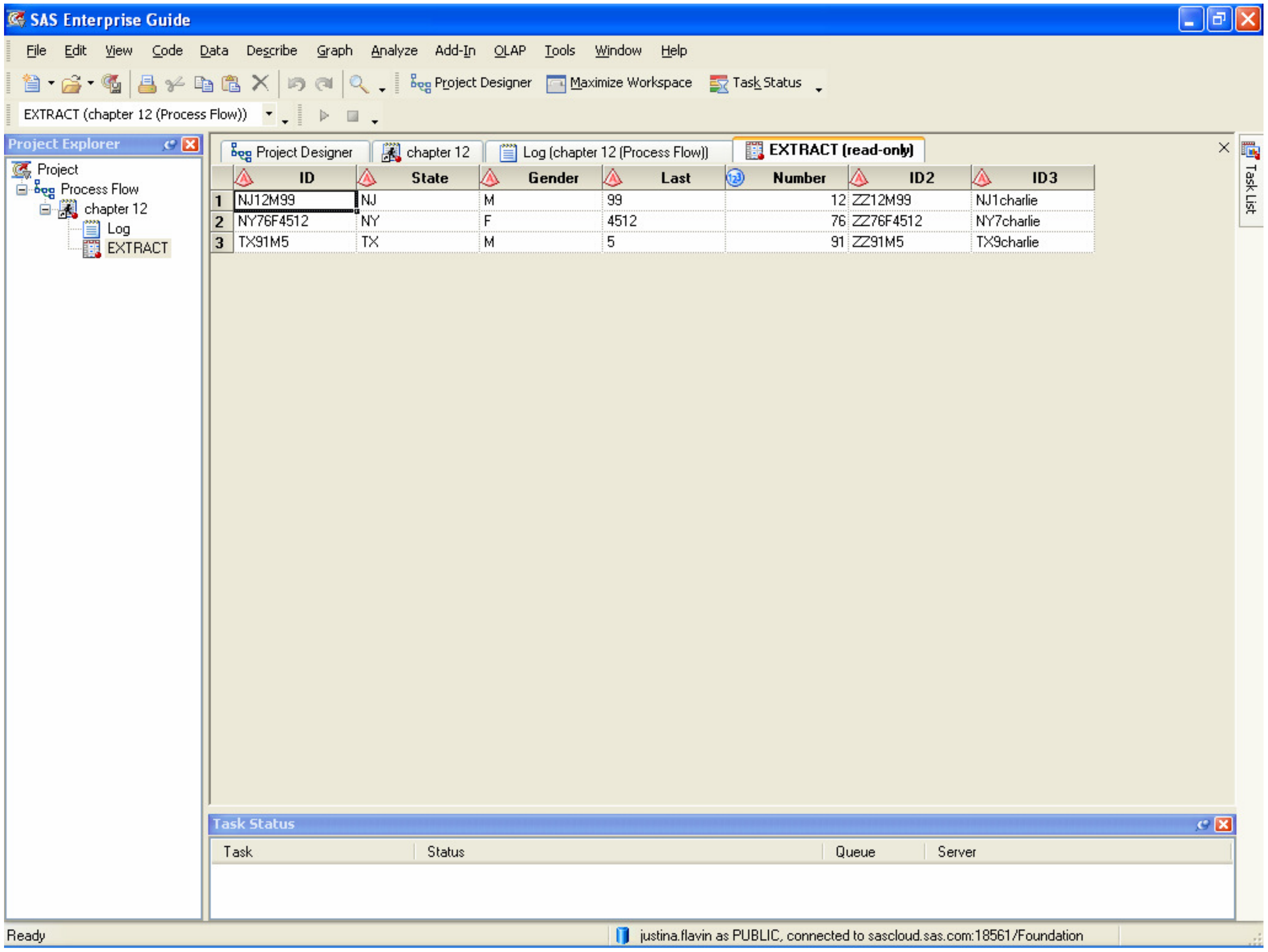
```
*Program 12-13 Using the SUBSTR function to extract substrings - page 229;
* modified from text example;
data extract;
   input ID : $10. @@;
   length State $ 2 Gender $ 1 Last $ 5;
   * using SUBSTR on the right;
   State = substr(ID,1,2);
   Number = input(substr(ID,3,2),3.);
   Gender = substr(ID,5,1);
   Last = substr(ID,6);
   * using SUBSTR on the left;
   ID2=ID;
   substr(ID2,1,2)='ZZ';
   ID3=ID;
   substr(ID3,4)='charlie';
datalines;
NJ12M99 NY76F4512 TX91M5
;
run;
```

In this example, the variable State is created by selecting the first two characters in ID, Number is created by selecting two characters starting at position 3 and then applying the input function to convert the value to a numeric, Gender is created by selecting a single character at position 5, and Last is created by selecting everything from position 6 to the end of the value of ID.

Illustrating the use of the SUBSTR function on the left side of the equation, in ID2, the character string ZZ replaces the first two characters in ID.

In ID3, the character string “charlie’ replaces all characters starting at the 4th character.

Here is the data set EXTRACT


SAS Enterprise Guide
File Edit View Code Data Describe Graph Analyze Add-In OLAP Tools Window Help
Project Designer Maximize Workspace Task Status
EXTRACT (chapter 12 (Process Flow))
Project Explorer
Project
Process Flow
chapter 12
Log
EXTRACT
Project Designer
chapter 12
Log (chapter 12 (Process Flow))
EXTRACT (read-only)
Task List
Task Status
Task
Status
Queue
Server

	ID	State	Gender	Last	Number	ID2	ID3
1	NJ12M99	NJ	M	99	12	ZZ12M99	NJ1charlie
2	NY76F4512	NY	F	4512	76	ZZ76F4512	NY7charlie
3	TX91M5	TX	M	5	91	ZZ91M5	TX9charlie

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

12.15 Dividing Strings into Words

The SCAN function is used to extract words from a character string.

Syntax:

`SCAN(`*`string ,n`*`<, `*`delimiter(s)`*`>)`

string specifies any character expression

n specifies a numeric expression that produces the number of the word in the character string you want SCAN to select

delimiter specifies a character expression that produces characters that you want SCAN to use as a word separator in the character string

If a delimiter is not specified, SAS will treat any of the following as delimiters:

blank . < (+ & ! $ *) ; ^ - / , % |

```
*Program 12-14 Demonstrating the SCAN function - page 230;
data original;
   input Name $ 30.;
datalines;
Jeffrey Smith
Ron Cody
Alan Wilson
Alfred E. Newman
;
run;

data first_last;
   set original;
   length First Last $ 15;
   First = scan(Name,1,' ');
   Last = scan(Name,2,' ');
run;
```

In this example, the variables First and Last are created using the SCAN function, specifying a blank as a word delimiter.

First is created by scanning in the first word, and Last is created by scanning in the second word.

The results are shown on page 231.

Notice that the fourth data value actually contains 3 words: Alfred, E., and Newman.

So for that value, Last=E. since that is the second word in the character string.

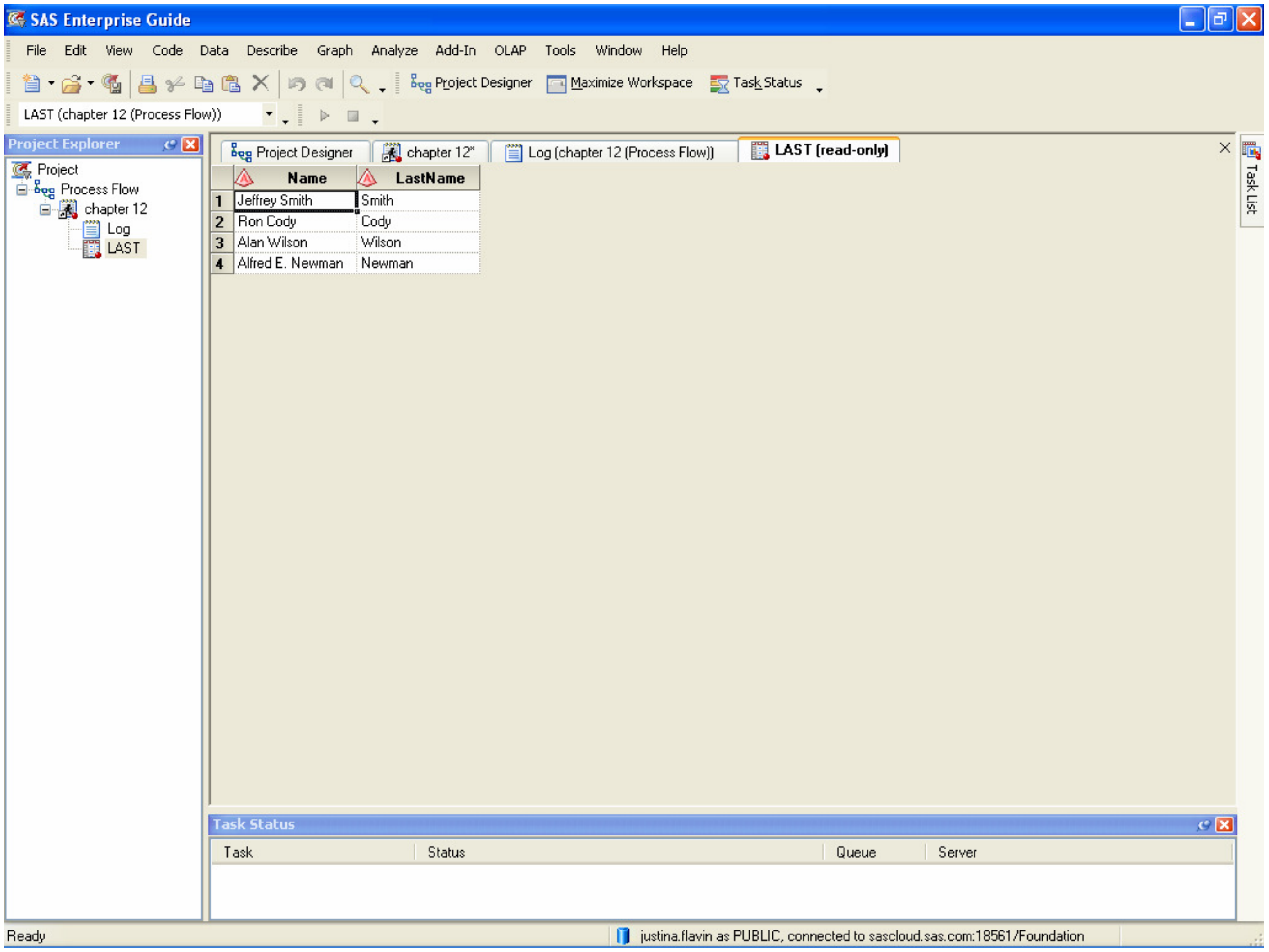
If a negative number is specified in the SCAN function, then the function scans in reverse, starting at the end of the string.

So scan(Name,-1) would scan in the first word starting at the end of the string.

This provides a way to correctly scan in the last name for all of the data values.

```
*Program 12-15 Using the SCAN function to extract the last name - page 231;
data last;
   set original;
   length LastName $ 15;
   LastName = scan(Name,-1,' ');
run;
```

Notice that LastName for Alfred E. Newman is now correct.



	Name	LastName
1	Jeffrey Smith	Smith
2	Ron Cody	Cody
3	Alan Wilson	Wilson
4	Alfred E. Newman	Newman

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

12.16 Comparing Strings

This section is omitted.

12.17 Performing a Fuzzy Match

This section is omitted.

12.18 Substituting Characters or Words

Another set of functions that are similar to the SUBSTR function are the TRANSLATE and TRANWRD functions.

TRANSLATE - replaces specific characters in a character expression

TRANSLATE(*source,to-1,from-1*<,...*to-n,from-n*>)
"to" specifies the characters that you want TRANSLATE to use as substitutes
"from" specifies the characters that you want TRANSLATE to replace

TRANWRD - replaces or removes all occurrences of a word in a character string

TRANWRD(*source,target,replacement*) specifies the source string that you want to translate
target specifies the string searched for in *source*
replacement specifies the string that replaces *target*

```
*Program 12-19 Demonstrating the TRANSLATE function - page 236;
data trans;
   input Answer : $5.;
   Answer = translate(Answer,'ABCDE','12345');
datalines;
14325
AB123
51492
;
run;
```

In this example, the characters 12345 will be replaced by the characters ABCDE on a 1-to-1 basis.

That is, 1 is replaced by A, 2 is replaced by B, etc.

The result is shown on page 236.

This example illustrates the use of the TRANWRD function and the concept of reading multiple lines of data into a single observation.

```
*Program 12-20 Using the TRANWRD function to standardize an address - page 237;
data address;
   infile datalines dlm=' ,';
   *Blanks or commas are delimiters;
   input #1 Name $30.
         #2 Line1 $40.
         #3 City & $20. State : $2. Zip : $5.;

   Name = tranwrd(Name,'Mr.',' ');
   Name = tranwrd(Name,'Mrs.',' ');
   Name = tranwrd(Name,'Dr.',' ');
   Name = tranwrd(Name,'Ms.',' ');
   Name = left(Name);

   Line1 = tranwrd(Line1,'Street','St.');
   Line1 = tranwrd(Line1,'Road','Rd.');
   Line1 = tranwrd(Line1,'Avenue','Ave.');
datalines;
...
run;
```

When there are multiple lines of data that need to be read into a single observation, a line pointer must be used on the input statement.

The symbol for a line pointer is #.

The line pointer is followed by the line number or row in the raw data, followed by the variables to be read from that line of data.

Any input method can be used to read in the variables.

Syntax:

```
input #1 variablename1 col1-col2
         variablename2 $ col3-col4
         variablename3 col5
      #2 @columnnumber1 variablename1 informat. @columnnumber2 variablename2 $informat.
      #3 @columnnumber1 variablename1 informat.;
```

In this example, the TRANWRD function is used to remove titles from the data lines, so Mr., Mrs., Dr. etc are replaced with a blank.

Then the LEFT function is used to left-align those values. Similarly, street types are replaced with shorter abbreviations.

The resulting data set is shown on page 238.