# Learning SAS: A Comprehensive Guide to the COMPRESS Function with Practical Examples

Authored by

**Mohammed looti**

October 27, 2025

In the realm of SAS programming, the ability to perform efficient data cleaning and manipulation is absolutely paramount for ensuring accurate and reliable analytical results. Raw data often contains inconsistencies, extraneous spaces, and unwanted symbols that hinder proper processing. To address these issues, one of the most versatile and frequently utilized tools available to programmers is the **COMPRESS** function.

This powerful function enables users to precisely control the content of character or string variables by systematically removing specified characters, thereby preparing the data for rigorous analysis, reporting, or database integration. Mastering the COMPRESS function can significantly optimize your SAS workflows, whether the task involves basic removal of trailing whitespace or complex exclusion of character patterns. This comprehensive article will guide you through its fundamental syntax, explore its powerful modifiers, and demonstrate its most common applications through practical, step-by-step examples.

## Deconstructing the COMPRESS Function Syntax

The COMPRESS function is engineered to simplify character removal from a string. Its structure is highly flexible, allowing developers to define the source data, the exact characters to target, and optional instructions for character classes. This flexibility is achieved through three primary arguments that provide granular control over the compression process.

The general structure of the **COMPRESS** function call is:

```
COMPRESS(source, characters_to_remove, modifiers)
```

Understanding the role of each argument is essential for effective data manipulation:

**`source`**: This is the mandatory first argument. It defines the character variable or constant from which characters will be eliminated. If the source argument is missing or contains a null value, the function will logically return a missing or null result.
**`characters_to_remove`** (Optional): This second argument specifies a set of one or more individual characters that should be removed from the `source` string. If this argument is entirely omitted, **COMPRESS** defaults to removing all blank spaces from the source data, which is a common data cleaning requirement.
**`modifiers`** (Optional): The third argument is a character constant, variable, or expression that supplies additional instructions for the function's operation. Modifiers are extremely powerful as they allow you to target entire character classes (e.g., all letters or all numbers) without listing them individually. Key modifiers include:

**`'s'`**: Instructs the function to remove all blank spaces (functionally equivalent to omitting the `characters_to_remove` argument).

`'a'`: Removes all alphabetical characters (A-Z and a-z).
`'d'`: Removes all numeric values (digits 0-9).
`'p'`: Removes all punctuation characters.
`'k'`: The "Keep" modifier. This inverts the function's logic, instructing it to keep the characters specified in the `characters_to_remove` argument while removing all others.

## Method 1: Standardizing Data by Removing Whitespace

One of the simplest yet most crucial data cleaning operations is the elimination of superfluous blank spaces. Extraneous leading, trailing, or internal spaces can cause significant inconsistencies, preventing accurate matching or comparison of text values. For example, "ID 123 " and "ID 123" are treated as distinct values until the spacing is standardized. When the COMPRESS function is called with only the mandatory source argument, it automatically defaults to removing all occurrences of blank spaces.

This method is highly recommended for preparing text fields for concatenation, database lookups, or when creating unique keys based on character variables. The resulting string is compact and consistent, ensuring that values are treated identically regardless of original spacing.

The concise syntax for this default behavior is:

```
data new_data;
set original_data;
compressed_string = compress(string_variable);
run;
```

## Method 2: Precision Removal of Specific Symbols

Beyond standard blank spaces, data often contains unique symbols, punctuation, or unwanted alphanumeric codes that must be purged. The **COMPRESS** function provides the flexibility to target these elements precisely using the second argument, `characters_to_remove`. By explicitly listing the characters within this argument, you maintain fine-grained control over the data transformation process, ensuring only specific unwanted characters are eliminated.

This technique is essential for standardizing inputs like product codes, financial identifiers, or addresses where inconsistent punctuation or currency symbols might interfere with later analysis. When listing the characters, they should be grouped together inside single quotes. SAS interprets each character within this group as a separate target for removal.

The syntax below demonstrates how to remove specific characters, such as common symbols like exclamation points, question marks, and hash symbols:

```
data new_data;
set original_data;
compressed_string = compress(string_variable, '!?@#');
run;
```

## Method 3: Class-Based Removal Using Modifiers ('a' and 'd')

When the requirement is to isolate or remove entire categories of characters, relying on modifiers simplifies the process significantly. Instead of manually listing every letter or digit, the third argument allows us to specify character classes. This approach is fundamental for transforming mixed alphanumeric data into pure numerical or pure textual fields.

For instance, if you need to extract pure numerical information from a mixed string--perhaps a house number embedded in an address line--you would use the `'a'` modifier. The `'a'` modifier instructs SAS to remove all alphabetical characters (A-Z, a-z), leaving only digits and other symbols behind. Conversely, if you need to retain only the textual components and eliminate all measurements or codes, you would use the `'d'` modifier, which targets and removes all numeric values (0-9).

When applying a modifier, the second argument (`characters_to_remove`) is typically left as an empty string (`''`) to indicate that the removal logic is entirely driven by the character class defined in the modifier. This ensures the function focuses solely on the class defined by the modifier.

Example of removing **Alphabetical Characters** (using 'a' modifier):

```
data new_data;
set original_data;
compressed_string = compress(string_variable, '', 'a');
run;
```

Example of removing **Numeric Values** (using 'd' modifier):

```
data new_data;
set original_data;
compressed_string = compress(string_variable, '', 'd');
run;
```

## Setting Up the Demonstration Dataset in SAS

To provide clear, practical illustrations of the COMPRESS function's capabilities, we must first

establish a sample dataset containing varied, messy data. This dataset, which we will name `original_data`, includes a character variable called `name`. The content of this variable is intentionally complex, containing a mix of names, numbers, excessive spacing, and special characters, allowing us to demonstrate the effect of each compression technique effectively.

In SAS, dataset creation for demonstrations is handled efficiently using a DATA step. We use the **INPUT** statement to define the `name` variable as a character type (`$25.` indicating a length of 25) and then populate it using the **DATALINES** statement. The semicolon on a new line signals the end of the inline data input. Following the data creation, we employ the **PROC PRINT** procedure to generate a clean, readable tabular output, allowing us to inspect the raw data before any transformation occurs. This initial view is vital for appreciating the subsequent data cleaning results.

```
/*Create demonstration dataset containing mixed characters*/
data original_data;
input name $25.;
datalines;
Andy Lincoln4 Bernard!
Barren Michael55 Smith!
Chad Simpson7 Arnolds?
Derrick Parson2 Henry
Eric Miller2 Johansen!
Frank Giovanni5 Goode
;
run;

/*View the raw dataset contents*/
proc print data=original_data;
```

| Obs | name |
|---|---|
| 1 | Andy Lincoln4 Bernard! |
| 2 | Barren Michael55 Smith! |
| 3 | Chad Simpson7 Arnolds? |
| 4 | Derrick Parson2 Henry |
| 5 | Eric Miller2 Johansen! |
| 6 | Frank Giovanni5 Goode |

The image above clearly displays the structure of `original_data`. Observe the presence of

excessive blank spaces, embedded numeric values, and special characters within the `name` variable. These elements represent common challenges faced during data preparation, which we will address in the following examples using the **COMPRESS** function.

## Example 1: Implementing Default Whitespace Removal

The simplest and perhaps most frequently executed operation with the COMPRESS function is eliminating all internal and external blank spaces. This standardization is vital for creating standardized fields that are suitable for comparison and joining across different tables. In this demonstration, we read the `name` variable from `original_data` and generate a new variable, `compressed_name`, where all whitespace has been completely removed.

The following SAS code illustrates this process within a DATA step. By calling `compress(name)` without any subsequent arguments, we leverage the default behavior of the function, which is optimized solely for blank space removal. This provides the most efficient way to achieve a tightly concatenated string representation of the original data.

```
/*Remove all blank spaces from the name column*/
data new_data;
set original_data;
compressed_name = compress(name);
run;

/*View the results of the transformation*/
proc print data=new_data;
```

| Obs | name | compressed_name |
|---|---|---|
| 1 | Andy Lincoln4 Bernard! | AndyLincoln4Bernard! |
| 2 | Barren Michael55 Smith! | BarrenMichael55Smith! |
| 3 | Chad Simpson7 Arnolds? | ChadSimpson7Arnolds? |
| 4 | Derrick Parson2 Henry | DerrickParson2Henry |
| 5 | Eric Miller2 Johansen! | EricMiller2Johansen! |
| 6 | Frank Giovanni5 Goode | FrankGiovanni5Goode |

Observing the output, it is evident that every blank space has been successfully removed, resulting in a completely concatenated string in the newly created `compressed_name` column. For instance, the value "Andy Lincoln4 Bernard!" is compacted into "AndyLincoln4Bernard!". This transformation

is often critical in preparing data for text mining or creating lookup keys where absolute consistency is required.

## Example 2: Targeted Removal of Punctuation and Special Characters

Data imported from external systems frequently contains unwanted punctuation or symbols that must be systematically eliminated to achieve clean data. In our sample data, certain observations contain symbols like exclamation points (!) and question marks (?). Our objective in this example is to use the second argument of the COMPRESS function to precisely target and remove these specific symbols from the `name` column.

By defining the set of characters to be removed as `'?!'`, we instruct the **COMPRESS** function to scan the input variable and eliminate any matching instances. This method ensures that the remaining characters, including spaces, numeric values, and letters, are preserved exactly as they were, granting high precision in the data cleaning process. The syntax below demonstrates this targeted approach within a DATA step:

```
/*Remove specific symbols (question marks and exclamation points)*/
data new_data;
set original_data;
compressed_name = compress(name, '?!');
run;

/*View the results of the transformation*/
proc print data=new_data;
```

| Obs | name | compressed_name |
|---|---|---|
| 1 | Andy Lincoln4 Bernard! | Andy Lincoln4 Bernard |
| 2 | Barren Michael55 Smith! | Barren Michael55 Smith |
| 3 | Chad Simpson7 Arnolds? | Chad Simpson7 Arnolds |
| 4 | Derrick Parson2 Henry | Derrick Parson2 Henry |
| 5 | Eric Miller2 Johansen! | Eric Miller2 Johansen |
| 6 | Frank Giovanni5 Goode | Frank Giovanni5 Goode |

The resulting `compressed_name` column confirms the successful removal of the targeted punctuation. For example, "Chad Simpson7 Arnolds?" is transformed to "Chad Simpson7 Arnolds". This ability to specify and remove non-essential symbols is highly beneficial for preparing textual

data that might contain noise from differing source formats, ensuring clean and comparable string variables.

## Example 3: Extracting Pure Numeric Data (Removing Alphabetical Characters)

A frequent challenge in data analysis is isolating numerical identifiers or measurements that are embedded within a larger descriptive text. To achieve this, we need a method to strip away all alphabetical characters while preserving the digits and other symbols. The **COMPRESS** function handles this efficiently using the `'a'` modifier, which represents the entire class of letters (A-Z and a-z).

In this example, we apply the `'a'` modifier to the `name` column of our `original_data`. Note that the second argument is specified as an empty string (`''`); this tells the function that no individual characters are targeted, and the removal logic is entirely governed by the modifier. This technique is indispensable when you need to convert a mixed field into a purely numerical format, often as a precursor to converting the character variable into a numeric value for calculations.

The SAS code below demonstrates the power of class-based removal:

```
/*Remove all alphabetical characters using the 'a' modifier*/
data new_data;
set original_data;
compressed_name = compress(name, '', 'a');
run;

/*View the results of the transformation*/
proc print data=new_data;
```

| Obs | name | compressed_name |
|---|---|---|
| 1 | Andy Lincoln4 Bernard! | 4! |
| 2 | Barren Michael55 Smith! | 55! |
| 3 | Chad Simpson7 Arnolds? | 7? |
| 4 | Derrick Parson2 Henry | 2 |
| 5 | Eric Miller2 Johansen! | 2! |
| 6 | Frank Giovanni5 Goode | 5 |

The output validates the operation: all alphabetical characters have been successfully removed, leaving only the digits, spaces, and special symbols (like the exclamation point). For example, "Derrick Parson2 Henry" is reduced to "2". This is a highly efficient method for isolating numerical data required for quantitative analysis.

## Example 4: Extracting Pure Textual Data (Removing Numeric Values)

Conversely, isolating the pure textual components of a string often requires the elimination of all embedded digits. This is standard practice when preparing names, descriptions, or comments for analysis where numerical noise might obscure patterns. The COMPRESS function facilitates this with the `'d'` modifier, which specifically targets and removes all digits (0-9).

In this final illustration, we apply the `'d'` modifier to the `name` column. By using the empty string for the second argument and supplying `'d'` as the modifier, we instruct SAS to remove all numeric values. This leaves behind only the alphabetical characters, spaces, and punctuation, ensuring the resultant variable contains only the textual information intended for linguistic or categorical analysis.

The corresponding syntax is as follows:

```
/*Remove all numeric values using the 'd' modifier*/
data new_data;
set original_data;
compressed_name = compress(name, '', 'd');
run;

/*View the results of the transformation*/
proc print data=new_data;
```

| Obs | name | compressed_name |
|---|---|---|
| 1 | Andy Lincoln4 Bernard! | AndyLincolnBernard! |
| 2 | Barren Michael55 Smith! | BarrenMichaelSmith! |
| 3 | Chad Simpson7 Arnolds? | ChadSimpsonArnolds? |
| 4 | Derrick Parson2 Henry | DerrickParsonHenry |
| 5 | Eric Miller2 Johansen! | EricMillerJohansen! |
| 6 | Frank Giovanni5 Goode | FrankGiovanniGoode |

As you can see from the output, all numeric values have been successfully removed from each string in the `compressed_name` column. For example, "Andy Lincoln4 Bernard!" becomes "Andy Lincoln Bernard!". This powerful operation is essential for tasks requiring the isolation of purely textual data from mixed alphanumeric fields.

## Conclusion: Mastering String Manipulation with COMPRESS

The COMPRESS function stands as an indispensable utility within SAS for high-precision data cleaning and string manipulation. Its robust design allows programmers to tackle a wide spectrum of data quality issues, ranging from simple whitespace standardization to complex, class-based removal of numeric values or alphabetical characters.

By effectively utilizing its three arguments--the source string, the list of specific characters to remove, and powerful modifiers--you can significantly improve the integrity and usability of your SAS datasets. This mastery ensures that subsequent analyses are based on standardized and accurate data, leading to more reliable business intelligence and statistical outcomes.

Always evaluate your specific data cleaning goals before implementing the function. Whether you need to concatenate fields, extract numerical codes, or remove noise, the **COMPRESS** function, when integrated into your DATA step logic, provides a concise and powerful mechanism for achieving optimal data quality.

## Additional Resources

The following tutorials explain how to perform other common tasks in SAS: