# How to Easily Concatenate Strings in SAS Using the Operator and CATX Function

Authored by

**stats writer**

December 1, 2025

In the world of data manipulation and statistical programming, the ability to combine text fields--a process known as string concatenation--is a fundamental requirement. Within the SAS environment, data analysts and programmers frequently need to merge multiple character variables into a single, cohesive string for reporting, labeling, or creating unique identifiers. This operation is essential when generating full names from separate first and last name fields, constructing addresses, or assembling specific metadata strings.

SAS provides several powerful and flexible tools to achieve concatenation, primarily relying on the use of built-in functions or the dedicated concatenation operator. Understanding the nuances between these methods is crucial, as each handles missing values, delimiters, and trailing blanks differently, which can profoundly impact the resulting output. The two primary approaches are utilizing the concatenation operator (`||`) or employing the versatile family of CAT functions, such as CAT, `CATS`, and CATX. These techniques must be applied strategically within the SAS Data Step to ensure data integrity and desired formatting.

The choice between the operator and the functions often boils down to how trailing spaces are managed. The standard concatenation operator (`||`) preserves all trailing blanks attached to character variables, often necessitating explicit trimming functions like `TRIM` before concatenation. Conversely, the CAT family of functions, designed specifically for combining text, automatically handles the removal of leading and trailing spaces, leading to cleaner and more efficient code. This guide details the structure and application of the most effective methods for combining strings within a SAS programming environment.

## The Concatenation Operator (||) vs. SAS Functions

While the goal of combining strings remains constant, the methodology employed significantly affects the outcome, particularly concerning data cleanliness. The double vertical bar (`||`) operator is the most traditional way to perform string concatenation in SAS. When using this operator within a Data Step, the resulting string is an exact combination of the source strings, including any padding or trailing spaces defined by the variable's length. For instance, if `var1` has a length of 10 and contains "John" (followed by six spaces), concatenating it with `var2` containing "Doe" using `var1 || var2` results in "John Doe"--often an undesirable outcome for readable text.

To mitigate the issue of unwanted trailing blanks when using the `||` operator, programmers must often wrap each variable argument in the `TRIM()` function before concatenation. This adds complexity and verbosity to the code, which is why the specialized CAT functions are usually preferred. The primary advantage of the CAT functions (`CAT`, `CATS`, `CATT`, `CATX`) is their inherent ability to strip trailing and leading blanks from arguments before they are combined. This automatic cleansing dramatically simplifies the process of creating professional, neatly formatted output strings.

Furthermore, the CAT functions offer superior handling of numeric variables that are included in the list of arguments. If a numeric variable is passed to a CAT function, SAS automatically performs an implicit character conversion, whereas the `||` operator typically requires an explicit conversion function like `PUT`, adding yet another layer of required coding. For streamlined and robust string manipulation in SAS, the CAT function family is highly recommended, as they are designed to manage common data imperfections efficiently and effectively.

You can use the following highly efficient methods to quickly concatenate strings in SAS, leveraging the powerful CAT function family.

## Method 1: Using the CAT Function for Space Delimitation

The CAT function is the go-to utility when you need to combine multiple strings while ensuring that a single space is inserted between non-missing argument values. This function automatically removes leading and trailing blanks from the arguments before concatenation and inserts a blank separator. This behavior is incredibly useful for standard text concatenation tasks, such as merging a first name and a last name where a space is logically required for readability.

The syntax for the `CAT` function is straightforward: `CAT(argument-1, argument-2, ..., argument-n)`. Each argument can be a character variable, a numeric variable, or a quoted string literal. The function evaluates the arguments sequentially, strips their blanks, and combines them, inserting a space only when the resulting string is not null. It is important to note that if an argument is entirely missing (null or blank), the `CAT` function gracefully ignores it, preventing extraneous spaces in the output.

While effective for standard concatenation with a space delimiter, the `CAT` function is limited to using a space as its separator. For scenarios requiring no delimiter or a custom separator, alternative functions like `CATS` or `CATX` must be utilized. Nonetheless, for the vast majority of name and address formatting tasks in SAS, `CAT` offers the perfect balance of simplicity and functionality.

**new_variable = CAT(var1, var2);**

## Method 2: Using the CATS Function for Clean Concatenation

The `CATS` (Concatenate and Trim Space) function provides the cleanest form of string concatenation available in SAS. Unlike `CAT`, which introduces a space, `CATS` combines arguments without inserting any delimiter between them. This is achieved by stripping leading and trailing blanks from all arguments and then joining them immediately. This method is essential when the resulting text requires tight joining, such as creating unique keys, generating file paths, or merging codes where no separation is permitted.

For instance, if you are combining an area code (`"555"`) and a local number (`"1234"`), the `CATS` function will produce `"5551234"` without any intervening spaces or delimiters. This contrasts sharply with the concatenation operator (`||`), which would likely leave several trailing blanks from the source variables embedded in the middle of the result unless `TRIM()` was used extensively.

The superior blank handling of `CATS` makes it ideal for generating concise, machine-readable output. It ensures that the combined string is exactly the length of the constituent non-blank parts, providing high confidence in the output formatting, especially when dealing with data derived from disparate sources that might have inconsistent variable padding. The function's structure is analogous to `CAT`: `CATS(argument-1, argument-2, ..., argument-n)`.

**new_variable = CATS(var1, var2);**

## Method 3: Using the CATX Function for Custom Delimiters

The CATX function stands out as the most versatile concatenation tool in SAS, specifically designed for scenarios where a custom separator is required. CATX accepts three types of arguments: first, the specified delimiter; and second, the list of character arguments to be concatenated. Like `CAT` and `CATS`, it automatically trims leading and trailing spaces from all component strings, but it then inserts the designated delimiter between them.

The standard syntax is `CATX(delimiter, argument-1, argument-2, ...)`. The delimiter can be any character string, such as a hyphen (`"-"`), comma (`","`), colon (`":"`), or a combination of characters (e.g., `" / "`). Crucially, `CATX` only inserts the delimiter between arguments that are non-missing. If an argument in the sequence is missing or null, `CATX` skips that argument entirely and does not place a delimiter where the missing argument would have been, preventing double delimiters or separators at the beginning/end of the string.

This intelligent handling of missing data makes `CATX` indispensable for tasks like creating mailing addresses or constructing complex reporting titles, where fields may sometimes be empty. Using `CATX` ensures a clean, professional output regardless of the completeness of the source data, dramatically reducing the need for conditional logic (IF-THEN statements) that would otherwise be necessary to manage delimiters manually.

**new_variable = CATX("-", var1, var2);**

## Practical Implementation: Setting Up the Sample Dataset

To illustrate the functionality of the three primary concatenation functions, we will utilize a simple sample dataset containing basic user information. This initial setup is crucial for demonstrating how

variable concatenation works within the SAS Data Step environment. The dataset, named `my_data1`, includes three variables: `firstName` (character), `lastName` (character), and `points` (numeric). Although `points` is numeric, it helps illustrate that these functions are typically used for character manipulation.

We use the `DATALINES` statement within the Data Step to populate the dataset quickly. Note the use of the `$` sign following the variable names `firstName` and `lastName` in the `INPUT` statement, which explicitly defines them as character variables. The structure of this sample data ensures we have distinct first and last names that will be merged into a new, single variable in the subsequent examples.

After creating and populating the dataset, we use the `PROC PRINT` procedure to display the contents of the newly created table. This provides a baseline reference, allowing us to confirm the input data structure before any transformation occurs. Understanding the structure of the input variables is key to anticipating the output generated by the various string functions.

```
/*create dataset*/
data my_data1;
input firstName $ lastName $ points;
datalines;
Austin Smith 15
Brad Stevens 31
Chad Miller 22
Dave Michaelson 19
Eric Schmidt 29
Frank Wright 20
Greg Gunner 40
Harold Anderson 35
;
run;

/*view dataset*/
proc print data=my_data1;
```

| Obs | firstName | lastName | points |
|---|---|---|---|
| 1 | Austin | Smith | 15 |
| 2 | Brad | Stevens | 31 |
| 3 | Chad | Miller | 22 |
| 4 | Dave | Michaels | 19 |
| 5 | Eric | Schmidt | 29 |
| 6 | Frank | Wright | 20 |
| 7 | Greg | Gunner | 40 |
| 8 | Harold | Anderson | 35 |

## Example 1: Concatenating Strings with Space in Between (CAT)

This example demonstrates the primary usage of the CAT function, which is to combine character strings separated by a single blank space. We are creating a new dataset, `my_data2`, by reading `my_data1` and applying the `CAT` function to merge the `firstName` and `lastName` variables into a new variable called `fullName`. This is the most common requirement in data processing when preparing human-readable labels or report outputs.

The code `fullName = CAT(firstName, lastName);` is executed for every observation in the dataset. Because the `CAT` function automatically handles the trimming of trailing spaces from both `firstName` and `lastName`, we are guaranteed a clean output where the first name and last name are separated by exactly one space, regardless of the defined length of the input variables. This simplicity ensures code reliability and minimizes debugging related to unexpected white space.

When SAS creates the new variable `fullName`, it defaults the length of this character variable based on the lengths of the variables used in the concatenation. It is good practice, especially in production environments, to explicitly define the length of the new concatenated string using a `LENGTH` statement if the default length (often 200 bytes for CAT functions) is inefficient or too restrictive. However, for demonstration purposes, the implicit handling by the Data Step suffices to show the desired formatted output.

```
/*create new dataset with concatenated strings*/
data my_data2;
set my_data1;
fullName = CAT(firstName, lastName);
run;
```

```
/*view new dataset*/
proc print data=my_data2;
```

| Obs | firstName | lastName | points | fullName |
|---|---|---|---|---|
| 1 | Austin | Smith | 15 | Austin Smith |
| 2 | Brad | Stevens | 31 | Brad Stevens |
| 3 | Chad | Miller | 22 | Chad Miller |
| 4 | Dave | Michaels | 19 | Dave Michaels |
| 5 | Eric | Schmidt | 29 | Eric Schmidt |
| 6 | Frank | Wright | 20 | Frank Wright |
| 7 | Greg | Gunner | 40 | Greg Gunner |
| 8 | Harold | Anderson | 35 | Harold Anderson |

## Example 2: Concatenating Strings with No Space (CATS)

In situations where no space or delimiter is desired between combined strings, the `CATS` function provides the necessary precision. This example modifies the previous code to use `CATS`, demonstrating how to merge `firstName` and `lastName` into `fullName` without any separation. This is typically required when creating composite keys, email addresses, or internal identifier strings.

The crucial difference here is the function call: `fullName = CATS(firstName, lastName);`. Like `CAT`, `CATS` handles all necessary trimming of leading and trailing blanks from the input variables. However, after trimming, it joins the resulting strings directly end-to-end, resulting in a single continuous string. For instance, "Austin" and "Smith" become "AustinSmith".

This method is highly valuable when dealing with raw data feeds or system requirements that mandate specific, compact string formats. If we had attempted this using the `||` operator without the `TRIM` function, the output would be cluttered with internal padding spaces. `CATS` ensures the most concise and accurate string concatenation, making the resulting variable ideal for use in comparisons, indexing, or database key generation within SAS analysis.

```
/*create new dataset with concatenated strings*/
data my_data2;
set my_data1;
fullName = CATS(firstName, lastName);
run;

/*view new dataset*/
```

proc print data=my_data2;

| Obs | firstName | lastName | points | fullName |
|---|---|---|---|---|
| 1 | Austin | Smith | 15 | AustinSmith |
| 2 | Brad | Stevens | 31 | BradStevens |
| 3 | Chad | Miller | 22 | ChadMiller |
| 4 | Dave | Michaels | 19 | DaveMichaels |
| 5 | Eric | Schmidt | 29 | EricSchmidt |
| 6 | Frank | Wright | 20 | FrankWright |
| 7 | Greg | Gunner | 40 | GregGunner |
| 8 | Harold | Anderson | 35 | HaroldAnderson |

## Example 3: Concatenating Strings with Custom Delimiter (CATX)

The flexibility of string manipulation in SAS is best demonstrated by the CATX function, which enables the specification of any character or string as a separator. In this final practical example, we demonstrate how to use a hyphen (`-`) as a delimiter to combine `firstName` and `lastName`. This mimics common formatting requirements, such as generating login IDs or standardized composite identifiers.

The syntax for this operation is `fullName = CATX("-", firstName, lastName);`. The hyphen is passed as the first argument, defining the separator that will be placed between all subsequent non-missing arguments. As observed in the resulting output, "Austin" and "Smith" are joined to form "Austin-Smith".

The power of `CATX` lies in its robustness against missing data. If, hypothetically, `firstName` was missing for a record, `CATX` would not place the hyphen before `lastName`, ensuring the output is simply `lastName` (or `-lastName` if the delimiter was not handled correctly). By skipping the missing argument and placing the delimiter only between valid fields, `CATX` provides a clean, conditional concatenation that is superior to manual handling using the `||` operator combined with multiple `IF-THEN` checks. This functionality is essential for high-quality data reporting and formatting.

```
/*create new dataset with concatenated strings*/
data my_data2;
set my_data1;
fullName = CATX("-", firstName, lastName);
run;
```

```
/*view new dataset*/
proc print data=my_data2;
```

| Obs | firstName | lastName | points | fullName |
|---|---|---|---|---|
| 1 | Austin | Smith | 15 | Austin-Smith |
| 2 | Brad | Stevens | 31 | Brad-Stevens |
| 3 | Chad | Miller | 22 | Chad-Miller |
| 4 | Dave | Michaels | 19 | Dave-Michaels |
| 5 | Eric | Schmidt | 29 | Eric-Schmidt |
| 6 | Frank | Wright | 20 | Frank-Wright |
| 7 | Greg | Gunner | 40 | Greg-Gunner |
| 8 | Harold | Anderson | 35 | Harold-Anderson |

## Summary of String Concatenation Methods in SAS

Mastering string manipulation techniques is a cornerstone of effective SAS programming. While the traditional concatenation operator (`||`) offers basic combining capabilities, it is often hampered by its inability to automatically manage the trailing blanks inherent in fixed-length character variables, requiring additional functions like `TRIM` for clean results. This is why the specialized CAT function family is almost universally preferred by experienced SAS developers for robust, production-ready code.

The three main functions discussed--CAT, `CATS`, and CATX--provide specific solutions for virtually all string combining requirements. `CAT` excels when a single space delimiter is needed, `CATS` is perfect for tight, no-delimiter joining, and `CATX` provides unparalleled flexibility by allowing any custom separator while intelligently handling missing values. Choosing the correct function based on the required output format drastically simplifies the Data Step logic and improves code maintainability.

In summary, for any task involving combining character variables, always prioritize the CAT family of functions. They eliminate manual blank management and provide clean output, thereby enhancing data quality and programmer efficiency. By integrating these functions into your standard coding practices, you ensure that your string concatenation operations are performed accurately and reliably across diverse datasets.

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