How to Concatenate Strings in SAS: A Step-by-Step Guide to CAT, CATT, CATS, and CATX

Authored by

stats writer

November 19, 2025

Welcome to this comprehensive technical guide on the fundamental SAS Programming Procedures designed for robust string manipulation. Data analysts and programmers frequently encounter scenarios requiring the combination of multiple character variables into a single, cohesive string. In the SAS environment, this crucial task is handled by a family of highly efficient concatenation functions: **CAT**, **CATT**, **CATS**, and **CATX**. While seemingly similar, each function possesses unique properties regarding how they handle spacing and delimiters, making the choice dependent on the specific data cleaning requirements.

This resource offers a detailed, step-by-step examination of these four primary functions. We will explore the theoretical differences between them, provide practical coding examples to illustrate their application, and deliver expert guidelines to help you determine which function is optimally suited for various data preparation tasks. Mastery of these concatenation methods is vital for ensuring data integrity and producing clean, usable character fields for analysis and reporting.

Understanding String Concatenation Functions in SAS

In SAS, the process of combining two or more character strings into a single, longer string is known as concatenation. This operation is indispensable when merging descriptive fields, creating unique identifiers, or formatting output labels. The suite of CAT functions provides sophisticated control over how whitespace (blanks) is handled during this merger, which is often the most critical factor distinguishing the functions.

It is important to recognize that character variables in SAS often retain their defined length, meaning that unused space at the end of the data field is padded with trailing blanks. These trailing blanks can interfere with desired concatenation results, leading to unwanted gaps in the output string. The following list summarizes the core differentiation strategy employed by the **CAT** family of functions concerning whitespace management:

CAT: Designed for simple aggregation, the CAT function concatenates string variables exactly as they are defined, including all trailing and leading blanks present in the input strings.
CATT: This function specializes in cleaning up data by automatically removing only the **trailing spaces** from each input string before performing the concatenation operation.
CATS: The CATS function offers more aggressive cleaning by removing both **leading and trailing spaces** from the constituent strings prior to their combination.
CATX: Serving the most complex requirement, the CATX function also removes both leading and trailing spaces but critically allows the user to specify a custom delimiter (separator) to be placed between the concatenated strings.

Choosing the appropriate function depends entirely on whether you need to preserve existing spacing (often necessary for fixed-width data formats) or eliminate superfluous blanks (standard practice for creating readable labels or paths). We will now delve deeper into the specific syntax

and use cases for each function, culminating in a practical example demonstrating their distinct outputs.

Deep Dive: The CAT Function (Preserving All Blanks)

The **CAT** function represents the most straightforward approach to combining strings in SAS. Its primary purpose is additive: it takes the full content of the specified character variables and joins them sequentially. Crucially, it does not modify the whitespace associated with the input variables. This behavior means that any trailing spaces inherent to a variable's defined length will be included in the final concatenated string.

Consider a scenario where you have variables defined with fixed lengths, perhaps `Var1` (length 10) containing "Hello" and `Var2` (length 10) containing "World". When using `CAT(Var1, Var2)`, the resulting string will look like "Hello World " (where the spaces represent the padding up to the defined length 10). Because **CAT** includes these blanks, the resulting string often contains large, undesirable gaps.

While this feature might seem counterintuitive for general-purpose string creation, it is invaluable in highly specific programming contexts, such as debugging where the exact representation of source data buffers is required, or when working with legacy systems that depend on positional data fields defined by fixed lengths. Unless the preservation of existing spacing is a strict necessity, analysts typically opt for one of the trimming functions (CATT, CATS, or CATX) to produce cleaner output.

Mastering the CATT Function (Targeting Trailing Spaces)

The **CATT** function introduces the first layer of data cleansing into the concatenation process. The letter 'T' often signifies 'Trim Trailing' spaces. When this function is executed, SAS internally checks each argument passed to the function, identifies any trailing blanks that are merely padding the variable length, and effectively removes them before the strings are joined.

The ability of **CATT** to remove trailing whitespace is immensely useful, as trailing blanks are the most common source of unwanted gaps when combining character fields derived from raw data input or preceding data steps. By eliminating these padded spaces, **CATT** ensures that the resulting string is tightly packed, improving readability and reducing the overall length of the newly created variable compared to the output of **CAT**.

However, it is vital to remember that **CATT** focuses only on trailing spaces. If an input variable contains intentional or unintentional leading spaces (spaces preceding the actual value), **CATT** will preserve those leading spaces in the final concatenated result. For data fields that might contain alignment issues or user input errors resulting in leading blanks, a more comprehensive trimming function is necessary.

Leveraging the CATS Function (Comprehensive Space Removal)

The **CATS** function is often considered the default choice for general, clean string combination, as the 'S' typically stands for 'Strip' or 'Space Removal'. This function performs a complete cleanup of whitespace surrounding the value of each input variable. Specifically, CATS removes both **leading and trailing spaces** from all arguments before the concatenation occurs.

This aggressive trimming ensures that the resulting string contains only the meaningful characters of the combined variables, eliminating any gaps caused by variable padding or extraneous spaces at the beginning or end of the source data elements. For instance, if `VarA` contains " Value1 " (with leading and trailing spaces) and `VarB` contains "Value2 " (with only trailing spaces), `CATS(VarA, VarB)` will yield a contiguous string: "Value1Value2".

The **CATS** function is the preferred tool when the goal is maximum data compression and cleanliness, especially when the resulting string is intended for subsequent functions, database storage, or presentation where aesthetics matter. It is a powerful utility for standardizing data format across disparate fields.

Utilizing the CATX Function (The Power of Delimiters)

The **CATX** function is arguably the most versatile and frequently used of the concatenation family, adding the essential feature of a user-defined delimiter. The 'X' denotes 'eXplicit delimiter'. Like **CATS**, CATX automatically removes both leading and trailing spaces from all arguments. However, it requires the programmer to specify a character string (the delimiter) as its first argument, and this delimiter is inserted between every concatenated element.

The syntax for **CATX** is distinct: `CATX(delimiter, argument1, argument2, ...)`. The specified delimiter can be any character or string, such as a comma (,), a hyphen (-), a pipe (|), or a space. This capability is indispensable for creating structured output, such as full names (First Name, Last Name), file paths (Folder/Subfolder/File), or comma-separated value (CSV) lists.

A key advantage of the **CATX** function over manually inserting delimiters (e.g., using `VAR1 || '-' || VAR2`) is its intelligence in handling missing values. If an argument passed to **CATX** is entirely missing (null or blank after trimming), CATX will not place the delimiter immediately adjacent to the missing value's space. This prevents redundant or dangling delimiters, resulting in much cleaner output, particularly when dealing with incomplete data rows.

Practical Demonstration: Setting up the Sample Dataset

To fully appreciate the differences between **CAT**, **CATT**, **CATS**, and **CATX**, we must observe how they process data with inherent spacing issues. We will create a sample dataset containing three

character variables: `player`, `team`, and `conf`. These variables are intentionally defined such that they contain implicit trailing spaces due to the way SAS handles character variable storage during the data input process.

The following code initiates the data step, defining the structure and populating the rows. Pay close attention to how the input format (`$`) implies character strings that may be padded to a default length, setting the stage for the concatenation nuances we are exploring. This small but representative dataset allows for a clear visual comparison of the results produced by the four functions.

Here is the SAS code used to construct the initial data structure:

```
/*create dataset*/
data my_data;
input player $ team $ conf $;
datalines;
Andy Mavs West
Bob Lakers West
Chad Nuggets West
Doug Celtics East
Eddy Nets East
;
run;

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

The execution of the `PROC PRINT` step confirms the input data structure, illustrating the distinct columns before any transformation occurs. This view is crucial for establishing the baseline lengths and recognizing the presence of any implicit trailing spaces that SAS maintains for each character column.

Obs	player	team	conf
1	Andy	Mavs	West
2	Bob	Lakers	West
3	Chad	Nuggets	West
4	Doug	Celtics	East
5	Eddy	Nets	East

Applying the Four Concatenation Functions

Our objective now is to apply **CAT**, **CATT**, **CATS**, and **CATX** simultaneously to the `player`, `team`, and `conf` variables within a new data step. By creating four new variables (`cat`, `catt`, `cats`, and `catx`) side-by-side, the resulting output will dramatically highlight the operational differences between the functions, particularly concerning whitespace management.

For the **CATX** function, we have selected the hyphen (`-`) as our explicit delimiter. This choice is arbitrary but effective in visually separating the combined components. Note how the syntax requires the delimiter string to be the first argument, followed by the variables to be concatenated.

The following code block executes the core string manipulation logic:

```
/*create new dataset that concatenates columns*/
data new_data;
set my_data;
cat = cat(player, team, conf);
catt = catt(player, team, conf);
cats = cats(player, team, conf);
catx = catx('-', player, team, conf);
run;

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

Upon execution, we generate a new dataset, `new_data`, which includes the original input variables alongside the four newly generated concatenated columns. Reviewing the `PROC PRINT` output immediately following this step provides the conclusive evidence necessary to differentiate the functional behaviors.

Obs	player	team	conf	cat	catt	cats	catx
1	Andy	Mavs	West	Andy Mavs West	AndyMavsWest	AndyMavsWest	Andy-Mavs-West
2	Bob	Lakers	West	Bob Lakers West	BobLakersWest	BobLakersWest	Bob-Lakers-West
3	Chad	Nuggets	West	Chad Nuggets West	ChadNuggetsWest	ChadNuggetsWest	Chad-Nuggets-West
4	Doug	Celtics	East	Doug Celtics East	DougCelticsEast	DougCelticsEast	Doug-Celtics-East
5	Eddy	Nets	East	Eddy Nets East	EddyNetsEast	EddyNetsEast	Eddy-Nets-East

Analyzing the Concatenation Results

Observing the output table is critical for understanding why and when to choose each function. The horizontal spread of the columns visually represents the impact of whitespace treatment on the final string length and appearance.

A careful examination of the resulting variables reveals distinct patterns of string formation, confirming the specific rules governing each function:

The CAT Output: The `cat` variable clearly demonstrates the inclusion of all trailing blanks from the original `player`, `team`, and `conf` variables. The resulting string is exceptionally long and contains large, noticeable gaps between the values (e.g., "Andy Mavs West"). This makes it generally unsuitable for human-readable output but confirms its role in preserving source data fidelity.
The CATT Output: The `catt` variable shows significant improvement over **CAT**, as the trailing spaces have been removed from each component before combination. Assuming the input data had no leading spaces, the result is tightly concatenated (e.g., "AndyMavsWest"), but lacks any natural separation.
The CATS Output: The `cats` variable is identical to the `catt` output in this specific scenario because the input variables lacked any leading spaces. If leading spaces had been present, **CATS** would have removed them, resulting in the cleanest possible combination without an external separator.
The CATX Output: The `catx` variable provides the most readable and structured output. It performed the same full trimming (leading and trailing space removal) as **CATS**, but then inserted the specified hyphen (`-`) between the non-blank components (e.g., "Andy-Mavs-West"). This result is highly suitable for generating structured identifiers or compound variables.

Conclusion and Best Practices for String Manipulation

The decision among the **CAT** family of functions hinges entirely on two critical factors: the requirement for space preservation and the necessity for a delimiter. Data scientists must weigh the need for data cleaning against the functional requirement of the output variable.

If the goal is to produce a single, professional, and easily parseable string--which is the case for the vast majority of analytical tasks--you should almost always default to the **CATX** function. CATX handles necessary trimming (leading and trailing spaces) and ensures logical separation of components, while intelligently managing missing values to prevent dangling delimiters.

Conversely, **CAT** should be reserved only for highly specialized tasks where every byte of storage, including padding, must be maintained. **CATT** is useful if you are certain your input data contains only trailing, but never leading, extraneous spaces. By understanding the subtle yet powerful differences in how these functions manage whitespace, SAS programmers can write more

efficient, robust, and error-resistant data manipulation code. When working with these concatenation functions on your own data, always select the one most suitable for your specific data cleaning and formatting situation.

Further SAS Tutorials

To continue advancing your proficiency in data preparation and manipulation within the SAS environment, consider exploring other specialized character functions and data steps. Mastery of string utilities is a foundational skill for effective data governance and analysis.

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

How to handle date and time formats in the DATA step.
Techniques for using `SUBSTR` and `INDEX` functions for complex string searching.
Methods for merging and joining datasets using `PROC SQL` or the DATA step.