Learning SAS: Mastering String Concatenation with CAT, CATT, CATS, and CATX Functions

Authored by

Mohammed looti

November 15, 2025

Mastering Character Manipulation: The Essential SAS Concatenation Functions

In the demanding environment of SAS programming, the efficient handling and manipulation of textual data are crucial for everything from routine data cleansing to sophisticated analytical reporting. A fundamental requirement in this process is combining or joining multiple text fields, an operation universally known as concatenation. While the concept of linking strings appears straightforward, practical implementation frequently encounters a significant challenge: managing inherent, unwanted whitespace--specifically leading and trailing spaces--that often accompanies character data fields sourced from fixed-length files or databases. To solve this complexity with precision and efficiency, SAS provides a highly specialized and powerful suite of character functions: **CAT**, **CATT**, **CATS**, and **CATX**.

These four primary functions are indispensable tools for any experienced SAS developer, as each is engineered to execute concatenation while applying a distinct, specific strategy for addressing extraneous blanks. Understanding the nuanced behavior of each function is not merely helpful--it is critical. Choosing the wrong function can easily lead to "messy" data plagued by excessive internal spacing, which can complicate subsequent processing and reduce readability, or, conversely, result in strings that are unintentionally compressed without necessary separators. This comprehensive guide aims to thoroughly demystify these differences, providing clear explanations of their syntax, internal trimming mechanisms, and optimal practical applications in complex data preparation tasks.

We will systematically explore how the base function, **CAT**, executes a raw, unmodified join that preserves all spacing; how **CATT** offers a targeted approach by efficiently eliminating only trailing blanks; how **CATS** delivers a truly clean, space-free combination by removing both leading and trailing whitespace; and finally, how **CATX** elevates the process entirely by combining robust trimming with the intelligent insertion of a customizable delimiter. Upon concluding this article, you will possess the foundational knowledge required to confidently select and implement the most appropriate SAS string function for any character manipulation challenge your data may present.

The CAT() Function: Raw Joining and Space Preservation

The **CAT()** function serves as the bedrock of character combination within the SAS language, executing the most fundamental form of concatenation. Its defining operational characteristic is its absolute commitment to preservation: it joins the input arguments exactly as they are defined in memory, including any embedded, leading, or trailing spaces associated with fixed-length character fields. This raw joining mechanism means that if an input string variable is allocated a length of 50 characters but only contains 10 characters of actual data, the remaining 40 trailing spaces are carried forward and preserved within the resulting concatenated string.

The function's syntax is remarkably straightforward: `CAT(argument-1, argument-2, ..., argument-n)`. Every argument provided must evaluate to a character value, whether it is a defined variable name, a literal string enclosed in quotes, or the result of a more complex character expression. While the simplicity of **CAT()** makes it easy to implement, developers must remain acutely aware of its implications regarding variable length. If the combined total length of the resulting string exceeds the defined length of the target variable, SAS will silently truncate the output from the right side. This lack of warning can potentially lead to critical data loss or corruption if the target variable length is not meticulously managed and adequately defined prior to the concatenation step.

A vital distinction of **CAT()**, a behavior shared by **CATT()** and **CATS()**, is its specific handling of missing data. When **CAT()** encounters an input argument that is a missing character value (or a numeric value that is automatically converted to a missing character representation), it treats that argument as if it were a zero-length string. Crucially, the missing value is ignored entirely during the joining process. This behavior ensures that **CAT()** will never insert a space or any placeholder character where a missing value occurred, which is a key difference from other programming environments and a behavior that might be unexpected if the programmer was relying on the original variable definition's padding to provide necessary separation between non-missing components.

The CATT() Function: Precision Trimming of Trailing Blanks

When the core concern shifts from raw preservation to the targeted cleanup of extraneous padding found at the end of character fields, the **CATT()** function offers an elegant and precise solution. **CATT()** executes concatenation only after performing an internal operation that removes all trailing spaces from each individual argument provided to the function. This specific functionality is invaluable in data environments, particularly within SAS, where source data fields are commonly defined with fixed lengths, leading inevitably to undesirable gaps and excessive spacing when those fields are joined together using the standard **CAT()** function.

Structurally, **CATT()** adheres to the identical basic syntax as **CAT()**: `CATT(argument-1, argument-2, ..., argument-n)`. However, the internal mechanism preceding the actual joining differs significantly. Before combining the arguments, **CATT()** applies an automatic trimming process solely to the right-hand side of each input string variable. This action ensures that components are joined immediately following their last non-blank character, effectively bridging the internal gaps that are typically created by fixed-length padding. The resulting string is invariably cleaner and significantly more readable than its **CAT()** counterpart, but critically, any existing leading spaces (blanks at the beginning of the string) remain untouched.

Consider a typical scenario in a DATA step: merging an employee's first name and last name,

where the first name variable is defined as `CHAR(30)`. If the actual first name is "John", 26 trailing spaces follow that text. Using **CAT()** would result in a massive gap: "John Doe". Conversely, using **CATT()** automatically removes those 26 trailing spaces, yielding the compact string "JohnDoe" (or "John Doe" if a space literal is deliberately inserted between the arguments). This efficiency makes **CATT()** highly suitable for preliminary data standardization when only the right-hand padding, a common artifact of data importing, needs to be addressed and eliminated.

The CATS() Function: Achieving Universal Cleanliness

For the vast majority of general-purpose data standardization and text formatting tasks in SAS, the **CATS()** function is widely regarded as the preferred and most reliable choice. It provides the most aggressive and comprehensive whitespace removal among the non-delimited concatenation functions, actively eliminating both leading and trailing spaces from every single argument before those components are combined. This operation results in the most compact string possible, ensuring the final output is entirely free from extraneous whitespace that could negatively affect subsequent analysis, comparisons, or display formatting.

The function's syntax remains consistent with its siblings: `CATS(argument-1, argument-2, ..., argument-n)`. The key internal difference, however, is its trimming scope. Unlike **CATT()**, which only focuses on the right-hand padding, **CATS()** performs what is essentially a full, symmetrical trim on each input component. This makes it exceptionally effective when dealing with data sources such as free-form user input fields or data imported from external, less controlled systems where inconsistent spacing, including unintentional leading spaces before the meaningful content, is a pervasive issue. By rigorously cleaning both the beginning and the end of the input string variables, **CATS()** guarantees that only the core, essential characters of each component are joined together.

The strategic use of **CATS()** significantly streamlines the DATA step coding process by inherently eliminating the need for iterative application of manual trimming functions like `TRIM()` or `LEFT()` immediately prior to joining strings. If the task involves combining address components, product descriptions, or any textual data where maximal cleanliness and compactness are desired without the intentional addition of a separator, **CATS()** is the most efficient and robust solution. Consistent with the other CAT functions, **CATS()** treats missing arguments as zero-length strings, ensuring they do not introduce any unwanted characters, including single spaces, into the final output string.

The CATX() Function: Structured Output with Intelligent Delimiters

The **CATX()** function stands as the most flexible and sophisticated tool for character concatenation in SAS, specifically engineered for scenarios that demand structured and standardized output. **CATX()** performs two critical operations simultaneously that make it uniquely powerful: first, it

automatically removes both leading and trailing blanks from every input string variable (mimicking the comprehensive cleanup of **CATS()**); and second, it inserts a user-specified delimiter between the concatenated arguments.

Due to the inclusion of the required separator, the syntax for **CATX()** differs slightly from the others: `CATX(delimiter, argument-1, argument-2, ..., argument-n)`. The `delimiter` must be provided as the very first argument and can be any character literal, such as a comma (`','`), a hyphen (`'-'`), a vertical bar (`'|'`), or even a space (`' '`). The true architectural strength of **CATX()** lies in its intelligent handling of missing or empty arguments. Unlike manual concatenation methods that might inadvertently result in double delimiters when data is sparse (e.g., `'A,,B'`), **CATX()** only inserts the specified delimiter between arguments that are confirmed to be non-missing and non-empty after the trimming process. If an argument is missing or contains only blanks, **CATX()** skips both the argument and the delimiter that would have been required to separate it, thereby preventing sequences of unwanted separators and guaranteeing a clean, logically structured string.

This intelligent, self-cleaning behavior makes **CATX()** absolutely indispensable for tasks requiring standardized output, such as generating comma-separated value (CSV) lists, dynamically constructing file paths using slashes, or synthesizing composite keys used for merging and joining disparate datasets. Whether the goal is formatting a human-readable address line or creating a clean, machine-readable identifier, **CATX()** reliably guarantees a clean, delimited string without the need for complex, manual conditional logic to manage and exclude missing components.

Practical Demonstration: Comparing the Four Functions

To clearly and definitively illustrate the operational differences between **CAT**, **CATT**, **CATS**, and **CATX**, let us examine a practical, working example using a small SAS dataset. We begin the demonstration by creating sample data that intentionally contains potential spacing issues due to the way variables are defined and populated. Our objective is to concatenate three fixed-length character fields: `player`, `team`, and `conf`.

```
/*create dataset with implicit trailing spaces*/
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 to confirm structure*/
proc print data=my_data;
```

The initial DATA step defines these variables as character types (indicated by the `$` sign). When data is read using the `DATALINES` statement without explicit length specifications, SAS assigns a default length based on the longest value encountered in the input, inevitably padding shorter values with trailing spaces. The subsequent PROC PRINT statement confirms the initial structure of `my_data`, which visually demonstrates how variable padding introduces these implicit blanks that must be managed.

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

Next, we apply all four concatenation functions within a new DATA step. We create four new string variables (`cat`, `catt`, `cats`, and `catx`) that clearly illustrate the distinct output behaviors when joining the `player`, `team`, and `conf` fields.

```
/*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;
```

Note the essential difference in the definition of `catx`, where the hyphen (`'-'`) is specified as the mandatory first argument, acting as the delimiter that instructs SAS to insert this character between

the components. The final PROC PRINT command displays the combined results, offering a direct, side-by-side comparison of the four resulting variables and powerfully highlighting the impact of each function's unique space-handling strategy.

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 resulting output variables clearly demonstrates the specific operational behaviors:

The `cat` variable exhibits significant internal spacing because **CAT()** meticulously preserved all the original trailing spaces from the fixed-length `player` and `team` variables, resulting in an overly padded string (e.g., "AndyMavsWest").

The `catt` variable shows greatly reduced spacing. **CATT()** successfully removed the trailing spaces from each component before joining, making the result far more compact than `cat`, though any original leading spaces (if present) would still be retained.

The `cats` variable produces the most condensed non-delimited result. **CATS()** removed both leading and trailing blanks from all arguments, creating a tightly joined string like "AndyMavsWest" (assuming no intentional space literals were added between arguments).

The `catx` variable demonstrates the utility of structured output. **CATX()** performed the same comprehensive trimming as **CATS()** but then intelligently inserted the hyphen delimiter between the non-missing components, resulting in the clean structure "Andy-Mavs-West".

Choosing the Right Tool for the Job

Selecting the appropriate SAS string function for character concatenation is entirely dependent on the desired output format and the necessary level of control over whitespace. Each function is finely optimized for a specific scenario, and recognizing these distinct capabilities is the fundamental key to writing efficient, robust, and maintainable SAS code.

When should you opt for **CAT()**? This function should be reserved exclusively for highly specialized cases where preserving the exact allocated length and position of every single character, including all fixed-width padding, is absolutely critical. If your subsequent data processing steps rely on specific column alignments, fixed-width file output, or internal spacing requirements, **CAT()** is the

only choice, as it guarantees a raw, unmodified join of the input string variables.

Opt for **CATT()** primarily when you are dealing with character variables defined by fixed lengths that introduce unwanted trailing padding. If your data structure may contain intentional leading spaces that must be retained for meaning, but the variable's trailing whitespace needs aggressive cleanup, **CATT()** provides the necessary targeted trimming without affecting the beginning of the strings.

Employ **CATS()** as the general-purpose, default function for achieving the cleanest possible, non-delimited string output. If the requirement is to combine textual elements and eliminate any possibility of accidental leading or trailing blanks without introducing a separator, **CATS()** streamlines the process by handling the full trim automatically. This is essential for standardizing text fields before performing comparisons or preparing data for general display.

Finally, choose **CATX()** when structured, delimited output is the absolute priority. Whether the requirement is to generate clear, human-readable lists, construct hierarchical file paths using slashes, or create structured database keys, **CATX()** is indispensable. Its powerful combination of aggressive whitespace removal and the intelligent insertion of a customizable delimiter solely between non-missing arguments makes it the most versatile function for complex output formatting.

Conclusion and Additional Resources

The specialized **CAT**, **CATT**, **CATS**, and **CATX** functions represent foundational elements of the SAS language, providing programmers with precise, granular control over character concatenation and crucial whitespace management. By developing a clear understanding of the unique trimming behavior and intelligent delimiter capabilities of each function, programmers can efficiently transform raw, often messy data into clean, structured, and report-ready textual components. Mastering these four core tools ensures data integrity and substantially reduces the complexity typically associated with comprehensive string manipulation tasks in SAS.

To further enhance your data processing expertise, we strongly encourage reviewing the official SAS documentation, which offers deep dives into these and many other essential string functions. Continuous exploration of these core features will solidify your ability to handle complex data preparation tasks with absolute confidence and maximal efficiency.

Additional Resources

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