Learning to Convert Character Variables to Date Variables in SAS

Authored by

Mohammed looti

October 31, 2025

Introduction to Date Handling in SAS

Handling temporal data correctly is a cornerstone of effective statistical programming, and within the SAS environment, this process requires careful attention to data types. Unlike most programming languages that might store dates as complex strings or objects, SAS fundamentally stores every date variable as a numeric value representing the number of days elapsed since January 1, 1960. This internal numeric representation is efficient for calculations (such as finding the difference between two dates) but poses a challenge when raw data is imported. Often, dates arrive in the form of a **character variable**--a string of numbers or text--that SAS cannot immediately interpret for mathematical operations. Consequently, a crucial step in data preparation involves converting this character string into the recognized SAS date format.

The necessity for this conversion arises because an unformatted character string, even if it looks like a date (e.g., '20231026'), is treated by SAS merely as text. If you attempt to sort, filter, or perform date arithmetic on character data, the results will be logically incorrect. For instance, character sorting might place '01/10/2023' before '12/01/2022' because it compares the first digit '0' to '1'. To enable true temporal integrity, we must utilize specialized functions that instruct SAS how to interpret the exact structure of the character input and translate it into its native numeric date format. This translation is primarily accomplished using the powerful **input() function**, paired with the appropriate informat specification.

This guide focuses specifically on utilizing the **input() function** within a SAS Data Step to achieve this transformation seamlessly. Mastering this technique ensures that your data is correctly categorized, thereby unlocking the full potential of SAS procedures for time-series analysis, aging reports, and sequential data processing. The transition from raw, messy character data to clean, calculable date variables is essential for maintaining data quality and ensuring the reliability of downstream analysis.

The Essential SAS INPUT() Function

The core mechanism for converting a **character variable** to a numeric or date variable in SAS is the **input() function**. This function is vital for reading raw data values using an explicit instruction set known as an **informat**. An informat acts as a blueprint, telling SAS exactly how the characters in the string are arranged (e.g., is it Month-Day-Year, or Year-Month-Day?) and how long the string is. Without a matching informat, SAS cannot correctly parse the character data into the internal numeric representation it requires for date processing.

The structure of the **input() function** is straightforward yet precise. It requires two primary arguments: the source character variable you wish to convert, and the specific informat that matches the structure of that character data. The result of this function is a new numeric value--the SAS date--which must then be assigned to a new variable. It is a common mistake for users to

forget the second step required to make the output human-readable, which is applying a **format**, discussed in the next section.

The basic syntax used for date conversion is demonstrated below. This syntax illustrates how the function is used within a Data Step to create the new date variable, followed immediately by the required formatting statement to ensure proper display.

date_var = input(character_var, MMDDYY10.);
format date_var MMDDYY10.;

In this structure, the `MMDDYY10.` specification serves two distinct roles: first, as the **informat** inside the `INPUT()` function, guiding the conversion; and second, as the **format** in the subsequent `FORMAT` statement, ensuring the resulting numeric date is displayed in a recognizable Month/Day/Year structure. Understanding the difference between these two concepts is fundamental to successful date manipulation in SAS.

Understanding SAS Date Formats and Informats

A key concept often confusing to new SAS users is the distinction between **informats** and **formats**. While they often share similar naming conventions (e.g., `MMDDYY10.`), they serve opposing purposes. An **informat** is the instruction used by SAS to read data--it dictates how the external or character data should be read and converted into the internal numeric format. Conversely, a **format** is an instruction used by SAS to display data--it dictates how the internal numeric date value should be presented back to the user (e.g., as '01/15/2022' or '15-JAN-2022').

The specific informat chosen, such as **MMDDYY10.**, must precisely match the structure and width of the incoming character string. In the case of `MMDDYY10.`, the '10' denotes the total width of the character string being read, implying a structure where two digits represent the month, two the day, and four the year, plus two optional separators (like slashes or hyphens, though in our example, the input is continuous). If the input data were structured differently--say, Year-Month-Day with hyphens (e.g., '2022-01-15')--we would need a different informat, such as `YYMMDD10.`, to ensure accurate interpretation. Using the wrong informat is the most common cause of date conversion failure, resulting in missing values or incorrect dates.

Once the conversion via the `INPUT()` function and informat is complete, the resulting numeric variable holds the count of days since January 1, 1960. If you were to print this variable without applying a **format**, you would see a large integer (e.g., 22650). This number is meaningless to a human reader but is essential for SAS calculations. The `FORMAT` statement is therefore mandatory for usability. SAS offers a wide array of date formats--including `DATE9.` (e.g., 15JAN2022), `DDMMYY8.`, and `MMDDYY10.`--allowing analysts to choose the display style best suited for reporting

or visualization purposes. A complete understanding of the available formats is crucial for professional SAS usage.

Step-by-Step Example: Converting Character Data to Date Format

To solidify the concepts of conversion, let us work through a practical example using a small retail dataset. Suppose we have compiled sales data where the transaction date was initially logged as a continuous character string without any separators. This raw data structure, while concise, prevents immediate analysis. The following code demonstrates the creation of our initial **SAS dataset**, `original_data`, containing the day as a character string and the corresponding sales volume.

The variable `day` is defined using the dollar sign (`$`) in the `INPUT` statement, explicitly designating it as a character type. Notice that the values, such as `01012022`, represent January 1, 2022, following the MMDDYYYY structure. Our goal is to transform this string into a numeric date field, making it compatible with SAS date functions.

```
/*create dataset*/
data original_data;
input day $ sales;
datalines;
01012022 15
01022022 19
01052022 22
01142022 11
01152022 26
01212022 28
;
run;

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

After executing this code, the output generated confirms that the `day` variable is indeed a **character variable**. This is evident because SAS aligns character data to the left in the output window. More importantly, attempting to use this variable in any temporal calculation would fail or yield erroneous results. The image below illustrates the structure of our initial dataset, clearly showing the text-based nature of the date field.

Obs	day	sales
1	01012022	15
2	01022022	19
3	01052022	22
4	01142022	11
5	01152022	26
6	01212022	28

Our next step involves creating a new dataset, `new_data`, where we apply the conversion logic. We will use the `SET` statement to read records from `original_data` one by one. For each record, the `INPUT()` function will be used to read the character string in `day` and interpret it using the `MMDDYY10.` informat, storing the resulting numeric date in a new variable called `new_day`. Following the conversion, the `FORMAT` statement ensures that `new_day` is displayed correctly, and finally, the `DROP` statement cleans up the dataset by removing the superfluous original character variable.

Analyzing the Conversion Results

The transformation process is executed within a second Data Step, meticulously converting the text field into a functional date type. This is the heart of the solution, where the **input() function** performs its primary task. We leverage the `MMDDYY10.` informat because our character data is 8 digits long and structured as Month-Day-Year. The inclusion of '10' in the informat, even though our data only uses 8 digits, is acceptable; SAS handles the lack of separators gracefully, interpreting the first two digits as Month, the next two as Day, and the final four as Year.

The code below outlines the complete conversion process, including the application of the display format and the removal of the redundant character column. Note the critical use of the `FORMAT` statement: `format new_day MMDDYY10.;`. This line is what translates the internal numeric date (e.g., 22631) back into the visually familiar date (e.g., 01/01/2022). Without this statement, the output would be correct internally but unusable for reporting.

```
/*create new dataset where 'day' is in date format*/
data new_data;
set original_data;
new_day = input(day, MMDDYY10.);
format new_day MMDDYY10.;
drop day;
run;
```

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

The resultant dataset, shown in the image below, confirms the successful conversion. The new variable, `new_day`, is now properly formatted and aligned to the right, indicating its numeric nature. Furthermore, the values are displayed in the recognizable date format specified by **MMDDYY10.**. This transformed variable is now ready for any necessary statistical computations, such as calculating elapsed time, determining the day of the week, or grouping sales data by specific time periods. The use of the **drop** statement ensured a clean output, removing the now-obsolete character column and preventing potential confusion in future analyses.

Obs	sales	new_day
1	15	01/01/2022
2	19	01/02/2022
3	22	01/05/2022
4	11	01/14/2022
5	26	01/15/2022
6	28	01/21/2022

Advanced Considerations and Best Practices

While the `INPUT()` function provides a robust solution for conversion, practitioners should be aware of several advanced considerations to handle real-world data complexities. One critical area is error handling. If the incoming character string contains non-date characters (e.g., '01/XX/2022') or represents an impossible date (e.g., February 30th), the `INPUT()` function will automatically assign a missing value (represented as a dot '.') to the resulting date variable. For robust production code, it is best practice to include an `IFC( )` function check or use the `PUT()` function to debug why specific observations failed conversion, allowing analysts to identify and clean up problematic source data.

For data sources where the date format might be inconsistent across records--for example, some dates use slashes ('01/01/2022') and others use hyphens ('01-01-2022')--relying on a single informat like `MMDDYY10.` might prove insufficient. In such scenarios, SAS provides more flexible functions. For instance, the `ANYDTDTE.` informat is designed to intelligently parse various date formats, making it highly valuable when dealing with heterogenous or messy source files. Although `ANYDTDTE.` provides flexibility, it often comes with a slight performance cost compared to explicitly defined informats, so choosing the most specific informat remains the best practice when data

structure is known and consistent.

Finally, always remember the fundamental data type rule: character variables must be read using an informat to produce a numeric variable (which is how SAS dates are stored), and that numeric variable must then be assigned a format to be human-readable. Failing to apply the format means the data is mathematically sound but visually opaque. Furthermore, when dealing with dates that include time components, specialized informats like `DATETIME.` or `ANYDTDTM.` must be employed, as the SAS internal representation for datetime variables counts seconds since January 1, 1960, rather than days.

Additional Resources

Mastering date conversion is just one skill necessary for advanced data manipulation in SAS. The following resources offer further guidance on related data management tasks and advanced programming techniques within the SAS environment:

How to Handle Missing Values in SAS
Using PROC SQL for Data Aggregation in SAS
Understanding the SAS Data Step vs. PROC Steps
Applying Conditional Logic with IF-THEN/ELSE Statements