Learning How to Convert Datetime Variables to Date in SAS

Authored by

Mohammed looti

May 13, 2026

Working efficiently with temporal data is fundamental in modern data analysis, and the SAS system provides a powerful suite of functions to manage these complex data types. A requirement encountered frequently by data professionals is the need to isolate the date component from a comprehensive datetime variable, effectively discarding the time stamp. This conversion is critically important for standardizing data for aggregate analyses, such as grouping records by day, filtering specific daily events, or simply ensuring data presentation is clean and easily readable.

The standard, most reliable, and highly recommended approach for executing this conversion within the SAS environment involves leveraging the built-in **DATEPART** function. This function serves as the definitive tool for extracting the pure date component from any SAS datetime value, making it an indispensable part of the data scientist's toolkit when manipulating temporal data.

The Necessity of Converting Datetime Data

In many real-world datasets, especially those derived from transactional systems, sensors, or logs, time information is recorded down to the second or millisecond. While this precision is valuable for auditing, it often becomes cumbersome or irrelevant when analyzing trends over longer periods. For example, if you are attempting to calculate daily sales totals or track monthly user activity, the specific minute and second of an event provide noise rather than insight.

The process of converting datetime to date streamlines data for these analytical tasks. By converting the time-based data into a standardized date format, data records can be accurately aligned and grouped. This is essential for generating accurate summary reports, performing time-series comparisons, and ensuring that statistical models are based on consistent daily observations rather than minute-by-minute fluctuations.

Furthermore, managing date values rather than datetime values significantly simplifies data management and storage. The resulting date variable requires less computational overhead and is far more intuitive for non-technical stakeholders reading reports. Therefore, mastering the efficient extraction of the date component is a core skill for any professional working with large-scale data in SAS.

Internal Representation: How SAS Stores Temporal Values

To effectively utilize conversion functions like **DATEPART**, it is vital to understand the underlying structure of SAS date and datetime values. Unlike simple text strings, both are stored as numeric variables relative to a fixed epoch, or reference point: January 1, 1960.

A SAS datetime value is fundamentally the total number of seconds that have elapsed since January 1, 1960, at midnight. This second-level granularity means that a single datetime entry, such as '14OCT2022:14:30:00', is internally stored as a very large integer corresponding to the

total seconds passed since the epoch. This provides the necessary precision for detailed event tracking.

In contrast, a SAS date value simplifies this by representing the total number of days that have passed since January 1, 1960. It focuses only on the day, month, and year, completely ignoring the time component. When the **DATEPART** function is executed, it performs the mathematical conversion from the seconds-based count (datetime) to the days-based count (date), effectively zeroing out the time information.

Introducing the DATEPART Function: Syntax and Purpose

The DATEPART function is specifically engineered to extract the date portion from a SAS datetime value, returning a numeric SAS date value. Its simplicity is its strength: it takes one argument--the datetime variable--and returns the corresponding date value, stripped of all time-of-day data.

While the **DATEPART** function successfully performs the conversion, the output itself is a numeric integer (the number of days since January 1, 1960), which is not typically suitable for reporting or direct human consumption. This is why **DATEPART** is almost always used in conjunction with the **PUT** function, which is responsible for applying a human-readable SAS format, such as MM/DD/YYYY, to the resulting numeric date value.

The core structure of this conversion, typically implemented within a SAS DATA step, is highly standardized. It ensures that the calculation (extracting the date) and the presentation (formatting the date) are handled simultaneously in an efficient manner.

date = put(datepart(some_datetime), mmddyy10.);

In this fundamental command, `some_datetime` is your source datetime variable. **DATEPART** extracts the numeric date integer, which is then passed immediately to the **PUT** function. The format `mmddyy10.` then converts this numeric integer into a character string that displays as a date.

Combining DATEPART and PUT: Mastering Date Formatting

The synergy between **DATEPART** and the PUT function is what transforms raw numeric data into meaningful dates. While **DATEPART** handles the mathematical truncation, the **PUT** function dictates the aesthetic and structural presentation of the output, which is crucial for data reporting.

The **PUT** function requires two main arguments: the numeric value to be formatted (the output of **DATEPART**) and the desired SAS format. Failing to include **PUT** means SAS will simply print the raw integer count, which is rarely the desired outcome for final analysis or presentation.

The flexibility of SAS is highlighted by the wide array of date formats available, allowing users to customize the display based on regional conventions or internal standards. Some of the most frequently used date formats include:

`mmddyy10.`: This format presents the date in the common U.S. style of MM/DD/YYYY (e.g., 10/15/2022). The specified width, `10`, ensures there is enough space to accommodate the date separators.
`yymmdd10.`: Often favored for international use or for datasets intended for sorting, this format yields YYYY-MM-DD (e.g., 2022-10-15).
`date9.`: A classic and highly readable SAS format that displays the date as DDMMMYYYY (e.g., 15OCT2022).
`ddmmyy10.`: Standardized for many European regions, this format arranges the date as DD/MM/YYYY (e.g., 15/10/2022).

The selection of the correct format determines the appearance of the resulting variable. It is important to note that when **PUT** is used to create a new variable, that variable becomes a character string, which may affect subsequent numeric calculations.

A Step-by-Step Practical Implementation in SAS

To demonstrate the versatility of the **DATEPART** function, let us examine a complete working example. We start with a SAS dataset named `original_data` that contains a datetime column, `some_datetime`, which we intend to transform into several date formats.

First, we create and populate the initial dataset:

```
/*create dataset*/
data original_data;
format some_datetime datetime23.;
input some_datetime :datetime23.;
datalines;
14OCT2022:0:0:0
09NOV2022:0:0:0
14NOV2022:0:0:0
15NOV2022:0:0:0
22DEC2022:0:0:0
24DEC2022:0:0:0
04JAN2023:0:0:0
;
run;
```

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

The execution of this first code block establishes the data and provides an initial view of the `some_datetime` column, confirming the presence of the full datetime values:

Obs	some_datetime
1	14OCT2022:00:00:00
2	09NOV2022:00:00:00
3	14NOV2022:00:00:00
4	15NOV2022:00:00:00
5	22DEC2022:00:00:00
6	24DEC2022:00:00:00
7	04JAN2023:00:00:00

Next, we proceed to the conversion within a new DATA step, creating four new columns demonstrating different applications of the function and various formats:

```
/*create new dataset with datetime formatted as date*/
data new_data;
set original_data;
date_mmddyyyy = put(datepart(some_datetime), mmddyy10.);
date_yyyymmdd = put(datepart(some_datetime), yymmdd10.);
date_date9 = put(datepart(some_datetime), date9.);
date_default = datepart(some_datetime);
run;

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

The resulting output, generated using PROC PRINT, clearly confirms that **DATEPART** successfully isolated the date, and **PUT** applied the requested display formats:

Obs	some_datetime	date_mmddyyyy	date_yyyymmdd	date_date9	date_default
1	14OCT2022:00:00:00	10/14/2022	2022-10-14	14OCT2022	22932
2	09NOV2022:00:00:00	11/09/2022	2022-11-09	09NOV2022	22958
3	14NOV2022:00:00:00	11/14/2022	2022-11-14	14NOV2022	22963
4	15NOV2022:00:00:00	11/15/2022	2022-11-15	15NOV2022	22964
5	22DEC2022:00:00:00	12/22/2022	2022-12-22	22DEC2022	23001
6	24DEC2022:00:00:00	12/24/2022	2022-12-24	24DEC2022	23003
7	04JAN2023:00:00:00	01/04/2023	2023-01-04	04JAN2023	23014

Understanding Raw Numeric Output and Common Errors

The output column `date_default` in the practical example above is highly instructive. It was generated by calling the DATEPART function without coupling it with the PUT function or applying a separate format statement. The result is a series of large integer numbers that do not resemble a conventional calendar date format.

These integers are the raw, numeric representation of the date: the count of days elapsed since the SAS epoch of January 1, 1960. For internal calculations, comparison, and sorting within SAS, this numeric value is ideal. However, for reporting, these raw numbers are completely unintelligible to human readers. This discrepancy highlights the critical necessity of always coupling the **DATEPART** extraction step with a presentation step (via **PUT** or a `FORMAT` statement) if the output is intended for display in a standard calendar date format.

To ensure robust SAS programming when handling temporal data, adhere to these best practices and avoid common pitfalls:

Never Omit Formatting for Display: If the resulting date is to be seen by anyone, ensure you use **PUT** with a specific format (e.g., `date9.`) to convert the numeric date value into a readable character string. Alternatively, if the variable must remain numeric for calculations, use a `FORMAT` statement in the DATA step to apply the display format without changing the underlying data type.
Understand Data Type Implications: Remember that if you use the **PUT** function to create a new variable (as in `date_mmddyyyy = put(...)`), the new variable is a character type. If you only use the SAS DATEPART function (e.g., `date_numeric = datepart(...)`), the new variable remains numeric, which is often preferable for subsequent arithmetic operations.
Prioritize International Standards: When choosing a format, prefer unambiguous formats like `yymmdd10.` (YYYY-MM-DD) over formats that can be confused regionally, such as `mmddyy10.` or `ddmmyy10.`.

Conclusion

Converting a datetime value into a simple date in the SAS environment is a straightforward process made efficient by the use of the **DATEPART** function. This function serves as the critical extraction tool, transforming the seconds-based datetime value into a days-based numeric date value.

While **DATEPART** performs the necessary calculation, successful implementation requires pairing it with the **PUT** function and an appropriate SAS format. This combination ensures that data is not only accurately processed but also presented clearly and professionally. By understanding the underlying numeric storage of temporal values and correctly applying these core functions, analysts can achieve precise data segmentation and enhanced reporting capabilities, vital for effective data utilization within the SAS ecosystem.

Additional Resources

For those seeking to deepen their expertise in handling temporal data and advanced SAS programming techniques, the following authoritative resources are highly recommended:

Official SAS Documentation: The definitive and most up-to-date source for detailed technical specifications on all functions, procedures, and statements within the SAS language.
SAS Communities: An invaluable platform for consulting with peers, resolving complex coding challenges, and sharing domain-specific knowledge.
Advanced SAS Training Materials: Resources that offer structured learning paths covering complex data manipulation, statistical modeling, and reporting methodologies.