How to Easily Convert Datetime to Date in SAS

Authored by

stats writer

November 22, 2025

Working with time series and temporal data is a fundamental requirement in data analysis, and SAS provides robust tools for handling these complexities. A common task involves stripping the time component from a combined date and time variable--known as a datetime value--to obtain a pure date. While advanced functions like INTNX can be utilized for date manipulation, such as calculating intervals or shifts (e.g., `INTNX('datetime-value', 'day', 0)`), the simplest and most direct approach for conversion relies on a specialized, built-in function designed specifically for this purpose. This guide details the expert method for performing this crucial transformation efficiently within the SAS environment.

The Primary Tool: Utilizing the DATEPART Function

The most straightforward and highly recommended method to convert a datetime value into a simple date value in SAS is through the use of the **DATEPART function**. This function is specifically engineered to extract the date component from a combined date and time stamp, discarding the hours, minutes, and seconds. It is essential to understand that SAS stores date and datetime values internally as numeric variables, representing time intervals since a specific starting point (the SAS epoch). A date value is the number of days since January 1, 1960, while a datetime value is the number of seconds since January 1, 1960.

When the DATEPART function processes a datetime variable, it performs the necessary calculation to isolate the date component and presents it as a standard SAS date value. This resulting date value, while numerically accurate, typically requires a subsequent step--the application of a date format--to be displayed in a human-readable form, such as DDMMYY or YYYY-MM-DD. Ignoring the formatting step will result in the output displaying the internal numeric representation, which is usually not what the analyst intends.

The efficiency of the DATEPART function stems from its simplicity and directness. Unlike manual calculations involving division by 86,400 (the number of seconds in a day) and truncation, **DATEPART** handles these underlying complexities internally. This makes the code cleaner, less prone to error, and significantly easier to maintain and debug, ensuring that data analysts can focus on the interpretation rather than the mechanics of temporal data conversion.

Syntax Breakdown and Formatting Considerations

To successfully convert a datetime value into a date, the **DATEPART** function must be used, often in conjunction with the **PUT function** to apply a display format. The PUT function is vital because, as mentioned, DATEPART returns a numeric date value, not a formatted string. The PUT function takes the numeric output of DATEPART and applies the specified format mask.

The general syntax for this conversion process is as follows:

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

Let's break down the components of this essential line of code. The `date =` part assigns the resulting formatted date to a new variable named 'date'. The inner function, `DATEPART(some_datetime)`, takes your existing datetime variable (`some_datetime`) and extracts its date component, returning the numerical SAS date value. The outer function, `PUT(...)`, then takes that numerical value and applies the specified format, which in this case is `mmddyy10.`. The argument **mmddyy10.** specifies that the date should be formatted like 10/15/2022, displaying the month, day, and four-digit year, separated by slashes.

Choosing the correct date format is critical for clarity and adherence to regional standards. SAS offers a wide array of date formats to suit various needs. Analysts commonly rely on the following built-in formats:

DATE9.: Displays the date as DD MMM YYYY (e.g., 14OCT2022). This is often preferred for international clarity.
YYMMDD10.: Presents the date in the ISO standard format YYYY-MM-DD (e.g., 2022-10-14), excellent for sorting and consistency.
MMDDYY10.: The US-standard format M/D/YYYY (e.g., 10/14/2022).
DOWNAME.: Displays the full name of the day of the week, useful for reporting.

Understanding the interplay between **DATEPART** (which extracts the numeric value) and **PUT** (which applies the visual mask) is key to successful temporal data transformation in SAS programming.

Setting the Stage: Creating Sample Datetime Data in SAS

To demonstrate how to use this powerful function, we must first establish a representative dataset containing datetime values. In real-world scenarios, these datetime variables are often imported directly from database systems or log files, usually represented as long numerical strings or complex timestamps. In the following example, we create a small dataset named `original_data` specifically to mimic this structure, ensuring our sample data uses the `DATETIME23.` format to properly handle both the date and time components down to the second.

The creation process involves defining the input format and then reading the specific timestamps using the `DATALINES` statement. It is crucial that the `FORMAT` statement within the SAS DATA step correctly identifies `some_datetime` as a datetime field, allowing SAS to interpret the data strings accurately before any conversion takes place. If the variable is not correctly defined as a datetime input, the **DATEPART** function will return missing values or incorrect results.

The following code shows how to define and populate our starting dataset:

```
/*create dataset containing sample datetime values*/
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 to confirm datetime structure*/
proc print data=original_data;
```

Once this code is executed, the `original_data` dataset is created. Inspecting this dataset using `PROC PRINT` confirms that the variable `some_datetime` contains the combined date and time stamps, formatted correctly. Observe the output structure:

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

As illustrated in the output, all values are true datetime stamps, even if the time component is zeroed out (0:0:0). Our goal now is to remove the time component entirely and store only the date part in various display formats.

Implementation Example: Converting Datetime to Date

With the source data prepared, we can now proceed to apply the **DATEPART** function within a

new SAS DATA step. In this step, we read the `original_data` and create four new variables, each demonstrating a different aspect of the conversion and subsequent formatting. This comprehensive approach highlights not only the function's core purpose but also the necessity of applying appropriate formats.

We define the following four new date variables:

date_mmddyyyy: Formatted using `mmddyy10.` (Month/Day/Year).
date_yyyymmdd: Formatted using `yymmdd10.` (Year-Month-Day, ISO standard).
date_date9: Formatted using `date9.` (DDMMMYYYY, SAS standard).
date_default: Left unformatted after the **DATEPART** extraction, revealing the underlying numerical value.

The following code shows how to use the **DATEPART** function in combination with the **PUT** function to achieve these formatted results. Notice that only the first three variables use **PUT**; the last one, `date_default`, uses **DATEPART** alone to demonstrate the native output of the function.

```
/*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;
```

Executing this code creates the `new_data` dataset, which now includes the original datetime value alongside the four new date variables. The conversion is successful, resulting in clean date values ready for further analysis or reporting.

Exploring Output Formats: Displaying Dates Correctly

Reviewing the output from `PROC PRINT DATA=new_data;` clearly illustrates the effect of applying different date formats. While the source information (the date itself) is identical across the first three columns, the presentation varies dramatically based on the format mask provided to the **PUT** function.

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

Notice that the four new columns display the date extracted from the original datetime column in various formats. The `date_mmddyyyy` column is displayed in the familiar M/D/YYYY format, suitable for US reports. The `date_yyyymmdd` column uses dashes and the year first, which is ideal for datasets destined for systems requiring ISO standard timestamps. Finally, `date_date9` provides the compact, readable SAS default date style.

Understanding these format differences is essential for analysts. For instance, if you intend to merge datasets based on date columns, ensure that the underlying variable type (numeric date value) is consistent. If you only rely on the display format (which is a character representation created by the PUT function), you might encounter issues if you later try to perform date arithmetic on the formatted variable itself, as it is no longer a pure numeric date value unless a format is explicitly applied to the variable definition rather than using PUT.

Why DATEPART Returns Numerical Values: The SAS Epoch

A key concept that often confuses new SAS users is the nature of the unformatted date output. Observe the column named `date_default` in the output table. Instead of a calendar date, this column displays large integer values (e.g., 22934, 22960, etc.). This numerical output is the true, raw result of the **DATEPART function**.

By default, the **DATEPART** function converts a datetime to the number of days since the SAS epoch, which is **January 1, 1960**. This specific reference date serves as the zero point (Day 0) for all SAS date values. Every subsequent day is incremented by one. Therefore, the resulting value for `date_default` displays the number of days since January 1, 1960, for each respective date. For example, a value of 22934 represents the 22,934th day following the epoch, which corresponds precisely to October 14, 2022.

This internal representation is highly efficient for computation. SAS can easily perform date arithmetic (e.g., calculating the difference between two dates) simply by subtracting these integers,

without needing complex calendar logic. However, for reporting and visual interpretation, this numerical value is meaningless to humans, underscoring the necessity of applying a human-readable format using the **PUT** function or a **FORMAT** statement later in the process.

Alternative Methods: Using INTNX for Date Manipulation

Although **DATEPART** is the gold standard for simple extraction, it is worthwhile to briefly consider the **INTNX function**, which was mentioned in the introductory material. While less direct for pure conversion, INTNX is an incredibly powerful tool for shifting or aligning dates and timestamps based on intervals.

The **INTNX** function operates by calculating a date or datetime that is a specified number of time intervals away from a starting date or datetime. Its structure is typically `INTNX(interval, start-from, increment)`. We can use INTNX to convert a datetime value to a date by setting the interval to a discrete unit, such as 'DAY', and the increment to zero. The syntax looks like this:

date_intnx = intnx('day', some_datetime, 0, 'SAME');

When INTNX is applied to a datetime variable using the 'DAY' interval, it effectively returns the starting point of that day (i.e., midnight of the current date), which corresponds precisely to the date component. While technically functional, using **DATEPART** is semantically clearer and generally preferred when the only requirement is stripping the time component, reserving INTNX for more complex tasks like determining the start of the month, the end of the quarter, or shifting dates forward or backward.

Best Practices and Summary

Converting datetime variables to date variables is a common and essential step in cleaning and preparing temporal data for analysis. To ensure robustness and clarity in your SAS code, adhere to these key best practices:

Prioritize DATEPART: Always use the **DATEPART function** for straightforward extraction of the date component from a datetime variable.
Format for Readability: Recognize that **DATEPART** returns a numeric value (days since 1960). Utilize the **PUT function** or a **FORMAT statement** to apply a human-readable format (e.g., `date9.` or `yymmdd10.`).
Verify Variable Types: Ensure that the original variable is correctly defined as a datetime type (usually 8 bytes) before attempting conversion.
Documentation: For complex temporal manipulations beyond simple extraction, consult the official SAS documentation. You can find the complete documentation for the SAS **DATEPART**

function and related time functions on the official SAS support website, providing full details on syntax and behavior.

Mastering the **DATEPART** function is a foundational skill in SAS programming, enabling precise control over how temporal data is stored, manipulated, and presented.

The following code shows how to use the **DATEPART** function to create a new dataset in which the values in the datetime column are formatted as dates with various formats: