Learning to Create Empty Datasets in SAS: A Step-by-Step Guide

Authored by

Mohammed looti

May 14, 2026

Understanding the Necessity of Empty Datasets in SAS

In the realm of **SAS** programming and data management, the ability to intentionally create an **empty dataset** is not merely an academic exercise; it is a fundamental and frequently utilized technique. An empty dataset is structurally complete, meaning it possesses defined **variables** (columns) and their associated attributes (type, length, format, and label), but crucially contains zero **observations** (rows).

This capability provides programmers with essential control over the data workflow. Empty datasets function ideally as templates for future data loading, ensuring data integrity by pre-defining the required structure. They are also invaluable when preparing a target dataset for processes that involve aggregation, iterative calculations, or merging data from diverse external sources where the structure must be established before the content arrives. By defining the schema upfront, even without data, you significantly streamline subsequent data manipulation tasks and minimize potential errors related to structure mismatch.

Method 1: Defining Structure from the Ground Up (Using ATTRIB)

The first and most comprehensive method for generating an empty dataset involves manually defining every variable and its attributes within a dedicated **DATA step**. This approach grants the programmer maximum control and is necessary when the desired structure does not exist in any current SAS dataset. The core mechanism relies on the **ATTRIB statement** to specify characteristics like **length**, **format**, and **label** for each variable.

The `ATTRIB` statement is highly versatile, allowing precise definition of how data should be stored and displayed. For example, setting a numeric variable's length to 8 is standard for high-precision storage, while defining a character variable with a length such as `$30` sets its maximum capacity. The format governs the visual representation of the data in outputs and reports, and the label provides a human-readable description, significantly enhancing documentation and clarity.

To ensure this structure is created without any data rows, the process culminates with the crucial **STOP statement**, which halts the execution of the `DATA` step immediately after the variable attributes have been processed. This combination is the programmatic signature for building an empty dataset from scratch.

The following code illustrates the fundamental structure for defining three variables and ensuring the resulting dataset contains zero observations:

```
data empty_data;
attrib
var1 length=8 format=best12. label="var1"
```

```
var2 length=$30 format=$30. label="var2"
var3 length=8 format=best12. label="var3"
stop;
run;
```

In this block, the `DATA` step is initiated, the variables are defined via `ATTRIB`, and the `STOP` statement ensures that the **RUN statement** executes a program that has only processed the structural metadata, yielding a fully defined but empty dataset.

Method 2: Leveraging Existing Metadata for Structure (Using SET)

When the goal is to create a template that exactly mirrors the structure of an existing SAS dataset, relying on manual attribute definition is inefficient and prone to error. Method 2 provides a highly efficient alternative by leveraging the **SET statement** to inherit all necessary structural information.

The primary function of the `SET` statement is to read data from an existing source dataset. However, when used at the beginning of a `DATA` step, SAS first reads the structural components, or **metadata**, of the source dataset. This metadata includes the names, data types, lengths, formats, and labels of all variables. By immediately following the `SET` statement with a `STOP` statement, we capture this inherited structure while preventing the subsequent transfer of any data rows (observations).

This technique is a powerful tool for maintaining consistency across large projects, ensuring that staging datasets or output templates conform perfectly to the schema of a master input file. It drastically reduces the lines of code required compared to manually defining every attribute.

The simplified code required for this method is remarkably concise:

```
data empty_data;
set existing_data;
stop;
run;
```

The resulting dataset, `empty_data`, will be a perfect structural clone of `existing_data`, ready to accept new data that adheres to the established schema, but without carrying over any original observations.

The Essential Role of the STOP Statement

While the `ATTRIB` and `SET` statements handle the definition or inheritance of variable structure,

the process of creating a truly empty dataset hinges entirely on the strategic placement of the **STOP statement**. This statement is critical because it controls the flow of execution within the `DATA` step.

A standard `DATA` step operates iteratively, executing all subsequent commands for every observation it processes. If the `STOP` statement were omitted in Method 1 (creating from scratch), the `DATA` step would execute the implicit loop once, resulting in a dataset with one observation of missing values. If the `STOP` statement were omitted in Method 2 (using `SET`), the `DATA` step would proceed to read and copy every single observation from the source dataset, defeating the purpose of creating an empty template.

By placing `STOP` immediately after the structural definitions are completed, we instruct SAS to terminate the data processing cycle before the first iteration of observation writing can occur. This ensures that only the metadata--the fundamental blueprint of the dataset--is saved, leaving the count of observations firmly at zero. Thus, the deliberate combination of defining structure and immediately halting execution is the universal principle for generating empty datasets in SAS.

Practical Implementation: Building a Template Dataset (Example 1)

To solidify understanding, let us examine a practical example of creating a brand-new, empty template. We aim to construct a dataset named `empty_data` designed to hold employee sales records, necessitating four distinct variables: a numeric employee ID, a character employee name, a numeric sales amount, and a numeric variable formatted as a date for the sales date.

The following **code** snippet utilizes the `ATTRIB` statement to precisely define these characteristics. Note how we specify the appropriate lengths and formats for both character (`$30.`) and numeric (e.g., `date9.`) variables, ensuring the resulting structure is robust and ready for data entry.

```
/*create empty dataset*/
data empty_data;
attrib
employee_ID length=8 format=best12. label="Employee ID"
employee_Name length=$30 format=$30. label="Employee Name"
sales length=8 format=best12. label="Sales"
sales_date length=8 format=date9. label="Sales Date";
stop;
run;
```

To confirm that the execution was successful and that the dataset is indeed empty, we employ the

PROC CONTENTS procedure. This utility is the standard method for inspecting the metadata of any SAS file, providing crucial information about the structure, including the total number of observations and the definitions of all variables.

```
/*view contents of dataset*/
proc contents data=empty_data;
```

The output from `PROC CONTENTS` (as shown below) confirms two vital pieces of information: first, that the "Number of Observations" is zero, proving its emptiness; and second, that the four variables we defined are correctly listed with their assigned attributes. This validation step is essential to ensure the template is accurately configured before it is used for data population.

The CONTENTS Procedure

Data Set Name	WORK.EMPTY_DATA	**Observations**	0
Member Type	DATA	**Variables**	4
Engine	V9	**Indexes**	0
Created	11/17/2022 10:30:43	**Observation Length**	56
Last Modified	11/17/2022 10:30:43	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

A further look at the variable listing section verifies the specific details of the structure, including lengths, types (Num or Char), formats (e.g., DATE9.), and descriptive labels, confirming the success of the manual definition process.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	employee_ID	Num	8	BEST12.	Employee ID
2	employee_Name	Char	30	$30.	Employee Name
3	sales	Num	8	BEST12.	Sales
4	sales_date	Num	8	DATE9.	Sales Date

Practical Implementation: Replicating Structure from a Source (Example 2)

In contrast to building a dataset from scratch, often a programmer needs an empty dataset that adheres strictly to the layout of an existing file. This scenario is perfectly suited for Method 2, utilizing the efficiency of the `SET` and `STOP` statements. For this demonstration, we will use the sample dataset `sashelp.Comet`, which is readily available in the built-in `sashelp` library, as our source template.

By referencing `sashelp.Comet` in the `SET` statement, we instruct SAS to read all its structural **metadata**. The immediate execution of the `STOP` statement ensures that this structural information is written to the new dataset, `empty_dat`, while preventing the transfer of any actual data observations. This method guarantees structural conformity without the overhead of data copying.

```
/*create empty dataset from existing dataset*/
data empty_dat;
set sashelp.Comet;
stop;
run;
```

To verify the outcome, we again rely on the descriptive power of `PROC CONTENTS`. We confirm that `empty_dat` has successfully inherited the complex structure of the source file while remaining completely empty of data rows.

```
/*view contents of dataset*/
proc contents data=empty_dat;
```

The resulting output clearly shows that `empty_dat` has the same variable count as `sashelp.Comet` but confirms the desired zero observations. This streamlined approach underscores the power of using existing structural templates for rapid and accurate dataset generation.

The CONTENTS Procedure

Data Set Name	WORK.EMPTY_DAT	**Observations**	0
Member Type	DATA	**Variables**	4
Engine	V9	**Indexes**	0
Created	11/17/2022 10:41:24	**Observation Length**	32
Last Modified	11/17/2022 10:41:24	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

The detailed variable listing confirms that all variable attributes from `sashelp.Comet` have been perfectly replicated in the new, empty template, making it ready for any data processing that requires this specific schema.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
1	Dose	Num	8	1,2 Dimethylhydrazine dihydrochloride Dose Level
4	Length	Num	8	Tail Length of the Comet
2	Rat	Num	8	Rat Index
3	Sample	Num	8	Slide Index of Grouped Cells from a Rat

Conclusion and Summary of Best Practices

The creation of empty datasets in **SAS** is a foundational skill that supports efficiency and integrity across advanced data workflows. Whether you choose the explicit definition route using the `ATTRIB` statement or the structure inheritance approach using the `SET` statement, the success of the operation relies fundamentally on the judicious application of the **STOP statement**.

These empty datasets serve multiple critical functions: they act as robust templates for data input, ensuring that data collected or loaded conforms to predefined standards; they provide essential staging areas for complex data manipulation steps, such as merging or aggregation; and they ensure consistent schema definitions across disparate parts of a large programming project. Mastering these simple yet powerful techniques allows SAS programmers to design reliable, scalable, and highly maintainable programs, significantly enhancing the overall quality of their data

management processes.

Additional Resources

To further expand your SAS programming expertise, consider exploring the following tutorials that cover other common data manipulation and analytical tasks:

SAS Official Documentation: https://documentation.sas.com/

SAS Communities: https://communities.sas.com/