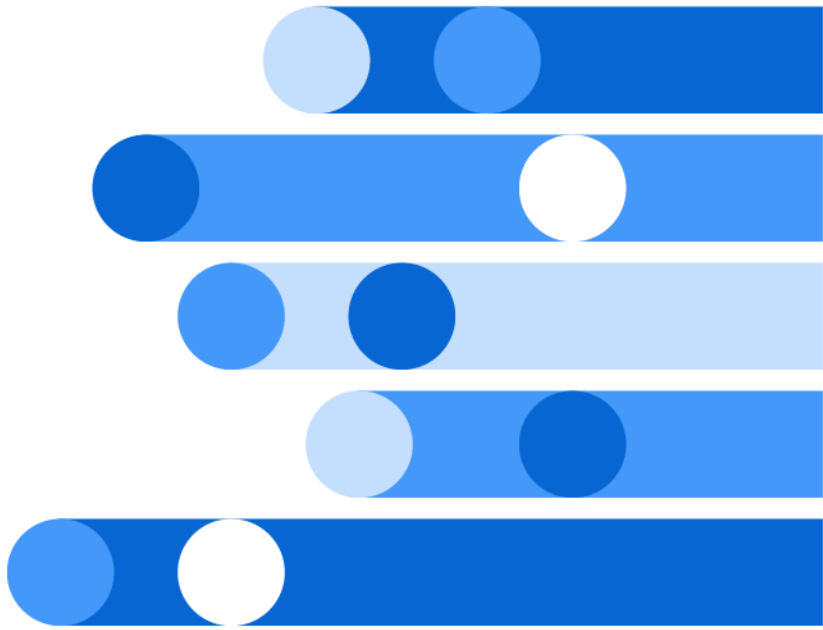
SAS® In-Database Products: User's Guide

2026.06 - 2026.08*



* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

SAS® Documentation
August 19, 2026

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2026. *SAS® In-Database Products: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® In-Database Products: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

August 2026

v_003-P1:indbug

Contents

PART 1

Introduction

1

In-Database Processing

Overview of In-Database Processing

SAS Data Accelerator and SAS In-Database Technologies improve performance by running some code inside the data source. In contrast, under conventional processing, the SAS/ACCESS engine generates an SQL SELECT * statement that is passed to the data source. That SELECT statement fetches all the rows in the table, which can cause a considerable amount of data transfer. In-database processing greatly reduces data transfer and network latency.

The third column in the table below indicates which components of in-database processing use SAS Embedded Process, which is a SAS server process that is deployed in and runs within your data source to read and write data. SAS Embedded Process is delivered with SAS Data Accelerator or SAS In-Database Technologies, depending on your environment. Contact your SAS representative for licensing details. See "Deployment and Administration for In-Database Processing" on page 5.

Note: For the sake of brevity, in this document, the term *database* is often used to denote both relational database management systems and file systems.

The following table lists the data sources that are supported for each of the in-database processing features.

Table 1.1 *In-Database Processing Features and Supported Interfaces*

In-Database Feature	Supported Interfaces	Requires Deployment of SAS Embedded Process
SQL Pass-Through facility (PROC SQL) Processing with PROC FEDSQL and PROC DS2[1] SQL Functions	See the documentation for each data source in *SAS/ACCESS for Relational Databases: Reference*.	No
DATA step programs For more information, see Part 2, "In-Database DATA Step Processing," on page 7.	Azure Synapse Databricks SingleStore[2]	Yes
Data Connector provides parallel data transfer between a data source and CAS. For requirements, see each data source in *SAS Cloud Analytic Services: User's Guide*.	Cloudera SingleStore[2] SPD Engine files on HDFS Teradata	Yes
Scoring models using PROC ACCELERATOR For more information, see Part 3, "In-Database Model Scoring," on page 33.	Azure Synapse Databricks SingleStore[2] Teradata	Yes
Scoring models using PROC SCOREACCEL or CAS actions (These features are deprecated for Databricks and Synapse. Use PROC ACCELERATOR instead.) For more information, see Part 3, "In-Database Model Scoring," on page 33.	Amazon EMR Cloudera Databricks SingleStore[2] Teradata CAS	Yes
Base SAS procedures: COPY[3] FREQ MEANS RANK[4, 6] REPORT SORT[6, 7, 5] SUMMARY TABULATE	Amazon Redshift DB2 Google BigQuery Greenplum Hadoop Impala Microsoft SQL Server MySQL[8] Netezza	No

In-Database Feature	Supported Interfaces	Requires Deployment of SAS Embedded Process
For more information, see Part 4, "In-Database Procedures," on page 245.	Oracle PostgreSQL[9] SAP HANA SingleStore[10] Snowflake Spark Teradata Vertica Yellowbrick	

1 The FedSQL language provides SQL pass-through. For more information, see "FedSQL Implicit Pass-Through Facility" in *SAS FedSQL Language Reference* and "FedSQL Explicit Pass-Through Facility" in *SAS FedSQL Language Reference*. Several DS2 language elements accept embedded FedSQL syntax, and the runtime-generated queries can exchange data interactively between DS2 and any supported database. For more information, see "Dynamically Executing FedSQL Statements from DS2" in *SAS DS2 Programmer's Guide*.

2 SingleStore: In-database DATA step, parallel data transfer, and in-database scoring are available if you license the product SAS SpeedyStore. That product includes an instance of SingleStore that has SAS Embedded Process installed already.

3 Greenplum, PostgreSQL, Vertica: In-database processing for PROC COPY is supported only for Greenplum, PostgreSQL, and Vertica. For more information, see the documentation for your SAS/ACCESS interface.

4 Google BigQuery: In-database processing for PROC RANK requires a VAR variable that is not of type FLOAT64.

5 Google BigQuery: In-database processing for PROC SORT requires a BY variable that is not of type FLOAT64.

6 Spark: In-database processing for PROC RANK and PROC SORT requires Spark 2.4 and later.

7 In-database processing for PROC SORT is supported only for the NODUPKEY option.

8 MySQL: PROC RANK and PROC SORT are supported for MySQL 8.0.17 and later.

9 In-database processing is not supported for CockroachDB, which is accessed via a PostgreSQL LIBNAME statement.

10 SingleStore: In-database procedures do not require SAS SpeedyStore.

Deployment and Administration for In-Database Processing

For information about deployment and configuration, see *SAS Viya Platform: Deployment Guide* and *SAS In-Database Products: Deployment and Administration Guide*.

For data source requirements, see "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.

PART 2

In-Database DATA Step Processing

2

DATA Step Acceleration

Introduction to DATA Step Processing in the Data Source

The SAS DATA step can export some code to be processed in the following data sources:

- Databricks as of 2025.03
- Microsoft Azure Synapse Analytics as of 2025.12
- SAS SpeedyStore (SingleStore database) as of 2024.04

If a DATA step program does not meet the requirements for running in-database, the code executes in the SAS Compute Server session. For more information, see "Requirements for DATA Step Processing" on page 13 and "Restrictions in DATA Step Processing" on page 14.

Why Use In-Database DATA Step Processing?

Benefits of In-Database DATA Step Processing

The in-database DATA step can often improve performance compared to conventional processing. When you use in-database processing, you move the processing to the data, rather than moving the data to the processing. Reducing the movement of data can result in processing that is cheaper, quicker, and more secure.

In general, processing big data in a single thread of execution is slow, especially for heavy computation. When the DATA step can execute on multiple rows in parallel on multiple database nodes or executors, in multiple partitions, the processing workload is divided among threads on those nodes or executors.

An example of *heavy computation* is operating on multiple columns. If you have to transfer the data to SAS, you are operating on it in serial mode in a single thread.

In the statement above, *big data* means more data than can fit in the available memory in SAS. When you exceed memory, SAS must perform Read and Write operations between the buffer and the storage medium. In addition, transferring big data from the database to SAS uses more resources than transferring smaller data.

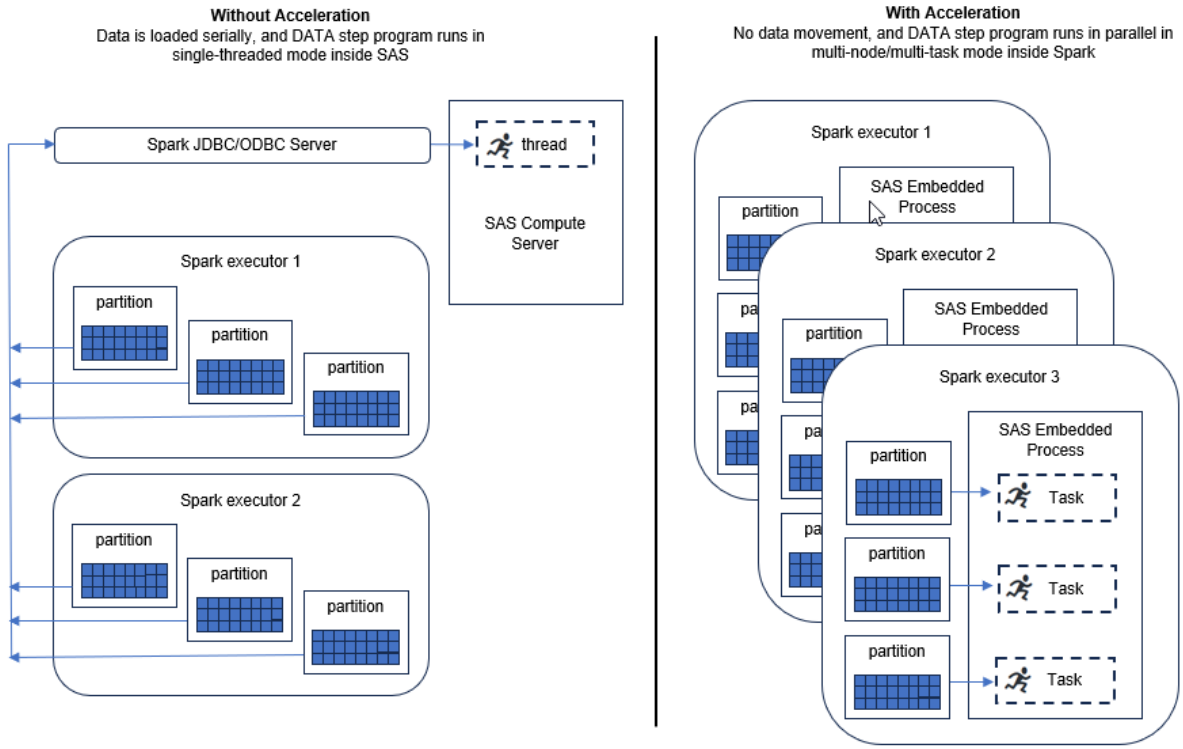
Note that for simple operations on smaller data, you might get better performance by avoiding in-database processing. Performance testing is recommended.

Processing in Databricks

The following illustration shows the difference when the SAS Embedded Process runs in the Spark executors. The input data is broken into multiple partitions. The partitions are distributed to multiple executors, where a Spark task is assigned to each partition. During the DATA step, if the code meets the requirements for running in-database, the data does not leave the cluster.

For simplicity, the graphic does not show the output data: Each Spark task produces an output partition. All the output partitions combine to make up the output table.

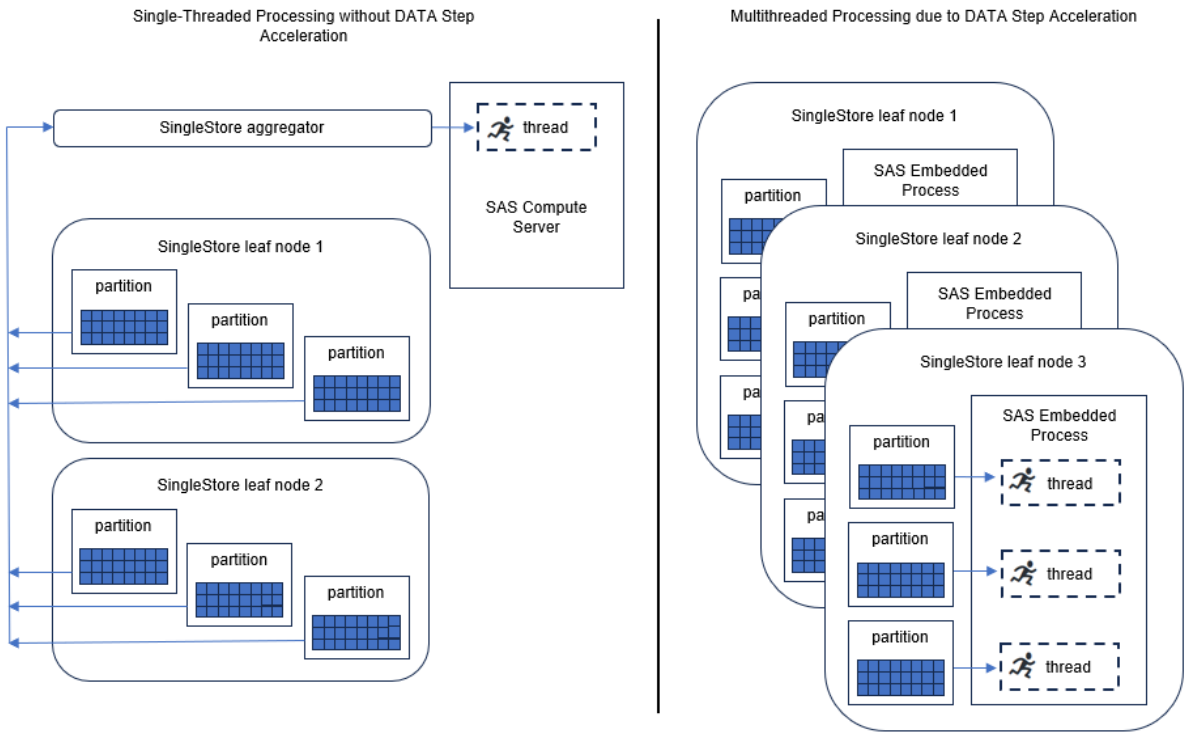
Figure 2.1 *Comparison of Single-Threaded and Multithreaded Processing for Databricks*



Processing in SingleStore

The following illustration shows the difference when the SAS Embedded Process runs inside the SingleStore leaf nodes.

Figure 2.2 *Comparison of Single-Threaded and Multithreaded Processing for SingleStore*

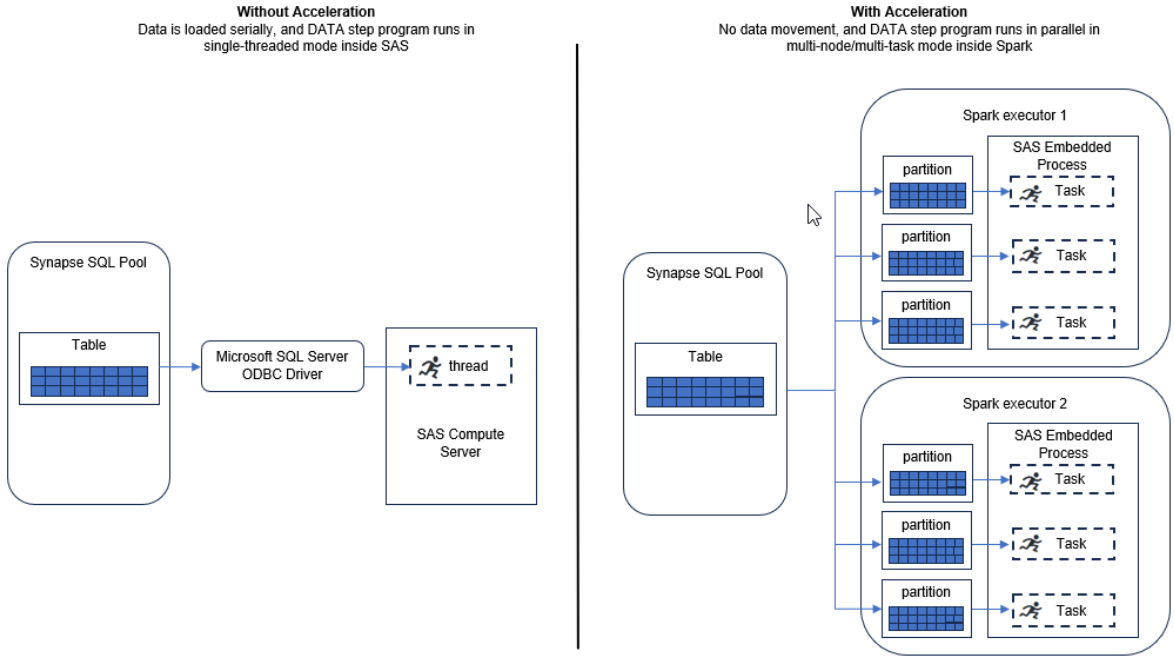


Processing in Synapse

The following illustration shows the difference when the SAS Embedded Process runs in the Spark executors. The input data is broken into multiple partitions. The partitions are distributed to multiple executors, where a Spark task is assigned to each partition. During the DATA step, if the code meets the requirements for running in-database, the data does not leave the cluster.

For simplicity, the graphic does not show the output data: Each Spark task produces an output partition. All the output partitions combine to make up the output table.

Figure 2.3 Comparison of Single-Threaded and Multithreaded Processing for Synapse



Requirements for DATA Step Processing

In order to run a DATA step program in the data source, the following are required:

- For Databricks or Synapse, you must license and deploy SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06). See "Deployment and Administration for In-Database Processing" on page 5. You must start the SAS Embedded Process continuous session before you submit DATA step code. Use the ACCELERATOR procedure with the STARTSESSION statement.
- For SingleStore, you must license and deploy SAS SpeedyStore. The SAS Embedded Process is already installed in the included instance of SingleStore, and you are not required to start the continuous session. See *SAS SpeedyStore: Administration and Configuration Guide*.
- Set the system option DSACCEL=ANY. See "DSACCEL= System Option" on page 255.
- The code must reference a LIBNAME assignment for the data source.
  - For Databricks, see "SAS/ACCESS Interface to Spark" in *SAS/ACCESS for Relational Databases: Reference*.
  - For SingleStore, see "SAS/ACCESS Interface to SingleStore" in *SAS/ACCESS for Relational Databases: Reference*.
  - For Synapse, see "SAS/ACCESS Interface to Microsoft SQL Server" in *SAS/ACCESS for Relational Databases: Reference*.

- For more code requirements, and DATA step processing restrictions, see "Restrictions in DATA Step Processing" on page 14.

Restrictions in DATA Step Processing

Here are the restrictions for running the DATA step in-database. For most of these restrictions, if the DATA step code does not meet the requirements for running in-database, the code executes in your SAS Compute Server session. Exceptions are noted in the list.

- For Databricks, you can reference Spark tables only. For Synapse, you reference a SQL Server table, also known as a dedicated SQL pool table. You must start the continuous session before you submit DATA step code. Use the ACCELERATOR procedure with the STARTSESSION statement. When you are finished running code, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS. For more information, see Chapter 8, "ACCELERATOR Procedure," on page 119. For SingleStore, you are not required to start the continuous session.
- You can use one input table and one output table only. Both tables must be in the same data source. As of 2024.07, the input and output LIBNAME statements can reference two different databases, but they must reference the same server, port number, and user credentials.
- The DATA statement must be followed immediately by the SET statement. More than one SET statement is not supported.
- No SET statement options are supported, including data set options for input tables.
- Data set options for output tables are ignored. The DATA step runs in-database but does not apply the options.
- The following statements are not supported:

CARDS	INPUT	REMOVE
DATALINES	LIST	RENAME
DESCRIBE	LOSTCARD	REPLACE
EXECUTE	MERGE	SYSTASK
FILE	MODIFY	UPDATE
INFILE	PUT	
INFORMAT	REDIRECT	

- The ABORT statement does not accept arguments.
- The ATTRIB statement does not support the INFORMAT= attribute.
- The RETAIN statement does not support variable list syntax in any of these formats:

```
x1-x10
_numeric_
a-numeric-b
```

- Many functions and CALL routines have restrictions or are not supported. These restrictions are documented in each affected function or CALL routine.
- Customized functions that you create by using the FCMP procedure are not supported.
- The DATA step code cannot assign a value to a variable from the input table.
- The TIMEZONE= system option does not affect in-database DATA step processing. If you want to change the time zone, set it in the data source configuration. The SAS Embedded Process uses the time zone of the data source.
- Under multithreaded processing, rows are not necessarily processed in the order of the input table. Therefore, any processing that relies on information from one row to another might return unexpected results.
- Component objects are not supported.
- For the date, datetime, and time column types that are supported by each LIBNAME engine, beginning in 2025.07 the following formats are retained in the output data: DATE*w*. format. DATETIME columns retain the DATETIME*w.d* format. TIME columns retain the TIME*w.d* format.

Note: The SAS_EP_PROPERTIES environment variable can be used to set the logger.level= and cmp.threads= properties. However, most customers do not need to set these properties. For more information, see "SAS_EP_PROPERTIES Environment Variable" on page 261.

Data Type Considerations

Data Type Considerations for Cloudera and Databricks

Data Type Guidelines for Cloudera and Databricks

These topics cover data type considerations when the accelserver RPM or the accelserver-uia RPM is deployed.

Here are the key takeaways:

- All data types map cleanly between input and output types.
- ARRAY, BINARY, GEOGRAPHY, GEOMETRY, INTERVAL, MAP, OBJECT, STRUT, VARIANT, and VOID types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from Databricks, the input data type is converted internally by SAS Embedded Process for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a Databricks data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions for Cloudera and Databricks

The following data types map from input data types to output data types with no conversion issues:

- BIGINT
- BOOLEAN
- DATE
- DECIMAL(10,2)
- DOUBLE
- FLOAT
- INT
- SMALLINT
- STRING [1]
- TIMESTAMP
- TIMESTAMP_NTZ
- TINYINT

[1]For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22. For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Data Type Considerations for SingleStore

Data Type Guidelines for SingleStore

These topics cover data type considerations for SAS SpeedyStore.

Here are the key takeaways:

- All data types map cleanly between input and output types.
- BINARY, BIT, BSON, ENUM, GEOGRAPHY, GEOGRAPHYPOINT, JSON, PSQL types, SET, and VECTOR types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from SingleStore, the input data type is converted internally by SAS Embedded Process for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a SingleStore data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions for SingleStore

The table below summarizes the input and output data types for SingleStore. As shown in the table, some output data types are not the same as the input type, due to limitations or conventions in SingleStore or in SAS SpeedyStore.

Table 2.1 *Input and Output Data Types*

Input SingleStore Data Type	Output SingleStore Data Type
Character[1]	
CHAR(*n*)	CHAR(*n*)
LONGTEXT[2]	TEXT
MEDIUMTEXT[2]	TEXT
TEXT[2]	TEXT
TINYTEXT[2]	VARCHAR(63)
VARCHAR(*n*)	VARCHAR(*n*)
Numeric	
BIGINT(*n*)	BIGINT(*n*)
DECIMAL(*p*,*s*)	DOUBLE
DOUBLE	DOUBLE
FLOAT	FLOAT
INT(*n*)	INT(*n*)
MEDIUMINT(*n*)	INT(*n*)
SMALLINT(*n*)	SMALLINT(*n*)
TINYINT(*n*)	TINYINT(*n*)
Date and Time	
DATE	DATE
DATETIME	DATETIME(6)
DATETIME(6)	DATETIME(6)
TIME	TIME(6)
TIME(6)	TIME(6)

Input SingleStore Data Type	Output SingleStore Data Type
TIMESTAMP	DATETIME(6)
TIMESTAMP(6)	DATETIME(6)
YEAR	BIGINT(20)

1 For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22.
2 For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Data Type Considerations for Synapse

Data Type Guidelines for Synapse

These topics cover data type considerations when the accelserver RPM is deployed.

Here are the key takeaways:

- Most date and numeric data types map cleanly between input and output types.
- String types are converted to NVARCHAR to preserve Unicode and length (up to 4000 characters).
- Date and time types are standardized to DATETIME2 for compatibility and precision.
- BINARY and VARBINARY types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from Microsoft SQL Server, the input data type is converted internally by SAS Embedded Process to an Apache Spark data type for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a Microsoft SQL Server data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions and Limitations for Synapse

The table below summarizes the input and output data types for Microsoft SQL Server. As shown in the table, some output data types are not the same as the input type, due to limitations or conventions in Spark or in Microsoft SQL Server. For simplicity, the table does not show how the input data types are converted to Spark data types in Synapse during processing.

Table 2.2 *Input and Output Data Types*

Input Microsoft SQL Server Data Type	Output Microsoft SQL Server Data Type	Notes
	Character	
CHAR(10)	NVARCHAR(4000)	For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22.
NCHAR(10)	NVARCHAR(4000)	
VARCHAR(100)	NVARCHAR(4000)	For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.
NVARCHAR(100)	NVARCHAR(4000)	
	Numeric	
BIGINT	BIGINT	
BIT	BIT	
DECIMAL(18,4)	DECIMAL(18,4)	Conversion can result in a loss of precision or scale because the value is converted to a DOUBLE internally during processing by the SAS Embedded Process for Spark. For more information about these concepts, see also "Numeric Precision" in *SAS Programmer's Guide: Essentials*.

Input Microsoft SQL Server Data Type	Output Microsoft SQL Server Data Type	Notes
FLOAT	FLOAT	The output column in Microsoft SQL Server is a double-precision floating-point data type.
INT	INT	
MONEY	DECIMAL(19,4)	
NUMERIC(18,4)	DECIMAL(18,4)	
REAL	REAL	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
TINYINT	INT	The usable range is 0–127, because TINYINT has a different range in Microsoft SQL Server (unsigned, 0–255) than in the SAS Embedded Process for Spark (signed, -128–127).
Date and Time		
DATE	DATETIME2	The following formats are retained in the output data: DATE columns retain the DATE*w*. format. DATETIME columns retain the DATETIME*w.d* format. TIME columns retain the TIME*w.d* format.
DATETIME	DATETIME2	
DATETIME2	DATETIME2	
SMALLDATETIME	DATETIME2	
TIME	DATETIME2	
Other		
BINARY		Not supported.
VARBINARY		Not supported.

Data Type Conversions: NULLs, Empty Strings, and Padding

This topic covers Read and Write behavior for character data types when the accelserver RPM is deployed. The information also applies to SingleStore, although you do not deploy an RPM. The information in this topic applies to PROC ACCELERATOR and the in-database DATA step, and the information is applicable to several data sources. Some details are noted as being specific to one data source.

For Read behavior of unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Table 2.3 *Read Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is Read as a NULL and is padded with spaces to the defined length.	Value is Read as an empty string and is padded with spaces to the defined length.	Value is Read as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT VARCHAR	Value is Read as a NULL.	Value is Read as a zero-length empty string.	Value is Read as is. Any blank spaces (including leading or trailing blanks) are preserved.

1 For supported data types, see the in-database conversion topic for your data source.

2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.

3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Table 2.4 *Write Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is padded with spaces to the defined length.	Value is padded with spaces to the defined length.	Value is stored as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT VARCHAR	Value is stored as a NULL.	Value is stored as a zero-length empty string.	Value is stored as is. Any blank spaces (including leading or trailing blanks) are preserved.

1 For supported data types, see the in-database conversion topic for your data source.

2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.

3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

3

Examples

Example: Run a DATA Step Program in Databricks

This example demonstrates executing an in-database DATA step program in Databricks.

```
options dsaccel=any;                                          /*1*/

libname spklib spark platform=databricks                      /*2*/
        user=token
        password="authentication-token"
        server="nnnn.azuredatabricks.net"
        catalog="my-catalog-name"
        schema="my-schema-name"
        port=10016
        httppath="my-http-path"
        preserve_names=yes
        insertbuff=32767
        bulkload=no
        ;

proc accelerator library=spklib;                              /*3*/
   startsession
      platformendpoint="https://my-identifier.databricks.net"
      ;
run;
quit;

data spklib.mycars;                                           /*4*/
```

```
   set sashelp.cars;
run;

data spklib.mycars_out;                                          /* 5 */
   set spklib.mycars;
   by make;
   keep make makeCount averageMsrp;
   if first.make then do;
      makeCount=0;
      totalMsrp=0;
   end;
   makeCount+1;
   totalMsrp+msrp;
   if last.make and makeCount > 0 then do;                       /* 6 */
      averageMsrp = totalMsrp / makeCount;
      output;
   end;
run;

proc print data=spklib.mycars_out (obs=5) noobs;                 /* 7 */
run;

proc accelerator library=spklib;                                 /* 8 */
   stopsession;
run;
quit;
```

1 The DSACCEL=ANY option enables in-database DATA step processing.

2 The LIBNAME statement associates a SAS libref with a Databricks cluster, using the Spark LIBNAME engine.

3 The ACCELERATOR procedure starts a SAS Embedded Process continuous session, and associates the specified endpoint with the spklib libref. The PLATFORMENDPOINT= value is the Databricks workspace URL.

4 Copy a table from the Sashelp library to Databricks. In-database processing cannot occur if the input table does not exist in the data source.

5 This DATA step executes in Databricks and demonstrates several DATA step features. The BY statement enables the IF-THEN clauses to reference FIRST. and LAST. temporary variables. The variables averageMsrp, makeCount, and totalMsrp are created in the DATA step.

6 The second IF-THEN clause in the DATA step calculates averageMsrp for each row. Then the OUTPUT statement writes each row to the output table.

7 The PRINT procedure prints the first five observations.

8 Stop the SAS Embedded Process continuous session. When you are finsished running DATA step code, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

Output 3.1 *PROC PRINT Output of Mycars_out Table*

Make	makeCount	averageMsrp
Volkswagen	15	32248.67
Toyota	28	22524.46
Suzuki	8	16230.25
Isuzu	2	26149.00
Jeep	3	24518.33

Example: Run a DATA Step Program in SingleStore

This example demonstrates executing an in-database DATA step program in SingleStore.

```
options dsaccel=any;                                    /*1*/
libname mylib sstore user="myuser" pw="mypw"
        server="myserver" database="mydb";

data mylib.myclass;
   set sashelp.class (keep=name age);                   /*2*/
run;
proc print data=mylib.myclass (obs=5) noobs;            /*3*/
run;

libname mylib2 sstore user="myuser" pw="mypw" server="myserver"
database="my_output_db";
data mylib2.myclass_ages;                               /*4*/
   set mylib.myclass;
   length Age_Group $9;
   if age < 14 then age_group="Group A";
      else age_group="Group B";
run;
proc print data=mylib.myclass_ages (obs=5) noobs;       /*5*/
run;
```

1 The DSACCEL=ANY option enables in-database DATA step processing.

2 This SET statement includes an option, which is not supported for in-database DATA step processing. A note is written in the SAS log stating that the SAS Embedded Process is not used for this operation. See "SAS Log Message When SAS Embedded Process Is Not Used" on page 28. In addition, note that the input table is in the Sashelp library and is not in SingleStore. For in-database processing to occur, both the input table and the output table must be in SingleStore.

3 See Output 3.2 on page 28. The printed table is limited to five rows for simplicity.

4 This DATA step creates a new column. A note is written in the SAS log stating that the SAS Embedded Process is used for this operation. See "SAS Log Message When SAS Embedded Process Is Used" on page 29. Notice the output table uses a different libref than the input table. As of 2024.07, the input and output tables can be in different SingleStore databases. The input and output LIBNAME statements must reference the same server, port number, and user credentials.

5 See Output 3.3 on page 28. Notice that the rows are not in the same order as in the Myclass table. In-database processing does not maintain row order.

Output 3.2 *PROC PRINT Output of Myclass Table*

Name	Age
Janet	15
Alfred	14
Henry	14
Jeffrey	13
John	12

Output 3.3 *PROC PRINT Output of Myclass_ages Table with New Column*

Name	Age	Age_Group
Janet	15	Group B
Alfred	14	Group B
Jeffrey	13	Group A
John	12	Group A
Henry	14	Group B

Example Code 3.1 *SAS Log Message When SAS Embedded Process Is Not Used*

```
89   data mylib.myclass;
90      set sashelp.class (keep=name age);
                                         -
                                         811
NOTE 811-185: The DATA step contains a SET statement options. The SAS Embedded
Process will not be used for this operation.
91   run;
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set MYLIB.myclass has 19 observations and 2 variables.
```

Example Code 3.2 *SAS Log Message When SAS Embedded Process Is Used*

```
95   data mylib.myclass_ages;
96      set mylib.myclass;
97      length Age_Group $9;
98      if age < 14 then age_group="Group A";
99         else age_group="Group B";
100  run;
NOTE: The SAS Embedded Process is being used for this operation.
```

Example: Run a DATA Step Program in Synapse

This example demonstrates executing an in-database DATA step program in Microsoft Azure Synapse.

```
options dsaccel=any;                                                    /*1*/

libname mylib sqlsvr                                                    /*2*/
        user=my-user-name
        password="my-password"
        schema="my-schema"
        noprompt="DRIVER=SAS ACCESS to MS SQL Server;
                  Hostname=synapse-workspace-name.sql.azuresynapse.net;
                  Port=my-port-number;
                  Database=my-database-name
                 ...;";

proc accelerator library=mylib;                                         /*3*/
   startsession
      platform=synapse
      platformendpoint=""
      platformuser="endpoint/livyApi/versions/livyApiVersion/
sparkPools/sparkPoolName"
      platformpassword="my-secret";
run;
quit;

data mylib.mycars;                                                      /*4*/
   set sashelp.cars;
run;

data mylib.mycars_out;                                                  /*5*/
   set mylib.mycars;
   by make;
   keep make makeCount averageMsrp;
   if first.make then do;
      makeCount=0;
      totalMsrp=0;
```

```
   end;
   makeCount+1;
   totalMsrp+msrp;
   if last.make and makeCount > 0 then do;                    /*6*/
      averageMsrp = totalMsrp / makeCount;
      output;
   end;
run;

proc print data=mylib.mycars_out (obs=5) noobs;               /*7*/
run;

proc accelerator library=mylib;                               /*8*/
   stopsession;
run;
quit;
```

1 The DSACCEL=ANY option enables in-database DATA step processing.

2 The LIBNAME statement associates a SAS libref with a dedicated SQL pool in the Synapse workspace, using SAS/ACCESS Interface to Microsoft SQL Server. The dedicated SQL pool must be unpaused. Specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option. For additional options, see "SAS/ACCESS Interface to Microsoft SQL Server" in *SAS/ACCESS for Relational Databases: Reference*.

3 The ACCELERATOR procedure starts a SAS Embedded Process continuous session, and associates the specified endpoint with the SAS library location. The PLATFORM=, PLATFORMENDPOINT=, PLATFORMUSER=, and PLATFORMPASSWORD= options are required for Synapse. For more information about the endpoint, see "Forming the Livy REST URL for Synapse" on page 74.

4 Copy a table from the Sashelp library to Synapse. In-database processing cannot occur if the input table does not exist in the data source.

5 This DATA step executes in Synapse and demonstrates several DATA step features. The BY statement enables the IF-THEN clauses to reference FIRST. and LAST. temporary variables. The variables averageMsrp, makeCount, and totalMsrp are created in the DATA step.

6 The second IF-THEN clause in the DATA step calculates averageMsrp for each row. Then the OUTPUT statement writes each row to the output table.

7 The PRINT procedure prints the first five observations.

8 Stop the SAS Embedded Process continuous session. When you are finsished running DATA step code, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

Output 3.4 *PROC PRINT Output of Mycars_out Table*

Make	makeCount	averageMsrp
Volkswagen	15	32248.67
Toyota	28	22524.46
Suzuki	8	16230.25
Isuzu	2	26149.00
Jeep	3	24518.33

PART 3

In-Database Model Scoring

4

Introduction to In-Database Scoring

Overview of In-Database Scoring in the SAS Viya Platform

When you use in-database processing, the scoring is done inside the data source. Performance is improved by reducing the transfer of data. For more information, see "Overview of In-Database Processing" on page 3.

In the SAS Viya platform, models are created and published in the following ways:

Build
: You can build scoring models by using SAS Model Studio or SAS Visual Analytics, or by submitting modeling procedures or CAS actions within SAS Data and AI Studio (known as SAS Studio prior to 2026.06). In addition, in-database scoring continues to support models that are created in SAS Enterprise Miner. For more information, see Chapter 5, "Models Supported for Scoring," on page 39.

Register (optional)
: If you want to share or deploy a model, you can register it from SAS Model Studio. SAS Model Manager can also import some models that are not created in SAS Model Studio. SAS Model Manager enables you to register, deploy, monitor, and manage a broad variety of model types. Registering a model is not required. See SAS Model Studio documentation and SAS Model Manager documentation. Some model types that are supported by SAS Model Manager are not supported for in-database scoring.

Publish
: After a model is registered, you can use SAS Model Manager to publish, run, and verify a model.

  As an alternative to SAS Model Manager, you can use SAS code to publish a model. You can write a program and submit it in SAS Data and AI Studio (known as SAS Studio prior to 2026.06). See "Publishing and Running Models Programmatically" on page 36.

Run
: After a supported scoring model is published, you can score data in an accelerated way by using SAS code. See "Publishing and Running Models Programmatically" on page 36.

  SAS Embedded Process is the underlying technology that runs the published scoring models inside the data source. SAS Embedded Process is one of the deployed components for in-database processing. For more information, see "Deployment and Administration for In-Database Processing" on page 5.

Publishing and Running Models Programmatically

The table below lists the SAS code that you can use to publish, run, and delete scoring models in a database. See also "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

Table 4.1 *SAS Code for In-Database Scoring*

SAS Code	Supported Data Sources	Documentation
PROC ACCELERATOR	■ AWS Databricks ■ Azure Databricks ■ Azure Synapse ■ SingleStore ■ Teradata	■ Syntax and examples ■ Usage
PROC SCOREACCEL	■ CAS ■ AWS Databricks ■ Azure Databricks ■ Hadoop ■ SingleStore ■ Teradata	■ Syntax and examples ■ Usage

SAS Code	Supported Data Sources	Documentation
Model Publishing and Scoring action set	■ CAS ■ AWS Databricks ■ Azure Databricks ■ Hadoop ■ SingleStore ■ Teradata	■ Syntax and examples ■ Usage
Python and Scala API for the accelserver RPM (Runs scoring; does not publish or delete.)	■ AWS Databricks ■ Azure Databricks ■ Azure Synapse	■ Syntax and examples
Scala and Python API for the sepcorespark RPM (Runs scoring; does not publish or delete.)	■ AWS Databricks ■ Azure Databricks ■ Azure Synapse	■ Syntax and examples

5

Models Supported for Scoring

Models Created in the SAS Viya Platform with CAS Actions, SAS Procedures, or SAS Model Studio

In the SAS Viya platform, scoring models can be created by submitting a CAS action or a SAS procedure, or by using SAS Model Studio. For information about SAS Model Studio nodes, see "Scoring Your Models" in *SAS Viya: Machine Learning Advanced Topics*.

The following table lists the SAS Viya platform models that are currently supported by SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06). (For legacy models that are not created in the SAS Viya platform, see the topics that follow.)

The "Cadence Validated" column lists the cadence when the model was thoroughly tested for in-database scoring with supported data sources.

For the best results, you are encouraged to update SAS Embedded Process to the most recent release. For more information, see *SAS In-Database Products: Deployment and Administration Guide*.

Table 5.1 *Models Built by Actions, Procedures, or SAS Model Studio*

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
annTrain of the Neural Network action set in *SAS Viya: Machine Learning Programming Guide*	NNET in *SAS Viya: Machine Learning Procedures*	Neural Network	Analytic store	2024.02
bartGauss of the Bayesian Additive Regression Trees action set in *SAS Visual Statistics: Programming Guide*	BART in *SAS Visual Statistics: Procedures*	Bayesian Additive Regression Trees	Analytic store	2024.02
bartProbit of the Bayesian Additive Regression Trees action set in *SAS Visual Statistics: Programming Guide*	BART in *SAS Visual Statistics: Procedures*	Bayesian Additive Regression Trees	Analytic store	2024.04
bnet of the Bayesian Net Classifier action set in *SAS Viya: Machine Learning Programming Guide*	BNET in *SAS Viya: Machine Learning Procedures*	Bayesian Network	Analytic store	2024.02
dnnExportModel of the Deep Neural Network action set in *SAS Viya: Deep Learning Programming Guide*	(None)	Neural Network	Analytic store	2024.02
dtreeExportModel, dTreePrune,	TREESPLIT procedure	Decision Tree	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
dTreeSplit, or dtreeTrain actions of the Decision Tree action set in *SAS Visual Analytics: Programming Guide*	(generating some of the same models as dtreeExportModel , dTreePrune, or dtreeTrain only) in *SAS Visual Statistics: Procedures*			
dsAutoMl of the Data Science Pilot action set in *SAS Viya: Machine Learning Programming Guide*	(None)	(None)	Analytic store	2024.10
factmac of the Factorization Machine action set in *SAS Viya: Machine Learning Programming Guide*	FACTMAC in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
featureMachine of the Data Science Pilot action set in *SAS Viya: Machine Learning Programming Guide*	(None)	Feature Machine	Analytic store	2024.02
forestTrain of the Decision Tree action set in *SAS Visual Analytics: Programming Guide*	FOREST in *SAS Viya: Machine Learning Procedures*	Forest	Analytic store	2024.02
gampl of the Generalized Additive Models action set in *SAS Visual Statistics: Programming Guide*	GAMMOD in *SAS Visual Statistics: Procedures*	GAM	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
gamselect of the Generalized Additive Models action set in *SAS Visual Statistics: Programming Guide*	GAMSELECT in *SAS Visual Statistics: Procedures*	GAM	Analytic store	2024.02
gbtreeTrain of the Decision Tree action set in *SAS Visual Analytics: Programming Guide*	GRADBOOST in *SAS Viya: Machine Learning Procedures*	Gradient Boosting	Analytic store	2024.02
genmod of the Regression action set in *SAS Visual Statistics: Programming Guide*	GENSELECT in *SAS Visual Statistics: Procedures*	Logistic Regression	DATA step	2024.02
glm of the Regression action set in *SAS Visual Statistics: Programming Guide*	REGSELECT in *SAS Visual Statistics: Procedures*	Linear Regression	DATA step	2024.02
gmm of the Nonparametric Bayes action set in *SAS Viya: Machine Learning Programming Guide*	GMM in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
gpClass of the Nonparametric Bayes action set in *SAS Viya: Machine Learning Programming Guide*[1]	GPCLASS in *SAS Viya: Machine Learning Procedures*	Gaussian Process Classification	Analytic store	2024.10
gpReg of the Nonparametric Bayes action set in *SAS Viya:*	(None)	Gaussian Process Regression	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
Machine Learning Programming Guide				
graphMultiReg of the Generalized Linear Multitask Learning action set in *SAS Viya: Machine Learning Programming Guide*	MTLEARN in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
kClus of the Clustering action set in *SAS Visual Statistics: Programming Guide*	KCLUS in *SAS Visual Statistics: Procedures*	Clustering	Analytic store	2024.02
kpca of the Kernel Principal Component Analysis action set in *SAS Viya: Machine Learning Programming Guide*	KPCA in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
logistic of the Regression action set in *SAS Visual Statistics: Programming Guide*	LOGSELECT in *SAS Visual Statistics: Procedures*	(None)	Analytic store	2024.02
mbAnalysis of the Association Rule Mining action set in *SAS Viya: Machine Learning Programming Guide*	MBANALYSIS in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
mbcFit of the Model-Based Clustering action set in *SAS Visual Statistics:*	MBC in *SAS Visual Statistics: Procedures*	(None)	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
Programming Guide				
mixed of the Mixed Modeling action set in *SAS Visual Statistics: Programming Guide*	LMIXED in *SAS Visual Statistics: Procedures*	(None)	Analytic store	2024.02
optBinning of the Risk Modeling and Decisioning action set in *SAS Viya: Machine Learning Programming Guide*	OPTBINNING in *SAS Viya: Machine Learning Procedures*	Interactive Grouping	Analytic store	2024.02
Principal Component Analysis action set, actions eig, itergs, nipals, or randompca in *SAS Visual Statistics: Programming Guide*	PCA in *SAS Visual Statistics: Procedures*	(None)	Analytic store	2024.02
quantreg of the Quantile Regression Modeling action set in *SAS Visual Statistics: Programming Guide*	QTRSELECT in *SAS Visual Statistics: Procedures*	Quantile Regression	Analytic store	2024.02
rlExportModel of the Reinforcement Learning action set in *SAS Viya: Reinforcement Learning Programming Guide*	(None)	(None)	Analytic store	2024.02
robustpca of the Robust PCA action set in *SAS Viya: Machine*	RPCA in *SAS Viya: Machine Learning Procedures*	Feature Extraction	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
Learning Programming Guide[2]				
saveas of the Analytic Store Scoring action set in *SAS Viya: Machine Learning Programming Guide*	ASTORE in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.04
seqRuleGen of the Sequence Rule Mining action set in *SAS Visual Analytics: Programming Guide*	(None)	(None)	Analytic store	2024.02
sparseSvmTrain of the Machine Learning for Sparse Data action set in *SAS Viya: Machine Learning Programming Guide*	SPARSEML in *SAS Viya: Machine Learning Procedures*	(None)	Analytic store	2024.02
svddTrain of the Support Vector Data Description action set in *SAS Viya: Machine Learning Programming Guide*	SVDD in *SAS Viya: Machine Learning Procedures*	Anomaly Detection	Analytic store	2024.02
svmTrain of the Support Vector Machine action set in *SAS Viya: Machine Learning Programming Guide*	SVMACHINE in *SAS Viya: Machine Learning Procedures*	SVM	Analytic store	2024.02
tmAstore when used in the	(None)	(None)	Analytic store	2024.02

Action	Procedure	SAS Model Studio Node	Type	Cadence Validated
following sequence: 1 tpParse of the Text Parse action set 2 tpAccumulate of the Text Parse action set 3 tmSvd of the Text Mining action set 4 tmAstore of the Text Utilities action set in *SAS Visual Text Analytics: Programming Guide*				
tmMine of the Text Mining action set in *SAS Visual Text Analytics: Programming Guide*	TEXTMINE in *SAS Viya: Machine Learning Procedures*	Text Mining	Analytic store	2024.02
transform of the Data Preprocess action set in *SAS Visual Analytics: Programming Guide*	SVDD in *SAS Viya: Machine Learning Procedures*	Anomaly Detection	Analytic store	2024.02
transform of the Data Preprocess action set in *SAS Visual Analytics: Programming Guide*	TREESPLIT in *SAS Visual Statistics: Procedures*	Decision Tree	DATA step	2024.02
(Multiple)	(Multiple)	Ensemble	DS2 multi-type	2024.02

1 For gpClass, modifying the processing environment, such as updating to a newer version of the SAS Viya platform or moving model scoring to an in-database procedure, can slightly change the numeric results. These types of differences are

due to changes in the operating system version. For more information, see "Numeric Precision" in *SAS Programmer's Guide: Essentials*.

2 For robustpca, when the input data is an encoded image, the model does not support in-database processing. For more information, see "Models Not Supported for In-Database Processing" on page 47.

The following model is supported by PROC SCOREACCEL and the CAS actions. However, the only supported data source is the CAS server.

Table 5.2 *Model Supported in CAS Only*

Action	Procedure	SAS Model Studio Node	Type
iml of the IML action set in *SAS IML: Programming Guide*	(None)	(None)	Analytic store

Models Not Supported for In-Database Processing

The following table lists the details that that cause certain models to not be supported by SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).

Table 5.3 *Limitations of SAS Viya Platform Models*

Limitation	Action	Procedure	SAS Model Studio Node
The model cannot be executed in a massively parallel processing (MPP) environment, which is required for in-database scoring.	dlExportModel	(None)	(None)
	filterDesign	DFIL	(None)
	rnnExportModel	(None)	(None)
	smcalib	SMCALIB	(None)
The analytic store model uses multiple output rows per input row and does not support in-database scoring. When you attempt to run the model with in-database processing, an error is printed in the SAS log:	hmm	HMM	(None)
	recBpr	RECENGINE	(None)
	recDtos	RECENGINE	(None)

Limitation	Action	Procedure	SAS Model Studio Node
`The flag one_to_many is not supported.`	recKnn	RECENGINE	(None)
Any model that uses the GAN library is not supported, because the SAS Embedded Process does not have GPU support. When you attempt to run the model with in-database processing, an error is printed in the SAS log: `Cannot load the GAN library.`	lgbmTrain	LIGHTGRADBOOST	Gradient Boosting
	robustpca, when input data is an encoded image	RPCA	(None)
	styleGanTrain	STYLEGAN	(None)
	tabularGanTrain	TABULARGAN	(None)
The following type of error message indicates that the environment running the SAS Embedded Process does not have the required level of glibc. Ensure that the operating system is at the appropriate level. Depending on the data source, the appropriate OS level might not be available to match the required level of glibc in order to run the model. `/lib64/libc.so.6: version `GLIBC_2.14' not found`	(Several models)		
The model produces a data-only analytic store, which is not supported for in-database scoring.	ktTrain	KTTRAIN	(None)

Models Created in SAS Enterprise Miner

Overview of the Score Code Export Node

Users of SAS Enterprise Miner develop data mining models that use measured attributes to either characterize or predict the value of an event. These models are developed on historical data where an event has been measured or inferred. The models are then applied to new data for which the attributes are known, but the event has not yet occurred. For example, a model can be created based on a credit institution's records of payments that customers made and missed last year. The

model can then be used to predict which customers are expected to miss payments this year.

SAS Enterprise Miner creates SAS language score code for the purpose of scoring new data. Users run this code in production systems to make business decisions for each record of new data.

The Score Code Export node is an extension for SAS Enterprise Miner that exports files that are necessary for score code deployment. Extensions are programmable add-ins for the SAS Enterprise Miner environment.

The following icon is the Score Code Export node as it appears in a SAS Enterprise Miner process flow diagram.

The following files are exported by the Score Code Export node:

- the SAS scoring model program (score.sas).
- an XML file that contains the scoring variables and other properties that are used and created by the scoring code (score.xml).
- a format catalog, if the scoring program contains user-defined formats.
- an XML file containing descriptions of the final variables that are created by the scoring code. This file can be kept for decision-making processes.
- a ten-row sample of the scored data set showing typical cases of the input attributes, intermediate variables, and final output variables. This data set can be used to test and debug new scoring processes.
- a ten-row sample table of the training data set showing the typical cases of the input attributes used to develop the score code.

For more information about the exported files, see "Output Files" on page 50. For more information about using SAS Enterprise Miner, see the SAS Enterprise Miner online Help.

Using the Score Code Export Node in a Process Flow Diagram

The **Score Code Export node** icon is located on the **Utility** tab, as shown in Figure 3.1:

Figure 5.1 *The Diagram Toolbar with the SAS Score Code Export Node Icon Highlighted*

To use the Score Code Export node, you need a process flow diagram that contains nodes that produce score code and that flow to a Score node. The Score node aggregates the score code for the entire process flow diagram and transfers it to the Score Code Export node. The Score node must precede the Score Code Export node in the process flow diagram.

This is a valid data mining process for exporting score code:

Figure 5.2 *Data Mining Process Flow Diagram*

Requirement: The Score Code Export node exports score code that contains only one DATA step. To see the SAS Enterprise Miner nodes that produce score code, see the *SAS Enterprise Miner: Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS 9.4*.

After the process flow diagram is in place, set the properties for the Score node and the Score Code Export node:

1. Select the **Score** node. Ensure that each of the following properties is set to the default value of Yes:
   - **Use Output Fixed Names**
   - **C Score**
2. Select the **Score Code Export** node and set the properties. The **Output Directory** property specifies the directory to store the export files. The **Name** property specifies the folder that contains the output files created by the Score Code Export node. For information about the properties, see the *SAS Enterprise Miner: Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS 9.4*.

After the properties are set, you are ready to export the score code. Right-click the **Score Code Export** node and select **Run**. When SAS Enterprise Miner completes processing, the Run Status window appears and indicates that the run completed. Click the **Results** button to view the output variables and the listing output. For information about the output, see "Output Created by the Score Code Export Node" on page 50.

Output Created by the Score Code Export Node

Output Files

The Score Code Export node writes the following output files, and a format catalog, if applicable, to the location specified by the Output Directory property. These files are used for publishing a model.

Table 5.4 *Score Code Export Node Output Files*

File or Folder	Description

score.sas	SAS language score code created by SAS Enterprise Miner. This code can be used directly in a SAS program. A sample program looks like this:
	`data testout ;`
	`  set simpletest.scoredata ;`
	`  %include "c:\models\simpletest\score.sas";`
	`run;`

score.xml	A description of the variables that are used and created by the scoring code. XML files are created by a machine process for the use of machine processes. Do not edit the XML file. **Restriction:** The maximum number of input variables for a scoring function is 128.
emoutput.xml	A description of the final variables that are created by the scoring code. This file can be kept for decision-making processes. These variables include the primary classification, prediction, probability, segment, profit, and loss variables created by a data mining process. The list does not include intermediate variables created by the analysis. For more information about these variables, see "Fixed Variable Names" on page 53. **Note:** The emoutput.xml file is not used for publishing a model.
scoredata.sas7bdat	A ten-row sample of the scored data set showing typical cases of the input attributes, intermediate variables, and final output variables. Use this data set to test and debug new scoring processes. **Note:** The scoredata.sas7bdat file is not used for publishing a model.
traindata.sas7bdat	A ten-row sample table of the training data set showing typical cases of the input attributes used to develop the score code. **Note:** The traindata.sas7bdat file is not used for publishing a model.
Format catalog	If the training data contains SAS user-defined formats, the Score Code Export node creates a format catalog. The catalog contains the user-defined formats in the form of a lookup table. This file has an extension of .sas7bcat.
Analytic store output	When your process flow diagram contains a node that creates an analytic store, three files are created. The score.sasast file contains the analytic store. The epscore.sas file contains the DS2 scoring model program. The score.sas file contains the SAS language score code. Together, these files contain the

File or Folder	Description
	aggregated score code for an entire process flow diagram that can be used by SAS in-database scoring to deploy the score code in a database.

Output Variables

The score code produced by SAS Enterprise Miner creates both intermediate variables, such as imputed values of missing values, transformations, and encodings; and output variables, such as predicted value and probability. Any of these created variables can be used in a scoring process.

> **TIP** The number of input parameters on a scoring function has a direct impact on performance. The more parameters there are, the more time it takes to score a row. A recommended best practice is to make sure that only variables that are involved in a model score evaluation are exported from SAS Enterprise Miner.

The most important output variables for the scoring process follow a naming convention using a prefix, as shown in the following table.

Table 5.5 *Output Variables*

Role	Type	Prefix	Key	Suffix	Example
Prediction	N	P_	Target variable name		P_amount
Probability	N	P_	Target variable name	Predicted event value	P_purchaseYES P_purchaseNO
Classification	$	I_	Target variable name		I_purchase
Expected Profit	N	EP_	Target variable name		EP_conversion
Expected Loss	N	EL_	Target variable name		EL_conversion

Role	Type	Prefix	Key	Suffix	Example
Return on Investment	N	ROI_	Target variable name		ROI_conversion
Decision	$	D_	Target variable name		D_conversion
Decision Tree Leaf	N	_NODE_			_NODE_
Cluster number or SOM cell ID	N	_SEGMENT_			_SEGMENT_

Fixed Variable Names

The Score node of SAS Enterprise Miner maps the output variable names to fixed variable names. This mapping is appropriate in cases where there is only one prediction target or one classification target. In other cases, refer to the output variable names described in the previous table.

Using the fixed variable names enables scoring users to build processes that can be reused for different models without changing the code that processes the outputs. These fixed names are listed in the emoutput.xml file and are described in the following table. Most scoring processes return one or more of these variables.

Table 5.6 *Fixed Variable Names*

Role	Type	Fixed Name	Description
Prediction	N	EM_PREDICTION	The prediction value for an interval target.
Probability	N	EM_PROBABILITY	The probability of the predicted classification, which can be any one of the target variable values.
Probability	N	EM_EVENTPROBABILITY	The probability of the target event. By default this is the first value in descending order. This is often the event of interest. The user can control the ordering in SAS Enterprise Miner.

Role	Type	Fixed Name	Description
Classification	$	EM_CLASSIFICATION	The predicted target class value.
Expected Profit	N	EM_PROFIT	Based on the selected decision.
Expected Loss	N	EM_LOSS	Based on the selected decision.
Return on Investment	N	EM_ROI	Based on the selected decision.
Decision	$	EM_DECISION	Optimal decision based on a function of probability, cost, and profit or loss weights.
Decision Tree Leaf, Cluster number, or SOM cell ID	N	EM_SEGMENT	Analytical customer segmentation.

SAS Enterprise Miner Tools Production of Score Code

Each node in SAS Enterprise Miner creates different types of score code.

These types can include the following:

SAS DATA Step
SAS Program
PMML
C
Java
DBMS

Users can develop their own nodes, known as extension nodes, which can create either SAS DATA step or SAS program score code. However, this code is not converted to PMML, C, or Java.

Note: There is limited support for user-written code in the Variable Clustering and Rules Builder nodes. User-written code could produce errors or unexpected results.

For information about the Enterprise Miner nodes and the type of score code that each node produces, see the *SAS Enterprise Miner: Reference Help* and *SAS Enterprise Miner High-Performance Data Mining Node Reference for SAS 9.4*.

Models Created by High-Performance Procedures

SAS in-database scoring continues to support legacy scoring models that are created by high-performance procedures.

The files that are required for analytic store scoring can be generated by SAS Factory Miner HPFOREST and HPSVM components or by SAS Enterprise Miner PROC HPFOREST or PROC HPSVM. The files must be accessible by the SAS Compute Server.

In addition, the SAS Factory Miner HPFOREST or HPSVM components generate a ten-row sample of the scored data set and a ten-row sample table of the training data set.

Files Used in Model Publishing

PROC SCOREACCEL and the CAS actions support DATA step, DS2, and analytic store model types. The following files are used as input for publishing a model:

DATA step or DS2 program
: If you provide DATA step code instead of DS2 code, the DATA step code is automatically converted to DS2.

analytic store
: The following files are used for analytic store scoring by in-database scoring:

  - An analytic store is a binary file that contains information about the state of an analytic object. It stores information that enables you to load and restore the state of the analytic object and set it in a score-ready mode. The analytic store is transportable. That is, it can be produced on one host and consumed on others without the need of the traditional SAS export or import. The file extension is **`.sasast`**.
  - A DS2 scoring model program can be specified to accompany the analytic store. The file extension is usually **`.sas`** or **`.ds2`**. If you specify an analytic store without specifying a DS2 model program, then a DS2 program is generated. However, automatic generation of DS2 model programs is not supported for models that include multiple analytic stores.

scoring variables and other properties
: In some cases, an XML file is present that contains the scoring variables and other properties. This file is used by some DATA step models.

user-defined formats
: In some cases, a scoring model program references SAS user-defined formats. If so, this information is available as an XML file (extension `.xml`) or an item store (extension `.is`).

Files Produced by Model Publishing

When you publish a model in the form of DS2 code, DATA step code, or an analytic store, in the SAS Viya platform the publishing operation generates an item store with file extension `.is`. The item store file includes, as part of its contents, an analytic store file, the scoring model program, and a format XML file if one exists. In CAS or Teradata, this item store is written as a blob to a VARBINARY column of the model table. In Hadoop, the item store is copied to the HDFS file system on the cluster. When you run the model, SAS can read the item store, unpack it, and run the model in the data source.

If you publish a model in the form of DATA step code, the publishing operation translates the DATA step code to DS2 code on the SAS client before publishing the model. Some SAS language elements and syntax are not supported. See "Considerations When Creating or Modifying DATA Step Score Code" on page 56. For more information about the translation from DATA step to DS2, see "DSTODS2 Procedure" in *Base SAS Procedures Guide*.. If the model is published to CAS, a DS2 thread program is generated. If the model is published to Hadoop or Teradata, a DS2 serial program is generated (in other words, a sequential program without THREAD and ENDTHREAD statements). For more information about DS2 thread programs, as well as serial, parallel, and parallel-serial programs, see "Threaded Processing" in *SAS DS2 Programmer's Guide*.

Considerations When Creating or Modifying DATA Step Score Code

DATA step code is one type of scoring program that can be used as input by in-database scoring. Only the SAS language elements and syntax that are required to run critical data transformations and model scoring functions are available. If you use a statement or function that is not supported, an error occurs and your model is not published to the data source.

The following items describe the syntax that is supported:

- macro references (code that includes % and &).
- braces ({}), brackets ([ ]), or parentheses (( )) in a DATA step ARRAY statement.

- declaration statements using LENGTH.

  This means declarations including lists of variables with the standard DATA step numeric or character length specifications (for example, 8 or $32).

- attribute statements, although these have no semantic meaning.

  Only the syntax is supported. That is, you do not get a syntax error for them.

- array statements and array initializers.

  This includes both temporary and variable arrays. For variable arrays, DATA step aliasing variables are available. However, these variable arrays are implemented literally as array references instead of as variable aliases as in the DATA step.

- standard control structures, including IF, THEN, ELSE, DO, WHILE, UNTIL, SELECT/WHEN/OTHERWISE, CONTINUE, LEAVE, RETURN, LINK, and GOTO.

  The standard `do i = 1 to n` syntax is supported. WHILE and UNTIL are also supported. However, features such as list syntax, `do i= 1,3,5`, are not supported.

- STOP and RUN are syntactically supported but have no semantic meaning.

- assignment statements using arrays and scalars.

- SUBSTR references, including left-hand-side "pseudo" substring.

- basic syntax for the PUT statement using lists of variables.

  Line and column controls are not supported.

- DROP, FORMAT, and LABEL statements.

  FORMAT and LABEL syntax is supported, but the format or label information is not used.

  DROP is supported both syntactically and semantically. Dropped variables are made into local variables and are not included in any output table. Basic lists of variables are supported for DROP. Some syntax is supported for variable enumeration such as `drop A1-A3`. Colon syntax (`A:`) is not supported.

- variable array syntax, variable range lists, and OF lists (for example, `a1-a10` or `sum(of a[*])`).

- all DATA step expressions.

- constant lists (as used in IN clauses and array initializations).

  This includes standard lists such as (1,2,3,4) and those including iterators such as (4 * 99). Array initializations are translated into DS2 array assignment statements.

- some hash object syntax.

  This includes the basic declaration constructor and the DEFINEKEY, DEFINEDATA, DEFINEDONE, ADD, REPLACE, FIND, and CLEAR methods.

- Format justifiers (L, C, R) and some PUT modifiers (?) are syntactically supported.

- If you use the SAS Embedded Process to run your scoring model, you can use any function that is supported by the DS2 language. For more information, see "DS2 Functions" in *SAS DS2 Language Reference*.
- If you use scoring functions to run your scoring model, only the following functions are supported:

  ABS
  ARCOS
  ARSIN
  ATAN
  ATAN2
  CEIL
  COS
  COSH
  DMNORM
  DMRAN
  DMINIT
  EXP
  FLOOR
  INDEX
  INT
  LEFT
  LENGTH
  LOG
  LOG10
  LOWCASE
  MAX
  MIN
  MISSING
  MOD
  N
  NMAX
  SIN
  SINH
  SQRT
  STRIP
  SUBSTR
  SUM
  TAN
  TANH
  TRIM
  UPCASE

Note: The KLEFT, KTRIM, KLENGTH, KLOWCASE, KUPCASE, and KINDEX functions are syntactically supported by mapping each to its corresponding standard function.

6

Using PROC ACCELERATOR for Databricks, SingleStore, Synapse, or Teradata

Running Scoring Models in Databricks Using PROC ACCELERATOR

Requirements for Databricks Using PROC ACCELERATOR

SAS in-database processing supports PROC ACCELERATOR for scoring in Databricks, starting with 2024.11. Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of Databricks for Microsoft Azure or Amazon Web Services. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed in Databricks from the accelserver RPM file. See *SAS In-Database Products: Deployment and Administration Guide*.
- For in-database model scoring, the Databricks cluster must be started.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing to a Spark Table and Running in Databricks with PROC ACCELERATOR

Here are some details for Databricks:

- Connect to Databricks by submitting a Spark LIBNAME statement. You specify that libref in the LIBRARY= option every time you submit PROC ACCELERATOR.

  Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

- Submit PROC ACCELERATOR with the PUBLISHMODEL statement to publish a model.
- You must start a SAS Embedded Process continuous session before you run models. Submit PROC ACCELERATOR with the STARTSESSION statement. The PLATFORMENDPOINT= option is required, in order to specify the workspace URL. See "Specifying the Workspace URL for Databricks" on page 63.
- Submit PROC ACCELERATOR with the RUNMODEL statement to run the model.
- When you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.
- Submit PROC ACCELERATOR with the DELETEMODEL statement to delete a model.

For syntax and examples, see Chapter 8, "ACCELERATOR Procedure," on page 119.

Specifying the Workspace URL for Databricks

To start the SAS Embedded Process continuous session, submit PROC ACCELERATOR with the STARTSESSION statement. Use the PLATFORMENDPOINT= option of the STARTSESSION statement to specify the Databricks workspace URL

In Databricks, you must start the SAS Embedded Process continuous session before you run models or attempt DATA step processing in the data source.

- For Azure, you can find this URL in the Azure portal. Navigate to the Azure Databricks Service resource and copy the value of the **URL** field.
- For AWS, contact your administrator to request the workspace URL.

Data Type Considerations for Databricks

Data Type Guidelines for Databricks

These topics cover data type considerations when the accelserver RPM is deployed.

Here are the key takeaways:

- All data types map cleanly between input and output types.
- ARRAY, BINARY, GEOGRAPHY, GEOMETRY, INTERVAL, MAP, OBJECT, STRUT, VARIANT, and VOID types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from Databricks, the input data type is converted internally by SAS Embedded Process for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a Databricks data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions for Databricks

The following data types map from input data types to output data types with no conversion issues:

- BIGINT
- BOOLEAN
- DATE

- DECIMAL(10,2)
- DOUBLE
- FLOAT
- INT
- SMALLINT
- STRING [1]
- TIMESTAMP
- TIMESTAMP_NTZ
- TINYINT

[1]For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22. For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Data Type Conversions: NULLs, Empty Strings, and Padding

This topic covers Read and Write behavior for character data types when the accelserver RPM is deployed. The information also applies to SingleStore, although you do not deploy an RPM. The information in this topic applies to PROC ACCELERATOR and the in-database DATA step, and the information is applicable to several data sources. Some details are noted as being specific to one data source.

For Read behavior of unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Table 6.1 *Read Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is Read as a NULL and is padded with spaces to the defined length.	Value is Read as an empty string and is padded with spaces to the defined length.	Value is Read as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT	Value is Read as a NULL.	Value is Read as a zero-length empty string.	Value is Read as is. Any blank spaces (including leading or trailing blanks) are preserved.

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
STRING[3] TEXT TINYTEXT VARCHAR			

1 For supported data types, see the in-database conversion topic for your data source.
2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.
3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Table 6.2 *Write Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is padded with spaces to the defined length.	Value is padded with spaces to the defined length.	Value is stored as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT VARCHAR	Value is stored as a NULL.	Value is stored as a zero-length empty string.	Value is stored as is. Any blank spaces (including leading or trailing blanks) are preserved.

1 For supported data types, see the in-database conversion topic for your data source.
2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.
3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Running Scoring Models in SingleStore Using PROC ACCELERATOR

Requirements for SingleStore Using PROC ACCELERATOR

SAS in-database processing supports PROC ACCELERATOR for scoring in SingleStore, starting with 2025.07. Here are the basic requirements:

- You must license and deploy the product SAS SpeedyStore.
- PROC ACCELERATOR supports the integrated instance of the SingleStore database. The procedure does not support an external database.
- See "Requirements for SAS SpeedyStore" in *System Requirements for the SAS Viya Platform*.
- When you order SAS SpeedyStore, installation of SAS Embedded Process is not required as for other data sources. SingleStore is delivered with the SAS Viya platform order, and SAS Embedded Process is already installed in SingleStore. For more information about deployment, see *SAS SpeedyStore: Administration and Configuration Guide*.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing and Running Models in SingleStore Using PROC ACCELERATOR

Here are some details for SingleStore:

- Connect to your data source by submitting a SingleStore LIBNAME statement. You specify that libref in the LIBRARY= option every time you submit PROC ACCELERATOR.
- Use PROC ACCELERATOR to publish, run, or delete a model.
- For SingleStore, you do not start or stop a SAS Embedded Process continuous session.

For syntax and examples, see Chapter 8, "ACCELERATOR Procedure," on page 119.

Data Type Considerations for SingleStore

Data Type Guidelines for SingleStore

These topics cover data type considerations for SAS SpeedyStore.

Here are the key takeaways:

- All data types map cleanly between input and output types.
- BINARY, BIT, BSON, ENUM, GEOGRAPHY, GEOGRAPHYPOINT, JSON, PSQL types, SET, and VECTOR types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from SingleStore, the input data type is converted internally by SAS Embedded Process for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a SingleStore data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions for SingleStore

The table below summarizes the input and output data types for SingleStore. As shown in the table, some output data types are not the same as the input type, due to limitations or conventions in SingleStore or in SAS SpeedyStore.

Table 6.3 *Input and Output Data Types*

Input SingleStore Data Type	Output SingleStore Data Type
Character[1]	
CHAR(*n*)	CHAR(*n*)
LONGTEXT[2]	TEXT
MEDIUMTEXT[2]	TEXT
TEXT[2]	TEXT
TINYTEXT[2]	VARCHAR(63)
VARCHAR(*n*)	VARCHAR(*n*)
Numeric	
BIGINT(*n*)	BIGINT(*n*)
DECIMAL(*p*,*s*)	DOUBLE
DOUBLE	DOUBLE
FLOAT	FLOAT
INT(*n*)	INT(*n*)
MEDIUMINT(*n*)	INT(*n*)
SMALLINT(*n*)	SMALLINT(*n*)
TINYINT(*n*)	TINYINT(*n*)

Input SingleStore Data Type	Output SingleStore Data Type
Date and Time	
DATE	DATE
DATETIME	DATETIME(6)
DATETIME(6)	DATETIME(6)
TIME	TIME(6)
TIME(6)	TIME(6)
TIMESTAMP	DATETIME(6)
TIMESTAMP(6)	DATETIME(6)
YEAR	BIGINT(20)

1 For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22.

2 For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Data Type Conversions: NULLs, Empty Strings, and Padding

This topic covers Read and Write behavior for character data types when the accelserver RPM is deployed. The information also applies to SingleStore, although you do not deploy an RPM. The information in this topic applies to PROC ACCELERATOR and the in-database DATA step, and the information is applicable to several data sources. Some details are noted as being specific to one data source.

For Read behavior of unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Table 6.4 *Read Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is Read as a NULL and is padded with	Value is Read as an empty string and is padded with spaces to	Value is Read as is, but padded if necessary with spaces to the defined length. Any blank spaces

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
	spaces to the defined length.	the defined length.	(including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT VARCHAR	Value is Read as a NULL.	Value is Read as a zero-length empty string.	Value is Read as is. Any blank spaces (including leading or trailing blanks) are preserved.

1 For supported data types, see the in-database conversion topic for your data source.

2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.

3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Table 6.5 *Write Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is padded with spaces to the defined length.	Value is padded with spaces to the defined length.	Value is stored as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT	Value is stored as a NULL.	Value is stored as a zero-length empty string.	Value is stored as is. Any blank spaces (including leading or trailing blanks) are preserved.

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
VARCHAR			

1 For supported data types, see the in-database conversion topic for your data source.

2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.

3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Running Scoring Models in Synapse Using PROC ACCELERATOR

Requirements for Synapse Using PROC ACCELERATOR

IMPORTANT Changes are required for SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06) beginning with 2026.04. Microsoft has announced an end of support for Azure Synapse Runtime for Apache Spark 3.4 as of March 31, 2026. Here are the details:

- You must deploy the SAS Embedded Process from the accelserver RPM. You can no longer deploy the sepcorespark RPM in Synapse.
- You must perform a pre-installation task to download an additional object library and create a wheel file. Then you install that wheel file when you install the sasindb wheel. See "Deploying SAS Embedded Process for Synapse" in *SAS In-Database Products: Deployment and Administration Guide*.
- The SCOREACCEL procedure, the Model Publishing and Scoring action set, and the SAS Embedded Process for Spark action set are no longer supported for Synapse. Instead, use the ACCELERATOR procedure for in-database scoring and to stop and start the SAS Embedded Process.
- When you deploy the accelserver RPM, you can also use the in-database DATA step, which was not supported under the sepcorespark RPM.

SAS in-database processing supports PROC ACCELERATOR for scoring in Microsoft Azure Synapse Analytics, starting with 2025.12. Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of Synapse. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed in Synapse from the accelserver RPM file. See *SAS In-Database Products: Deployment and Administration Guide*.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing to an SQL Server Table and Running in Synapse with PROC ACCELERATOR

Here are some details for Synapse:

- Use the Microsoft SQL Server LIBNAME statement for your connection. You specify that libref in the LIBRARY= option every time you submit PROC ACCELERATOR. You must connect to a dedicated SQL pool in the Synapse workspace. The pool must be unpaused. In the LIBNAME statement, specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option.
- Submit PROC ACCELERATOR with the PUBLISHMODEL statement to publish a model to an SQL Server table (also known as a dedicated SQL pool table).

- You must start a SAS Embedded Process continuous session before you run models. Submit PROC ACCELERATOR with the STARTSESSION statement. The PLATFORM=, PLATFORMENDPOINT=, PLATFORMUSER=, and PLATFORMPASSWORD= options are required for Synapse. For information about specifying the PLATFORMENDPOINT= option, see "Forming the Livy REST URL for Synapse" on page 74.
- Submit PROC ACCELERATOR with the RUNMODEL statement to run the model.
- When you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.
- Submit PROC ACCELERATOR with the DELETEMODEL statement to delete a model.

For syntax and examples, see Chapter 8, "ACCELERATOR Procedure," on page 119.

Forming the Livy REST URL for Synapse

To start the SAS Embedded Process continuous session, submit PROC ACCELERATOR with the STARTSESSION statement. Use the PLATFORMENDPOINT= option of the STARTSESSION statement to specify the Livy REST URL.

In Synapse, you must start the SAS Embedded Process continuous session before you run models or attempt DATA step processing in the data source.

The Livy REST URL for the Synapse Spark pool includes three parameters:

endpoint/livyApi/versions/*livyApiVersion*/sparkPools/*sparkPoolName*

- The endpoint is the workspace development endpoint. An example is `https://myworkspace.dev.azuresynapse.net`.
- To find the Livy API version, see the Synapse Livy API documentation. An example is `2019-11-01-preview`.
- The final segment is the name of the Spark pool.

Data Type Considerations for Synapse

Data Type Guidelines for Synapse

These topics cover data type considerations when the accelserver RPM is deployed.

Here are the key takeaways:

- Most date and numeric data types map cleanly between input and output types.
- String types are converted to NVARCHAR to preserve Unicode and length (up to 4000 characters).
- Date and time types are standardized to DATETIME2 for compatibility and precision.
- BINARY and VARBINARY types are not supported.

Note: These topics apply to PROC ACCELERATOR and in-database DATA step, and the content is identical.

Read and Write Behavior

Data type conversion and limitations can vary, depending on whether you are Reading or Writing.

When you Read data from Microsoft SQL Server, the input data type is converted internally by SAS Embedded Process to an Apache Spark data type for processing. Some examples of Reading include:

- You use in-database DATA step to Read a table in the database.
- You use PROC ACCELERATOR to Read the input data during model scoring.

When you Write data from the SAS Embedded Process, the output data type is converted to a Microsoft SQL Server data type. Some examples of Writing include:

- You use in-database DATA step to create a new table in the database.
- You use PROC ACCELERATOR to create output data from a model.

Data Type Conversions and Limitations for Synapse

The table below summarizes the input and output data types for Microsoft SQL Server. As shown in the table, some output data types are not the same as the input type, due to limitations or conventions in Spark or in Microsoft SQL Server. For simplicity, the table does not show how the input data types are converted to Spark data types in Synapse during processing.

Table 6.6 *Input and Output Data Types*

Input Microsoft SQL Server Data Type	Output Microsoft SQL Server Data Type	Notes
Character		

Input Microsoft SQL Server Data Type	Output Microsoft SQL Server Data Type	Notes
CHAR(10)	NVARCHAR(4000)	For more information, see "Data Type Conversions: NULLs, Empty Strings, and Padding" on page 22.
NCHAR(10)	NVARCHAR(4000)	
VARCHAR(100)	NVARCHAR(4000)	For unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.
NVARCHAR(100)	NVARCHAR(4000)	
	Numeric	
BIGINT	BIGINT	
BIT	BIT	
DECIMAL(18,4)	DECIMAL(18,4)	Conversion can result in a loss of precision or scale because the value is converted to a DOUBLE internally during processing by the SAS Embedded Process for Spark. For more information about these concepts, see also "Numeric Precision" in *SAS Programmer's Guide: Essentials*.
FLOAT	FLOAT	The output column in Microsoft SQL Server is a double-precision floating-point data type.
INT	INT	
MONEY	DECIMAL(19,4)	
NUMERIC(18,4)	DECIMAL(18,4)	
REAL	REAL	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
TINYINT	INT	The usable range is 0–127, because TINYINT has a different range in Microsoft SQL Server (unsigned, 0–255) than in the SAS Embedded Process for Spark (signed, -128–127).

Input Microsoft SQL Server Data Type	Output Microsoft SQL Server Data Type	Notes
	Date and Time	
DATE	DATETIME2	The following formats are retained in the output data: DATE columns retain the DATE*w*. format. DATETIME columns retain the DATETIME*w.d* format. TIME columns retain the TIME*w.d* format.
DATETIME	DATETIME2	
DATETIME2	DATETIME2	
SMALLDATETIME	DATETIME2	
TIME	DATETIME2	
	Other	
BINARY		Not supported.
VARBINARY		Not supported.

Data Type Conversions: NULLs, Empty Strings, and Padding

This topic covers Read and Write behavior for character data types when the accelserver RPM is deployed. The information also applies to SingleStore, although you do not deploy an RPM. The information in this topic applies to PROC ACCELERATOR and the in-database DATA step, and the information is applicable to several data sources. Some details are noted as being specific to one data source.

For Read behavior of unbounded data types in PROC ACCELERATOR, see also "MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*" on page 129.

Table 6.7 *Read Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is Read as a NULL and is padded with	Value is Read as an empty string and is padded with spaces to	Value is Read as is, but padded if necessary with spaces to the defined length. Any blank spaces

Data Type of Input Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
	spaces to the defined length.	the defined length.	(including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT VARCHAR	Value is Read as a NULL.	Value is Read as a zero-length empty string.	Value is Read as is. Any blank spaces (including leading or trailing blanks) are preserved.

1 For supported data types, see the in-database conversion topic for your data source.

2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.

3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Table 6.8 *Write Behavior of SAS Embedded Process with Accelserver RPM Deployed*

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
CHAR NCHAR	Value is padded with spaces to the defined length.	Value is padded with spaces to the defined length.	Value is stored as is, but padded if necessary with spaces to the defined length. Any blank spaces (including leading or trailing blanks) are preserved.[2]
NVARCHAR LONGTEXT MEDIUMTEXT STRING[3] TEXT TINYTEXT	Value is stored as a NULL.	Value is stored as a zero-length empty string.	Value is stored as is. Any blank spaces (including leading or trailing blanks) are preserved.

Data Type of Scoring Model Column or DATA Step Output Table Column[1]	When You Provide a NULL Value	When You Provide an Empty Value	When You Provide a Non-Empty Value
VARCHAR			

1 For supported data types, see the in-database conversion topic for your data source.
2 For SingleStore: For information about trailing blanks, see the data types documentation on the SingleStore website.
3 For Spark: When you initially create a table, you could define CHAR or VARCHAR columns (for example, by using SQL). In a Spark dataset, a CHAR or VARCHAR column is stored as a STRING column. However, in SAS, DS2 stores the column as a CHAR or VARCHAR *data type* in the STRING *column* of the Spark dataset. For output processing, if a model or DATA step creates a CHAR or VARCHAR column, that column is stored in the output Spark dataset as a STRING.

Running Scoring Models in Teradata Using PROC ACCELERATOR

Requirements for Teradata Using PROC ACCELERATOR

SAS in-database processing supports PROC ACCELERATOR for scoring in Teradata, starting with 2026.08. Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of the Teradata client and server environment. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process 2026.08 or later must be installed in Teradata. See *SAS In-Database Products: Deployment and Administration Guide*.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing and Running Models in Teradata Using PROC ACCELERATOR

Here are some details for Teradata:

- Connect to your data source by submitting a Teradata LIBNAME statement. You specify that libref in the LIBRARY= option every time you submit PROC ACCELERATOR.
- Use PROC ACCELERATOR to publish, run, or delete a model.
- For Teradata, you do not start or stop a SAS Embedded Process continuous session.

For syntax and examples, see Chapter 8, "ACCELERATOR Procedure," on page 119.

SAS_SCORE_EP Stored Procedure

In the SAS Viya platform, you do not submit the SAS_SCORE_EP stored procedure to run a scoring model. Instead, the stored procedure is called on your behalf when you submit PROC ACCELERATOR, PROC SCOREACCEL, or the runModelExternal CAS action. Therefore, it is unlikely that you need the syntax for the stored procedure. Most users rely on the syntax of PROC ACCELERATOR, PROC SCOREACCEL, or the runModelExternal action.

If needed, the stored procedure syntax is supported in the following way: In PROC ACCELERATOR, use the OPTIONS= argument. In PROC SCOREACCEL, use the EPOPTIONS= argument. Use the argument in the RUNMODEL statement to submit a string of options. If the argument is specified, then the string is passed as the value of the OPTIONS parameter on the resulting call to the SAS_SCORE_EP stored procedure.

Use the following syntax in the string.

SAS_SYSFNLIB.SAS_SCORE_EP

```
('MODELTABLE="database"."model-table-name" ',
 'MODELNAME=model-name',
 'INQUERY=SELECT ...'
```

'OUTTABLE= "*output-database-name*". "*output-table-name*" ',
'OUTKEY=*column*<..., *column*,> | NO PRIMARY INDEX,
NULL | 'OPTIONS=' | 'OPTIONS=*options-list*'
);

Parameters

"database"
specifies the name of the database that contains the model table.

Default — Database in the current session

Requirements — The database must be the same as the one specified in the DATABASE argument.

The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"model-table-name"
specifies the name of the model table where the scoring files were published.

Requirements — The model name must be the same as the one specified in the MODELNAME argument.

The maximum table name length is 128 characters, and it must be a valid Teradata table name.

model-name
specifies the name of the model.

SELECT ...
specifies a SELECT statement that defines the inputs to the SAS Embedded Process.

Range — The INQUERY= parameter string can be up to 30,000 characters long.

Restrictions — The maximum number of characters in the query is 30,000.

The maximum number of input and output columns is 1024.

Requirements — If the query is greater than 1,000 characters, the INQUERY= parameter must be the first parameter listed in the stored procedure.

The SELECT statement must be syntactically correct SQL.

Interaction — A query can reference tables or views, except if you specify the DIRECT option. If you specify the DIRECT option, the query can reference only tables.

Tips — If you want to query all data from the input data without any filtering or subsetting (`'INQUERY=SELECT * FROM` *table*'), you can use the table name in the INQUERY argument. However, you must also add DIRECT=YES to the OPTIONS argument. Here is an example:

```
...
'inquery="myDatabase"."myTableName"',
'options=direct=yes',
...
```

To pass single quotation marks around character literals, use two adjacent single quotation marks. This is an example.

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%'''
```

"output-database-name". "output-table-name"

specifies the location of the scoring model output table.

"output-database-name"

specifies the database where the table is created or currently exists.

Default: Current database

Requirement: The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"output-table-name"

specifies the name of the table to be created or the table that currently exists.

Requirement: The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Requirement: The output table can already exist. If the output table already exists, scored rows are inserted into the table along with any existing rows. If the output table already exists, the output columns must match the existing table's columns for the insert to succeed.

Interaction: The output table can be a temporary table by adding VOLATILE=YES in the OPTIONS parameter. The temporary table can be used only for the duration of the SQL session where it is created.

column

specifies the column or columns that are used as the primary index for the output table.

Requirements: The column must exist in the output table.

If there are multiple primary index columns, the column names must be separated by commas.

Tip: Specifying the same primary index for both the input and output tables enables Teradata to avoid redistribution of data across its AMPs.

NO PRIMARY INDEX

specifies that there is no primary index for the output table. Output rows are placed on the same Teradata Access Module Processor (AMP) that ran the scoring code for the corresponding input row.

NULL | 'OPTIONS=' | 'OPTIONS=*options-list*'

NULL | 'OPTIONS='
specifies that no options are used. You can use either NULL or 'OPTIONS=' to indicate that no options are used.

Example These two code lines are identical.

```
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
null);
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
'options=');
```

options-list
specifies one or more additional options for the stored procedure. If more than one option is specified, use a semicolon to separate the options. The *options-list* can include any of the following options:

CACHEEXPIRE=*hours*
specifies the number of hours to cache models in the file system.

Default 48

Range 0–720

Requirement To use this option, set CACHEMODEL=YES.

Note This option was added in 2023.05.

CACHEMODEL=YES | NO
specifies whether to cache models in the file system. Models are cached in /opt/SAS/glopdata on the Teradata database nodes. To cache models, you must specify an uppercase YES value. If you specify a lowercase or mixed-case value, or a NO value, or any other value, then models are removed from the file system after scoring is complete.

Default NO

Note This option was added in 2023.05.

CONTRACT=YES | NO
specifies whether to write the output metadata to the table specified in the OUTTABLE= parameter.

Default NO

Interaction If you specify CONTRACT=YES, the OUTKEY= parameter is ignored.

Tip The output is written to the table in the form of one row per output value with the sequence, name, data type, and length for each output.

DIRECT=YES | NO
specifies whether direct retrieve optimization is enabled.

Default NO

Interaction This option affects the stored procedure SQL generation.

Tip The direct retrieve optimization improves performance in the case where the input to the SAS Embedded Process is a table and the input query is `SELECT * FROM table`. When DIRECT=YES, the INQUERY= parameter is only the table name. No SELECT statement is needed.

DS2_KEEP=*column-name<... column-name>*
specifies the column or columns that are passed to the SAS_SCORE_EP procedure and are applied as a dynamic KEEP= statement in the sasscore_*modelname*.ds2 file.

Requirements If more than one column is specified, column names must be separated with spaces.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction Specify CONTRACT=YES to preview the available output columns without executing the model.

ENCODING=LATIN | UNICODE
specifies the character data encoding for the column data. This setting is for internationalization purposes.

Default LATIN

See *SAS National Language Support (NLS): Reference Guide*

EPTRACE=YES
specifies that journal messages are written to the journal table. Use the JOURNALTABLE= option to specify the journal table.

HASHBY=*column-name<..., column-name>*
specifies one or more columns to use for the HASH BY clause.

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction This option affects the stored procedure SQL generation.

Note Data is redistributed by hash code to the TERADATA AMPs based on this column or columns although there is no implied ordering to the groups.

JOURNALTABLE="*database-name*"."*journal-table-name*"
specifies the name and location of a table that the stored procedure creates. This table holds any journal messages and notes from the SAS journal facility that are produced when executing the store procedure.

"database-name"
specifies the database where the journal table is created.

Default Current database

Requirement The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"journal-table-name"
specifies the name of the journal table.

Requirement The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Requirement To use a journal table, you must set EPTRACE=YES.

Note Use a SELECT statement to retrieve the journal messages from the table after the stored procedure call is complete.

LOCALE=*sas-locale*
specifies a set of attributes in the SAS session that reflect the language, local conventions, and culture for a geographical region.

Requirement *sas-locale* must be one of the five-character POSIX values (for example, fr_FR).

See *SAS National Language Support (NLS): Reference Guide*

LOGLEVEL=*value*
specifies the logger level. The following levels are supported: allon, trace, debug, info, warn, error, fatal, off. The log file is created in the `/ep/home/log/` directory within the root folder where the SAS Embedded Process RPM file is installed.

Note This option was added in 2025.06.

ORDERBY=*column-name<..., column-name>*
specifies one or more columns to use for the LOCAL ORDER BY (BY groups) clause.

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction This option affects the stored procedure SQL generation.

SELECT_LIST=*column-name<..., column-name>*
specifies the column or columns that are used in the SQL that is generated by the SAS_SCORE_EP stored procedure.

Default * (asterisk) which indicates all columns

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

SQLTRACE="*database-name*"."*table-name*"
specifies the name and location of a table to hold the generated SQL code.

"database-name"
specifies the database where the journal table is created.

Default Current database

Requirement The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"table-name"
specifies the name of the table.

Requirement The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Tip This table is useful for stored procedure debugging or to reference later if you want to customize the SQL code that is used to call the SAS Embedded Process.

UNIQUE=YES | NO
specifies whether the primary index of the output table is unique.

Default NO

VOLATILE=YES | NO
specifies whether the output table is created as a temporary table.

Default NO

Interaction This option affects the stored procedure SQL generation.

Length The OPTIONS= parameter string can be from 0–20,000 characters long.

Requirements Each option must end with a semicolon, including the last option in the list.

If the OPTIONS= parameter string is greater than 1,000 characters, the OPTIONS= parameter must be the last one.

The options-list can be blank or NULL if no options are needed.

Tip Options that are not recognized as directives to the stored procedure are passed to the SAS Embedded Process as Query Band name-value pairs. If the SAS Embedded Process does

not recognize them, they are ignored. Up to ten user-defined Query Band name-value pairs can be specified in addition to the options listed here that are Query Band name-value pairs. The maximum length of the query band is 2048 characters. User-defined Query Band information is logged in Teradata Database Query Log (DBQL) that makes it useful for workload analysis and reporting.

Here are some tips for using the syntax:

- No specific parameter order is required. However, the INQUERY parameter must be the first parameter if its string is greater than 1,000 characters. Similarly, if the OPTIONS parameter string is greater than 1,000 characters, it must be the last parameter.
- Database object names (for example, tables and columns) must be enclosed in double quotation marks if they are Teradata reserved words. Otherwise, quotation marks are optional.
- Tables should be qualified with a database name. If a table name is not qualified with a database name, how the table name is resolved depends on which version of Teradata is used.
- All parameters are passed as strings to the SAS_SCORE_EP stored procedure, so they must be enclosed in single quotation marks. To pass a single quotation mark as part of the SQL within a parameter, use two adjacent single quotation marks as shown in the following example:

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%''',
```

7

Using PROC SCOREACCEL or CAS Actions for Amazon EMR, Cloudera, Databricks, Synapse, SingleStore, or Teradata

Running Scoring Models in Amazon EMR or Cloudera Using PROC SCOREACCEL or CAS Actions

Requirements for Hadoop Using PROC SCOREACCEL or CAS Actions

Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of the Hadoop client and server environment. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed on the Hadoop cluster. See *SAS In-Database Products: Deployment and Administration Guide*.

Publishing and Running Models in Amazon EMR or Cloudera

As of 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. With this update, SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06) no longer uses the Java API that is provided in the set of Hadoop JAR files. SAS Embedded Process jobs run on Spark only. Jobs on Hadoop are now dispatched using the REST API that is provided by Apache Livy. The execution of SAS Embedded Process jobs on MapReduce is no longer supported.

The details in this topic reflect those changes in Hadoop in 2023.09 and later.

Here are some requirements that apply to all of in-database scoring (publishing, deleting, and running models):

- Specify your connection options in the Hadoop caslib assignment, and specify the caslib= option in the model publishing and scoring actions or in PROC SCOREACCEL. Specifying the connection options in the actions or in PROC SCOREACCEL is deprecated. Using a caslib for connection options is strongly recommended and will be required in the future. See "Hadoop Data Connector" in *SAS Cloud Analytic Services: User's Guide*.

  Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

- In the caslib assignment, use the hadoopConfigDir= option to specify the client-side location of the spark-defaults.conf and sparkep-defaults.conf files.
- The deployment of SAS Embedded Process has changed. The deployment is now delivered from a SAS Viya platform repository. In previous releases, the deployment was delivered from a SAS Viya 3.5 repository. For more details about changes to installation and configuration, see *SAS In-Database Products: Deployment and Administration Guide*.

The following details are specific to the storage location of models in Hadoop:

- You can now publish a model to a Hive table, and this method is recommended. In order to publish to a Hive table, omit the MODELDIR= option from the publishModelExternal action or PROC SCOREACCEL. In the caslib, you must specify the Hive server. You can use the server= and port= options, or the uri= option, to connect to Hive.
- In the model publishing and scoring actions or in PROC SCOREACCEL, you can use the MODELDATABASE= option to specify the database where the model is stored. If MODELDATABASE= is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored. When you publish a model, if MODELDATABASE= is specified, the database must exist already.
- In prior releases, models were published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to publish or delete models in HDFS, specify the MODELDIR= option in the publishModelExternal action or PROC SCOREACCEL. In the caslib, you must specify the webHdfsUrl= option. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

The following details are specific to running models in Hadoop:

- To start a continuous session, submit the startSparkEP action before you run scoring models. If you do not submit startSparkEP, then the Embedded Process starts automatically, using the default resource settings that are configured for the cluster. To stop the Embedded Process for Spark, submit the stopSparkEP

action. In addition, the SAS Embedded Process stops if you terminate the CAS session.

- If you need to run existing models that are stored in HDFS, then include the MODELDIR= option in your scoring action or PROC SCOREACCEL. As noted above, be aware that storing models in HDFS is deprecated.
- In-database model scoring now uses Livy. You must specify the jobManagementUrl= in the caslib.
- SAS Embedded Process uses Spark to run a scoring model in Hadoop. MapReduce is no longer supported. In the runModelExternal action and in PROC SCOREACCEL, the PLATFORM=SPARK value is always used. If you specify PLATFORM=MAPRED, a warning is printed to the SAS log, and Spark is used instead of MapReduce. You can remove the PLATFORM= option from your code.
- The CLASSPATH= option is no longer supported in the runModelExternal= action or in PROC SCOREACCEL.

For more information about in-database scoring, see Chapter 4, "Introduction to In-Database Scoring," on page 35.

Deprecated: Reading HCatalog File Types

Overview of HCatalog File Types

IMPORTANT When you update to the 2023.09 release of SAS Embedded Process for Hadoop, MapReduce is no longer supported, and Spark is used instead. As a result, scoring that uses HCatalog is not supported.

HCatalog is a table management layer that presents a relational view of data in the HDFS to applications within the Hadoop ecosystem. With HCatalog, data structures that are registered in the Hive metastore, including SAS data, can be accessed through standard MapReduce code and Pig. HCatalog is part of Apache Hive.

In-database scoring in Hadoop uses HCatalog to read the native Hive file types Avro, ORC, Parquet, and RCFile.

By default, an output file is delimited. You can use the OUTRECORDFORMAT option in the RUNMODEL statement of PROC SCOREACCEL to write a binary file.

Consider these requirements when using HCatalog:

- Data that you want to access with HCatalog must first be registered in the Hive metastore.
- Support for HCatalog varies by vendor. For more information, see the documentation for your Hadoop vendor.

- For non-delimited file types such as Avro or Parquet, see "Additional Configuration Steps When Accessing Files That Are Processed Using HCatalog" on page 93.

Additional Configuration Steps When Accessing Files That Are Processed Using HCatalog

If you plan to access complex, non-delimited file types such as Avro or Parquet, through HCatalog, there are additional prerequisites:

- On the Hadoop cluster, the SAS Embedded Process for Hadoop install script automatically adds HCatalog JAR files to its configuration file. The HCatalog JAR files are added to the Embedded Process Map Reduce job class path during job submission.

  For information about installing and configuring the SAS Embedded Process for Hadoop, see *SAS In-Database Products: Deployment and Administration Guide*.

- You must include the HCatalog JAR files in your SAS_HADOOP_JAR_PATH environment variable. See "SAS_HADOOP_JAR_PATH Environment Variable" in *SAS Global Statements: Reference*.

- The hive-site.xml file must be in your SAS_HADOOP_CONFIG_PATH. See "SAS_HADOOP_CONFIG_PATH Environment Variable" in *SAS Global Statements: Reference*.

Troubleshooting Scoring in Hadoop

When either MSGLEVEL=I or in-database scoring fails, a message is written to the SAS log that contains an HTTP link to the YARN job log. Here is an example.

```
ERROR: Job job_1424277669708_2919 has failed. Please, see job log for
       details. Job tracking URL :
       http://name.unx.company.com:8088/proxy/application_1424277669708_2919/
```

The HTTP link is to a site that contains the job summary. On the job summary page, you can see the number of failed and successful tasks. If you click on the failed tasks, you see a list of task attempts. A log is assigned to each attempt. Once in the log page, you are able to see the error messages.

In addition, If you visit the Livy URL, you can see the sessions that are actively running.

Hadoop Permissions

Contact your Hadoop administrator to obtain any needed permissions.

If you use the MODELDIR= option, you need Write permission when writing files to that location.

Running Scoring Models in Databricks Using PROC SCOREACCEL or CAS Actions

Requirements for Databricks Using PROC SCOREACCEL or CAS Actions

IMPORTANT Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

In addition, note that Databricks has announced an end of support for Databricks Runtime 13.3 LTS on August 22, 2026. The sepcorespark RPM does not support Databricks versions later than 13. Customers are encouraged to adopt the accelserver RPM, which supports more-recent Databricks versions.

SAS in-database processing supports PROC SCOREACCEL or CAS actions for scoring in Databricks. Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of Databricks for Microsoft Azure or Amazon Web Services. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed in the Databricks cluster. See *SAS In-Database Products: Deployment and Administration Guide*.
- For in-database model scoring, the Databricks cluster must be started.
- For the platforms that PROC SCOREACCEL supports for storage and execution, see "Supported Platforms for In-Database Model Scoring" on page 155. The platforms are also supported for the Model Publishing and Scoring action set.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing to a Spark Table and Running in Databricks with PROC SCOREACCEL

As of 2023.10, you can publish a model to a Spark table, and this method is recommended. Previously you could publish a model only to an Amazon S3 or ADLS file system in order to run the model in Databricks. Here are the details for publishing and running a model that is stored in a Spark table:

- You can use the same Spark caslib to publish, run, and delete a model. In the caslib, specify platform=databricks. You must also specify the clusterId=, httpPath=, jobManagementUrl=, password, schema=, server=, and userName= data connector options.

  Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

- To publish a model to a Spark table, in PROC SCOREACCEL, or in the model publishing and scoring actions, specify EXTTYPE=DATABRICKS. Reference the Spark caslib. Do not specify the MODELDIR= option. You can use the MODELDATABASE= option to specify the database where the model is stored. The database must exist already. If MODELDATABASE= is not specified, then the database that is specified in the SCHEMA= option of the caslib is used.
- Start a continuous session by submitting the startSparkEP action before you run models. If you do not submit startSparkEP, then the SAS Embedded Process executes in a Databricks notebook, which causes the Embedded Process to

start and stop for each execution request. To stop the Embedded Process, submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

- To run the model, in PROC SCOREACCEL, or in the model publishing and scoring actions, specify EXTTYPE=SPARK. Reference the Spark caslib. If you specified the MODELDATABASE= option to publish the model, specify the same value to run the model.

See this example: "Example 5: Publish a Model to a Spark Table and Run in Azure Databricks" on page 205.

To run a model from a notebook instead of from SAS, see "Python and Scala API for the Sepcorespark RPM" on page 221.

Specifying the REST URL

To run models in Databricks with PROC SCOREACCEL, you must set the JOBMANAGEMENTURL= (alias RESTURL=) option in the Spark caslib. This value is the Databricks workspace URL.

- For Azure, you can find this URL in the Azure portal. Navigate to the Azure Databricks Service resource and copy the value of the **URL** field.
- For AWS, contact your administrator to request the workspace URL.

Assigning Role-Based Access Control in Azure

SAS in-database scoring can use Microsoft Entra application registration and role-based access control (RBAC). To run a model in Azure Databricks, the storage object must be mounted to Databricks File System (DBFS).

An on-site administrator can use the Azure portal to create the application registration, which automatically creates a service principal. An administrator can assign RBAC by using the Azure portal or the command-line interface (CLI). You must assign a role of Storage Blob Data Contributor to the application ID.

The following code is an example of using the CLI. (If you run this code on an existing application registration, the code resets the password only.)

```
#> az ad sp create-for-rbac -n my_app_name --skip-assignment
```

The command returns information like the following example:

```
{
"appId": "xxxx-xxxx-xxxx",
"displayName": "MyApp",
"name": "xxxx-xxxx-xxxx",
"password": "xxxx-xxxx-xxxx",
"tenant": "xxxx-xxxx-xxxx"
}
```

From the returned information, take note of the RBAC values appId, password, and tenant, which are required when you run a model. In some Azure interfaces and documentation, the application ID is referred to as the client ID.

Note: For publishing and deleting models in ADLS, but not running models, an alternative is device-code authentication. Several of the values are required when you submit the CASLIB statement. For more information, see "Requirements for Device Code Flow Authentication" in *SAS Cloud Analytic Services: User's Guide*.

Interactions for PROC SCOREACCEL

Interaction with SAS Data Connector to Spark

For Databricks or Synapse, the SAS Data Connector to Spark supports a PLATFORM= option that interacts with PROC SCOREACCEL. In the runModelExternal action, or in the RUNMODEL statement of PROC SCOREACCEL, you identify the external type as SPARK. In the Spark caslib, use the PLATFORM= option to specify DATABRICKS or SYNAPSE. For a summary of SRCTYPE= values, and the platform= value when it is appropriate, see "Supported Platforms for In-Database Model Scoring" on page 155.

Specify all connection options in the caslib. Connection options that are specified in PROC SCOREACCEL or in the runModelExternal action are ignored.

TIP

In the Spark caslib, the HADOOPCONFIGDIR= parameter is optional. If you want to pass additional Spark configuration properties to the execution environment or to the SAS Embedded Process, create a file named spark-defaults.conf that specifies the additional properties. You can specify this path name in the HADOOPCONFIGDIR= parameter.

SAS Embedded Process for Spark Action Set

The following actions are available in the Embedded Process for Spark action set. These actions are used with PROC SCOREACCEL or the runModelExternal action to run scoring programs in Databricks or Synapse. You can run multiple scoring programs and submit Scala code within one continuous session for better performance. For more information about the actions, see "SAS Embedded Process for Spark Action Set" in *SAS Visual Analytics: Programming Guide*.

Table 7.1 SAS Embedded Process for Spark

CAS Actions	Task
executeProgram	Execute user-written Scala code before or after scoring. An example is running an SQL join as input for scoring. Another example is saving the scoring output as a Parquet table.
startSparkEP and stopSparkEP	Start or stop a SAS Embedded Process for Spark continuous session. For Synapse, if you do not submit startSparkEP, then the Embedded Process starts automatically, using the default resource settings that are configured for the cluster. For Databricks, if you do not submit startSparkEP, then the Embedded Process executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request. To stop the Embedded Process for Spark, submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

Deprecated: Publishing in ADLS or S3 and Running in Databricks with PROC SCOREACCEL

You use different syntax for publishing and running:

- When you publish or delete a model, you interact with a storage location:
  - If you plan to run a model in Databricks on Amazon Web Services (AWS), you can publish the scoring model in Amazon Simple Storage Service (Amazon S3). For details, see "Deprecated: Requirements for Amazon S3" on page 100.
  - If you plan to run in Databricks on Azure, you can publish the scoring model in Azure Data Lake Storage Gen2 (ADLS). For details, see "Deprecated: Requirements for ADLS" on page 99.
- When you run a model, you interact with Databricks by using the SAS Embedded Process for Spark:
  - In PROC SCOREACCEL or in the runModelExternal action, set the external type as SPARK. To specify the storage location of the model, use the MODELDIR= option. The value for MODELDIR= must begin with `dbfs:` and must reference a storage object that has been mounted to Databricks File System (DBFS). For details about mounting a data source, see Databricks documentation. See also Chapter 9, "SCOREACCEL Procedure," on page 153

or "Model Publishing and Scoring Action Set" in *SAS Visual Analytics: Programming Guide*.

- A Spark caslib is required. In the Spark caslib, specify PLATFORM=DATABRICKS. For more information, see "Interactions for PROC SCOREACCEL" on page 97.
- To start a continuous session, submit the startSparkEP action before you run scoring models. If you do not submit startSparkEP, then the Embedded Process executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request. To stop the Embedded Process for Spark, you must submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

See these examples:

- "Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS" on page 208
- "Example 7: Publish a Model to ADLS and Run in Azure Databricks" on page 210

To run a model from a notebook instead of from SAS, see "Python and Scala API for the Sepcorespark RPM" on page 221.

Deprecated: Requirements for Publishing Models to ADLS or Amazon S3

Deprecated: Requirements for ADLS

Note: As of 2023.10 for Databricks and 2023.12 for Synapse, you are no longer required to publish models to a file system.

If you plan to run a model in Databricks or Synapse on Azure, you can choose to publish the scoring model in Azure Data Lake Storage Gen2 (ADLS). Here are the basic requirements to publish or delete a scoring model in ADLS:

- An ADLS caslib is required. In the caslib, specify the application ID.
- You must set the AZURETENANTID= server option or the AZURETENANTID= session option.
- In PROC SCOREACCEL or in the publishModelExternal action, set the external type as FILESYSTEM. Specify the ADLS caslib. Specify the password from role-based access control (RBAC). You cannot specify the RBAC password in the caslib assignment. For more information, see . See also Chapter 9, "SCOREACCEL Procedure," on page 153 or "Model Publishing and Scoring Action Set" in *SAS Visual Analytics: Programming Guide*.

ADLS Gen1 and Azure Blob Storage are not supported as a location to publish models.

Deprecated: Requirements for Amazon S3

Note: As of 2023.10 for Databricks, you can use a Spark caslib to publish models to a Spark table instead of publishing to a file system. For more information, see "Publishing to a Spark Table and Running in Databricks with PROC SCOREACCEL" on page 95.

If you plan to run a model in Databricks on Amazon Web Services (AWS), you can choose to publish the scoring model in Amazon Simple Storage Service (Amazon S3). Here are the requirements to publish or delete a scoring model in Amazon S3:

- In PROC SCOREACCEL or in the publishModelExternal action, set the external type as FILESYSTEM. See Chapter 9, "SCOREACCEL Procedure," on page 153 or "Model Publishing and Scoring Action Set" in *SAS Visual Analytics: Programming Guide*.
- An Amazon S3 caslib is required. For authentication options and other details, see "Amazon S3 Data Source" in *SAS Cloud Analytic Services: User's Guide*.

Running Scoring Models in Synapse Using PROC SCOREACCEL or CAS Actions

Requirements for Synapse Using PROC SCOREACCEL or CAS Actions

IMPORTANT Changes are required for SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06) beginning with 2026.04. Microsoft has announced an end of support for Azure Synapse Runtime for Apache Spark 3.4 as of March 31, 2026. Here are the details:

- You must deploy the SAS Embedded Process from the accelserver RPM. You can no longer deploy the sepcorespark RPM in Synapse.
- You must perform a pre-installation task to download an additional object library and create a wheel file. Then you install that wheel file when you install the sasindb wheel. See "Deploying SAS Embedded

Process for Synapse" in *SAS In-Database Products: Deployment and Administration Guide*.

- The SCOREACCEL procedure, the Model Publishing and Scoring action set, and the SAS Embedded Process for Spark action set are no longer supported for Synapse. Instead, use the ACCELERATOR procedure for in-database scoring and to stop and start the SAS Embedded Process.
- When you deploy the accelserver RPM, you can also use the in-database DATA step, which was not supported under the sepcorespark RPM.

SAS in-database processing supports scoring in Microsoft Azure Synapse Analytics. Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.06).
- In-database processing requires a specific version of Synapse. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed in the Synapse cluster. See *SAS In-Database Products: Deployment and Administration Guide*.
- As a requirement before you can publish a model to an SQL Server table, you must download the Microsoft JDBC Driver for SQL Server. For instructions, see "Configuration for In-Database Model Scoring" in *SAS In-Database Products: Deployment and Administration Guide*.
- For the platforms that PROC SCOREACCEL supports for storage and execution, see "Supported Platforms for In-Database Model Scoring" on page 155. The platforms are also supported for the Model Publishing and Scoring action set.

Overview of Publishing to an SQL Server Table and Running in Synapse

As of 2023.12, you can publish a model to an SQL Server table (also known as a dedicated SQL pool table), and this method is recommended. Previously you could publish a model only to the file system in ADLS in order to run the model in Synapse.

You need two different caslib assignments, one for the JDBC data connector and one for the Spark data connector:

- To publish the model to an SQL Server table, assign a JDBC caslib. For Synapse, you must identify both a schema and a database in the caslib in the following ways: Specify the schema value in the SCHEMA= option of the caslib. For the database value, the JDBC caslib does not support the DATABASE= option, so you must use the URI= option to specify a connection string for the Microsoft JDBC Driver for SQL Server. (The connection string includes the database value, but not the schema value.) A prerequisite is to download the Microsoft JDBC

Driver for SQL Server driver by following the link in the Synapse workspace. Then, in the Synapse workspace, in the JDBC connection strings, copy the connection string that is provided for JDBC (SQL authentication). Specify that connection string in the URI= option of the caslib. In addition, in the connection string, you must add the property useBulkCopyForBatchInsert=true.

In the PUBLISHMODEL statement of PROC SCOREACCEL, or in the publishModelExternal action, the MODELSCHEMA= option is required. Do not specify the MODELDIR= option. Specify EXTTYPE=SYNAPSE and reference your JDBC caslib.

- To start a continuous session and run the model, assign a Spark caslib. Specify platform=synapse in the caslib. For the jobManagementUrl= parameter, specify the Livy REST URL that is specific to the Spark pool that you want to connect to. Specify the necessary connection properties.

  Start a continuous session by submitting the startSparkEP action before you run models. If you do not submit startSparkEP, then the SAS Embedded Process starts automatically, using the default resource settings that are configured for the cluster. To stop the Embedded Process, submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

  In the RUNMODEL statement of PROC SCOREACCEL, or in the runModelExternal action, reference the Spark caslib. Specify EXTTYPE=SPARK. The MODELSCHEMA= option is required. For the INTABLE= and OUTTABLE= options, use a two-part name in the form *myschema.mytable*.

See this example: "Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse" on page 212.

To run a model from a notebook instead of from SAS, see "Python and Scala API for the Sepcorespark RPM" on page 221.

Deprecated: Overview of Publishing in ADLS and Running in Synapse

You use different syntax for publishing and running:

- If you plan to run in Synapse on Azure, you can publish the scoring model in Azure Data Lake Storage Gen2 (ADLS). For details, see "Deprecated: Requirements for Publishing Models to ADLS" on page 106.
- When you run a model, you interact with Synapse by using the SAS Embedded Process for Spark. In PROC SCOREACCEL or the runModelExternal action, specify SPARK for the external type. A Spark caslib is required. In the Spark caslib, specify PLATFORM=SYNAPSE. Fore more information, see "Interactions for PROC SCOREACCEL" on page 105. See also Chapter 9, "SCOREACCEL Procedure," on page 153 or "Model Publishing and Scoring Action Set" in *SAS Visual Analytics: Programming Guide*.
- To start a continuous session, submit the startSparkEP action before you run scoring models. If you do not submit startSparkEP, then the Embedded Process starts automatically, using the default resource settings that are configured for

the cluster. To stop the Embedded Process for Spark, submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

See this example: "Example 9: Publish a Model to ADLS and Run in Azure Synapse" on page 214.

To run a model from a notebook instead of from SAS, see "Python and Scala API for the Sepcorespark RPM" on page 221.

> **TIP** You might need to size your Spark session differently than the default. By default, the Spark session driver container starts one executor with one core and a maximum of 1GB of memory. You can specify the executorInstances=, executorCores=, and executorMemory= parameters when you submit the startSparkEP action. The maximum number of executors, cores, and memory depends on the size of your Spark pool.

Forming the Livy REST URL for Synapse

To run models in Synapse, you must set the JOBMANAGEMENTURL= (alias RESTURL=) option. A best practice is to set the value in the caslib that is passed to PROC SCOREACCEL or to the runModelExternal action. The Livy REST URL for the Synapse Spark pool includes three parameters:

endpoint`/livyApi/versions/`*livyApiVersion*`/sparkPools/`*sparkPoolName*

- The endpoint is the workspace development endpoint. An example is `https://myworkspace.dev.azuresynapse.net`.
- To find the Livy API version, see the Synapse Livy API documentation. An example is `2019-11-01-preview`.
- The final segment is the name of the Spark pool.

Running Models in Synapse against SQL Server Data

SAS in-database scoring supports reading from and writing to SQL tables. The SQL pool tables are accessed by the Microsoft Synapse SQL Connector. Here are some key concepts:

- You must use a Spark caslib with the following parameters: the schema= is the SQL database name, the username= is the application ID, and the password= is the application ID's secret.

- For the INTABLE= and OUTTABLE= values in PROC SCOREACCEL or in the runModelExternal action, use a two-part name. Specify the SQL server schema name followed by a dot and then the table name.
- Output tables cannot be overwritten. In order to write to an SQL table, the table must not exist.

Here are the basic requirements to run models against SQL tables:

- Users must be created in the dedicated SQL pool.
- Users must have the db_exporter role in the database or dedicated SQL pool that you want to transfer data to and from.
- Users must have the Storage Blob Data Contributor role on the default storage account.

Assigning Role-Based Access Control in Azure

SAS in-database scoring can use Microsoft Entra application registration and role-based access control (RBAC). To run a model in Azure Synapse, the appropriate privileges must be assigned in Synapse to access the Spark pool.

An on-site administrator can use the Azure portal to create the application registration, which automatically creates a service principal. An administrator can assign RBAC by using the Azure portal or the command-line interface (CLI). You must assign a role of Storage Blob Data Contributor to the application ID.

The following code is an example of using the CLI. (If you run this code on an existing application registration, the code resets the password only.)

```
#> az ad sp create-for-rbac -n my_app_name --skip-assignment
```

The command returns information like the following example:

```
{
"appId": "xxxx-xxxx-xxxx",
"displayName": "MyApp",
"name": "xxxx-xxxx-xxxx",
"password": "xxxx-xxxx-xxxx",
"tenant": "xxxx-xxxx-xxxx"
}
```

From the returned information, take note of the RBAC values appId, password, and tenant, which are required when you run a model. In some Azure interfaces and documentation, the application ID is referred to as the client ID.

Note: For publishing and deleting models in ADLS, but not running models, an alternative is device-code authentication. Several of the values are required when you submit the CASLIB statement. For more information, see "Requirements for Device Code Flow Authentication" in *SAS Cloud Analytic Services: User's Guide*.

Interactions for PROC SCOREACCEL

Interaction with SAS Data Connector to Spark

For Databricks or Synapse, the SAS Data Connector to Spark supports a PLATFORM= option that interacts with PROC SCOREACCEL. In the runModelExternal action, or in the RUNMODEL statement of PROC SCOREACCEL, you identify the external type as SPARK. In the Spark caslib, use the PLATFORM= option to specify DATABRICKS or SYNAPSE. For a summary of SRCTYPE= values, and the platform= value when it is appropriate, see "Supported Platforms for In-Database Model Scoring" on page 155.

Specify all connection options in the caslib. Connection options that are specified in PROC SCOREACCEL or in the runModelExternal action are ignored.

> **TIP**
>
> In the Spark caslib, the HADOOPCONFIGDIR= parameter is optional. If you want to pass additional Spark configuration properties to the execution environment or to the SAS Embedded Process, create a file named spark-defaults.conf that specifies the additional properties. You can specify this path name in the HADOOPCONFIGDIR= parameter.

SAS Embedded Process for Spark Action Set

The following actions are available in the Embedded Process for Spark action set. These actions are used with PROC SCOREACCEL or the runModelExternal action to run scoring programs in Databricks or Synapse. You can run multiple scoring programs and submit Scala code within one continuous session for better performance. For more information about the actions, see "SAS Embedded Process for Spark Action Set" in *SAS Visual Analytics: Programming Guide*.

Table 7.2 *SAS Embedded Process for Spark*

CAS Actions	Task
executeProgram	Execute user-written Scala code before or after scoring. An example is running an SQL join as input for scoring. Another example is saving the scoring output as a Parquet table.
startSparkEP and stopSparkEP	Start or stop a SAS Embedded Process for Spark continuous session.

CAS Actions	Task
	For Synapse, if you do not submit startSparkEP, then the Embedded Process starts automatically, using the default resource settings that are configured for the cluster. For Databricks, if you do not submit startSparkEP, then the Embedded Process executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request. To stop the Embedded Process for Spark, submit the stopSparkEP action. In addition, the Embedded Process stops if you terminate the CAS session.

Deprecated: Requirements for Publishing Models to ADLS

Note: As of 2023.10 for Databricks and 2023.12 for Synapse, you are no longer required to publish models to a file system.

If you plan to run a model in Databricks or Synapse on Azure, you can choose to publish the scoring model in Azure Data Lake Storage Gen2 (ADLS). Here are the basic requirements to publish or delete a scoring model in ADLS:

- An ADLS caslib is required. In the caslib, specify the application ID.
- You must set the AZURETENANTID= server option or the AZURETENANTID= session option.
- In PROC SCOREACCEL or in the publishModelExternal action, set the external type as FILESYSTEM. Specify the ADLS caslib. Specify the password from role-based access control (RBAC). You cannot specify the RBAC password in the caslib assignment. For more information, see . See also Chapter 9, "SCOREACCEL Procedure," on page 153 or "Model Publishing and Scoring Action Set" in *SAS Visual Analytics: Programming Guide*.

ADLS Gen1 and Azure Blob Storage are not supported as a location to publish models.

Running Scoring Models in SingleStore Using PROC SCOREACCEL or CAS Actions

Requirements for SingleStore Using PROC SCOREACCEL or CAS Actions

IMPORTANT

The SCOREACCEL procedure and the Model Publishing and Scoring action set are deprecated. Instead, customers are encouraged to use the ACCELERATOR procedure for in-database scoring. SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Here are the basic requirements:

- You must license and deploy the product SAS SpeedyStore.
- PROC ACCELERATOR supports the integrated instance of the SingleStore database. The procedure does not support an external database.
- See "Requirements for SAS SpeedyStore" in *System Requirements for the SAS Viya Platform*.
- When you order SAS SpeedyStore, installation of SAS Embedded Process is not required as for other data sources. SingleStore is delivered with the SAS Viya platform order, and SAS Embedded Process is already installed in SingleStore. For more information about deployment, see *SAS SpeedyStore: Administration and Configuration Guide*.

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing and Running in SingleStore Using PROC SCOREACCEL or CAS Actions

For SingleStore, you do not start or stop the SAS Embedded Process continuous session.

SAS in-database processing supports PROC SCOREACCEL or CAS actions for model scoring in SingleStore. You connect to the data source by using the SAS SpeedyStore data connector. You must license and deploy the product SAS SpeedyStore. For more information, see "SAS SpeedyStore Data Connector" in *SAS Cloud Analytic Services: User's Guide*. In-database model scoring is not supported by the SingleStore standard data connector.

For an in-database scoring example, see "Example 10: Publish, Run, and Delete a Model in SingleStore" on page 218.

Running Scoring Models in Teradata Using PROC SCOREACCEL or CAS Actions

Requirements for Teradata Using PROC SCOREACCEL or CAS Actions

IMPORTANT

The SCOREACCEL procedure and the Model Publishing and Scoring action set are deprecated. Instead, customers are encouraged to use the ACCELERATOR procedure for in-database scoring. SAS Embedded Process

requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Here are the basic requirements:

- You must license SAS Data Accelerator (known as SAS In-Database Technologies prior to 2026.08).
- In-database processing requires a specific version of the Teradata client and server environment. See "Requirements for SAS In-Database Technologies" in *System Requirements for the SAS Viya Platform*.
- SAS Embedded Process must be installed in Teradata. See *SAS In-Database Products: Deployment and Administration Guide*

Choosing PROC ACCELERATOR or PROC SCOREACCEL

Customers are encouraged to adopt PROC ACCELERATOR. PROC SCOREACCEL has been deprecated for Databricks, SingleStore, Synapse, and Teradata.

See "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.

IMPORTANT For some of the supported data sources, SAS Embedded Process requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Publishing and Running Models in Teradata Using PROC SCOREACCEL

SAS Embedded Process is a SAS server process that can run scoring models inside the Teradata Enterprise Data Warehouse (EDW).

- See Chapter 9, "SCOREACCEL Procedure," on page 153.
- See the RunModelExternal, DeleteModelExternal, and PublishModelExternal actions in *SAS Visual Analytics: Programming Guide*.
- For more information about in-database scoring, see Chapter 4, "Introduction to In-Database Scoring," on page 35.

- It is recommended to provide all the connection options in a Teradata caslib assignment. Specify the CASLIB= option in the model publishing and scoring actions or in PROC SCORECACCEL. Specifying connection options in the actions or in PROC SCORECACCEL is deprecated.

SAS_SCORE_EP Stored Procedure

In the SAS Viya platform, you do not submit the SAS_SCORE_EP stored procedure to run a scoring model. Instead, the stored procedure is called on your behalf when you submit PROC ACCELERATOR, PROC SCOREACCEL, or the runModelExternal CAS action. Therefore, it is unlikely that you need the syntax for the stored procedure. Most users rely on the syntax of PROC ACCELERATOR, PROC SCOREACCEL, or the runModelExternal action.

If needed, the stored procedure syntax is supported in the following way: In PROC ACCELERATOR, use the OPTIONS= argument. In PROC SCOREACCEL, use the EPOPTIONS= argument. Use the argument in the RUNMODEL statement to submit a string of options. If the argument is specified, then the string is passed as the value of the OPTIONS parameter on the resulting call to the SAS_SCORE_EP stored procedure.

Use the following syntax in the string.

SAS_SYSFNLIB.SAS_SCORE_EP

```
('MODELTABLE="database"."model-table-name" ',
 'MODELNAME=model-name',
 'INQUERY=SELECT ...'
 'OUTTABLE= "output-database-name". "output-table-name" ',
 'OUTKEY=column<..., column,> | NO PRIMARY INDEX,
 NULL | 'OPTIONS=' | 'OPTIONS=options-list'
 );
```

Parameters

"database"

specifies the name of the database that contains the model table.

Default: Database in the current session

Requirements: The database must be the same as the one specified in the DATABASE argument.

The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"model-table-name"

specifies the name of the model table where the scoring files were published.

Requirements: The model name must be the same as the one specified in the MODELNAME argument.

The maximum table name length is 128 characters, and it must be a valid Teradata table name.

model-name
specifies the name of the model.

SELECT ...
specifies a SELECT statement that defines the inputs to the SAS Embedded Process.

Range — The INQUERY= parameter string can be up to 30,000 characters long.

Restrictions — The maximum number of characters in the query is 30,000.

The maximum number of input and output columns is 1024.

Requirements — If the query is greater than 1,000 characters, the INQUERY= parameter must be the first parameter listed in the stored procedure.

The SELECT statement must be syntactically correct SQL.

Interaction — A query can reference tables or views, except if you specify the DIRECT option. If you specify the DIRECT option, the query can reference only tables.

Tips — If you want to query all data from the input data without any filtering or subsetting (`'INQUERY=SELECT * FROM` *table*'), you can use the table name in the INQUERY argument. However, you must also add DIRECT=YES to the OPTIONS argument. Here is an example:

```
...
'inquery="myDatabase"."myTableName"',
'options=direct=yes',
...
```

To pass single quotation marks around character literals, use two adjacent single quotation marks. This is an example.

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%'''
```

"output-database-name"."output-table-name"
specifies the location of the scoring model output table.

"output-database-name"
specifies the database where the table is created or currently exists.

Default — Current database

Requirement — The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"output-table-name"
specifies the name of the table to be created or the table that currently exists.

Requirement The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Requirement The output table can already exist. If the output table already exists, scored rows are inserted into the table along with any existing rows. If the output table already exists, the output columns must match the existing table's columns for the insert to succeed.

Interaction The output table can be a temporary table by adding VOLATILE=YES in the OPTIONS parameter. The temporary table can be used only for the duration of the SQL session where it is created.

column
specifies the column or columns that are used as the primary index for the output table.

Requirements The column must exist in the output table.

If there are multiple primary index columns, the column names must be separated by commas.

Tip Specifying the same primary index for both the input and output tables enables Teradata to avoid redistribution of data across its AMPs.

NO PRIMARY INDEX
specifies that there is no primary index for the output table. Output rows are placed on the same Teradata Access Module Processor (AMP) that ran the scoring code for the corresponding input row.

NULL | 'OPTIONS=' | 'OPTIONS=*options-list*'

NULL | 'OPTIONS='
specifies that no options are used. You can use either NULL or 'OPTIONS=' to indicate that no options are used.

Example These two code lines are identical.

```
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
null);
call sas_sysfnlib.sas_score_ep ('modeltable=...', modelname=...',
'inquery=...', 'outtable=scored_output1', 'outkey=no primary index',
'options=');
```

options-list
specifies one or more additional options for the stored procedure. If more than one option is specified, use a semicolon to separate the options. The *options-list* can include any of the following options:

CACHEEXPIRE=*hours*
specifies the number of hours to cache models in the file system.

Default 48

Range 0–720

Requirement To use this option, set CACHEMODEL=YES.

Note This option was added in 2023.05.

CACHEMODEL=YES | NO
specifies whether to cache models in the file system. Models are cached in /opt/SAS/glopdata on the Teradata database nodes. To cache models, you must specify an uppercase YES value. If you specify a lowercase or mixed-case value, or a NO value, or any other value, then models are removed from the file system after scoring is complete.

Default NO

Note This option was added in 2023.05.

CONTRACT=YES | NO
specifies whether to write the output metadata to the table specified in the OUTTABLE= parameter.

Default NO

Interaction If you specify CONTRACT=YES, the OUTKEY= parameter is ignored.

Tip The output is written to the table in the form of one row per output value with the sequence, name, data type, and length for each output.

DIRECT=YES | NO
specifies whether direct retrieve optimization is enabled.

Default NO

Interaction This option affects the stored procedure SQL generation.

Tip The direct retrieve optimization improves performance in the case where the input to the SAS Embedded Process is a table and the input query is `SELECT * FROM table`. When DIRECT=YES, the INQUERY= parameter is only the table name. No SELECT statement is needed.

DS2_KEEP=*column-name*<... *column-name*>
specifies the column or columns that are passed to the SAS_SCORE_EP procedure and are applied as a dynamic KEEP= statement in the sasscore_*modelname*.ds2 file.

Requirements If more than one column is specified, column names must be separated with spaces.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction Specify CONTRACT=YES to preview the available output columns without executing the model.

ENCODING=LATIN | UNICODE
specifies the character data encoding for the column data. This setting is for internationalization purposes.

Default LATIN

See *SAS National Language Support (NLS): Reference Guide*

EPTRACE=YES
specifies that journal messages are written to the journal table. Use the JOURNALTABLE= option to specify the journal table.

HASHBY=*column-name<..., column-name>*
specifies one or more columns to use for the HASH BY clause.

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction This option affects the stored procedure SQL generation.

Note Data is redistributed by hash code to the TERADATA AMPs based on this column or columns although there is no implied ordering to the groups.

JOURNALTABLE=*"database-name"."journal-table-name"*
specifies the name and location of a table that the stored procedure creates. This table holds any journal messages and notes from the SAS journal facility that are produced when executing the store procedure.

"database-name"
specifies the database where the journal table is created.

Default Current database

Requirement The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"journal-table-name"
specifies the name of the journal table.

Requirement The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Requirement To use a journal table, you must set EPTRACE=YES.

Note Use a SELECT statement to retrieve the journal messages from the table after the stored procedure call is complete.

LOCALE=*sas-locale*
specifies a set of attributes in the SAS session that reflect the language, local conventions, and culture for a geographical region.

Requirement *sas-locale* must be one of the five-character POSIX values (for example, fr_FR).

See *SAS National Language Support (NLS): Reference Guide*

LOGLEVEL=*value*
specifies the logger level. The following levels are supported: allon, trace, debug, info, warn, error, fatal, off. The log file is created in the **`/ep/home/log/`** directory within the root folder where the SAS Embedded Process RPM file is installed.

Note This option was added in 2025.06.

ORDERBY=*column-name<..., column-name>*
specifies one or more columns to use for the LOCAL ORDER BY (BY groups) clause.

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

Interaction This option affects the stored procedure SQL generation.

SELECT_LIST=*column-name<..., column-name>*
specifies the column or columns that are used in the SQL that is generated by the SAS_SCORE_EP stored procedure.

Default * (asterisk) which indicates all columns

Requirements If more than one column is specified, column names must be separated with commas.

The maximum column name length is 128 characters, and it must be a valid Teradata column name.

SQLTRACE=*"database-name"."table-name"*
specifies the name and location of a table to hold the generated SQL code.

"database-name"
specifies the database where the journal table is created.

Default Current database

Requirement The maximum database name length is 128 characters, and it must be a valid Teradata database name.

"table-name"
specifies the name of the table.

Requirement The maximum table name length is 128 characters, and it must be a valid Teradata table name.

Tip This table is useful for stored procedure debugging or to reference later if you want to customize the SQL code that is used to call the SAS Embedded Process.

UNIQUE=YES | NO
specifies whether the primary index of the output table is unique.

Default NO

VOLATILE=YES | NO
specifies whether the output table is created as a temporary table.

Default NO

Interaction This option affects the stored procedure SQL generation.

Length The OPTIONS= parameter string can be from 0–20,000 characters long.

Requirements Each option must end with a semicolon, including the last option in the list.

If the OPTIONS= parameter string is greater than 1,000 characters, the OPTIONS= parameter must be the last one.

The options-list can be blank or NULL if no options are needed.

Tip Options that are not recognized as directives to the stored procedure are passed to the SAS Embedded Process as Query Band name-value pairs. If the SAS Embedded Process does not recognize them, they are ignored. Up to ten user-defined Query Band name-value pairs can be specified in addition to the options listed here that are Query Band name-value pairs. The maximum length of the query band is 2048 characters. User-defined Query Band information is logged in Teradata Database Query Log (DBQL) that makes it useful for workload analysis and reporting.

Here are some tips for using the syntax:

- No specific parameter order is required. However, the INQUERY parameter must be the first parameter if its string is greater than 1,000 characters. Similarly, if the OPTIONS parameter string is greater than 1,000 characters, it must be the last parameter.
- Database object names (for example, tables and columns) must be enclosed in double quotation marks if they are Teradata reserved words. Otherwise, quotation marks are optional.

- Tables should be qualified with a database name. If a table name is not qualified with a database name, how the table name is resolved depends on which version of Teradata is used.
- All parameters are passed as strings to the SAS_SCORE_EP stored procedure, so they must be enclosed in single quotation marks. To pass a single quotation mark as part of the SQL within a parameter, use two adjacent single quotation marks as shown in the following example:

```
'INQUERY=select * from my_input_tbl where name like ''%Jones%''',
```

8

ACCELERATOR Procedure

Overview: ACCELERATOR Procedure

What Does the ACCELERATOR Procedure Do?

PROC ACCELERATOR is an interactive procedure that provides an interface to external data sources for model publishing and scoring.

- PROC ACCELERATOR is an advanced replacement for the SCOREACCEL procedure. For differences between the two procedures, see "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124.
- Models are supported in the form of DATA step code, DS2 code, or analytic stores.
- The procedure supports the following data sources:
  - Databricks on Microsoft Azure as of 2024.11.
  - Microsoft Azure Synapse Analytics as of 2025.12.
  - SingleStore as of 2025.07. You must license and deploy the product SAS SpeedyStore.
  - Teradata as of 2026.08.
- After a model is published to an external data source, running the model invokes the SAS Embedded Process.
- For Databricks or Synapse, you use PROC ACCELERATOR to start and stop the SAS Embedded Process continuous session. For SingleStore or Teradata, you do not start or stop a session.
- Models can be removed from an external data source by using the DELETEMODEL statement.
- For a list of the supported scoring models and other information, see Part 3, "In-Database Model Scoring," on page 33.

Concepts: ACCELERATOR Procedure

Specifying One-Part, Two-Part, or Three-Part Table Names

Overview of Table Name Rules

Data sources support either two-part or three-part table names. Depending on the data source, you specify a catalog, database, schema, or a combination of these values, in the LIBNAME assignment. PROC ACCELERATOR uses the LIBNAME values as the default values for catalog, database, or schema in the MODELTABLENAME=, INPUTTABLENAME=, and OUTPUTTABLENAME= options.

For simplicity, it is recommended that you specify a one-part name in the PROC ACCELERATOR options, and take the defaults from the LIBNAME assignment for the catalog, database, or schema values.

PROC ACCELERATOR enables you to specify a one-part, two-part, or three-part table name in the MODELTABLENAME=, INPUTTABLENAME=, and OUTPUTTABLENAME= options. You can specify a two-part or three-part name if you need to override the defaults from the LIBNAME assignment. For example, you might want to create output data in a different database than the input data. An exception is Synapse, where you can override the database in the RUNMODEL= statements but not in the PUBLISHMODEL or DELETEMODEL statements. The behavior for each data source is described in the following topics.

PROC ACCELERATOR uses the authentication information specified in the LIBNAME to connect to the data source.

If a valid table name cannot be formed, PROC ACCELERATOR writes an error to the SAS log.

For all data sources, you must surround a name with backticks (`) if the name contains periods, reserved words, or special characters. The following example requires backticks around the table name **`my.table`** because the name contains a period:

```
modeltablename="myschema.`my.table`"
```

Table Names for Databricks

Databricks supports three-part table names, in this form:

```
catalog.schema.table
```

In the Spark LIBNAME statement for Databricks, use the SCHEMA= option, and do not use the DATABASE= option. Starting in 2025.12, you can use the CATALOG= LIBNAME option to specify the Databricks Unity Catalog.

In releases prior to 2025.12, you must use JDBC driver properties to specify the catalog. If you prefer, you can continue to use JDBC driver properties to specify the catalog, or specify both catalog and schema. The ConnCatalog= and ConnSchema= properties are supported in the URI= or PROPERTIES= LIBNAME options.

If you specify the catalog and schema in the PROPERTIES= option and you also specify them in the CATALOG= and SCHEMA LIBNAME options, then the values in the PROPERTIES= option take precedence.

The following table describes the PROC ACCELERATOR behavior for Databricks.

Table 8.1 *Forming Three-Part Table Names for Databricks*

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Read the option value as the table name. Use the catalog from the CATALOG= LIBNAME option or from the ConnCatalog= driver property. Use the schema from the SCHEMA= LIBNAME option or from the ConnSchema= driver property.
Two-part name	Read the option value as *schema.table*. Use the catalog from the CATALOG= LIBNAME option or from the ConnCatalog= driver property.
Three-part name	Use the value as is.

Table Names for SingleStore (SAS SpeedyStore)

SingleStore supports two-part table names, in this form:

```
database.table
```

In the SingleStore LIBNAME statement, specify the database in the DATABASE= option. SingleStore does not support a schema name.

The following table describes the PROC ACCELERATOR behavior for SingleStore.

Table 8.2 *Forming Two-Part Table Names for SingleStore*

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Use the database from the LIBNAME statement.
Two-part name	Use the option value as is.
Three-part name	Three-part names are not supported in SingleStore. Write an error to the SAS log.

Table Names for Synapse

Microsoft Azure Synapse Analytics requires three-part table names, in this form:

```
database.schema.table
```

Use the Microsoft SQL Server LIBNAME statement for your connection. You must connect to a dedicated SQL pool in the Synapse workspace. The pool must be unpaused. Specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option.

The following table describes the PROC ACCELERATOR behavior for Synapse. When you publish or delete a model, you cannot override the database value from the LIBNAME assignment. However, when you run the model, you can specify a different database in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= value.

Table 8.3 *Forming Three-Part Table Names for Synapse*

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	**PROC ACCELERATOR Behavior**	
	PUBLISHMODEL or DELETEMODEL Statement	**RUNMODEL Statement**
One-part name	Use the schema and database from the LIBNAME statement.	Read the option value as the table name. Use the schema and database from the LIBNAME statement.
Two-part name	Use the value as *schema.table*. Use the database from the LIBNAME statement.	Read the option value as *schema.table*. Use the database from the LIBNAME statement.
Three-part name	Use the value if the database in the three-part name matches the database from the LIBNAME statement. If the database does not match, write an error to the SAS log.	Use the option value as is.

Table Names for Teradata

Teradata supports two-part table names, in this form:

```
database.table
```

In the Teradata LIBNAME statement, use the DATABASE= option, and do not use the SCHEMA= option. Although the SCHEMA= option is supported by the LIBNAME statement, it is used to view or modify another user's tables. For in-database processing, this usage is discouraged.

The following table describes the PROC ACCELERATOR behavior for Teradata.

Table 8.4 *Forming Three-Part Table Names for Teradata*

Value in the MODELTABLENAME=, INPUTTABLENAME=, or OUTPUTTABLENAME= Option	PROC ACCELERATOR Behavior
One-part name	Read the option value as the table name. Use the database from the LIBNAME statement.
Two-part name	Use the option value as is.
Three-part name	Use the option value as is. However, three-part names might cause errors.

Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL

The following table summarizes differences from PROC SCOREACCEL.

Table 8.5 *Support for Tasks and Features for In-Database Scoring*

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
Deploy the SAS Embedded Process.[1]	For Databricks or Synapse, you deploy the accelserver RPM file.	For Databricks or Synapse, you deploy the sepcorespark RPM file.
	For SingleStore, you must license and deploy the product SAS SpeedyStore, for both procedures.	

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
	For Teradata, you deploy the sepcoretera RPM file, for both procedures.	
Start or stop the SAS Embedded Process continuous session.	For Databricks or Synapse, submit the procedure with a STARTSESSION or STOPSESSION statement.	For Databricks or Synapse, submit the CAS action sparkEmbeddedProcess.startSparkEP.startSparkEP or sparkEmbeddedProcess.startSparkEP.stopSparkEP.
	For SingleStore or Teradata, you do not start or stop a continuous session.	
Establish a connection to the data source.	Submit a LIBNAME statement or use another method to assign a SAS library.	Submit a CASLIB statement or use another method to assign a CAS library.
Reference the data source within the procedure.	Specify the libref.	Specify the casref.
Reference the input data or output data.	One-part, two-part, or three-part table names are supported.	One-part name or a two-part table names are supported.
	For Databricks or Synapse, you can read the input data and write the output data by using custom Scala code.	Scala code is not supported.
Interact with a CAS server.	There is no interaction with CAS.	PROC SCOREACCEL calls CAS actions to complete the tasks. You can use the Model Publishing and Scoring action set directly, instead of using PROC SCOREACCEL.
Deprecated features	Deprecated options are omitted.	Deprecated options are available.
Supported data sources	Azure Databricks Azure Synapse SingleStore Teradata	CAS Amazon EMR AWS Databricks Azure Databricks Azure Synapse Cloudera SingleStore

Task or Feature	PROC ACCELERATOR	PROC SCOREACCEL
		Teradata

1 For more information, see "Deployment Details for PROC ACCELERATOR" on page 126.

Deployment Details for PROC ACCELERATOR

Deployment Details for Databricks and Synapse

It is important to note the difference in deployment for PROC ACCELERATOR compared to PROC SCOREACCEL. Here are the details:

- An additional RPM file is now available for the SAS Embedded Process for Spark. The new deployment is referred to as the *accelserver* RPM file. The previous deployment is referred to as the *sepcorespark* RPM file.
  - For Databricks, the accelserver RPM and the sepcorespark RPM are supported (on separate clusters). However, the sepcorespark RPM is deprecated, and customers are encouraged to use the accelserver RPM instead.
  - For Synapse, starting with 2026.04, only the accelserver RPM is supported. In addition, a new pre-installation task is required in order to download an additional object library and create a wheel file. Then you install that wheel file when you install the sasindb wheel. The sepcorespark RPM is no longer supported.
- PROC ACCELERATOR is supported if you install the Python wheel from the accelserver RPM file. The Python wheel from the sepcorespark RPM file does not support PROC ACCELERATOR.
- PROC SCOREACCEL, the Model Publishing and Scoring action set, and the SAS Embedded Process for Spark action set are supported if you install the Python wheel from the sepcorespark RPM file. The Python wheel from the accelserver RPM file does not support PROC SCOREACCEL or the action sets.
- For Databricks and Synapse, you cannot install both RPMs on the same cluster. If you need the functionality of both RPMs, they must be installed on separate clusters.

For deployment information, see *SAS In-Database Products: Deployment and Administration Guide*.

Deployment Details for SingleStore

The procedure supports SingleStore if you license and deploy the product SAS SpeedyStore. The procedure supports the integrated instance of SingleStore database only. The procedure does not support an external database. No additional deployment or configuration tasks are required for PROC ACCELERATOR.

For deployment information, see *SAS SpeedyStore: Administration and Configuration Guide*.

Deployment Details for Teradata

The SAS Embedded Process for Teradata is deployed from the sepcoretera RPM. No additional deployment or configuration tasks are required for PROC ACCELERATOR. You must install SAS Embedded Process 2026.08 or later.

For deployment information, see *SAS In-Database Products: Deployment and Administration Guide*.

Syntax: ACCELERATOR Procedure

PROC ACCELERATOR *required-argument*;

STARTSESSION *required-arguments* <*optional-arguments*>;

STOPSESSION;

PUBLISHMODEL *required-arguments* <*optional-argument*>;

RUNMODEL *required-arguments* <*optional-arguments*>;

DELETEMODEL *required-argument*;

Statement	Task	Example
PROC ACCELERATOR	Submit statements to start or stop a continuous session, or publish, run, or delete a model in a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
STARTSESSION	Start the SAS Embedded Process continuous session	Ex. 1, Ex. 2, Ex. 6
STOPSESSION	Stop the SAS Embedded Process continuous session	Ex. 1, Ex. 2, Ex. 6
PUBLISHMODEL	Publish a model to a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

Statement	Task	Example
RUNMODEL	Run a model in a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
DELETEMODEL	Delete a model from a data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6

PROC ACCELERATOR Statement

Starts or stops a continuous session, or publishes, runs, or deletes a model in a data source.

Syntax

PROC SCOREACCEL LIBRARY=*libref*;

Required Argument

LIBRARY=*libref*
specifies the name of the SAS library where the model is run. A libref is a name that you use to reference a SAS library. You must assign the SAS library before you can specify its libref in this option. The LIBNAME statement is a common way of assigning a SAS library. For more information, see *SAS/ACCESS for Relational Databases: Reference*.

STARTSESSION Statement

Starts the SAS Embedded Process continuous session.

Note: Do not use the STARTSESSION and STOPSESSION statements with SingleStore or Teradata.

Syntax

STARTSESSION
PLATFORM=DATABRICKS | SYNAPSE
PLATFORMENDPOINT="*rest-url*"
PLATFORMPASSWORD=*endpoint-password*
PLATFORMUSER=*endpoint-user-identifier*
<*startsession-optional-arguments*>;

Required Arguments

PLATFORM=CLOUDERA | DATABRICKS | SYNAPSE
specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

PLATFORMENDPOINT="*rest-url*"
specifies the URL that is used to submit execution requests over a REST interface to services such as Apache Livy or Databricks execution context. For Databricks, this value is the workspace URL. For Synapse, this value is the Livy REST URL.

See "Specifying the Workspace URL for Databricks" on page 63

"Forming the Livy REST URL for Synapse" on page 74

PLATFORMPASSWORD=*endpoint-password*
specifies the Livy endpoint application secret.

Requirement The PLATFORMPASSWORD= option is required for Azure Synapse.

PLATFORMUSER=*endpoint-user-identifier*
specifies the Livy endpoint user name.

This value is the application ID.

Requirement The PLATFORMUSER= option is required for Azure Synapse.

Optional Arguments

AUTORESTART=
automatically restarts a session if it is terminated on the server side due to a timeout or other events.

Default TRUE

Restriction The AUTORESTART= option is supported for Synapse only.

MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*
specifies the maximum length for unbounded data types (such as BLOB, CLOB, TEXT, or long STRING types) when they are Read into the SAS Embedded Process. The option ensures consistent handling of very large character or binary columns from the database. If data in an input column exceeds the MAXUNBOUNDEDLENGTH= value, truncation could occur during Read. Truncation could occur during Write, but only for an output column that is also an input column. The option does not apply to any output column that is not an input column.

You can set the MAXUNBOUNDEDLENGTH= option in either the STARTSESSION statement or RUNMODEL statement. When you specify the option in RUNMODEL, you override the value in STARTSESSION. Note that the DBMAX_TEXT= LIBNAME option does not affect processing in PROC ACCELERATOR.

Default 32767

Note The MAXUNBOUNDEDLENGTH= option was added in 2025.09.

TIMEOUT=*n*
specifies the number of minutes the Livy session waits for execution requests. If the time-out period is reached, the Livy session terminates. A value of zero disables the timeout. If the value is not zero, it must be at least 15 minutes.

Default 60 minutes

Restriction The TIMEOUT= option is not supported for Databricks.

TRACE
specifies whether the SAS Embedded Process runs with traces.

STOPSESSION Statement

Stops the SAS Embedded Process continuous session.

Notes:

For Databricks, when you are finished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

Do not use the STARTSESSION and STOPSESSION statements with SingleStore or Teradata.

Syntax

STOPSESSION;

PUBLISHMODEL Statement

Publishes a model to a data source.

Syntax

PUBLISHMODEL
MODELTABLENAME="*fully-qualified-table-name*"
MODELTYPE=DATASTEP | DS2
PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
STOREFILES=*file-specification* | PROGRAMFILE=*file-specification*
VARSFILE=*file-path*
<*publishmodel-optional-argument*>;

Required Arguments

MODELTABLENAME="*fully-qualified-table-name*"
specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see "Specifying One-Part, Two-Part, or Three-Part Table Names" on page 120.

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

Examples The following example references a catalog, a schema, and a table:

```
modeltablename="mycatalog.myschema.mytable"
```

The following example references a schema and a table:

```
modeltablename="myschema.mytable"
```

The following example requires backticks around the table name **`my.table`** because the name contains a period:

```
modeltablename="myschema.`my.table`"
```

MODELTYPE=DATASTEP | DS2
specifies the type of the input model program.

DATASTEP
specifies that the input model program is DATA step code.

DS2
specifies that the input model program is DS2 code.

Default DS2

Note The MODELTYPE= option is required only for DATA step models. The DATA step code is converted to DS2 code before being bundled into an item store.

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

STOREFILES=("*file-path-1*" | *fileref-1* <,"*file-path-2*" | *fileref-2*, ...>)
specifies a list of analytic store .sasast files, one for each analytic store to be published. The list of files can be a mixture of file path names and file references.

You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Interactions You must specify either STOREFILES= or PROGRAMFILE= or both options.

The STOREFILES= option is used only when MODELTYPE=DS2. The option is not used when MODELTYPE=DATASTEP.

PROGRAMFILE="*file-path*" | *fileref*
You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

Interaction You must specify either STOREFILES= or PROGRAMFILE= or both options.

VARSFILE="*file-path*" | *fileref*
specifies the file that contains the variable metadata XML to be used during translation of an input DATA step model program to DS2.

Interaction The VARSFILE= option is required by some models when MODELTYPE=DATASTEP. The option is not used when MODELTYPE=DS2.

Optional Argument

FORMATSFILE="*file-path*" | *fileref*
specifies the file that contains the user-defined format XML definition to be published.

Note This option is valid for DS2 models or DATA step models.

RUNMODEL Statement

Runs a model in a data source.

Syntax

RUNMODEL
INPUTQUERY='*sql-query*' | INPUTSCALACODE="*scala-code*" |
INPUTTABLENAME="*fully-qualified-table-name*"
MODELTABLENAME="*fully-qualified-table-name*"
OUTKEY="*column-specification*" | OUTPUTSCALACODE="*scala-code*" |
OUTPUTTABLENAME="*fully-qualified-table-name*"
PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
<*runmodel-optional-arguments*>;

Required Arguments

**INPUTQUERY='*sql-query*' | INPUTSCALACODE="*scala-code*" |
INPUTTABLENAME="*fully-qualified-table-name*"**

INPUTQUERY='*sql-query*'
is a SELECT statement that returns a table results set.

The SQL query must be supported by the data source.

Restriction The INPUTQUERY= option is not supported for Databricks.

Note This option was added in 2025.07.

Example Here is an example for SingleStore:

```
inputquery='select a, b, c from mytable where a > 1'
```

Example “Example 5: Use an SQL Query to Read Data for Scoring” on page 148

INPUTSCALACODE="*scala-code*"
is a string of Scala code.

Use this option if you want to read your input data by using custom Scala code. Your Scala code must be a single statement that returns a Spark dataset.

In the INPUTSCALACODE= string, do not include the variable declaration.

Restriction The INPUTSCALACODE= option is not supported for SingleStore or Teradata.

Note This option was added in 2025.07.

Tip If you use double quotation marks (") inside the Scala code string, and the entire string is contained within double quotation marks ("), then you must escape the internal quotation marks. To escape quotation marks, use a pair of quotation marks together ("") instead of one (").

Examples Here is an example of Scala code that you would submit in a Databricks notebook to read a Parquet file:

```
val data = spark.read.parquet("/data/myparquet")
```

Here is the code to read the same file from Databricks in PROC ACCELERATOR. The double quotation marks (") are not escaped, because the string is enclosed within single quotation marks (').

```
inputscalacode='spark.read.parquet("/data/myparquet")'
```

Here is an example of Scala code used in a Databricks notebook to read a JSON file:

```
val df = spark.read.format("json").load("myfile.json")
```

Here is the code to read the same file from Databricks in PROC ACCELERATOR. Notice that the quotation marks within the string are escaped by using a pair of quotation marks.

```
inputscalacode="spark.read.format(""json"").load(""myfile.json"")"
```

Example "Example 6: Use Scala Code to Read and Write Data for Scoring" on page 150

INPUTTABLENAME="*fully-qualified-table-name*"
specifies the name of the input table.

You can specify a one-part, two-part, or three-part name. For more information, see "Specifying One-Part, Two-Part, or Three-Part Table Names" on page 120.

Restriction You must specify either the INPUTQUERY= option or the INPUTSCALACODE= option or the INPUTTABLENAME= option. You cannot specify more than one of the input options.

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

MODELTABLENAME="*fully-qualified-table-name*"
specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see "Specifying One-Part, Two-Part, or Three-Part Table Names" on page 120.

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

Examples The following example references a catalog, a schema, and a table:

```
modeltablename="mycatalog.myschema.mytable"
```

The following example references a schema and a table:

```
modeltablename="myschema.mytable"
```

The following example requires backticks around the table name **my.table** because the name contains a period:

```
modeltablename="myschema.`my.table`"
```

OUTKEY="*column-specification*" | OUTPUTSCALACODE="*scala-code*" | OUTPUTTABLENAME="*fully-qualified-table-name*"

OUTKEY="*column-1 <column-2 ...)>*"
OUTKEY=(*column-1 <column-2 ...)>*
OUTKEY=("*column-1*" <"*column-2*" ...)>

specifies the name of one or more columns used for the primary index of the output table that is created by the SAS Embedded Process.

Restriction The OUTKEY= argument is supported for Teradata only.

OUPUTSCALACODE="*scala-code*"

is a string of Scala code.

Use this option if you want to write your output data by using custom Scala code.

The string must begin with the function.

Restriction The OUPUTSCALACODE= option is not supported for SingleStore, or Teradata.

Note This option was added in 2025.07.

Tip If you use double quotation marks (") inside the Scala code string, and the entire string is contained within double quotation marks ("), then you must escape the internal quotation marks. To escape quotation marks, use a pair of quotation marks together ("") instead of one (").

Example Here is an example of Scala code that uses the WRITE function, which you would submit in a Databricks notebook:

```
outputdataset.write.parquet("/data/myparquet_output")
```

Here is code to use the WRITE function in PROC ACCELERATOR for Databricks:

```
outputscalacode='write.parquet("/data/myparquet_output")'
```

Example "Example 6: Use Scala Code to Read and Write Data for Scoring" on page 150

OUTPUTTABLENAME="*fully-qualified-table-name*"

specifies the name of the output table.

You can specify a one-part, two-part, or three-part name. For more information, see "Specifying One-Part, Two-Part, or Three-Part Table Names" on page 120.

Restriction You must specify one of these options: OUTKEY=, OUTPUTSCALACODE=, or OUTPUTTABLENAME=. You cannot specify more than one of these options.

Requirement Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

Optional Arguments

KEEPLISTCOLUMNS=("*column-1*"<..."*column-n*">)
specifies the name of one or more columns to keep in the output data. If a column is not specified, it is dropped from the output data. If you want to keep all of the columns in the output data, then do not use the KEEPLISTCOLUMNS= option.

Note This option was added in 2025.06.

MAXUNBOUNDEDLENGTH=*maximum-unbounded-column-length*
specifies the maximum length for unbounded data types (such as BLOB, CLOB, TEXT, or long STRING types) when they are Read into the SAS Embedded Process. The option ensures consistent handling of very large character or binary columns from the database. If data in an input column exceeds the MAXUNBOUNDEDLENGTH= value, truncation could occur during Read. Truncation could occur during Write, but only for an output column that is also an input column. The option does not apply to any output column that is not an input column.

You can set the MAXUNBOUNDEDLENGTH= option in either the STARTSESSION statement or RUNMODEL statement. When you specify the option in RUNMODEL, you override the value in STARTSESSION. Note that the DBMAX_TEXT= LIBNAME option does not affect processing in PROC ACCELERATOR.

Default 32767

Note The MAXUNBOUNDEDLENGTH= option was added in 2025.09.

OPTIONS="*options-string*"
specifies properties that are passed to the SAS Embedded Process. For full list of supported options, see options-list on page 112 in the SAS_SCORE_EP stored procedure.

Restriction The OPTIONS= argument is supported for Teradata only.

Note This option was added in 2026.08.

Example
```
options = "loglevel=info;encoding=unicode;"
```

Details

You can mix input and output objects:

- You must specify one and only one of the input options: INPUTQUERY= or INPUTSCALACODE= or INPUTTABLENAME=.
- You must specify one and only one of the output options: OUTPUTSCALACODE= or OUTPUTTABLENAME=.
- You can mix the input and output options. For example, you can read your input data by using INPUTSCALACODE= and write your output data by using OUTPUTTABLENAME=.

DELETEMODEL Statement

Deletes a model from a data source.

Syntax

DELETEMODEL
MODELTABLENAME=*fully-qualified-table-name*
PLATFORM=DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA;

Required Arguments

MODELTABLENAME="*fully-qualified-table-name*"
specifies the name of the table to which the model is published. You can specify a one-part, two-part, or three-part name. For more information, see "Specifying One-Part, Two-Part, or Three-Part Table Names" on page 120.

Requirement: Surround a name with backticks (`) if the name contains periods, reserved words, or special characters.

Examples: The following example references a catalog, a schema, and a table:

```
modeltablename="mycatalog.myschema.mytable"
```

The following example references a schema and a table:

```
modeltablename="myschema.mytable"
```

The following example requires backticks around the table name **`my.table`** because the name contains a period:

```
modeltablename="myschema.`my.table`"
```

PLATFORM=CLOUDERA | DATABRICKS | SINGLESTORE | SYNAPSE | TERADATA
specifies the target platform for model publishing and scoring. This option is required for Azure Synapse, because PROC ACCELERATOR cannot determine the platform from the LIBNAME assignment. In that case, if you have not specified the PLATFORM= option, an error is written to the SAS log.

Requirement The PLATFORM= option is required for Azure Synapse and is optional for other data sources.

Examples: ACCELERATOR Procedure

Example 1: Publish, Run, and Delete a Model in Azure Databricks

Features:
LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
STARTSESSION statement
RUNMODEL statement
STOPSESSION statement
DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a Spark table in Databricks on Microsoft Azure. Then PROC ACCELERATOR runs the model in Databricks by using the SAS Embedded Process for Spark. Note that the AZURETENANTID= system option is not necessary for in-database model scoring with PROC ACCELERATOR, because the procedure publishes models to a Spark table, not to the file system.

IMPORTANT SAS Embedded Process for Spark requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information, see "Deployment Details for Databricks and Synapse" on page 126.

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes

might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

Program

```
libname spklib spark platform=databricks
       user=token
       password="authentication-token"
       server="myserver.azuredatabricks.net"
       catalog="my-catalog-name"
       schema="my-schema-name"
       port=10016
       httppath="my-http-path";

proc accelerator library=spklib;
   publishModel
      modeltablename="my-model-table"
      programfile="my-program-file"
      storefiles="my-analytic-store-file";
run;
quit;


proc accelerator library=spklib;
   startsession
      platformendpoint="https://my-identifier.databricks.net";
run;
quit;

proc accelerator library=spklib;
   runmodel
      modeltablename="my-model-table"
      inputtablename="catalog.schema.input-table"
      outputtablename="catalog.schema.output-table";
run;
quit;

proc accelerator library=spklib;
   stopsession;
run;
quit;

proc accelerator library=spklib;
   deletemodel
      modeltablename="my-model-table"
      ;
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Spark. Specify PLATFORM=DATABRICKS, connection options, and any other LIBNAME options

that you need. For more information about assigning this SAS library, see "SAS/ACCESS Interface to Spark" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname spklib spark platform=databricks
       user=token
       password="authentication-token"
       server="myserver.azuredatabricks.net"
       catalog="my-catalog-name"
       schema="my-schema-name"
       port=10016
       httppath="my-http-path";
```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref.

```
proc accelerator library=spklib;
   publishModel
      modeltablename="my-model-table"
      programfile="my-program-file"
      storefiles="my-analytic-store-file";
run;
quit;
```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORMENDPOINT= value is the Databricks workspace URL.

```
proc accelerator library=spklib;
   startsession
      platformendpoint="https://my-identifier.databricks.net";
run;
quit;
```

Run the model. You must start a SAS Embedded Process continuous session before you run a model.

```
proc accelerator library=spklib;
   runmodel
      modeltablename="my-model-table"
      inputtablename="catalog.schema.input-table"
      outputtablename="catalog.schema.output-table";
run;
quit;
```

Stop the SAS Embedded Process continuous session. When you are finsished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```
proc accelerator library=spklib;
   stopsession;
run;
quit;
```

Delete the model.

```
proc accelerator library=spklib;
   deletemodel
```

```
      modeltablename="my-model-table"
      ;
run;
quit;
```

Example 2: Publish, Run, and Delete a Model in Azure Synapse

Features:

LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to an SQL Server table (also known as a dedicated SQL pool table) in Microsoft Azure Synapse Analytics. Then PROC ACCELERATOR runs the model by using the SAS Embedded Process for Spark. The connection uses SAS/ACCESS Interface to Microsoft SQL Server.

IMPORTANT For Synapse, starting with 2026.06, changes are required due to an update in the DataDirect Microsoft SQL Server ODBC driver, which is used for connections to Microsoft SQL Server. If you use the NOPROMPT= LIBNAME option to supply connection values, then you must add three properties in the connection string. If you use a DSN connection, then the properties can be set on the specific DSN in that odbc.ini, rather than in the LIBNAME statement. The properties are `KeepConnectionActive=0;ConnectionRetryCount=3;ConnectionRetryDelay=2;`. Without these properties, you receive an SSL error message.

In addition, starting with 2026.04, a new pre-installation task is required in order to download an additional object library and create a wheel file. Then you install that wheel file when you install the sasindb wheel. For more information, see *SAS In-Database Products: Deployment and Administration Guide*.

Program

```
libname mylib sqlsvr
       user=my-user-name
```

```
         password="my-password"
         schema="my-schema"
         noprompt="DRIVER=SAS ACCESS to MS SQL Server;
                  KeepConnectionActive=0;
                  ConnectionRetryCount=3;
                  ConnectionRetryDelay=2;
                  Hostname=synapse-workspace-name.sql.azuresynapse.net;
                  Port=my-port-number;
                  Database=my-database-name
                  ...;";
proc accelerator library=mylib;
   publishModel
      platform=synapse
      modeltablename="my-model-table"
      programfile="my-program-file"
      storefiles="my-analytic-store-file";
run;
quit;

proc accelerator library=mylib;
   startsession
      platform=synapse
      platformendpoint=""
      platformuser="endpoint/livyApi/versions/livyApiVersion/
sparkPools/sparkPoolName"
      platformpassword="my-secret";
run;
quit;

proc accelerator library=mylib;
   runmodel
      platform=synapse
      modeltablename="my-model-table"
      inputtablename="my-input-table"
      outputtablename="my-output-table";
run;
quit;

proc accelerator library=mylib;
   stopsession;
run;
quit;

proc accelerator library=mylib;
   deletemodel
      platform=synapse
      modeltablename="my-model-table"
      ;
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Microsoft SQL Server. You must connect to a dedicated SQL pool in the Synapse workspace. The pool must be

unpaused. Specify the schema in the SCHEMA= LIBNAME option. You must specify the Database= connection option within the NOPROMPT= option. The database value is the name of the dedicated SQL pool. Do not use the DATASRC= option. For additional options, see "SAS/ACCESS Interface to Microsoft SQL Server" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname mylib sqlsvr
       user=my-user-name
       password="my-password"
       schema="my-schema"
       noprompt="DRIVER=SAS ACCESS to MS SQL Server;
                KeepConnectionActive=0;
                ConnectionRetryCount=3;
                ConnectionRetryDelay=2;
                Hostname=synapse-workspace-name.sql.azuresynapse.net;
                Port=my-port-number;
                Database=my-database-name
                ...;";
```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref. For Synapse, the PLATFORM= option is required in every PROC ACCELERATOR statement except STOPSESSION. You must specify either PROGRAMFILE= or STOREFILES= or both options, depending on your model. See the syntax in "PUBLISHMODEL Statement" on page 130 for more information.

```
proc accelerator library=mylib;
   publishModel
      platform=synapse
      modeltablename="my-model-table"
      programfile="my-program-file"
      storefiles="my-analytic-store-file";
run;
quit;
```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORM=, PLATFORMENDPOINT=, PLATFORMUSER=, and PLATFORMPASSWORD= options are required for Synapse. For more information about the endpoint, see "Forming the Livy REST URL for Synapse" on page 74.

```
proc accelerator library=mylib;
   startsession
      platform=synapse
      platformendpoint=""
      platformuser="endpoint/livyApi/versions/livyApiVersion/
sparkPools/sparkPoolName"
      platformpassword="my-secret";
run;
quit;
```

Run the model. You must start a SAS Embedded Process continuous session before you run a model.

```
proc accelerator library=mylib;
   runmodel
      platform=synapse
      modeltablename="my-model-table"
      inputtablename="my-input-table"
```

```
      outputtablename="my-output-table";
run;
quit;
```

Stop the SAS Embedded Process continuous session. When you are finsished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```
proc accelerator library=mylib;
   stopsession;
run;
quit;
```

Delete the model.

```
proc accelerator library=mylib;
   deletemodel
      platform=synapse
      modeltablename="my-model-table"
      ;
run;
quit;
```

Example 3: Publish, Run, and Delete a Model in SingleStore

Features:

LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a SingleStore database. Then PROC ACCELERATOR runs the model in SingleStore by using the SAS Embedded Process. The SingleStore instance that is included in the product SAS SpeedyStore is supported. An external SingleStore database is not supported. For more information, see “Deployment Details for SingleStore” on page 127.

Note: For SingleStore, you do not start the SAS Embedded Process continuous session before running a model.

Program

```
libname mylib sstore
       user=my-user-name
       password="my-password"
       database=my-database
       server=my-server
       port=my-port-number;

proc accelerator library=mylib;
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store_file";
run;
quit;

proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"
      inputtablename="my_input_table"
      outputtablename="output_table";
run;
quit;

proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to SingleStore. For more information about this LIBNAME statement, see "SAS/ACCESS Interface to SingleStore" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname mylib sstore
       user=my-user-name
       password="my-password"
       database=my-database
       server=my-server
       port=my-port-number;
```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```
proc accelerator library=mylib;
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store_file";
run;
quit;
```

Run the model.

```
proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"
      inputtablename="my_input_table"
      outputtablename="output_table";
run;
quit;
```

Delete the model.

```
proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Example 4: Publish, Run, and Delete a Model in Teradata

Features:

LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a Teradata database. Then PROC ACCELERATOR runs the model in Teradata by using the SAS Embedded Process.

Note: For Teradata, you do not start the SAS Embedded Process continuous session before running a model.

Program

```
libname mylib teradata
       user="my-user-name"
       password="my-password"
       database="my-database"
       server="my-server";

proc accelerator library=mylib;
```

```
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store_file";
run;
quit;

proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"
      inputtablename="my_input_table"
      outputtablename="output_table"
      options="loglevel=info;encoding=unicode;";
run;
quit;

proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to Teradata. For more information about this LIBNAME statement, see "SAS/ACCESS Interface to Teradata" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname mylib teradata
       user="my-user-name"
       password="my-password"
       database="my-database"
       server="my-server";
```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```
proc accelerator library=mylib;
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store_file";
run;
quit;
```

Run the model.

```
proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"
      inputtablename="my_input_table"
      outputtablename="output_table"
      options="loglevel=info;encoding=unicode;";
run;
quit;
```

Delete the model.

```
proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Example 5: Use an SQL Query to Read Data for Scoring

Features: LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC ACCELERATOR publishes a model to a table in a SingleStore database. Then PROC ACCELERATOR runs the model in SingleStore by using the SAS Embedded Process. The INPUTQUERY= option specifies a SELECT statement that uses an SQL query to read the data.

The SingleStore instance that is included in the product SAS SpeedyStore is supported. An external SingleStore database is not supported. For more information, see “Deployment Details for SingleStore” on page 127.

Note: For SingleStore, you do not start the SAS Embedded Process continuous session before running a model.

Program

```
libname mylib sstore
       user=my-user-name
       password="my-password"
       database=my-database
       server=my-server
       port=my-port-number;

proc accelerator library=mylib;
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store-file";
```

```
run;
quit;

proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"                          1
      inputquery="select sales, salary from mydb.mytable"   2
      outputtablename="output_database.output_table";       3
run;
quit;

proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Program Description

Assign a SAS library, using SAS/ACCESS Interface to SingleStore. For more information about this LIBNAME statement, see "SAS/ACCESS Interface to SingleStore" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname mylib sstore
       user=my-user-name
       password="my-password"
       database=my-database
       server=my-server
       port=my-port-number;
```

Publish the model. The LIBRARY= option in the PROC ACCELERATOR statement references the libref.

```
proc accelerator library=mylib;
   publishModel
      modeltablename="model_table"
      programfile="my_program_file"
      storefiles="my_analytic_store-file";
run;
quit;
```

Run the model.

```
proc accelerator library=mylib;
   runmodel
      modeltablename="model_table"                          1
      inputquery="select sales, salary from mydb.mytable"   2
      outputtablename="output_database.output_table";       3
run;
quit;
```

1 The MODELTABLENAME= option specifies a one-level name, so it references the Mydb database specified in the Mylib LIBNAME statement.

2 The INPUTQUERY= option specifies a SELECT statement. The SELECT query must be supported by SingleStore. You must specify the database in the

SELECT statement, even if it is the same database as in the LIBNAME statement.

3 The OUTPUTTABLENAME= option specifies a two-level name, so it stores the table in a different database than the Mylib database from the LIBNAME statement.

Delete the model.

```
proc accelerator library=mylib;
   deletemodel
      modeltablename="model_table";
run;
quit;
```

Example 6: Use Scala Code to Read and Write Data for Scoring

Features:

LIBNAME statement
PROC ACCELERATOR statement
PUBLISHMODEL statement
STARTSESSION statement
RUNMODEL statement
STOPSESSION statement
DELETEMODEL statement

Details

In this Databricks example, the input data is read by using custom Scala code, and the output data is written by using custom Scala code.

IMPORTANT SAS Embedded Process for Spark requires a different deployment for PROC ACCELERATOR than for PROC SCOREACCEL. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information, see "Deployment Details for Databricks and Synapse" on page 126.

Assign a SAS library, using SAS/ACCESS Interface to Spark. Specify PLATFORM=DATABRICKS, connection options, and any other LIBNAME options that you need. For more information about assigning this SAS library, see "SAS/ACCESS Interface to Spark" in *SAS/ACCESS for Relational Databases: Reference*.

```
libname spklib spark platform=databricks
```

```
      user=token
      password="authentication-token"
      server="myserver.azuredatabricks.net"
      catalog="my-catalog-name"
      schema="my-schema-name"
      port=10016
      httppath="my-http-path";
```

Publish the model. Use the LIBRARY= option in the PROC ACCELERATOR statement to reference the libref.

```
proc accelerator library=spklib;
   publishModel
      modeltablename="my-model-table"
      programfile="my-program-file"
      storefiles="my-analytic-store-file";
run;
quit;
```

Start the SAS Embedded Process continuous session. The procedure starts a continuous session and associates the specified endpoint with the SAS library location. The PLATFORMENDPOINT= value is the Databricks workspace URL.

```
proc accelerator library=spklib;
   startsession
      platformendpoint="https://my-identifier.databricks.net";
run;
quit;
```

Run the model. The INPUTSCALACODE= option reads a Spark table. The OUPUTSCALACODE= option writes a Spark table. Your input Scala string must be a single statement that returns a Spark dataset. Your output Scala string must begin with the function. Here, that function is WRITE.

```
%let
mydatabase=september;
                  1
proc accelerator library=spklib;
   runmodel
      modeltablename="my-model-table"
      inputscalacode="spark.table(""&mydatabase..input-
table"").select(""col1"", ""col2"")"   2

outputscalacode="write.mode(""overwrite"").saveAsTable(""&mydatabase..o
utput-table"")";3
run;
```

1 The LET statement defines a macro variable `mydatabase`, because the user wants to run the model against a different database each month.

2 Because the INPUTSCALACODE= option references a macro variable, double quotation marks are required around the entire string. Within the string, quotation marks must be escaped around the value that contains the macro variable. To escape a quotation mark in the string, you must use a pair of quotation marks ("") instead of one quotation mark ("). Quotation marks are escaped around the column names also.

3 Quotation marks in the OUTPUTSCALACODE= string are escaped similarly to the input string.

Stop the SAS Embedded Process continuous session. When you are finsished running models, you can stop the SAS Embedded Process continuous session by submitting PROC ACCELERATOR with the STOPSESSION statement. If you do not stop the session, it terminates a few minutes after you end SAS.

```
proc accelerator library=spklib;
   stopsession;
run;
quit;
```

Delete the model.

```
proc accelerator library=spklib;
   deletemodel
      modeltablename="my-model-table";
run;
quit;
```

9

SCOREACCEL Procedure

Overview: SCOREACCEL Procedure

What Does the SCOREACCEL Procedure Do?

PROC SCOREACCEL is an interactive procedure that provides an interface to the CAS server and external data sources for model publishing and scoring.

- Models are supported in the form of DATA step code, DS2 code, or analytic stores.
- Model code can be published to a session-local or global CAS table and executed in CAS. Session-local means it is accessible only to the current CAS session that is executing PROC SCOREACCEL.
- Alternatively, models can be published to external data sources. See "Supported Platforms for In-Database Model Scoring" on page 155. After a model is published to an external data source, running the model invokes the SAS Embedded Process.
- Models can be removed from CAS or from an external data source by using the DELETEMODEL statement.
- PROC SCOREACCEL calls CAS actions to complete the tasks. You can use the Model Publishing and Scoring action set directly, instead of using PROC SCOREACCEL.
- For a list of the supported scoring models and other information, see Part 3, "In-Database Model Scoring," on page 33.
- For information about PROC ACCELERATOR, an alternative to PROC SCOREACCEL, see "Feature Comparison for PROC ACCELERATOR and PROC SCOREACCEL" on page 124. PROC ACCELERATOR is supported for Databricks and SingleStore.

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for

Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Concepts: SCOREACCEL Procedure

Supported Platforms for In-Database Model Scoring

The following table shows processing environments other than CAS where you can run model scoring.

Table 9.1 *Platforms for Running a Model*

RUNMODEL Processing	RUNMODEL EXTTYPE= Value CASLIB Values	PUBLISHMODEL Model Storage	PUBLISHMODEL and DELETEMODEL EXTTYPE= Value CASLIB Values	Documentation
AWS Databricks	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Spark table[1]	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
		Amazon S3 (deprecated)[2]	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
Azure Databricks	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Spark table[1]	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=databricks.	Overview Example
		ADLS Gen2 (deprecated)[2, 3]	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=adls.	Overview Example

RUNMODEL Processing	RUNMODEL EXTTYPE= Value CASLIB Values	PUBLISHMODEL Model Storage	PUBLISHMODEL and DELETEMODEL EXTTYPE= Value CASLIB Values	Documentation
Azure Synapse[4]	EXTTYPE=SPARK In the CASLIB, specify srcType=spark and platform=synapse.	SQL Server table	EXTTYPE=SYNAPSE In the CASLIB, specify srcType=jdbc and other required arguments.	Overview Example
		ADLS Gen2 (deprecated)[3]	EXTTYPE=FILESYSTEM In the CASLIB, specify srcType=adls.	Overview Example
Hadoop	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Hive table[5]	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Overview Example
		Hadoop (deprecated)	EXTTYPE=HADOOP In the CASLIB, specify srcType=hadoop.	Overview
SingleStore (SAS SpeedyStore)[6]	EXTTYPE=SINGLESTORE In the CASLIB, specify srcType=singlestore.	SingleStore	EXTTYPE=SINGLESTORE In the CASLIB, specify srcType=singlestore.	Overview Example
Teradata	EXTTYPE=TERADATA In the CASLIB, specify srcType=teradata.	Teradata	EXTTYPE=TERADATA In the CASLIB, specify srcType=teradata.	Overview Example

1 As of 2023.10 for Databricks, you can publish a model to a Spark table, and this method is recommended.

2 Publishing a model to Databricks by using the DBFS REST API is not recommended.

3 ADLS Gen1 and Azure blob storage are not supported.

4 As of 2026.04 for Synapse, PROC SCOREACCEL is deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

5 As of 2023.09 for Hadoop, you can publish a model to a Hive table, and this method is recommended. Cloudera and Amazon EMR are supported.

6 In-database model scoring is supported if you license and deploy the product SAS SpeedyStore. For more information, see "SAS SpeedyStore Data Connector" in *SAS Cloud Analytic Services: User's Guide*. In-database model scoring is not supported by the SingleStore standard data connector.

Arguments Listed by Data Source and Statement

EXTTYPE=CAS

- DELETEMODEL statement on page 161

CASLIB=, DATABASE=, DELETEGLOBAL=, EXTTYPE=, MODELNAME=, MODELTABLE=, PERSISTTABLE=, PROMOTETABLE=, REPLACETABLE=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=, MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PERSISTTABLE=, PROGRAMFILE=, PROMOTETABLE=, PUBLISHGLOBAL=, REPLACEMODEL=, REPLACETABLE=, SCHEMA=, STORETABLES=, VARXMLFILE=

- RUNMODEL statement on page 182

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=, MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PERSISTTABLE=, PROGRAMFILE=, PROMOTETABLE=, PUBLISHGLOBAL=, REPLACEMODEL=, REPLACETABLE=, SCHEMA=, STORETABLES=, VARXMLFILE=

EXTTYPE=DATABRICKS

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELDATABASE=, MODELNAME=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELDATABASE=, MODELNAME=, MODELTYPE=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

EXTTYPE=FILESYSTEM (deprecated)

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELDIR=, MODELNAME=, PASSWORD=, SCHEMA=, TIMEOUT=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELDIR=, MODELNAME=, MODELTYPE=, OUTDIR=, PASSWORD=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, TIMEOUT=, VARXMLFILE=

EXTTYPE=HADOOP

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELDATABASE=, MODELDIR=, MODELNAME=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELDATABASE=, MODELDIR=, MODELNAME=, MODELTYPE=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

- RUNMODEL statement on page 182

CASLIB=, DATABASE=, DBMAXTEXT=, EXTTYPE=, FORCEOVERWRITE=, INDATASET=, INTABLE=, KEEPLISTCOLUMNS=, KEEPLISTFILE=, MODELDATABASE=, MODELDIR=, MODELNAME=, OUTDATASET=, OUTSCHEMA=, OUTTABLE=, PROPERTIES=, SCHEMA=, TRACE

EXTTYPE=SINGLESTORE

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=, MODELTYPE=, MODELUUID=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

- RUNMODEL statement on page 182

  CASLIB=, DATABASE=, EPOPTIONS=, EXTTYPE=, INQUERY=, INTABLE=, MODELNAME=, OUTTABLE=, OUTTABLEOPTIONS=, SCHEMA=

EXTTYPE=SPARK

- RUNMODEL statement on page 182

  CASLIB=, DATABASE=, EXTTYPE=, FORCEOVERWRITE= , INDATASET=, INTABLE=, KEEPLISTCOLUMNS=, KEEPLISTFILE=, MODELDATABASE=, MODELDIR=, MODELNAME=, OUTDATASET=, OUTSCHEMA=, OUTTABLE=, PROPERTIES=, SCHEMA=, TRACE

EXTTYPE=SYNAPSE

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, MODELSCHEMA=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELSCHEMA=, MODELTYPE=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

EXTTYPE=TERADATA

- DELETEMODEL statement on page 161

  CASLIB=, DATABASE=, EXTTYPE=, MODELNAME=, MODELTABLE=, SCHEMA=

- PUBLISHMODEL statement on page 169

  CASLIB=, DATABASE=, EXTTYPE=, FORMATFILE=, FORMATITEMSTOREFILE=, KEEPLIST=, MODELNAME=, MODELNOTES=, MODELTABLE=, MODELTYPE=, MODELUUID=, OUTDIR=, PROGRAMFILE=, REPLACEMODEL=, SCHEMA=, STOREFILES=, STORETABLES=, VARXMLFILE=

- RUNMODEL statement on page 182

CASLIB=, DATABASE=, EPOPTIONS=, EXTTYPE=, INQUERY=, INTABLE=, MODELNAME=, MODELTABLE=, OUTTABLE=, OUTTABLEOPTIONS=, SCHEMA=,

Publishing Models to a Model Table on the CAS Server

Publishing a model to CAS involves adding or replacing a row in a model table. In order to publish a model to an existing model table, the model table must first be loaded on the CAS server. If the specified model table has been loaded on the CAS server, then the model table is updated. Otherwise, a new model table is created.

A model can be published directly to a global model table in CAS. A model is published to a global model table in CAS when the PUBLISHMODEL PUBLISHGLOBAL= parameter is set to TRUE.

Naming Requirements

Data source column and table names must follow the vendor's naming conventions.

Syntax: SCOREACCEL Procedure

PROC SCOREACCEL <*optional-argument*>;

PUBLISHMODEL *required-arguments* <*publish-model-optional-arguments*>;

RUNMODEL *required-arguments* <*run-model-optional-arguments*>;

DELETEMODEL *required-arguments* <*delete-model-optional-arguments*>;

Statement	Task	Example
PROC SCOREACCEL	Publish, execute, or delete a model in CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10
PUBLISHMODEL	Publish a model in CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10

Statement	Task	Example
RUNMODEL	Run a model in CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10
DELETEMODEL	Delete a model from CAS or an external data source	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10

PROC SCOREACCEL Statement

Publishes, executes, or deletes a model in CAS or an external data source.

Examples:

"Example 1: Publish, Run, and Delete a DATA Step Model in CAS" on page 195
"Example 2: Publish, Run, and Delete a Model in Teradata" on page 197
"Example 3: Publish a Model to a Hive Table and Run in Cloudera" on page 200
"Example 4: Publish a Model to a Spark Table and Run in Databricks on AWS" on page 203
"Example 5: Publish a Model to a Spark Table and Run in Azure Databricks" on page 205
"Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS" on page 208
"Example 7: Publish a Model to ADLS and Run in Azure Databricks" on page 210
"Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse" on page 212
"Example 9: Publish a Model to ADLS and Run in Azure Synapse" on page 214
"Example 10: Publish, Run, and Delete a Model in SingleStore" on page 218

Syntax

PROC SCOREACCEL <SESSREF=*session-reference* | SESSUUID="*session-uuid*">;

Arguments

SESSREF=*session-reference* | SESSUUID="*session-uuid*"
specifies the name of a CAS session or the universally unique identifier (UUID) of an existing CAS session to which you want to connect. If no option is specified, the automatic CAS session, CASAUTO, is used.

SESSREF=*session-reference*
specifies the name of a CAS session to which you want to connect.

SESSUUID="*session-uuid*"
specifies the universally unique identifier (UUID) of an existing CAS session. You must obtain the SESSUUID from the existing session before you can specify it in this option. The engine connects to the session that is identified in the UUID.

DELETEMODEL Statement

Deletes a model from CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see "Publishing and Running Models in Amazon EMR or Cloudera" on page 90.

Syntax

DELETEMODEL
CASLIB="*caslib*"
MODELNAME="*model-name*"
MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
<*delete-model-optional-arguments*>;

Required Arguments

CASLIB="*casref*"
specifies the name of the caslib that is associated with the data source where the model is published. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to: CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions: Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required

for publishing and deleting models with EXTTYPE= values of FILESYSTEM, SINGLESTORE, SPARK, and SYNAPSE.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="*model-name*"
specifies the name of the model to be deleted.

Alias MODEL=

Applies to CAS

Filesystem

Databricks

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **`sasmodel_`** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **`sasmodel_`** prefix. Do not use the MODELTABLE= option.

MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
specifies the name of the table that contains the model to be deleted when deleting a model from CAS or Teradata.

For CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Requirements This parameter is required when deleting a model from CAS or Teradata.

Deleting a model from CAS involves removing a row from a model table. In order to remove the model, the model table must first be loaded to the CAS server.

Optional Arguments

AUTHDOMAIN="*authentication-domain*"
specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

AUTHTOKEN="*authentication-token*"
specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CLASSPATH="*class_path*"
specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="*cluster-id*"
specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

DATABASE="*database-name*"
specifies the name of the database.

Default "" (a zero-length string)

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that names the Teradata database to be accessed.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

DELETEGLOBAL=TRUE | FALSE
specifies whether the model is deleted from a global model table in CAS.

TRUE
deletes the model from the global model table.

Alias YES

FALSE
deletes the model from the model table in the local CAS session, leaving the global table unaltered.

Alias NO

Default FALSE

Applies to CAS

EXTTYPE=CAS | DATABRICKS | FILESYSTEM | HADOOP | SINGLESTORE | SYNAPSE | TERADATA
specifies the target environment to which the model is to be deleted.

CAS
specifies to delete the model from a model table in CAS.

DATABRICKS
specifies to delete the model from Databricks.

FILESYSTEM
specifies to delete the model from a cloud-based file system. FILESYSTEM supports Microsoft Azure Data Lake Storage Gen2 (ADLS) and Amazon S3.

Note: The FILESYSTEM value is deprecated as of 2023.12. For Databricks and Synapse, you are no longer required to publish a model to a file system.

HADOOP
specifies to delete the model from the Hadoop server.

SINGLESTORE
specifies to delete the model from the SingleStore cluster. This value is supported as of 2023.05.

SYNAPSE
specifies to delete the model from Azure Synapse. This value is supported as of 2023.12 to delete a model that was published to an SQL Server table.

TERADATA
specifies to delete the model from a model table in the Teradata database.

Alias TARGET=

Default CAS

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

MODELDATABASE="*model-database*"
specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to Databricks

Hadoop

Requirement When you publish a model, if MODELDATABASE= is specified, the database must exist already.

Note This option is supported as of 2023.09.

MODELDIR="*model-directory*"
specifies the root folder where the model directory is stored.

For Hadoop, in releases prior to 2023.09, models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of 2023.09 you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to Filesystem

Hadoop

MODELSCHEMA="*model-schema*"
specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSCHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSCHEMA= option must be specified.

Note This option is supported as of 2023.12.

PASSWORD="*password*"
is the password for the user ID on ADLS, the Hadoop server, or the Teradata server. On ADLS this is value known as the secret.

Aliases PASS=

PASSWD=

PWD=

Applies to Filesystem

Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PERSISTTABLE=TRUE | FALSE
specifies whether the updated model table, that results from deleting a model, should be saved to the caslib data source associated with the table.

TRUE
saves the updated model table to the data source.

Alias YES

FALSE
does not save the updated model table to the data source.

Alias NO

Default FALSE

Applies to CAS

Requirement If the model table already exists in the caslib data source, you must also specify REPLACETABLE=TRUE to save the table.

See REPLACETABLE= option

PROMOTETABLE=TRUE | FALSE
specifies whether the updated model table, that results from deleting a model, should be promoted to global scope on the CAS server.

TRUE
promotes the updated model table to global scope.

Alias YES

FALSE
does not promote the updated model table to global scope.

Alias NO

Default FALSE

Applies to CAS

REPLACETABLE=TRUE | FALSE
specifies whether to allow an existing model table, that results from deleting a model, to be replaced when the updated model table is saved to the caslib data source.

TRUE
replaces the model table.

Alias YES

FALSE
does not replace the model table.

Alias NO

Default TRUE

Applies to CAS

Requirement You must also specify PERSISTTABLE=TRUE to save the model table to the caslib data source.

See PERSISTTABLE= option

SCHEMA="*schema-name*"
specifies the name of the database or schema.

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

SERVER="*server*"
specifies the name of the Teradata server.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

TIMEOUT="*n*"
specifies the number of seconds for the procedure to attempt an operation. If the operation does not complete by *n* seconds, the operation is terminated.

Applies to Filesystem

Restriction This option is supported for ADLS only.

USERNAME="*id*"
is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=

USERID=

UID=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

WEBHDFSURL="*webhdfs-url*"
specifies the URL used to access the Hadoop distributed file system through the REST API.

Applies to Hadoop

Note Use this argument when the platform is configured to access the distributed file system through the REST API.

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PUBLISHMODEL Statement

Publishes a model in CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see "Publishing and Running Models in Amazon EMR or Cloudera" on page 90.

Syntax

PUBLISHMODEL
CASLIB="*caslib*"
MODELNAME="*model-name*"
MODELTYPE=DATASTEP | DS2
MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
STOREFILES=*file-specification* | PROGRAMFILE=*file-specification*
VARXMLFILE=*file-path*
< *publish-model-optional-arguments*> ;

Required Arguments

CASLIB="*casref*"
specifies the name of the caslib that is associated with the data source where the model is published. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to: CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions: Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required for publishing and deleting models with EXTTYPE= values of FILESYSTEM, SINGLESTORE, SPARK, and SYNAPSE.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="*model-name*"
specifies the name of the model to be published.

Alias MODEL=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **`sasmodel_`** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **`sasmodel_`** prefix. Do not use the MODELTABLE= option.

MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
specifies the name of the table to which the model is published in CAS or Teradata.

For CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Interaction The caslib that is specified as the first part of a two-part model table name takes precedence over the caslib that is specified with the CASLIB= option.

Note Publishing a model to CAS involves adding or replacing a row in a model table. In order to publish a model to an existing model table, the model table must first be loaded onto the CAS server. If the specified model table has been loaded onto the CAS server, then the model table is updated. Otherwise, a new model table is created.

MODELTYPE=DATASTEP | DS2
specifies the type of the input model program.

DATASTEP
specifies that the input model program is DATA step code.

Alias DS

DS2
specifies that the input model program is DS2 code.

Default DS2

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note The MODELTYPE= option is required only for DATA step models. The DATA step code is converted to DS2 code before being bundled into an item store

STOREFILES=("*file-path-1*" | *fileref-1* <,"*file-path-2*" | *fileref-2*, ...>)
specifies a list of analytic store .sasast files, one for each analytic store to be published. The list of files can be a mixture of file path names and file references.

You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Alias STOREFILE=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions If a single analytic store model is specified without an accompanying DS2 program, the KEEPLIST option can also be specified.

You must specify either STOREFILES= or PROGRAMFILE= or both options.

The STOREFILES= option is used only when MODELTYPE=DS2. The option is not used when MODELTYPE=DATASTEP.

PROGRAMFILE="*file-path*" | *fileref*
specifies the file that contains the model program to be published. You must specify either STOREFILES= or PROGRAMFILE= or both options. Some analytic store models require only the STOREFILES= option. Some DATA step and DS2 models require only the PROGRAMFILE= option. Some models consist of DATA step code, DS2 code, and one or more ASTORE files. In that case, you specify both STOREFILES= and PROGRAMFILE= options.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction You must specify either STOREFILES= or PROGRAMFILE= or both options.

VARXMLFILE="*file-path*" | *fileref*
specifies the file that contains the variable metadata XML to be used during translation of an input DATA step model program to DS2.

Alias XMLFILE=

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction The VARXMLFILE= option is required by some models when MODELTYPE=DATASTEP. The option is not used when MODELTYPE=DS2.

Optional Arguments

AUTHDOMAIN="*authentication-domain*"
specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

AUTHTOKEN="*authentication-token*"
specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CLASSPATH="*class_path*"
specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="*cluster-id*"
specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

DATABASE="*database-name*"
specifies the name of the database.

Default "" (a zero-length string)

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that names the Teradata database to be accessed.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

EXTTYPE=CAS | DATABRICKS | FILESYSTEM | HADOOP | SINGLESTORE | SYNAPSE | TERADATA
specifies the target environment to which the model is published.

CAS
specifies to publish to a model table in CAS.

DATABRICKS
specifies to publish the model to Databricks. This value is supported as of 2023.10 to publish a model to a Spark table.

FILESYSTEM
specifies to publish the model to a cloud-based file system. FILESYSTEM supports Microsoft Azure Data Lake Storage Gen2 (ADLS) and Amazon S3.

Note: The FILESYSTEM value is deprecated as of 2023.12. For Databricks and Synapse, you are no longer required to publish a model to a file system.

HADOOP
specifies to publish the model to the Hadoop server.

SINGLESTORE
specifies to publish the model to the SingleStore cluster. This value is supported as of 2023.05.

SYNAPSE
specifies to publish the model to Azure Synapse. This value is supported as of 2023.12 to publish a model to an SQL Server table.

TERADATA
specifies to publish to a model table in the Teradata database.

Alias TARGET=

Default CAS

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

FORMATFILE="*file-path*" | *fileref*
specifies the file that contains the user-defined format XML definition to be published.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Note This option is valid for DS2 models or DATA step models.

FORMATITEMSTOREFILE="*file-path*" | *fileref*
specifies the file that contains the format item store to be published.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

KEEPLIST=TRUE | FALSE
specifies whether to include a KEEP statement in the DS2 model program that is automatically generated from an analytic store model.

TRUE
includes a KEEP statement.

Alias YES

FALSE
does not include a KEEP statement.

Alias NO

Default FALSE

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interaction KEEPLIST= can be specified when the STOREFILES= or STORETABLES= option contains a single analytic store model, and an accompanying DS2 model program is not specified.

MODELDATABASE="*model-database*"
specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to Databricks

Hadoop

Requirement When you publish a model, if MODELDATABASE= is specified, the database must exist already.

Note This option is supported as of 2023.09.

MODELDIR="*model-directory*"
specifies the root folder where the model directory is stored.

For Hadoop, in releases prior to 2023.09, models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of 2023.09 you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to Filesystem

Hadoop

MODELNOTES="*model-notes*"
specifies the model notes to be written to the model table.

Applies to CAS

SingleStore (SAS SpeedyStore)

Teradata

MODELSCHEMA="*model-schema*"
specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSCHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSCHEMA= option must be specified.

Note This option is supported as of 2023.12.

MODELUUID="*model-uuid*"
specifies that the Model UUID is written to the model table.

Applies to CAS

SingleStore (SAS SpeedyStore)

Teradata

OUTDIR="*work-directory*"
specifies the local output directory that contains the program file that was converted from DATA step to DS2.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

PASSWORD="*password*"
is the password for the user ID on ADLS, the Hadoop server, or the Teradata server. On ADLS this is value known as the secret.

Aliases PASS=

PASSWD=

PWD=

Applies to Filesystem

Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PERSISTTABLE=TRUE | FALSE

specifies whether the updated model table should be saved to the caslib data source associated with the table.

TRUE

saves the updated model table to the data source.

Alias YES

FALSE

does not save the updated model table to the data source.

Alias NO

Default FALSE

Applies to CAS

Requirement If the model table already exists in the caslib data source, you must also specify REPLACETABLE=TRUE to save the table.

See REPLACETABLE= option

PROMOTETABLE=TRUE | FALSE

specifies whether the updated model table should be promoted to global scope on the CAS server.

TRUE

promotes the updated model table to global scope.

Alias YES

FALSE

does not promote the updated model table to global scope.

Alias NO

Default FALSE

Applies to CAS

PUBLISHGLOBAL=TRUE | FALSE
specifies whether the model is published to a global model table in CAS.

TRUE

Alias YES

FALSE
publishes the model to the model table in the local CAS session, leaving the global model table unaltered.

Alias NO

Default FALSE

Applies to CAS

REPLACEMODEL=TRUE | FALSE
specifies whether to allow an existing model in the model table to be replaced by the model being published.

TRUE
replaces the model in the model table.

Alias YES

FALSE
does not replace the model in the model table.

Alias NO

Default TRUE

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

REPLACETABLE=TRUE | FALSE
specifies whether to allow an existing model table to be replaced when the updated model table is saved to the caslib data source.

TRUE
replaces the model table.

Alias YES

FALSE
does not replace the model table.

Alias NO

Default TRUE

Applies to CAS

Requirement You must also specify PERSISTTABLE=TRUE to save the model table to the caslib data source.

See PERSISTTABLE= option

SCHEMA="*schema-name*"
specifies the name of the database or schema.

Applies to Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Filesystem, Hadoop, SingleStore, Spark, or Synapse, the DATABASE= option is an alias for the SCHEMA= option.

SERVER="*server*"
specifies the name of the Teradata server.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

STORETABLES=("*store-table-1*" | "*caslib.store-table-1*" <,"*store-table-2*" | "*caslib.store-table-2*", ...>)
specifies one or more CAS blob table names that contain the analytic stores to be published. Each table must include a VARBINARY column named "_state_" that contains the analytic store blob. If you publish an analytic store without a model program, then a DS2 model program is generated from the analytic store.

To specify an associated model program, use the PROGRAMFILE= option. When an analytic store is published, the associated model program is usually DS2 code.

Applies to CAS

Databricks

Filesystem

Hadoop

SingleStore (SAS SpeedyStore)

Synapse

Teradata

Interactions If a single analytic store model is specified without an accompanying DS2 program, then the KEEPLIST option can also be specified.

If a single analytic store is specified using the STORETABLES= option, then the PROGRAMFILE= option is not required.

TIMEOUT="*n*"
specifies the number of seconds for the procedure to attempt an operation. If the operation does not complete by *n* seconds, the operation is terminated.

Applies to Filesystem

Restriction This option is supported for ADLS only.

USERNAME="*id*"
is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=

USERID=

UID=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

WEBHDFSURL="*webhdfs-url*"
specifies the URL used to access the Hadoop distributed file system through the REST API.

Applies to Hadoop

Note Use this argument when the platform is configured to access the distributed file system through the REST API.

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

RUNMODEL Statement

Runs a model in CAS or an external data source.

Requirement: For Hadoop, beginning in 2023.09, security updates have been introduced by a newer version of the Java Runtime Environment. For a summary of changes in deployment, configuration, and usage, see "Publishing and Running Models in Amazon EMR or Cloudera" on page 90.

Syntax

RUNMODEL
CASLIB="*caslib*"
MODELNAME="*model-name*"
MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
<*run-model-optional-arguments*>;

Required Arguments

CASLIB="*casref*"
specifies the name of the caslib that is associated with the data source where the model is executed. PROC SCOREACCEL uses the caslib to obtain options that are specific to the data source. For example, the caslib can specify user credentials and other connection options.

Applies to: CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

Requirement: For EXTTYPE=SPARK, use the following parameters in the Spark caslib: The platform= option specifies the external target. The schema= is either the Synapse SQL database name or Spark database name. The username= is the application ID, and the password= is the application ID's secret. You must specify all connection options in the caslib.

Interactions: Specify connection options in a caslib assignment. Specifying connection options in the procedure is deprecated. The CASLIB= option is recommended and will be required in the future for all EXTTYPE= data sources. Currently, the CASLIB= option is required for publishing and deleting models with EXTTYPE= values of SINGLESTORE and SPARK.

For CAS or Teradata, a caslib that is specified as the first part of a two-part model table name in MODELTABLE= takes precedence over the value in this option.

MODELNAME="*model-name*"
specifies the name of the model to run.

Alias MODEL=

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

Note For SingleStore, SAS publishes each model as a table. For Hadoop, Databricks, and Synapse, if you omit the MODELDIR= option, SAS publishes each model as a table (Hive table, Spark table, or SQL Server table, depending on the data source). In all of these cases, SAS uses the model name as the table name and adds a prefix of **`sasmodel_`** in order to differentiate from other tables in the database. When you reference the model to publish it, delete it, or run it, do not include the **`sasmodel_`** prefix. Do not use the MODELTABLE= option.

MODELTABLE="*model-table*" | "*caslib.model-table*" | "*schema.model-table*"
specifies the name of the table that contains the model to run. When running a model in CAS, the model table can be specified as a one-part or a two-part name. The first part of a two-part name is interpreted as the name of the caslib that contains the model table.

Applies to CAS

Teradata

Interaction The caslib that is specified as the first part of a two-part model table name takes precedence over the caslib that is specified with the CASLIB= option.

Optional Arguments

AUTHDOMAIN="*authentication-domain*"
specifies the name of the authentication domain that contains the credentials (user name and password) that are used to access Teradata.

Applies to Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

AUTHTOKEN="*authentication-token*"
specifies the authentication token that is used to establish a connection with a Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CLASSPATH="*class_path*"
specifies the class path used in the Hadoop call context. The class path can be a folder or individual JAR files. The Hadoop configuration folder must be included in the class path if you are not specifying the CONFIGPATH= option. If you are using Spark, Hadoop JAR files and Spark JAR files must be located in separate folders, and the Spark JAR folder must be specified as the last entry in the class path.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

CLUSTERID="*cluster-id*"
specifies the identifier of the Spark cluster.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CONFIGPATH="*configuration-path*"
specifies a single folder where all the Hadoop and Spark configuration files reside.

Applies to Hadoop

Spark

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

CUSTOMJAR="*file-path*"
specifies the local JAR file that contains the user-provided custom reader. The custom JAR file is automatically copied to the Hadoop cluster during job submission.

Applies to Hadoop

CAUTION **This option is no longer supported.** Beginning with 2023.09, modify your code to remove this option.

DATABASE="*database-name*"
specifies the name of the database.

Default "" (a zero-length string)

Applies to Hadoop

Spark

SingleStore (SAS SpeedyStore)

Teradata

Notes For Teradata, this option provides the connection option that names the Teradata database to be accessed.

For Hadoop, SingleStore, or Spark, the DATABASE= option is an alias for the SCHEMA= option.

DBMAXTEXT=*number-of-bytes*
specifies the maximum number of bytes to allocate for STRING data type columns. This option does not apply to CHAR or VARCHAR columns.

Default 32767

Range 1–32767

Applies to Hadoop

EPOPTIONS="*options-string*"
specifies properties that are passed to the SAS Embedded Process. For SingleStore, the properties `ds2.dbMaxText=`*`number`* and `logger.level=`*`value`* are valid. For information about these properties, see "epProperties= Data Connector Option" in *SAS Cloud Analytic Services: User's Guide*.

For Teradata, this option is supported in legacy programs only. The properties are passed to the SAS_SCORE_EP stored procedure in its OPTIONS parameter. For more information, see *SAS In-Database Products: User's Guide*.

Applies to SingleStore (SAS SpeedyStore)

Teradata

Example `epOptions="ds2.dbMaxText=2048,logger.level=allon"`

EXTTYPE=CAS | HADOOP | SINGLESTORE | SPARK | TERADATA
specifies the target environment in which the model is to be run.

CAS
specifies to run the model in CAS.

HADOOP
specifies to run the model using the SAS Embedded Process for Hadoop.

SINGLESTORE
specifies to run the model using the SAS Embedded Process for SingleStore. This value is supported as of 2023.05.

SPARK
specifies to run the model using the SAS Embedded Process for Spark. The CASLIB= option is required. In the caslib use the PLATFORM= option to

specify the external target. For more information, see "Interaction with SAS Data Connector to Spark" on page 97.

Note: The DATABRICKS and SYNAPSE values have been deprecated. Use EXTTYPE=SPARK instead.

TERADATA
specifies to run the model using the SAS Embedded Process for Teradata.

Alias TARGET=

Default CAS

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

FORCEOVERWRITE=TRUE | FALSE
specifies whether to force deletion of the output data directory before running the Hadoop MapReduce job.

TRUE
forces deletion of the output data directory.

Alias YES

FALSE
does not force deletion of the output data directory.

Alias NO

Default FALSE

Applies to Hadoop

Spark

Restriction For running models against SQL tables in Synapse, FORCEOVERWRITE=FALSE is not supported. The output table must not exist.

HIVEPORT=*integer*
specifies the Hive server port number.

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

INDATASET="*input-dataset*"
specifies the name of the Spark dataset passed as input to the SAS Embedded Process.

Applies to Hadoop

Spark

Requirements When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

When running a model in Databricks or Synapse, either INDATASET= or INTABLE= must be specified.

See INHDMD= option

INTABLE= option

INHDMD="*file-path*"
specifies the name of the input Spark dataset file on HDFS.

Applies to Hadoop

Requirement When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

See INDATASET= option

INTABLE= option

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

INQUERY="*sql-query*"
specifies an SQL SELECT statement that defines the inputs to the SAS Embedded Process. The SQL query must be supported by the data source.

Applies to SingleStore (SAS SpeedyStore)

Teradata

Requirement When running a model in Teradata or SingleStore, either INTABLE= or INQUERY= must be specified.

See INTABLE= option

INTABLE="*input-table*" | "*caslib.input-table* " | "*schema.input-table*"
specifies the name of the input table. The input table can be specified as a one-part or a two-part name:

- When running a model in CAS, the first part of a two-part name is interpreted as the name of the caslib that contains the input table.
- When running in SingleStore, you must use a one-part name. Use the DATABASE= argument to specify the database name.

- When running in Synapse against an SQL Server table, use a two-part name. Specify the schema name followed by a dot and then the table name.
- When running a model in Teradata, the first part of a two-part name is interpreted as the name of the database schema that contains the input table.

Alias INPUTTABLE=

Applies to CAS

Hadoop

Spark

Teradata

Requirements When running a model in Hadoop, either INDATASET=, INHDMD=, or INTABLE= must be specified.

When running a model in Databricks or Synapse, either INDATASET= or INTABLE= must be specified.

When running a model in Teradata or SingleStore, either INTABLE= or INQUERY= must be specified.

See INDATASET= option

INHDMD= option

INQUERY= option

JOBMANAGEMENTURL="*rest-url*"
specifies the URL used to submit execution requests over a REST interface to services such as Apache Livy or Databricks Notebooks.

Alias RESTURL=

Applies to Hadoop

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

KEEPLISTCOLUMNS="*column-1 <column-2 ... >*"
KEEPLISTCOLUMNS=(*column-1 <column-2 ... >*)
KEEPLISTCOLUMNS=("*column-1*" <"*column-2*" ... >)
specifies the name of the column (or columns) to be kept by the DS2 program.

Alias KEEPLISTCOLS=

Applies to Hadoop

Spark

Interaction The parameters KEEPLISTFILE and KEEPLISTCOLUMNS are mutually exclusive.

KEEPLISTFILE="*file-path*"
specifies the name of the file that contains a list of columns to be kept by the DS2 program.

Applies to Hadoop

Spark

Interaction The parameters KEEPLISTFILE and KEEPLISTCOLUMNS are mutually exclusive.

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

MODELDATABASE="*model-database*"
specifies the database where the model is stored. If this option is not specified, then the database that is specified in the SCHEMA= option of the caslib is used. This option is supported for storing a model in a Hive table or in a Spark table. If you specify the MODELDIR= option, then the MODELDATABASE= option is ignored.

Applies to Databricks

Hadoop

Requirement When you publish a model, if MODELDATABASE= is specified, the database must exist already.

Note This option is supported as of 2023.09.

MODELDIR="*model-directory*"
specifies the root folder where the model directory is stored.

For Databricks as of 2023.10 and Synapse as of 2023.12, you are no longer required to publish a model to a file system. You are not required to specify the MODELDIR= option if you are not running models that have been published to ADLS or Amazon S3.

For Databricks, when you specify MODELDIR=, the model directory includes the prefix **`dbfs:`** and references a storage object that has been mounted to Databricks File System (DBFS). You can publish a model to Amazon S3 or Microsoft Azure Data Lake Storage Gen2 (ADLS) and run that model in Databricks. Here is an example:

```
modeldir="dbfs:/mnt/mycontainer/mymodels"
```

For Hadoop, in releases prior to 2023.09, models were always published to HDFS. Now SAS uses WebHDFS to access models that are stored in HDFS. If you want to access models in HDFS, you must specify the webHdfsUrl= option in the Hadoop caslib. However, this method is not recommended. In a future release, publishing to HDFS will not be supported.

For Hadoop, as of 2023.09 you can publish a model to a Hive table, and this method is recommended. In order to publish, delete, or run a model that is stored in a Hive table, omit the MODELDIR= option from PROC SCOREACCEL. When you omit the MODELDIR= option, the SERVER= parameter in the Hadoop caslib must specify the Hive server.

Applies to Hadoop

Spark

MODELSCHEMA="*model-schema*"
specifies the schema for the stored model. This option is supported for storing a model in an SQL Server table. If you specify the MODELDIR= option, then the MODELSCHEMA= option is ignored.

Applies to Synapse

Requirement For Synapse, when deleting, publishing, or running a model that is stored in an SQL Server table, the MODELSCHEMA= option must be specified.

Note This option is supported as of 2023.12.

OUTDATASET="*output-dataset*"
specifies the name of the output Spark dataset to be created by the SAS Embedded Process.

Applies to Hadoop

Spark

Requirements When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.

When running a model in Databricks or Synapse, either OUTDATASET= or OUTTABLE= must be specified.

See OUTHDMD= option

OUTTABLE= option

OUTHDMD="*file-path*"
specifies the name of the output Hadoop metadata file that is created by the SAS Embedded Process.

Applies to Hadoop

Requirement When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.

See OUTDATASET= option

OUTTABLE= option

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

OUTKEY="*column-1 <column-2 ...)*>"
OUTKEY=(*column-1 <column-2 ...)*>
OUTKEY=("*column-1*" <"*column-2*" ...)>
specifies the name of one or more columns used for the primary index of the output table that is created by the SAS Embedded Process.

Applies to SingleStore (SAS SpeedyStore)

Teradata

OUTPUTFOLDER="*directory-path*"
specifies the name of the directory where the output files are stored.

Applies to Hadoop

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

OUTPUTFORMATCLASS="*class-name*"
specifies the name of the output format class in dot notation that is used to write the output records.

Applies to Hadoop

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

OUTRECORDFORMAT=BINARY | DELIMITED
specifies the format of the output record that is produced by the SAS Embedded Process for Hadoop.

BINARY
specifies that the output record is binary.

DELIMITED
specifies that the output record is delimited.

Default DELIMITED

Applies to Hadoop

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

OUTSCHEMA="*output-schema*"
specifies the Hive output data source schema name when running a model. This argument is appropriate for Databricks, Hadoop, or Synapse.

Applies to Hadoop

Spark

OUTTABLE="*output-table*" | "*caslib.output-table* " | "*schema.output-table*"
specifies the name of the output table. The output table can be specified as a one-part or a two-part name:

- When running a model in CAS, the first part of a two-part name is interpreted as the name of the caslib that contains the output table.
- When running in SingleStore, you must use a one-part name. Use the DATABASE= argument to specify the database name.
- When running in Synapse against an SQL Server table, use a two-part name. Specify the schema name followed by a dot and then the table name.
- When running a model in Teradata, the output table must be in the same database as the input table. The first part of a two-part name is interpreted as the name of the database schema that contains the output table.

Alias OUTPUTTABLE=

Applies to CAS

Hadoop

SingleStore (SAS SpeedyStore)

Spark

Teradata

Requirements When running a model in Hadoop, either OUTHDMD=, OUTDATASET=, or OUTTABLE= must be specified.

When running a model in Databricks or Synapse, either OUTDATASET= or OUTTABLE= must be specified.

When running a model in SingleStore or Teradata, OUTTABLE= must be specified.

See OUTDATASET= option

OUTHDMD= option

OUTTABLEOPTIONS="*options-string*"
provides user-specified options that are appended to the Hive CREATE TABLE statement.

Applies to Hadoop

Spark

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

PASSWORD="*password*"
is the password for the user ID on the Hadoop or Teradata server.

Aliases PASS=

PASSWD=

PWD=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

PLATFORM=MAPRED | SPARK
specifies the platform where the SAS Embedded Process for Hadoop is to be executed.

MAPRED
specifies to run the model in MapReduce.

SPARK
specifies to run the model in Spark.

Default SPARK

Applies to Hadoop

CAUTION **This option is deprecated.** The value SPARK is now used automatically. Modify your code to remove this option.

PROPERTIES="*name1=value*" <PROPERTIES="*name2=value*"> | PROPERTIES=("*name=value*"<, "*name=value*", ...>)
specifies a Hadoop configuration property as a name-value pair. For multiple properties, specify a single argument containing a comma-separated list of name-value pairs enclosed in parentheses. Any Hadoop configuration property can be assigned using this option. The PROPERTIES parameter can also be specified multiple times, once for each property.

Applies to Hadoop

Spark

Requirement If your output is to a Hive table and the Hive database folder is under an HDFS encrypted zone, you must set the SAS Embedded Process temporary folder to a location that is under the same encrypted zone. To do this, set the sas.ep.tempdir configuration property. Here is an example:

```
properties="sas.ep.tempdir=yourSASEPTemporaryFolder"
```

SCHEMA="*schema-name*"
specifies the name of the database or schema.

Applies to Hadoop

Spark

SingleStore (SAS SpeedyStore)

Teradata

Notes For Teradata, this option provides the connection option that the Teradata database uses to qualify the Teradata tables.

For Hadoop, SingleStore, or Spark, the DATABASE= option is an alias for the SCHEMA= option.

SENDPROGRAM=TRUE | FALSE
specifies whether the model program source code should be sent back to the client and displayed for the user.

TRUE
sends the source code back to the client.

Alias YES

FALSE
does not send the source code back to the client.

Alias NO

Default FALSE

Applies to CAS

SERVER="*server*"
specifies the name of the Teradata server or Hive server.

Applies to Hadoop

Spark

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

TRACE
runs the SAS Embedded Process for Hadoop with traces on.

Applies to Hadoop

Spark

USERNAME="*id*"
is an authorized user ID on the Hadoop or Teradata server.

Aliases USER=

USERID=

UID=

Applies to Hadoop

Teradata

CAUTION **Connection options in the procedure are deprecated.** Specify this option in the caslib assignment instead.

VERBOSE
specifies that the SAS Embedded Process provide additional logging information.

Applies to Hadoop

Spark

CAUTION **This option is deprecated.** In the future, it will not be supported. Modify your code to remove this option.

Examples: SCOREACCEL Procedure

Example 1: Publish, Run, and Delete a DATA Step Model in CAS

Features:
CAS LIBNAME statement
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this PROC SCOREACCEL example, a DATA step model is published and run in CAS. The model is then deleted.

PROC SCOREACCEL translates DATA step code into DS2 code on the SAS client.

Program

```
cas mysess1;
libname mycaslib cas casref=mysess1;
```

```
proc delete data=mycaslib.model01_score_data; run;
libname eminput "/mydir/scoring/model01";
data mycaslib.model01_score_data;
   set eminput.traindata;
   id=_n_; run;
quit;

proc scoreaccel sessref=mysess1;
   publishmodel
      modelname="model01"
      modeltype=datastep
      modeltable="modeltable1"
      programfile="/mydir/scoring/model01/score.sas"
      xmlfile="/mydir/scoring/model01/score.xml"
      outdir="/user1/score/work/"
      modelnotes="Simple model01 test model"
      promotetable=no
      persisttable=no
      ;
quit;

proc delete data=mycaslib.model01out_data run;
proc scoreaccel sessref=mysess1;
   runModel
      modelname="model01"
      modeltable="modeltable1"
      intable="model01_score_data"
      outtable="model01_out_data"
      ;
quit;

proc scoreaccel sessref=mysess1;
   deletemodel
      modelname="model01"
      modeltable="modeltable1"
      ;
quit;
```

Program Description

Assign a CAS LIBNAME. The CAS LIBNAME is passed to the DATA step in PROC DELETE.

```
cas mysess1;
libname mycaslib cas casref=mysess1;
```

Create an input scoring table in CAS. The DELETE procedure removes an existing input scoring table.

```
proc delete data=mycaslib.model01_score_data; run;
libname eminput "/mydir/scoring/model01";
data mycaslib.model01_score_data;
   set eminput.traindata;
   id=_n_; run;
quit;
```

Publish a model in CAS. The DATA step model is converted to DS2 in this step and is published in CAS.

```
proc scoreaccel sessref=mysess1;
   publishmodel
      modelname="model01"
      modeltype=datastep
      modeltable="modeltable1"
      programfile="/mydir/scoring/model01/score.sas"
      xmlfile="/mydir/scoring/model01/score.xml"
      outdir="/user1/score/work/"
      modelnotes="Simple model01 test model"
      promotetable=no
      persisttable=no
      ;
quit;
```

Run the model in CAS. The DELETE procedure removes an existing CAS table before running the model.

```
proc delete data=mycaslib.model01out_data run;
proc scoreaccel sessref=mysess1;
   runModel
      modelname="model01"
      modeltable="modeltable1"
      intable="model01_score_data"
      outtable="model01_out_data"
      ;
quit;
```

Delete the model from the model table in CAS.

```
proc scoreaccel sessref=mysess1;
   deletemodel
      modelname="model01"
      modeltable="modeltable1"
      ;
quit;
```

Example 2: Publish, Run, and Delete a Model in Teradata

Features:

CAS procedure
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

This PROC SCOREACCEL example publishes and runs a DS2 model in Teradata. PROC SCOREACCEL invokes the model publishing and scoring action set in CAS to delete a model, publish a model, or run a model.

Program

```
libname mytdlib teradata server=mytdserver user=myuser password=XXXXX
   database=model;

proc delete data=mytdlib.model01_score_data;
run;
libname eminput "/mydir/scoring/model01";
data mytdlib.model01_score_data;
   set eminput.traindata;
   id=_n_;
run;
quit;

proc cas;
   session mysess1;
   action addCaslib /
       caslib="tdlib1"
       datasource={
       srcType="teradata",
       server="mytdserver",
       username="myuser",
       password="XXXXX",
       database="mymodel"
   };
run;

proc scoreaccel sessref=mysess1;
   publishmodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
      programfile="/mydir/scoring/model01/score.ds2"
      modelnotes="Simple model01 test model"
      ;
run;
quit;

proc scoreaccel sessref=mysess1;
   runmodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
```

```
      intable="model01_score_data"
      outtable="model01_out_data"
      outkey="id"
      ;
run;
quit;

proc scoreaccel sessref=mysess1;
   deletemodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
      ;
run;
quit;
```

Program Description

Create the Teradata LIBNAME reference. The Teradata libref is needed for PROC DELETE and the DATA step program.

```
libname mytdlib teradata server=mytdserver user=myuser password=XXXXX
   database=model;
```

Create an input scoring table. The DELETE procedure removes an existing input scoring table if it exists.

```
proc delete data=mytdlib.model01_score_data;
run;
libname eminput "/mydir/scoring/model01";
data mytdlib.model01_score_data;
   set eminput.traindata;
   id=_n_;
run;
quit;
```

Define a Teradata caslib. The addCaslib action adds a CAS library to the current mysess1 session.

```
proc cas;
   session mysess1;
   action addCaslib /
       caslib="tdlib1"
       datasource={
       srcType="teradata",
       server="mytdserver",
       username="myuser",
       password="XXXXX",
       database="mymodel"
   };
run;
```

Publish a DS2 model to Teradata. The PROGRAMFILE statement contains the DS2 model.

```
proc scoreaccel sessref=mysess1;
```

```
   publishmodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
      programfile="/mydir/scoring/model01/score.ds2"
      modelnotes="Simple model01 test model"
      ;
run;
quit;
```

Run the DS2 model in Teradata. The input table and output table must be in the same Teradata database.

```
proc scoreaccel sessref=mysess1;
   runmodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
      intable="model01_score_data"
      outtable="model01_out_data"
      outkey="id"
      ;
run;
quit;
```

If necessary, delete the model from the model table in Teradata.

```
proc scoreaccel sessref=mysess1;
   deletemodel
      caslib="tdlib1"
      exttype=teradata
      modelname="model01"
      modeltable="modeltable1"
      ;
run;
quit;
```

Example 3: Publish a Model to a Hive Table and Run in Cloudera

Features:

CASLIB statement
PROC CAS
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, a DS2 model is published to a Hive table. Then the model is run by using a Spark Livy interface. This example code is appropriate for Cloudera or Amazon EMR.

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

Program

```
cas mysess1;
caslib myhadoop datasource=(
       srcType="hadoop",
       server="Hive-server",
       port="port-number",
       userName="user-name",
       password="password",
       schema="database-name",
       hadoopConfigDir="Hadoop_config_path_directory",
       jobManagementURL="http://server.company.com:8998"
       );

proc scoreaccel sessref=mysess1;
   publishmodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase"
      programfile="/score/simple/simple.ds2";
run;
quit;

proc cas;
   sparkEmbeddedProcess.startSparkEP caslib="myhadoop"
      executorInstances=2 executorCores=2
      executorMemory=10 driverMemory=4;
run;
quit;

proc scoreaccel sessref=mysess1;
   runmodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase"
      intable="mytable"
```

```
      outtable="myoutput"
      ;
run;
quit;

proc scoreaccel sessref=mysess1;
   deletemodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase";
run;
quit;

proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myhadoop";
run;
quit;
```

Program Description

Start a CAS session and assign a caslib to Hadoop. The hadoopConfigDir= option specifies the folder where the sparkep-defaults.config file is stored. The server= option identifies the Hive server. You must set the jobManagementURL= option for scoring, in order to dispatch the job by using the Apache Spark Livy interface. If Kerberos is enabled, omit the userName= and password= options. Note that this caslib does not demonstrate the options that are needed for parallel data transfer.

```
cas mysess1;
caslib myhadoop datasource=(
       srcType="hadoop",
       server="Hive-server",
       port="port-number",
       userName="user-name",
       password="password",
       schema="database-name",
       hadoopConfigDir="Hadoop_config_path_directory",
       jobManagementURL="http://server.company.com:8998"
       );
```

Publish the model. The following PUBLISHMODEL statement does not specify the MODELDIR= option. Therefore, the webHdfsUrl= option is not required in the caslib assignment above. Without the MODELDIR= option, the model is published to a Hive table. This method is recommended. The optional MODELDATABASE= option specifies an existing database where the model is stored.

```
proc scoreaccel sessref=mysess1;
   publishmodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase"
      programfile="/score/simple/simple.ds2";
run;
quit;
```

Start the SAS Embedded Process for Spark continuous session. Use the CAS procedure to submit the startSparkEP action. In the caslib= parameter, specify the Hadoop caslib.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP caslib="myhadoop"
      executorInstances=2 executorCores=2
      executorMemory=10 driverMemory=4;
run;
quit;
```

Run the model.

```
proc scoreaccel sessref=mysess1;
   runmodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase"
      intable="mytable"
      outtable="myoutput"
      ;
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess1;
   deletemodel
      exttype=hadoop
      caslib="myhadoop"
      modelname="simple01"
      modeldatabase="mydatabase";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myhadoop";
run;
quit;
```

Example 4: Publish a Model to a Spark Table and Run in Databricks on AWS

Features:

CASLIB statement
CAS procedure
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement

DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to a Spark table. Then PROC SCOREACCEL runs the model in Databricks on AWS by using the SAS Embedded Process for Spark. For details about the caslib, see "Spark Data Connector" in *SAS Cloud Analytic Services: User's Guide*.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

Start a CAS session. Assign a caslib to Spark. Consult your Databricks workspace to find the values for the clusterId=, httpPath=, jobManagementUrl=, and server= options.

```
cas mysess;
caslib myspark datasource=(
       srcType="spark",
       platform="databricks",
       schema="default"
       clusterId="cluster-id",
       userName="token",
       authToken="authentication-token",
       password="authentication-token",
       jobManagementUrl="https://nnnnn.cloud.databricks.com/",
       server="https://nnnnn.cloud.databricks.com/",
       httpPath="my-http-path"
       );
```

Publish a model to a Spark table. Do not specify the MODELDIR= option. The optional MODELDATABASE= option specifies an existing database where the model is stored.

```
proc scoreaccel sessref=mysess;
   publishmodel
      exttype=databricks
      caslib="myspark"
      modelname="mymodel"
      storefiles="/myfiles/mystore.ast"
      programfile="/myfiles/myprogram.sas"
```

```
      modeldatabase="mydatabase";
   run;
quit;
```

Start a Databricks interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP
   caslib="myspark";
run;
quit;
```

Run the model. The INTABLE= option specifies the input table that contains the data to run the model against.

```
proc scoreaccel sessref=mysess;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldatabase="mydatabase"
      intable="mytable"
      outtable="mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
   deletemodel
      exttype=databricks
      caslib="myspark"
      modelname="mymodel"
      modeldatabase="mydatabase";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 5: Publish a Model to a Spark Table and Run in Azure Databricks

Features: CASLIB statement
CAS procedure

PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL

Details

In this example, PROC SCOREACCEL publishes a model to a Spark table. Then PROC SCOREACCEL runs the model in Databricks on Azure by using the SAS Embedded Process for Spark. For details about the caslib, see "Spark Data Connector" in *SAS Cloud Analytic Services: User's Guide*.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Note: In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For more information, see "Updates Might Be Needed Due to Driver Change" on page 243.

Start a CAS session. Assign a caslib to Spark. Consult your Azure Databricks workspace to find the values for the clusterId=, httpPath=, jobManagementUrl=, and server= options.

```
cas mysess;
caslib myspark datasource=(
       srcType="spark",
       platform="databricks",
       server="myserver.azuredatabricks.net",
       userName="token",
       password="authentication-token",
       clusterId="cluster-id",
       jobManagementUrl="https://nnnnn.cloud.databricks.net/",
       httpPath="my-http-path",
       schema="my-schema"
       );
```

Publish a model to a Spark table. Do not specify the MODELDIR= option. The optional MODELDATABASE= option specifies an existing database where the model is stored.

```
proc scoreaccel sessref=mysess;
   publishmodel
      exttype=databricks
      caslib="myspark"
```

```
      modelname="mymodel"
      storefiles="/myfiles/mystore.ast"
      programfile="/myfiles/myprogram.sas"
      modeldatabase="mydatabase";
   run;
quit;
```

Start a Databricks interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP
   caslib="myspark";
run;
quit;
```

Run the model. The INTABLE= option specifies the input table that contains the data to run the model against.

```
proc scoreaccel sessref=mysess;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldatabase="mydatabase"
      intable="mytable"
      outtable="mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
   deletemodel
      exttype=databricks
      caslib="myspark"
      modelname="mymodel"
      modeldatabase="mydatabase";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 6: Publish a Model to Amazon S3 and Run in Databricks on AWS

Features:
CASLIB statement
CAS procedure
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to Amazon S3. Then PROC SCOREACCEL runs the model in Databricks on Amazon Web Services (AWS) by using the SAS Embedded Process for Spark. For details about Amazon S3, see "Amazon S3 Data Source" in *SAS Cloud Analytic Services: User's Guide*.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Start a CAS session. Assign a caslib to Amazon S3 for publishing.

```
cas casauto;
caslib mys3 datasource=(
       srctype="s3",
       bucket="bucket-name",
       accesskeyid="access-key-id",
       secretaccesskey="secret-access-key",
       region="region-value"
       );
```

Publish a model to Amazon S3.

```
proc scoreaccel sessref=casauto;
   publishmodel
      exttype=filesystem
      caslib="mys3"
      modelname="mymodel"
      storefiles="/myfiles/mystore.ast"
      programfile="/myfiles/myprogram.sas"
```

```
      modeldir="/scoring/models";
   run;
quit;
```

To run a model in Databricks on AWS, a Spark caslib is required.

```
caslib myspark datasource=(
       srctype="spark",
       platform="databricks",
       schema="default",
       clusterid="cluster-id",
       username="token",
       authtoken="authentication-token",
       password="authentication-token",
       jobmanagementurl="https://nnnnn.cloud.databricks.com/",
       hadoopjarpath="path-to-hadoop-jar-files"
       );
```

Start a Databricks interactive session. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP
   caslib="myspark";
run;
quit;
```

Run the model in Databricks on AWS. The INTABLE= option specifies the input table that contains the data to run the model against.

```
proc scoreaccel sessref=casauto;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldir="dbfs:/mnt/s3-bucket-mount-point/scoring/models"
      intable="mytable"
      outtable="mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=casauto;
   deletemodel
      exttype=filesystem
      caslib="mys3"
      modelname="mymodel"
      modeldir="/scoring/models";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
```

```
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 7: Publish a Model to ADLS and Run in Azure Databricks

Features:
- CASLIB statement
- CAS procedure
- PROC SCOREACCEL statement
- PUBLISHMODEL statement
- RUNMODEL statement
- DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes a model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Then PROC SCOREACCEL runs the model in Azure Databricks by using the SAS Embedded Process for Spark.

Note: Starting with 2026.06 for Databricks, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

Prepare to publish the model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Start a CAS session. You must include the azuretenantid= session option.

```
cas casauto sessopts=(azuretenantid="tenant-ID");
```

Assign a caslib to ADLS for publishing.

```
caslib myadls datasource=(
       srctype="adls"
       accountname="account-name",
       applicationid="application-id",
       filesystem="mystorage",
       dnssuffix="dfs.core.windows.net"
       );
```

Publish the model to ADLS by specifying EXTTYPE=FILESYSTEM in the PUBLISHMODEL statement. In the CASLIB= argument, specify the ADLS caslib.

For the PASSWORD= argument, use the password from RBAC, which is also known as the secret.

```
proc scoreaccel sessref=casauto;
   publishmodel
      exttype=filesystem
      caslib="myadls"
      password="RBAC-client-secret"
      modelname="mymodel"
      modeldir="/scoring/models"
      storefiles="/myfiles/mystore.ast"
      programfile="/myfiles/myprogram.sas";
run;
quit;
```

To run a model in Azure Databricks, a Spark caslib is required.

```
caslib myspark datasource=(
       srctype='spark',
       platform=databricks,
       schema="default",
       clusterid="cluster-id",
       username="token",
       authtoken="authentication-token",
       password="authentication-token",
       jobmanagementurl="https://nnnnn.cloud.databricks.com/",
       hadoopjarpath="path-to-hadoop-jar-files"
       );
```

Start a Databricks interactive session. If you do not use an interactive session, then the SAS Embedded Process for Spark executes in a Databricks notebook, which causes the Embedded Process to start and stop for each execution request. Most customers choose to use an interactive session for better performance.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP
   caslib="myspark";
run;
quit;
```

Run the model in Azure Databricks. The INTABLE= option specifies the input table that contains the data to run the model against.

```
proc scoreaccel sessref=casauto;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldir="dbfs:/mnt/blob-storage-mount-point/scoring/models"
      intable="mytable"
      outtable="mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=casauto;
   deletemodel
      exttype=filesystem
```

```
        caslib="myadls"
        password="RBAC-client-secret"
        modelname="mymodel"
        modeldir="/scoring/models";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 8: Publish a Model to an SQL Server Table and Run in Azure Synapse

Features:

CASLIB statement
CAS procedure
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL

Details

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

In this example, PROC SCOREACCEL publishes a model to an SQL Server table. Then PROC SCOREACCEL runs the model in Synapse by using the SAS Embedded Process for Spark.

As a requirement before you can publish a model to an SQL Server table, you must download the Microsoft JDBC Driver for SQL Server. For instructions, see "Configuration for In-Database Model Scoring" in *SAS In-Database Products: Deployment and Administration Guide*.

Start a CAS session. Assign a JDBC caslib in order to publish the model. For Synapse, you must identify both a schema and a database in the caslib in the following ways: Specify the schema value in the SCHEMA= option of the caslib. For the database value, the JDBC caslib does not support the DATABASE= option, so you must use the URI= option to specify a connection string for the Microsoft JDBC Driver for SQL Server. (The connection string includes the database value, but not the schema value.) As noted above, a prerequisite is to download the Microsoft JDBC Driver for SQL Server driver. For instructions, see "Configuration for In-Database Model Scoring" in *SAS In-Database Products: Deployment and Administration Guide*. Then, in the Synapse workspace, in the JDBC connection strings, copy the connection string that is provided for JDBC (SQL authentication). Specify that connection string in the URI= option of the caslib. In addition, in the connection string, you must add the property useBulkCopyForBatchInsert=true.

```
cas mysess;
caslib myjdbc datasource=(
       srcType="jdbc",
       userName="user-name",
       password="password",
       uri="JDBC-connection-options;
            useBulkCopyForBatchInsert=true;BatchSize=400",
       schema="SQL-server-schema-name"
       );
```

Publish a model to an SQL Server table. To store the model in an SQL Server table, do not specify the MODELDIR= option. For Synapse, when you delete, publish, or run a model that is stored in an SQL Server table, the MODELSCHEMA= option is required. Specify EXTTYPE=SYNAPSE and reference your JDBC caslib.

```
proc scoreaccel sessref=mysess;
   publishmodel
      exttype=synapse
      caslib="myjdbc"
      modelname="mymodel"
      modelschema="myschema"
      storefiles="/myfiles/mystore.ast"
      programfile="/myfiles/myprogram.sas"
      ;
run;
quit;
```

Assign a Spark caslib in order to start an interactive session and run the model. In the caslib, specify platform="synapse". For the jobManagementUrl= parameter, specify the Livy REST URL that is specific to the Spark pool that you want to connect to. For more information about the jobManagementUrl= value, see "Forming the Livy REST URL for Synapse" on page 103.

```
cas mysess;
caslib myspark datasource=(
       srcType="spark",
       platform="synapse",
       userName="application-id",
       password="secret",
       server="myserver.xxxx.net",
       database="my-database-name",
       jobManagementUrl="Livy-rest-url"
       );
```

Start an interactive session before you run the model. If you do not use an interactive session, then the SAS Embedded Process for Spark starts automatically, using the default resource settings that are configured for the cluster.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP
   caslib="myspark";
run;
quit;
```

Run the model. For the INTABLE= and OUTTABLE= values, use a two-part name. Specify the SQL server schema name followed by a dot and then the table name.

```
proc scoreaccel sessref=mysess;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modelschema="myschema"
      intable="myschema.mytable"
      outtable="myschema.mytable_out";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
   deletemodel
      exttype=synapse
      caslib="myjdbc"
      modelname="mymodel"
      modelschema="myschema";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 9: Publish a Model to ADLS and Run in Azure Synapse

Features:

CASLIB statement
CAS procedure
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement

DELETEMODEL statement

Details

Note: Starting with 2026.04 for Azure Synapse, PROC SCOREACCEL is deprecated. The Model Publishing and Scoring action set and the SAS Embedded Process for Spark action set are also deprecated. The change is a result of a Microsoft announcement to end support for Azure Synapse Runtime for Apache Spark 3.4. You can no longer deploy the sepcorespark RPM in Synapse. You are encouraged to use PROC ACCELERATOR instead. To use PROC ACCELERATOR, you must deploy the SAS Embedded Process from the accelserver RPM. See "Deployment Details for Databricks and Synapse" on page 126.

In this example, PROC SCOREACCEL publishes a model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Then PROC SCOREACCEL runs the model in Azure Synapse by using the SAS Embedded Process for Spark. The model runs against an SQL Server table.

For Synapse, PROC SCOREACCEL uses Azure Active Directory (Azure AD) application registration and role-based access control (RBAC). For essential information, such as how to specify the Synapse Spark pool Livy REST URL, and the requirements for SQL Server, see Part 3, "In-Database Model Scoring," on page 33.

Program

```
cas mysess sessopts=(azuretenantid="tenant-id");

caslib myadls datasource=(
       srctype="adls"
       accountname="myaccount",
       applicationid="application-id",
       filesystem="mystorage",
       dnssuffix="dfs.core.windows.net"
       );

proc scoreaccel sessref=mysess;
   publishmodel
      exttype=filesystem
      caslib="myadls"
      password="RBAC-client-secret"
      modelname="mymodel"
      modeldir="/scoring/models"
      programfile="/myfiles/myprogram.ds2";
run;
quit;

caslib myspark datasource=(
       srctype="spark",
```

```
       platform="synapse",
       username="application-id",
       password="RBAC-client-secret",
       schema="SQL-server-database-name",
       hadoopjarpath="path-to-hadoop-jar-files",
       jobmanagementurl="Livy-rest-url"
       );

proc cas;
   sparkEmbeddedProcess.startSparkEP caslib="myspark"
      executorInstances=2 executorCores=2
      executorMemory=10 driverMemory=4;
run;
quit;

proc scoreaccel sessref=mysess;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldir="/scoring/models"
      intable="SQL-server-database-schema-name.in-table-name"
      outtable="SQL-server-database-schema-name.out-table-name";
run;
quit;

proc scoreaccel sessref=mysess;
   deletemodel
      exttype=filesystem
      caslib="myadls"
      password="RBAC-client-secret"
      modelname="mymodel"
      modeldir="/scoring/models";
run;
quit;

proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Program Description

Prepare to publish the model to Microsoft Azure Data Lake Storage Gen2 (ADLS). Start a CAS session. You must include the azuretenantid= session option.

```
cas mysess sessopts=(azuretenantid="tenant-id");
```

Assign a caslib to ADLS.

```
caslib myadls datasource=(
       srctype="adls"
       accountname="myaccount",
       applicationid="application-id",
       filesystem="mystorage",
       dnssuffix="dfs.core.windows.net"
       );
```

Publish the model to ADLS by specifying EXTTYPE=FILESYSTEM in the PUBLISHMODEL statement. In the CASLIB= argument, specify the ADLS caslib. For the PASSWORD= argument, use the password from RBAC, which is also known as the secret.

```
proc scoreaccel sessref=mysess;
   publishmodel
      exttype=filesystem
      caslib="myadls"
      password="RBAC-client-secret"
      modelname="mymodel"
      modeldir="/scoring/models"
      programfile="/myfiles/myprogram.ds2";
run;
quit;
```

To run a model in Synapse, a Spark caslib is required. The username= is the application ID. For the password= parameter, use the password from RBAC, which is also known as the secret. For SQL Server tables, the schema name is the database name. For the jobmanagementurl= parameter, specify the Synapse Spark pool Livy REST URL. For more information about the jobManagementUrl= value, see "Forming the Livy REST URL for Synapse" on page 103. The Hadoop JAR path is required. The Hadoop configuration path is not required.

```
caslib myspark datasource=(
       srctype="spark",
       platform="synapse",
       username="application-id",
       password="RBAC-client-secret",
       schema="SQL-server-database-name",
       hadoopjarpath="path-to-hadoop-jar-files",
       jobmanagementurl="Livy-rest-url"
       );
```

Start the SAS Embedded Process for Spark continuous session. Use the CAS procedure to submit the startSparkEP action. In the caslib= parameter, specify the Spark caslib.

```
proc cas;
   sparkEmbeddedProcess.startSparkEP caslib="myspark"
      executorInstances=2 executorCores=2
      executorMemory=10 driverMemory=4;
run;
quit;
```

Run the model in Synapse by specifying EXTTYPE=SPARK in the RUNMODEL statement. In the CASLIB= argument, specify the Spark caslib.

```
proc scoreaccel sessref=mysess;
   runmodel
      exttype=spark
      caslib="myspark"
      modelname="mymodel"
      modeldir="/scoring/models"
      intable="SQL-server-database-schema-name.in-table-name"
      outtable="SQL-server-database-schema-name.out-table-name";
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel sessref=mysess;
   deletemodel
      exttype=filesystem
      caslib="myadls"
      password="RBAC-client-secret"
      modelname="mymodel"
      modeldir="/scoring/models";
run;
quit;
```

To stop the SAS Embedded Process for Spark continuous session, submit the stopSparkEP action. The session is also terminated when the CAS session is terminated.

```
proc cas;
   sparkEmbeddedProcess.stopSparkEP caslib="myspark";
run;
quit;
```

Example 10: Publish, Run, and Delete a Model in SingleStore

Features:

CASLIB statement
PROC SCOREACCEL statement
PUBLISHMODEL statement
RUNMODEL statement
DELETEMODEL statement

Details

In this example, PROC SCOREACCEL publishes and runs a model in SingleStore. In-database model scoring is supported only if you license and deploy the product SAS SpeedyStore. For more information, see "SAS SpeedyStore Data Connector" in *SAS Cloud Analytic Services: User's Guide*. In-database model scoring is not supported by the SingleStore standard data connector.

For information about deployment, see *SAS SpeedyStore: Administration and Configuration Guide*.

Program

```
cas casauto;
```

```
caslib mycaslib datasource=(
     srcType="singlestore"
     database="my_database"
     host="svc-sas-singlestore-cluster-dml"
     port="3306"
     password="my_password"
     userName="my_username"
     );

proc scoreaccel;
   publishmodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
      programfile="/myfiles/myprogram.ds2"
     ;
run;
quit;

proc scoreaccel;
   runmodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
      intable="in-table-name"
      outtable="out-table-name";
     ;
run;
quit;

proc scoreaccel;
   deletemodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
     ;
run;
quit;
```

Program Description

Prepare to publish the model to SingleStore. Start a CAS session if one is not already started.

```
cas casauto;
```

Assign a caslib to SingleStore. If single sign-on is enabled, do not include the password= and userName= options or the authenticationDomain= option.

```
caslib mycaslib datasource=(
     srcType="singlestore"
     database="my_database"
     host="svc-sas-singlestore-cluster-dml"
     port="3306"
     password="my_password"
     userName="my_username"
```

```
      );
```

Publish the model to SingleStore by specifying EXTTYPE=SINGLESTORE in the PUBLISHMODEL statement. In the CASLIB= argument, specify the SingleStore caslib.

```
proc scoreaccel;
   publishmodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
      programfile="/myfiles/myprogram.ds2"
     ;
run;
quit;
```

Run the model by specifying EXTTYPE=SINGLESTORE in the PUBLISHMODEL statement. In the CASLIB= argument, specify the SingleStore caslib.

```
proc scoreaccel;
   runmodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
      intable="in-table-name"
      outtable="out-table-name";
     ;
run;
quit;
```

Delete the model if needed.

```
proc scoreaccel;
   deletemodel
      exttype=singlestore
      caslib="mycaslib"
      modelname="mymodel"
     ;
run;
quit;
```

10

Python and Scala API for the Sepcorespark RPM

Introduction to Scoring with the API for the Sepcorespark RPM

This document is for the Python and Scala API when the sepcorespark RPM is deployed. The API for the accelserver RPM is documented separately. See "Python and Scala API for the Accelserver RPM" on page 233.

With this API, you can run a scoring model from a Databricks or Synapse notebook without invoking SAS.

The API provides a way for users to easily integrate SAS model execution with their Spark processing.

Here are the requirements:

- You must install SAS Embedded Process. See *SAS In-Database Products: Deployment and Administration Guide*.

  Note: For Databricks or Synapse, this API is supported if you deploy the sepcorespark RPM file. This API is not supported if you have deployed the accelserver RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

- Scala or Python code is supported by the API.
- The model must be published already from SAS to the data source.
- The model must be supported for in-database scoring. See Chapter 5, "Models Supported for Scoring," on page 39.
- The input data is a Spark Dataset of rows (also known as a DataFrame) that can be loaded from any source that is supported by Spark. For example, a Spark table or a Parquet file can be used.
- You must import the package that contains the implementation of the Model class. See "Model Class and Methods" on page 222.
- To run a model inside the SAS Embedded Process for Spark, you must create a Model object. Associate the Model object with an input Spark Dataset and the model. You can run the model with Scala or Python by calling the run method in the Model class object.

Model Class and Methods

The Model class enables execution of SAS models inside the SAS Embedded Process for Spark. The create() method is a static method and it should be called directly from the Model class. Instantiate the Model class by calling into the Model.create() method.

You must import the Model class before using it:

- Scala: `import com.sas.spark.scoring.Model`
- Python: `from sasep.model import Model`

See also "Example 2: Scala Example" on page 232 and "Example 1: Python Example" on page 231.

Table 10.1 Tasks and Methods

Task	Method
Create a model	create()
Set additional model properties before execution (Setting additional properties after the model is executed has no effect.)	setDBMaxText() setOutputStorageLevelCacheDir() setOutputCacheDir() setTraceON() setTraceOFF()
Run a model	run()
Destroy a model	destroy()

Dictionary

create Method

Creates a Model object that is capable of running the associated model inside the SAS Embedded Process for Spark, using a Spark Dataset as input.

Syntax

Form 1: create(*inputDataset*, "*modelDatabaseName*", "*modelName*")

Form 2: create(*inputDataset*, "*modelFileName*")

Form 3: create(*inputDataset*, *modelDataset*)

Required Arguments

inputDataset

specifies the Spark Dataset that is used as input to the model.

- Python type: pyspark.sql.DataFrame
- Scala type: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>

modelDatabaseName
specifies the database where the model is stored, for a model that is published to a Spark table. Type: String.

modelFileName
specifies the model location in a file system. Models must be published already in a location that is accessible from the Spark driver and executor containers. Type: String.

modelDataset
specifies the Spark Dataset that contains the published model.

modelName
specifies the name of a model that is published to a Spark table. A model that has been published to a Spark table has a prefix of `sasmodel_` in its name, in order to differentiate from other tables in the database. When you reference the model name, do not include the `sasmodel_` prefix. Type: String.

Details

- Use Form 1 or Form 3 for a model that is published to a Spark or Hive table.
- Use Form 2 for a model that is published to a file system, such as ADLS or Azure S3.
- If you use Form 3, you must first load the Spark or Hive table to a DataFrame (for Python) or Dataset (for Scala). Form 3 is useful for a model that is persisted to a file type such as Parquet.
- This method returns an instance of the model class.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature for Form 1:

```
public static Model create( org.apache.spark.sql.Dataset
<org.apache.spark.sql.Row> inputDataset, String modelDatabaseName,
String modelName ) throws IOException
```

Java signature for Form 2:

```
public static Model create( org.apache.spark.sql.Dataset
<org.apache.spark.sql.Row> inputDataset, String modelFileName ) throws
IOException
```

Java signature for Form 3:

```
public static Model create( org.apache.spark.sql.Dataset
<org.apache.spark.sql.Row> inputDataset, org.apache.spark.sql.Dataset
<org.apache.spark.sql.Row> modelDataset ) throws java.io.IOException
```

Examples

Example 1: Form 1

Python:

```
mymodel = Model.create(inputDF, "mymodeldatabase", "inout")
```

Scala:

```
val mymodel = Model.create(inputDF, "mymodeldatabase", "inout")
```

Example 2: Form 2

Python:

```
mymodel = Model.create(inputDF, "/models/inout/inout.is")
```

Scala:

```
val mymodel = Model.create(inputDF, "/models/inout/inout.is")
```

Example 3: Form 3

Python:

```
mymodel = Model.create(inputDF, modelDF)
```

Scala:

```
val mymodel = Model.create(inputDF, modelDF)
```

destroy Method

Destroys the model and closes all resources used by the SAS Embedded Process for Spark.

Syntax

destroy()

Details

- This operation invalidates the current model.
- This operation does not unpersist the input Spark Dataset that is passed to the Model class during creation. If you want to unpersist the input Spark Dataset, call the Dataset unpersist method. For example, submit `ds.unpersist` where `ds` is a variable of type Dataset.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public void destroy() throws IOException
```

Example

Here are some examples:

- Python: `mymodel.destroy()`
- Scala: `mymodel.destroy()`

run Method

Runs the model that is associated with this Model class.

Syntax

run()

Details

- This method returns the output Spark Dataset that is produced by the model execution. Type: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public org.apache.spark.sql.Dataset<org.apache.spark.sql.Row> run()
throws java.io.IOException
```

Example

Here are some examples:

- Python: `dfout = mymodel.run()`
- Scala: `val dfout = mymodel.run()`

setDBMaxText Method

Sets the maximum length of STRING data type columns that are read into the model or written from the model.

Syntax

setDBMaxText(*value*)

Required Argument

value
specifies the maximum length of STRING data type columns. Type: int.

Default 1024 bytes

Details

Java signature:

```
public void setDBMaxText( int value )
```

Example

Here are some examples:

- Python: `mymodel.setDBMaxText(2048)`
- Scala: `mymodel.setDBMaxText(2048)`

setOutputStorageLevelCacheDir Method

Sets the directory on the cluster local file system used to cache output data that is generated by the model execution.

Syntax

setOutputStorageLevelCacheDir(*folder-name*)

Required Argument

folder-name
specifies the directory to cache output data. Type: String.

Default The default directory is `/tmp`.

Interaction The cache directory is used only when the output storage level is set to MMAP.

See "setOutputStorageLevel Method" on page 228

Details

Java signature:

```
public void setOutputStorageLevelCacheDir( String value )
```

Example

Here are some examples:

- Python: `mymodel.setOutputStorageLevelCacheDir("/tmp")`
- Scala: `mymodel.setOutputStorageLevelCacheDir("/tmp")`

setOutputStorageLevel Method

Sets the storage level used by the output Spark Dataset that is generated by the model execution.

Syntax

setOutputStorageLevel("HEAP" | "MMAP")

Required Argument

"HEAP" | "MMAP"
specifies the storage level. Type: String.

HEAP
uses an array of Java objects to store Spark rows in the JVM heap.

MMAP
uses an implementation of a Spark row that is backed by memory mapped file.

Default	HEAP
Interaction	If you set the storage level to MMAP, you can use the setOutputStorageLevelCacheDir method to set the location to cache output data.
See	"setOutputStorageLevelCacheDir Method" on page 227

Details

- Use MMAP when the amount of output data is expected to exceed the maximum amount of memory that is allowed by the Spark executor container. The MMAP disk location is specified by the setOutputStorageLevelCacheDir method.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public void setOutputStorageLevel( String value ) throws IOException
```

Example

Here are some examples:

- Python: `mymodel.setOutputStorageLevel("MMAP")`
- Scala: `mymodel.setOutputStorageLevel("MMAP")`

setTraceOFF Method

Turns off tracing for SAS Embedded Process.

Syntax

setTraceOFF()

Details

Java signature:

```
public void setTraceOFF()
```

Example

Here are some examples:

- Python: `mymodel.setTraceOFF()`
- Scala: `mymodel.setTraceOFF()`

setTraceON Method

Turns on tracing for SAS Embedded Process.

Syntax

setTraceON()

Details

Traces are written to the Spark containers' job log.

Java signature:

```
public void setTraceON()
```

Example

Here are some examples:

- Python: `mymodel.setTraceON()`
- Scala: `mymodel.setTraceON()`

Examples

Example 1: Python Example

This example demonstrates Form 1 of the `create` method syntax.

Note: For Databricks or Synapse, this API is supported if you deploy the sepcorespark RPM file. This API is not supported if you have deployed the accelserver RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

```
# ----------------------------------------------------------
# Import the Model class.
# ----------------------------------------------------------
from sasep.model import Model
# ----------------------------------------------------------
# Load a table into a Spark Dataset.
# ----------------------------------------------------------
intable = spark.table("nyctaxi.trip12258")
# ----------------------------------------------------------
# Create a model
# ----------------------------------------------------------
mymodel = Model.create(intable,"mymodeldatabase","inout")
# ----------------------------------------------------------
# Set additional optional model properties.
# ----------------------------------------------------------
mymodel.setDBMaxText(2000)
# ----------------------------------------------------------
# Run the model. Returns the output of scoring as a Spark DataFrame.
# ----------------------------------------------------------
trips = mymodel.run()
# ----------------------------------------------------------
# Save the Spark Dataset as a table and read it into the Python
runtime.
# ----------------------------------------------------------
trips.write().saveAsTable("default.tripsout")
tripsout=spark.table("default.tripsout")
display(default.tripsout)
# ----------------------------------------------------------
# Destroy the model.
# ----------------------------------------------------------
mymodel.destroy()
```

Example 2: Scala Example

This example demonstrates Form 1 of the `create` method syntax.

Note: For Databricks or Synapse, this API is supported if you deploy the sepcorespark RPM file. This API is not supported if you have deployed the accelserver RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

```
//------------------------------------------------------------
// Import the Model package.
//------------------------------------------------------------
import com.sas.spark.scoring.Model
//------------------------------------------------------------
// Load a table into a Spark Dataset.
//------------------------------------------------------------
val inDataset = spark.table("tablein")
//------------------------------------------------------------
// Create a model.
//------------------------------------------------------------
val mymodel = Model.create(inDataset,"mymodeldatabase","inout")
//------------------------------------------------------------
// Set additional optional model properties.
//------------------------------------------------------------
mymodel.setDBMaxText(2000)
//------------------------------------------------------------
// Run the model and produce the output Spark Dataset.
//------------------------------------------------------------
val dfout = mymodel.run
//------------------------------------------------------------
// Save the output Spark Dataset to a table.
//------------------------------------------------------------
dfout.write.mode("overwrite").saveAsTable("tableout")
//------------------------------------------------------------
// Destroy the model.
//------------------------------------------------------------
mymodel.destroy()
```

11

Python and Scala API for the Accelserver RPM

Introduction to Scoring with the API for the Accelserver RPM

This document is for the Python and Scala API when the accelserver RPM is deployed. The API for the sepcorespark RPM is documented separately. See "Python and Scala API for the Sepcorespark RPM" on page 221.

With this API, you can run a scoring model from a Databricks or Synapse notebook, without invoking SAS.

The API provides a way for users to easily integrate SAS model execution with their Spark processing.

Here are the requirements:

- You must install SAS Embedded Process. See *SAS In-Database Products: Deployment and Administration Guide*.

  Note: For Databricks or Synapse, this API is supported if you deploy the accelserver RPM file. This API is not supported if you have deployed the sepcorespark RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

- Scala or Python code is supported by the API.
- The model must be published already from SAS to the data source.
- The model must be supported for in-database scoring. See Chapter 5, "Models Supported for Scoring," on page 39.
- The input data is a Spark Dataset of rows (also known as a DataFrame) that can be loaded from any source that is supported by Spark. For example, a Spark table or a Parquet file can be used.
- You must import the package that contains the implementation of the Model class. See "Model Class and Methods" on page 234.
- To run a model inside the SAS Embedded Process for Spark, you must create a Model object. Associate the Model object with an input Spark Dataset and the model. You can run the model with Scala or Python by calling the score() method in the Model class object.

Model Class and Methods

The Model class enables execution of SAS models inside the SAS Embedded Process for Spark. The create() method is a static method and it should be called directly from the Model class. Instantiate the Model class by calling into the Model.create() method.

You must import the Model class before using it:

- Scala: `import sas.indb.mla.Model`
- Python: `from sasindb.mla import Model`

See also "Example 1: Python Example" on page 241 and "Example 2: Scala Example" on page 242.

Table 11.1 Tasks and Methods

Task	Method
Create a model	create()
Set additional model properties before execution (Setting additional properties after the model is executed has no effect.)	setKeepListColumns() setMaxUnboundedLength() setTraceON() setTraceOFF()
Run a model	score()
Destroy a model	destroy()

Dictionary

create Method

Creates a Model object that is capable of running the associated model inside the SAS Embedded Process for Spark, using a Spark Dataset as input.

Syntax

create(*modelDataset*, *inputDataset*)

Required Arguments

inputDataset
specifies the Spark Dataset that is used as input to the model.

- Python type: pyspark.sql.DataFrame
- Scala type: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>

modelDataset
specifies the Spark Dataset that contains the published model.

- Python type: pyspark.sql.DataFrame
- Scala type: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>

Details

- This method returns an instance of the model class.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public static Model create( Dataset<Row> modelDataset, Dataset<Row>
inputDataset ) throws java.io.IOException
```

Example

Here are some examples:

- Python: `mymodel = Model.create(modelDF, inputDF)`
- Scala: `val mymodel = Model.create(modelDF, inputDF)`

destroy Method

Destroys the model and closes all resources used by the SAS Embedded Process for Spark.

Syntax

destroy()

Details

- This operation invalidates the current model.
- This operation does not unpersist the input Spark Dataset that is passed to the Model class during creation. If you want to unpersist the input Spark Dataset, call the Dataset unpersist method. For example, submit `ds.unpersist` where `ds` is a variable of type Dataset.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public void destroy() throws IOException
```

Example

Here are some examples:

- Python: `mymodel.destroy()`
- Scala: `mymodel.destroy()`

score Method

Runs the model that is associated with this Model class.

Syntax

score()

Details

- This method returns the output Spark Dataset that is produced by the model execution.
  - Python type: pyspark.sql.DataFrame
  - Scala type: org.apache.spark.sql.Dataset<org.apache.spark.sql.Row>.
- An I/O exception `java.io.IOException` is thrown if an error occurs.

Java signature:

```
public org.apache.spark.sql.Dataset<org.apache.spark.sql.Row> score()
throws java.io.IOException
```

Example

Here are some examples:

- Python: `dfout = model.score()`
- Scala: `val dfout = model.score()`

setKeepListColumns Method

Sets one or more columns to keep in the output data.

Syntax

setKeepListColumns("*value*")

Required Argument

"value"
is a string that specifies the name of one or more columns to keep in the output data. Use a space to separate column names. If a column is not specified, it is dropped from the output data. If you want to keep all of the columns in the output data, then do not use this method.

Tip If the column contains multibyte characters, surround them with double quotation marks. Add the NCHAR indicator, which is the letter N, after the column name. Here is an example that specifies two columns. Each column is enclosed in double quotation marks and followed by the letter N:

```
mymodel.setKeepListColumns("\"姓\"N \"国\"N")
```

Each quotation mark inside the string is escaped with a backslash character (\):

```
mymodel.setKeepListColumns("\"姓\"N \"国\"N")
```

Details

Java signature:

```
public void setKeepListColumns( String value )
```

Example

Here are some examples:

- Python: `mymodel.setKeepListColumns("mycol1 mycol2 mycol3")`
- Scala: `mymodel.setKeepListColumns("mycol1 mycol2 mycol3")`

setMaxUnboundedLength Method

Sets the maximum length for unbounded data types.

Syntax

setMaxUnboundedLength (*value*)

Required Argument

value

specifies the maximum length, in bytes, for unbounded data types (such as BLOB, CLOB, TEXT, or long STRING types) when they are Read into the SAS Embedded Process. The method ensures consistent handling of very large character or binary columns from the database. If data in an input column exceeds the set maximum value, truncation could occur during Read. Truncation could occur during Write, but only for an output column that is also an input column. The option does not apply to any output column that is not an input column. Type: int.

Default 32767 bytes

Details

Java signature:

```
public void setMaxUnboundedLength( int value )
```

Example

Here are some examples:

- Python: `mymodel.setMaxUnboundedLength(2048)`
- Scala: `mymodel.setMaxUnboundedLength(2048)`

setTraceOFF Method

Turns off tracing for SAS Embedded Process.

Syntax

setTraceOFF()

Details

Java signature:

```
public void setTraceOFF()
```

Example

Here are some examples:

- Python: `mymodel.setTraceOFF()`
- Scala: `mymodel.setTraceOFF()`

setTraceON Method

Turns on tracing for SAS Embedded Process.

Syntax

setTraceON()

Details

Traces are written to the Spark containers' job log.

Java signature:

```
public void setTraceON()
```

Example

Here are some examples:

- Python: `mymodel.setTraceON()`
- Scala: `mymodel.setTraceON()`

Examples

Example 1: Python Example

Note: For Databricks or Synapse, this API is supported if you deploy the accelserver RPM file. This API is not supported if you have deployed the sepcorespark RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

```
# Import the Model class.
# ------------------------------------------------------------
from sasindb.mla import Model
# ------------------------------------------------------------
# Load model table into a Spark Dataset.
# ------------------------------------------------------------
modelDF = spark.table("nyctaximodels.publishedmodel")
# ------------------------------------------------------------
# Load input data into a Spark DataFrame.
# ------------------------------------------------------------
inputDF = spark.table("nyctaxi.trip12258")
# ------------------------------------------------------------
# Create a model
# ------------------------------------------------------------
mymodel = Model.create(modelDF, inputDF)
# ------------------------------------------------------------
# Set additional optional model properties.
# ------------------------------------------------------------
mymodel.setMaxUnboundedLength(2048)
# ------------------------------------------------------------
# Run the model. Returns the output of scoring as a Spark DataFrame.
# ------------------------------------------------------------
trips = mymodel.score()
# ------------------------------------------------------------
# Save the output of scoring to a Spark table..
# ------------------------------------------------------------
trips.write().saveAsTable("nyctaxi.trips")
display(trips)
# ------------------------------------------------------------
# Destroy the model.
# ------------------------------------------------------------
mymodel.destroy()
```

Example 2: Scala Example

Note: For Databricks or Synapse, this API is supported if you deploy the accelserver RPM file. This API is not supported if you have deployed the sepcorespark RPM file. If you need the functionality of both RPMs, they must be installed on separate clusters. For more information about the two RPMs, see "Databricks Deployment Steps" in *SAS In-Database Products: Deployment and Administration Guide*.

```
//------------------------------------------------------------
// Import the Model package.
//------------------------------------------------------------
import sas.indb.mla.Model
//------------------------------------------------------------
// Load a published model table into a Spark Dataset.
//------------------------------------------------------------
val modelDataset = spark.table(“mymodel”)
//------------------------------------------------------------
// Load input data into a Spark Dataset.
//------------------------------------------------------------
val inDataset = spark.table("tablein")
//------------------------------------------------------------
// Create a model.
//------------------------------------------------------------
val mymodel = Model.create(modelDataset, inDataset)
//------------------------------------------------------------
// Set additional optional model properties.
//------------------------------------------------------------
mymodel.setMaxUnboundedLength(2048)
//------------------------------------------------------------
// Run the model. Returns the output of scoring as a Spark Dataset.
//------------------------------------------------------------
val outDataset = mymodel.score()
//------------------------------------------------------------
// Save the output Spark Dataset to a table.
//------------------------------------------------------------
outDataset.write.mode("overwrite").saveAsTable("tableout")
//------------------------------------------------------------
// Destroy the model.
//------------------------------------------------------------
mymodel.destroy
```

12

Troubleshooting Scoring

Updates Might Be Needed Due to Driver Change

In 2023.10, SAS changed the redistributed JDBC drivers used by SAS/ACCESS Interface to Hadoop and SAS/ACCESS Interface to Spark. Changes might be needed in your connection options. For example, in the PROPERTIES= option, you might need to modify property names, because names can differ slightly depending on the driver. For more information, see "Troubleshooting Hadoop Connection Problems" in *SAS/ACCESS for Relational Databases: Reference* and "Troubleshooting Spark Connection Problems" in *SAS/ACCESS for Relational Databases: Reference*.

If you encounter timeout errors when publishing large models, try increasing the socketTimeout property.

If you encounter errors or performance issues when publishing large models, you can try increasing or decreasing the batch size. You can set the batchSize in the PROPERTY= option of a Spark LIBNAME statement or Spark CASLIB statement. Unlike most properties, batchSize is specific to the SAS Embedded Process and is not a property of the driver.

For Synapse, when publishing a model, you might experience an `OutOfMemoryError`. Try setting `BatchSize=400` in the URL= option of the JDBC caslib. The PROPERTY= option is not supported in a JDBC caslib.

For Synapse, when you are running models, note that the Spark cluster automatically pauses after 15 minutes of inactivity. SAS code detects that the SAS Embedded Process session does not exist or has timed out, attempts to a new session, and is unsuccessful. Instead, you must manually start a new session by submitting the sparkEmbeddedProcess.startSparkEP action.

For Databricks only, a configuration change might be needed. For more information, see "Fetch Error for Simba Databricks" in *SAS Viya Platform: Deployment Guide*.

PART 4

In-Database Procedures

13

Running SAS Procedures inside the Database

Introduction to In-Database Procedures

When you use in-database technology, the procedures that are enabled for processing inside the data source generate more sophisticated queries. These queries allow the aggregations and analytics to be run inside the data source. Some of the in-database procedures generate SQL procedure syntax and use implicit pass-through to generate the native SQL. Other in-database procedures generate native SQL and use explicit pass-through. For more information about how a specific procedure works inside the data source, see the documentation for that procedure.

For most in-database procedures, a much smaller result set is returned for the remaining analysis that is required to produce the final output. Reducing data movement can improve performance.

For a list of the supported procedures, see Table 1.1 on page 4.

Requirements for In-Database Processing of Procedures

The SQLGENERATION system option or the SQLGENERATION LIBNAME option must be set to DBMS or DBMS='*database-name*'. For the default value of the option, see SQLGENERATION= system option on page 256 or SQLGENERATION= LIBNAME statement option.

Conventional SAS processing is also used when specific procedure statements and options do not support in-database processing.

Procedure Considerations and Limitations

Overview

The considerations and limitations in the following sections apply to both Base SAS and SAS/STAT in-database procedures.

Note: Each in-database procedure has its own specific considerations and limitations. For more information, see the documentation for the procedure.

User-Defined Formats

If you use in-database procedures with user-defined formats that were published in the data source, you must have a local copy of the user-defined formats. Without the local copy, the procedure fails.

Note: The local copy of the user-defined format must be identical in both name and function to the format that is published to the data source. If they are not identical, the following actions occur.

- A "check sum ERROR" warning is produced. The warning indicates that the local and published formats differ.
- The local format is used, and the query is processed by SAS instead of inside the data source.

If this occurs, you can redefine the local format to match the published version and rerun the procedure inside the data source.

Note: Format publishing of user-defined formats is not available for Hadoop.

Row Order

- DBMS tables have no inherent order for the rows. Therefore, the BY statement with the OBS option and the FIRSTOBS option prevents in-database processing.
- If you specify the ORDER=DATA option for input data, the procedure might produce different results for separate runs of the same analysis.
- The order of rows written to a data source table from a SAS procedure is not likely to be preserved. For example, the SORT procedure can write a SAS data set that contains ordered observations. If the results are written to a data source table, the order of rows within that table might not be preserved because the DBMS has no obligation to maintain row order.
- You can print a table using the SQL procedure with an ORDER BY clause to get consistent row order. Another option is to use the SORT procedure to create an ordinary SAS data set and use the PRINT procedure on that SAS data set.

Maximum Number of Columns Using PROC SORT with DB2

If you want to run PROC SORT inside a DB2 database, the maximum number of columns that can be sorted is the DB2 column count limit minus one for each variable in the BY statement. For example, if the column count limit is 1012 columns and there is one variable in the BY statement, then the maximum number of columns that can be sorted is 1011. The DB2 column count limit is derived from the DB2 page size.

If the number of columns is too large, you must reduce the number of columns that are being sorted, otherwise, an error occurs. The other option is to set SQLGENERATION=NONE and run PROC SORT on the SAS client (that is, outside the database).

LIBNAME Statement

- These LIBNAME statement options and settings prevent in-database processing:
  - CONNECTION
  - CONNECTION_GROUP
  - DBCREATE_TABLE_OPTS
  - DBMSTEMP=YES
  - DBCONINIT
  - DBCONTERM
  - DBGEN_NAME=SAS
  - HDFS_METADIR
  - MODE=TERADATA
- LIBNAME concatenation prevents in-database processing.
- For Amazon Redshift, in-database procedures cannot access temporary tables that were created by using the LIBNAME connection to the database. This is because the LIBNAME connection and the in-database connection to the database are separate. Access to temporary tables is limited to the connection that created them.

Data Set Options

These data set options and settings prevent in-database processing:

- DBCONDITION
- DBFORCE
- DBLINK (Oracle only)
- DBNULL
- DBTYPE
- NULLCHAR
- NULLCHARVAL
- OBS= and FIRSTOBS= on DATA= data set
- OUT= data set on DBMS and DATA= data set not on DBMS

  For example, if *data=work.foo* and *out=tera.fooout* where WORK is assigned to the Base SAS engine, in-database processing does not occur.
- RENAME= on a data set

- SCHEMA

Column Names in Netezza

Column names that start with an underscore are not allowed in Netezza.

An error occurs if you try to create an output table in Netezza that contains a column whose name starts with an underscore. The workaround for this is to send the output table to the SAS Work library.

Table Names in Hive

Table names that start with an underscore are not allowed in Hive.

An error occurs if you try to create an output table in Hive whose name starts with an underscore. The workaround for this is to send the output table to the SAS Work directory.

Additional Limitations That Can Prevent In-Database Processing

These items prevent in-database processing:

- DBMSs do not support SAS passwords.
- SAS encryption requires passwords that are not supported.
- Teradata does not support generation options that are explicitly specified in the procedure step, and the procedure does not know whether a generation number is explicit or implicit.
- When the data source resolves function references. the data source searches in this order:

  1. fully qualified object name
  2. current data source
  3. SYSLIB

  If you need to reference functions that are published in a nonsystem, nondefault data source, you must use one of these methods:

  - Use explicit SQL.
  - Use the DATABASE= LIBNAME option.

- Oracle Version 10g supports only 4000 characters per input data item. If you are transcoding input data that has special characters, be aware that these characters might need more than one byte per character.

Using the MSGLEVEL Option to Control Messaging

The MSGLEVEL system option specifies the level of detail in messages that are written to the SAS log. When the MSGLEVEL option is set to N (the default value), the following messages are printed to the SAS log:

- A note that says SQL is used for in-database computations when in-database processing is performed.
- A message can indicate an error from an SQL command that is submitted for in-database computations.
- If there are SQL error messages, a note that says whether SQL is used.

When the MSGLEVEL option is set to I, all the messages that are printed when MSGLEVEL=N are printed to the SAS log. In addition, these messages are also printed to the SAS log:

- A note that explains why SQL was not used for in-database computations, if SQL is not used.

  Note: No note is printed if you specify SQLGENERATION=NONE.

- A note that says that SQL cannot be used because there are no observations in the database.

  Note: This information is not always available to the procedure.

- A note that says that the TYPE= attribute is not stored in DBMS tables. You see this note if you try to create a special SAS data set as a DBMS table for PROC MEANS or PROC SUMMARY.
- A note that says if the format was or was not found in the data source. You see this note if you use a format that SAS supplies or a user-defined format.

PART 5

System Options and Environment Variables

14

System Options That Affect In-Database Processing

Dictionary

DSACCEL= System Option

Specifies whether the DATA step is enabled for in-database processing in supported environments.

Valid in:	SAS Environment Manager, Autoexec file, OPTIONS statement, SASV9_OPTIONS environment variable
Category:	System Administration: Performance
PROC OPTIONS GROUP=	PERFORMANCE
Default:	The shipped default is NONE.
Requirement:	For information about requirements and limitations, see Part 2, “In-Database DATA Step Processing,” on page 7.
Note:	This option can be restricted by a site administrator. For more information, see “Restricted Options” in *SAS System Options: Reference*.

Syntax

DSACCEL=ANY | NONE

Syntax Description

ANY
enables the DATA step to execute with in-database processing.

NONE
disables the DATA step from executing with in-database processing.

Details

DSACCEL=ANY enables the DATA step to run, with limitations, inside the data source. A LIBNAME engine and SAS Embedded Process are used.

SQLGENERATION= System Option

Specifies whether and when SAS procedures generate SQL for in-database processing of source data.

Valid in: OPTIONS statement, SASV9_OPTIONS environment variable

Categories: Data Access
System Administration: Performance

Default: NONE DBMS='TERADATA DB2 ORACLE NETEZZA GREENPLM HADOOP SAPHANA IMPALA POSTGRES REDSHIFT SQLSVR VERTICA SNOW BIGQUERY SPARK YBRICK MYSQL DUCKDB'

Restrictions: Parentheses are required when this option value contains multiple keywords.
The maximum length of the option value is 4096 characters.
For DBMS= and EXCLUDEDB= values, the maximum length of an engine name is eight characters. For the EXCLUDEPROC= value, the maximum length of a procedure name is 16 characters. An engine can appear only once, and a procedure can appear only once for a given engine.
Not all procedures support SQL generation for in-database processing for every engine type. If you specify a value that is not supported, an error message indicates the level of SQL generation that is not supported. The procedure can then reset to the default so that source table records can be read and processed within SAS. If this is not possible, the procedure ends and sets SYSERR= as needed.

Requirement: You must specify NONE or DBMS as the primary state.

Interactions: Use this option with such procedures as PROC FREQ to indicate that SQL is generated for in-database processing of DBMS tables through supported SAS/ACCESS engines.

You can specify different SQLGENERATION= values for the DATA= and OUT= data sets by using different LIBNAME statements for each of these data sets.

Data source: Amazon Redshift, DB2, DuckDB, Google BigQuery, Greenplum, Hadoop, Impala, Microsoft SQL Server, MySQL, Netezza, Oracle, PostgreSQL, SAP HANA, Snowflake, Spark, Teradata, Vertica, Yellowbrick

Note: Support for DuckDB was added in 2026.08.

Tip: After you specify a required value (primary state), you can specify optional values (modifiers).

See: SQLGENERATION= LIBNAME option

"Running In-Database Procedures" in *SAS In-Database Products: User's Guide*

Syntax

SQLGENERATION=<(>NONE | DBMS

<DBMS='*engine1* <engine2 ...>'>

<EXCLUDEDB=*engine* | '*engine1* < engine2 ...>'>

<EXCLUDEPROC="*engine='proc1 <proc2 ...>' <engine2='proc1 <proc2 ...>'*">"> <)>

SQLGENERATION=' '

Required Arguments

NONE
prevents those SAS procedures that are enabled for in-database processing from generating SQL for in-database processing. This is a primary state.

DBMS
allows SAS procedures that are enabled for in-database processing to generate SQL for in-database processing of DBMS tables through supported SAS/ACCESS engines. This is a primary state.

Note: As a best practice, run as many calculations in-database as possible. Processing that is run in-database generally results in better performance.

' '
resets the value to the default that was shipped.

Optional Arguments

DBMS='*engine1 <engine2 ...>*'
specifies one or more SAS/ACCESS engines. It modifies the primary state.

EXCLUDEDB=*engine* | '*engine1<engine2 ...>*'
prevents SAS procedures from generating SQL for in-database processing for one or more specified SAS/ACCESS engines.

EXCLUDEPROC="*engine='proc1 <proc2 ...>' <engine2='proc1 <proc2 ...>'> *"
prevents one or more specified SAS procedures from generating SQL for in-database processing for one or more specified SAS/ACCESS engines.

Details

Here are the values that you specify for each engine. The values are not case-specific.

Table 14.1 *SQLGENERATION= Values to Specify for Each Engine*

Engine	SQLGENERATION= Value
Amazon Redshift	REDSHIFT
DB2	DB2
DuckDB	DUCKDB
Google BigQuery	BIGQUERY
Greenplum	GREENPLM
Hadoop	HADOOP
Impala	IMPALA
Microsoft SQL Server	SQLSVR
MySQL	MYSQL
Netezza	NETEZZA
Oracle	ORACLE
PostgreSQL	POSTGRES
SAP HANA	SAPHANA
Snowflake	SNOW
Spark	SPARK
Teradata	TERADATA
Vertica	VERTICA

Engine	SQLGENERATION= Value
Yellowbrick	YBRICK

Here is how SAS/ACCESS handles precedence between the LIBNAME and system option.

Table 14.2 *Precedence of Values for SQLGENERATION= LIBNAME and System Options*

LIBNAME Option	PROC EXCLUDE on System Option?	Engine Specified on System Option	Resulting Value	From (option)
not specified	yes	NONE	NONE	system
		DBMS	EXCLUDEPROC	
NONE		NONE	NONE	LIBNAME
		DBMS		
DBMS		NONE	EXCLUDEPROC	system
		DBMS		
not specified	no	NONE	NONE	
		DBMS	DBMS	
NONE		NONE	NONE	LIBNAME
		DBMS		
DBMS		NONE	DBMS	
		DBMS		

Examples

Example 1: View the Default Value

Here is code to see the default value that is shipped with the product.

```
options sqlgeneration='';
proc options option=sqlgeneration;
run;
```

Example 2: Restrict In-Database Processing for an Engine

SAS procedures generate SQL for in-database processing for all databases except DB2 in this example.

```
options sqlgeneration='';
options sqlgeneration=(DBMS EXCLUDEDB='DB2');
proc options option=sqlgeneration;
run;
```

Example 3: Restrict In-Database Processing for All Engines but One

In this example, in-database processing occurs only for Teradata. SAS procedures that are run on other databases do not generate SQL for in-database processing.

```
options sqlgeneration='';
options SQLGENERATION=(NONE DBMS='Teradata');
proc options option=sqlgeneration;
run;
```

Example 4: Enable In-Database Processing for Multiple Engines but Restrict Processing for Some Procedures

For this example, SAS procedures generate SQL for PostgreSQL and Oracle in-database processing. However, no SQL is generated for PROC1 and PROC2 in Oracle.

```
options sqlgeneration='';
options SQLGENERATION=(NONE DBMS='Postgres Oracle'
   EXCLUDEPROC="oracle='proc1 proc2'");
proc options option=sqlgeneration;
run;
```

15

Environment Variables That Affect In-Database Processing

Dictionary

SAS_EP_PROPERTIES Environment Variable

Passes properties to the SAS Embedded Process for SingleStore.

Valid in:	SAS configuration, SAS invocation, OPTIONS statement
Category:	Program Control
Default:	None
Data source:	SingleStore
Note:	This environment variable was added in 2024.04.

Syntax

SAS_EP_PROPERTIES=*properties*

Required Argument

properties

lists one or more properties, as "*name*=*value*" pairs, to pass to the SAS Embedded Process for SingleStore.

Enclose the list in single or double quotation marks. Use a semicolon to separate each *name*=*value* pair in the list. The following properties are supported:

cmp.threads=*number*

specifies the maximum number of CMP threads that are used to run the CMP program. The maximum number of CMP threads cannot exceed the number of cores that are available in the system. Most customers do not need to set this property.

logger.level=*value*

specifies the logger level. The following levels are supported: allon, trace, debug, info, warn, error, fatal, off. You can use this property for debugging errors. Use the trace value to see the most amount of information. Usually, the debug value is sufficient. "Accessing SAS Embedded Process Logs in SingleStore" in *SAS SpeedyStore: Administration and Configuration Guide*.

Details

Use the SAS_EP_PROPERTIES= environment variable to set certain properties for SAS Embedded Process for SingleStore. The properties are set for in-database DATA step processing only. For more information, see Chapter 2, "DATA Step Acceleration," on page 9.

Example: Set Properties for SAS Embedded Process

The following OPTIONS statement passes two properties to SAS Embedded Process in SingleStore:

```
options set=SAS_EP_PROPERTIES="cmp.threads=4;logger.level=debug";
```

PART 6

Appendix

Appendix 1

Scoring File Examples

Example of a DS2 Scoring File

This is an example of a DS2 scoring program. For more information, see "Files Used in Model Publishing" on page 55.

```
data &ASTER_OUTPUT;
 #_local _LPMAX;
 #_local _P4;
 #_local _P3;
 #_local _P2;
 #_local _P1;
 #_local _P0;
 #_local _IY;
 #_local _MAXP;
 #_local _LP3;
 #_local _LP2;
 #_local _LP1;
 #_local _LP0;
 #_local _TEMP;
 #_local _7_1;
 #_local _7_0;
 #_local _6_2;
 #_local _6_1;
 #_local _6_0;
 #_local _5_14;
 #_local _5_13;
 #_local _5_12;
 #_local _5_11;
```

```
 #_local _5_10;
 #_local _5_9;
 #_local _5_8;
 #_local _5_7;
 #_local _5_6;
 #_local _5_5;
 #_local _5_4;
 #_local _5_3;
 #_local _5_2;
 #_local _5_1;
 #_local _5_0;
 #_local _3_10;
 #_local _3_9;
 #_local _3_8;
 #_local _3_7;
 #_local _3_6;
 #_local _3_5;
 #_local _3_4;
 #_local _3_3;
 #_local _3_2;
 #_local _3_1;
 #_local _3_0;
 #_local _2_12;
 #_local _2_11;
 #_local _2_10;
 #_local _2_9;
 #_local _2_8;
 #_local _2_7;
 #_local _2_6;
 #_local _2_5;
 #_local _2_4;
 #_local _2_3;
 #_local _2_2;
 #_local _2_1;
 #_local _2_0;
 #_local _DM_FIND;
 #_local _1_14;
 #_local _1_13;
 #_local _1_12;
 #_local _1_11;
 #_local _1_10;
 #_local _1_9;
 #_local _1_8;
 #_local _1_7;
 #_local _1_6;
 #_local _1_5;
 #_local _1_4;
 #_local _1_3;
 #_local _1_2;
 #_local _1_1;
 #_local _1_0;
 #_local _DM_BAD;
dcl char(4) _WARN_;
dcl char(6) I_ATTACK;
dcl char(6) U_ATTACK;
dcl char(32) EM_CLASSIFICATION;
```

```
dcl double COUNT;
dcl double DIF_SRVR;
dcl char(32) FLAG;
dcl double HOT;
dcl double SAM_SRAT;
dcl char(32) SERVICE;
dcl double SRV_CNT;
method run();
 dcl char(8) _NORM8;
 dcl char(8) _NORM8;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(12) _DM12;
 dcl char(6) REGDRU[5];
 dcl char(6) REGDRF[5];
REGDRU:=('u2r   ', 'r2l   ', 'probe ', 'normal', 'dos   ');
REGDRF:=('U2R', 'R2L', 'PROBE', 'NORMAL', 'DOS');
set &ASTER_INPUT;
_WARN_ = ' ';
if (COUNT = .) then AOV16_COUNT = 8.0;
 else if (COUNT <= 31.9375) then AOV16_COUNT = 1.0;
 else if (COUNT <= 63.875) then AOV16_COUNT = 2.0;
 else if (COUNT <= 95.8125) then AOV16_COUNT = 3.0;
 else if (COUNT <= 127.75) then AOV16_COUNT = 4.0;
 else if (COUNT <= 159.6875) then AOV16_COUNT = 5.0;
 else if (COUNT <= 191.625) then AOV16_COUNT = 6.0;
 else if (COUNT <= 223.5625) then AOV16_COUNT = 7.0;
 else if (COUNT <= 255.5) then AOV16_COUNT = 8.0;
 else if (COUNT <= 287.4375) then AOV16_COUNT = 9.0;
 else if (COUNT <= 319.375) then AOV16_COUNT = 10.0;
 else if (COUNT <= 351.3125) then AOV16_COUNT = 11.0;
 else if (COUNT <= 383.25) then AOV16_COUNT = 12.0;
 else if (COUNT <= 415.1875) then AOV16_COUNT = 13.0;
 else if (COUNT <= 447.125) then AOV16_COUNT = 14.0;
 else if (COUNT <= 479.0625) then AOV16_COUNT = 15.0;
 else AOV16_COUNT = 16.0;
if (SRV_CNT = .) then AOV16_SRV_CNT = 7.0;
 else if (SRV_CNT <= 31.9375) then AOV16_SRV_CNT = 1.0;
 else if (SRV_CNT <= 63.875) then AOV16_SRV_CNT = 2.0;
 else if (SRV_CNT <= 95.8125) then AOV16_SRV_CNT = 3.0;
 else if (SRV_CNT <= 127.75) then AOV16_SRV_CNT = 4.0;
 else if (SRV_CNT <= 159.6875) then AOV16_SRV_CNT = 5.0;
 else if (SRV_CNT <= 191.625) then AOV16_SRV_CNT = 6.0;
 else if (SRV_CNT <= 223.5625) then AOV16_SRV_CNT = 7.0;
 else if (SRV_CNT <= 255.5) then AOV16_SRV_CNT = 8.0;
 else if (SRV_CNT <= 287.4375) then AOV16_SRV_CNT = 9.0;
 else if (SRV_CNT <= 319.375) then AOV16_SRV_CNT = 10.0;
 else if (SRV_CNT <= 351.3125) then AOV16_SRV_CNT = 11.0;
 else if (SRV_CNT <= 383.25) then AOV16_SRV_CNT = 12.0;
 else if (SRV_CNT <= 415.1875) then AOV16_SRV_CNT = 13.0;
 else if (SRV_CNT <= 447.125) then AOV16_SRV_CNT = 14.0;
 else if (SRV_CNT <= 479.0625) then AOV16_SRV_CNT = 15.0;
 else AOV16_SRV_CNT = 16.0;
```

```
if (SAM_SRAT = .) then AOV16_SAM_SRAT = 14.0;
 else if (SAM_SRAT <= 0.0625) then AOV16_SAM_SRAT = 1.0;
 else if (SAM_SRAT <= 0.125) then AOV16_SAM_SRAT = 2.0;
 else if (SAM_SRAT <= 0.1875) then AOV16_SAM_SRAT = 3.0;
 else if (SAM_SRAT <= 0.25) then AOV16_SAM_SRAT = 4.0;
 else if (SAM_SRAT <= 0.3125) then AOV16_SAM_SRAT = 5.0;
 else if (SAM_SRAT <= 0.375) then AOV16_SAM_SRAT = 6.0;
 else if (SAM_SRAT <= 0.4375) then AOV16_SAM_SRAT = 7.0;
 else if (SAM_SRAT <= 0.5) then AOV16_SAM_SRAT = 8.0;
 else if (SAM_SRAT <= 0.5625) then AOV16_SAM_SRAT = 9.0;
 else if (SAM_SRAT <= 0.625) then AOV16_SAM_SRAT = 10.0;
 else if (SAM_SRAT <= 0.6875) then AOV16_SAM_SRAT = 11.0;
 else if (SAM_SRAT <= 0.75) then AOV16_SAM_SRAT = 12.0;
 else if (SAM_SRAT <= 0.8125) then AOV16_SAM_SRAT = 13.0;
 else if (SAM_SRAT <= 0.875) then AOV16_SAM_SRAT = 14.0;
 else if (SAM_SRAT <= 0.9375) then AOV16_SAM_SRAT = 15.0;
 else AOV16_SAM_SRAT = 16.0;
if (DIF_SRVR = .) then AOV16_DIF_SRVR = 1.0;
 else if (DIF_SRVR <= 0.0625) then AOV16_DIF_SRVR = 1.0;
 else if (DIF_SRVR <= 0.125) then AOV16_DIF_SRVR = 2.0;
 else if (DIF_SRVR <= 0.1875) then AOV16_DIF_SRVR = 3.0;
 else if (DIF_SRVR <= 0.25) then AOV16_DIF_SRVR = 4.0;
 else if (DIF_SRVR <= 0.3125) then AOV16_DIF_SRVR = 5.0;
 else if (DIF_SRVR <= 0.375) then AOV16_DIF_SRVR = 6.0;
 else if (DIF_SRVR <= 0.4375) then AOV16_DIF_SRVR = 7.0;
 else if (DIF_SRVR <= 0.5) then AOV16_DIF_SRVR = 8.0;
 else if (DIF_SRVR <= 0.5625) then AOV16_DIF_SRVR = 9.0;
 else if (DIF_SRVR <= 0.625) then AOV16_DIF_SRVR = 10.0;
 else if (DIF_SRVR <= 0.6875) then AOV16_DIF_SRVR = 11.0;
 else if (DIF_SRVR <= 0.75) then AOV16_DIF_SRVR = 12.0;
 else if (DIF_SRVR <= 0.8125) then AOV16_DIF_SRVR = 13.0;
 else if (DIF_SRVR <= 0.875) then AOV16_DIF_SRVR = 14.0;
 else if (DIF_SRVR <= 0.9375) then AOV16_DIF_SRVR = 15.0;
 else AOV16_DIF_SRVR = 16.0;
if (HOT = .) then AOV16_HOT = 1.0;
 else if (HOT <= 1.875) then AOV16_HOT = 1.0;
 else if (HOT <= 3.75) then AOV16_HOT = 2.0;
 else if (HOT <= 5.625) then AOV16_HOT = 3.0;
 else if (HOT <= 7.5) then AOV16_HOT = 4.0;
 else if (HOT <= 9.375) then AOV16_HOT = 5.0;
 else if (HOT <= 11.25) then AOV16_HOT = 6.0;
 else if (HOT <= 13.125) then AOV16_HOT = 7.0;
 else if (HOT <= 15.0) then AOV16_HOT = 8.0;
 else if (HOT <= 16.875) then AOV16_HOT = 9.0;
 else if (HOT <= 18.75) then AOV16_HOT = 10.0;
 else if (HOT <= 20.625) then AOV16_HOT = 11.0;
 else if (HOT <= 22.5) then AOV16_HOT = 12.0;
 else if (HOT <= 24.375) then AOV16_HOT = 13.0;
 else if (HOT <= 26.25) then AOV16_HOT = 14.0;
 else if (HOT <= 28.125) then AOV16_HOT = 15.0;
 else AOV16_HOT = 16.0;
_NORM8 = DMNORM(SERVICE, 32.0);
select (_NORM8);
when ('IRC     ') G_SERVICE = 2.0;
when ('X11     ') G_SERVICE = 2.0;
when ('Z39_50  ') G_SERVICE = 1.0;
```

```
when ('AUTH    ') G_SERVICE = 2.0;
when ('BGP     ') G_SERVICE = 0.0;
when ('COURIER ') G_SERVICE = 1.0;
when ('CSNET_NS') G_SERVICE = 1.0;
when ('CTF     ') G_SERVICE = 0.0;
when ('DAYTIME ') G_SERVICE = 1.0;
when ('DISCARD ') G_SERVICE = 0.0;
when ('DOMAIN  ') G_SERVICE = 1.0;
when ('DOMAIN_U') G_SERVICE = 2.0;
when ('ECHO    ') G_SERVICE = 0.0;
when ('ECO_I   ') G_SERVICE = 2.0;
when ('ECR_I   ') G_SERVICE = 0.0;
when ('EFS     ') G_SERVICE = 1.0;
when ('EXEC    ') G_SERVICE = 0.0;
when ('FINGER  ') G_SERVICE = 2.0;
when ('FTP     ') G_SERVICE = 2.0;
when ('FTP_DATA') G_SERVICE = 2.0;
when ('GOPHER  ') G_SERVICE = 1.0;
when ('HOSTNAME') G_SERVICE = 0.0;
when ('HTTP    ') G_SERVICE = 2.0;
when ('HTTP_443') G_SERVICE = 0.0;
when ('IMAP4   ') G_SERVICE = 1.0;
when ('ISO_TSAP') G_SERVICE = 0.0;
when ('KLOGIN  ') G_SERVICE = 0.0;
when ('KSHELL  ') G_SERVICE = 0.0;
when ('LDAP    ') G_SERVICE = 0.0;
when ('LINK    ') G_SERVICE = 1.0;
when ('LOGIN   ') G_SERVICE = 0.0;
when ('MTP     ') G_SERVICE = 1.0;
when ('NAME    ') G_SERVICE = 0.0;
when ('NETBIOS_') G_SERVICE = 0.0;
when ('NETSTAT ') G_SERVICE = 0.0;
when ('NNSP    ') G_SERVICE = 0.0;
when ('NNTP    ') G_SERVICE = 1.0;
when ('NTP_U   ') G_SERVICE = 2.0;
when ('OTHER   ') G_SERVICE = 2.0;
when ('POP_2   ') G_SERVICE = 0.0;
when ('POP_3   ') G_SERVICE = 2.0;
when ('PRINTER ') G_SERVICE = 1.0;
when ('PRIVATE ') G_SERVICE = 1.0;
when ('RED_I   ') G_SERVICE = 2.0;
when ('REMOTE_J') G_SERVICE = 1.0;
when ('RJE     ') G_SERVICE = 1.0;
when ('SHELL   ') G_SERVICE = 0.0;
when ('SMTP    ') G_SERVICE = 2.0;
when ('SQL_NET ') G_SERVICE = 0.0;
when ('SSH     ') G_SERVICE = 1.0;
when ('SUNRPC  ') G_SERVICE = 1.0;
when ('SUPDUP  ') G_SERVICE = 1.0;
when ('SYSTAT  ') G_SERVICE = 1.0;
when ('TELNET  ') G_SERVICE = 2.0;
when ('TFTP_U  ') G_SERVICE = 2.0;
when ('TIM_I   ') G_SERVICE = 1.0;
when ('TIME    ') G_SERVICE = 2.0;
when ('URH_I   ') G_SERVICE = 2.0;
when ('URP_I   ') G_SERVICE = 2.0;
```

```
when ('UUCP    ') G_SERVICE = 1.0;
when ('UUCP_PAT') G_SERVICE = 0.0;
when ('VMNET   ') G_SERVICE = 1.0;
when ('WHOIS   ') G_SERVICE = 1.0;
otherwise _WARN_ = 'U';
end;
_NORM8 = DMNORM(FLAG, 32.0);
select (_NORM8);
when ('OTH     ') G_FLAG = 3.0;
when ('REJ     ') G_FLAG = 2.0;
when ('RSTO    ') G_FLAG = 2.0;
when ('RSTOS0  ') G_FLAG = 3.0;
when ('RSTR    ') G_FLAG = 3.0;
when ('S0      ') G_FLAG = 0.0;
when ('S1      ') G_FLAG = 3.0;
when ('S2      ') G_FLAG = 3.0;
when ('S3      ') G_FLAG = 3.0;
when ('SF      ') G_FLAG = 1.0;
when ('SH      ') G_FLAG = 3.0;
otherwise _WARN_ = 'U';
end;
_DM_BAD = 0.0;
_1_0 = 0.0;
_1_1 = 0.0;
_1_2 = 0.0;
_1_3 = 0.0;
_1_4 = 0.0;
_1_5 = 0.0;
_1_6 = 0.0;
_1_7 = 0.0;
_1_8 = 0.0;
_1_9 = 0.0;
_1_10 = 0.0;
_1_11 = 0.0;
_1_12 = 0.0;
_1_13 = 0.0;
_1_14 = 0.0;
if MISSING(AOV16_COUNT) then do ;
_1_0 = .;
_1_1 = .;
_1_2 = .;
_1_3 = .;
_1_4 = .;
_1_5 = .;
_1_6 = .;
_1_7 = .;
_1_8 = .;
_1_9 = .;
_1_10 = .;
_1_11 = .;
_1_12 = .;
_1_13 = .;
_1_14 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
```

```
 else do ;
_DM12 = put(AOV16_COUNT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
_DM_FIND = 0.0;
if _DM12 <= '16' then do ;
if _DM12 <= '12' then do ;
if _DM12 <= '10' then do ;
if _DM12 = '1' then do ;
_1_0 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '10' then do ;
_1_9 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '11' then do ;
_1_10 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '12' then do ;
_1_11 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
 else do ;
if _DM12 <= '14' then do ;
if _DM12 = '13' then do ;
_1_12 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '14' then do ;
_1_13 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '15' then do ;
_1_14 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '16' then do ;
_1_0 = -1.0;
_1_1 = -1.0;
_1_2 = -1.0;
_1_3 = -1.0;
_1_4 = -1.0;
```

```
_1_5 = -1.0;
_1_6 = -1.0;
_1_7 = -1.0;
_1_8 = -1.0;
_1_9 = -1.0;
_1_10 = -1.0;
_1_11 = -1.0;
_1_12 = -1.0;
_1_13 = -1.0;
_1_14 = -1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
end;
 else do ;
if _DM12 <= '5' then do ;
if _DM12 <= '3' then do ;
if _DM12 = '2' then do ;
_1_1 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '3' then do ;
_1_2 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
 else do ;
if _DM12 = '4' then do ;
_1_3 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '5' then do ;
_1_4 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
end;
 else do ;
if _DM12 <= '7' then do ;
if _DM12 = '6' then do ;
_1_5 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '7' then do ;
_1_6 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
```

```
 else do ;
if _DM12 = '8' then do ;
_1_7 = 1.0;
_DM_FIND = 1.0;
end;
 else do ;
if _DM12 = '9' then do ;
_1_8 = 1.0;
_DM_FIND = 1.0;
end;
end;
end;
end;
end;
if ^_DM_FIND then do ;
_1_0 = .;
_1_1 = .;
_1_2 = .;
_1_3 = .;
_1_4 = .;
_1_5 = .;
_1_6 = .;
_1_7 = .;
_1_8 = .;
_1_9 = .;
_1_10 = .;
_1_11 = .;
_1_12 = .;
_1_13 = .;
_1_14 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_2_0 = 0.0;
_2_1 = 0.0;
_2_2 = 0.0;
_2_3 = 0.0;
_2_4 = 0.0;
_2_5 = 0.0;
_2_6 = 0.0;
_2_7 = 0.0;
_2_8 = 0.0;
_2_9 = 0.0;
_2_10 = 0.0;
_2_11 = 0.0;
_2_12 = 0.0;
if MISSING(AOV16_DIF_SRVR) then do ;
_2_0 = .;
_2_1 = .;
_2_2 = .;
_2_3 = .;
_2_4 = .;
_2_5 = .;
_2_6 = .;
_2_7 = .;
```

```
_2_8 = .;
_2_9 = .;
_2_10 = .;
_2_11 = .;
_2_12 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_DIF_SRVR, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_2_0 = 1.0;
end;
 else if _DM12 = '2' then do ;
_2_1 = 1.0;
end;
 else if _DM12 = '16' then do ;
_2_0 = -1.0;
_2_1 = -1.0;
_2_2 = -1.0;
_2_3 = -1.0;
_2_4 = -1.0;
_2_5 = -1.0;
_2_6 = -1.0;
_2_7 = -1.0;
_2_8 = -1.0;
_2_9 = -1.0;
_2_10 = -1.0;
_2_11 = -1.0;
_2_12 = -1.0;
end;
 else if _DM12 = '11' then do ;
_2_10 = 1.0;
end;
 else if _DM12 = '8' then do ;
_2_7 = 1.0;
end;
 else if _DM12 = '10' then do ;
_2_9 = 1.0;
end;
 else if _DM12 = '3' then do ;
_2_2 = 1.0;
end;
 else if _DM12 = '7' then do ;
_2_6 = 1.0;
end;
 else if _DM12 = '4' then do ;
_2_3 = 1.0;
end;
 else if _DM12 = '9' then do ;
_2_8 = 1.0;
end;
 else if _DM12 = '5' then do ;
_2_4 = 1.0;
end;
```

```
 else if _DM12 = '12' then do ;
_2_11 = 1.0;
end;
 else if _DM12 = '6' then do ;
_2_5 = 1.0;
end;
 else if _DM12 = '13' then do ;
_2_12 = 1.0;
end;
 else do ;
_2_0 = .;
_2_1 = .;
_2_2 = .;
_2_3 = .;
_2_4 = .;
_2_5 = .;
_2_6 = .;
_2_7 = .;
_2_8 = .;
_2_9 = .;
_2_10 = .;
_2_11 = .;
_2_12 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_3_0 = 0.0;
_3_1 = 0.0;
_3_2 = 0.0;
_3_3 = 0.0;
_3_4 = 0.0;
_3_5 = 0.0;
_3_6 = 0.0;
_3_7 = 0.0;
_3_8 = 0.0;
_3_9 = 0.0;
_3_10 = 0.0;
if MISSING(AOV16_HOT) then do ;
_3_0 = .;
_3_1 = .;
_3_2 = .;
_3_3 = .;
_3_4 = .;
_3_5 = .;
_3_6 = .;
_3_7 = .;
_3_8 = .;
_3_9 = .;
_3_10 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_HOT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
```

```
if _DM12 = '1' then do ;
_3_0 = 1.0;
end;
 else if _DM12 = '2' then do ;
_3_1 = 1.0;
end;
 else if _DM12 = '15' then do ;
_3_10 = 1.0;
end;
 else if _DM12 = '3' then do ;
_3_2 = 1.0;
end;
 else if _DM12 = '4' then do ;
_3_3 = 1.0;
end;
 else if _DM12 = '11' then do ;
_3_7 = 1.0;
end;
 else if _DM12 = '12' then do ;
_3_8 = 1.0;
end;
 else if _DM12 = '10' then do ;
_3_6 = 1.0;
end;
 else if _DM12 = '8' then do ;
_3_5 = 1.0;
end;
 else if _DM12 = '16' then do ;
_3_0 = -1.0;
_3_1 = -1.0;
_3_2 = -1.0;
_3_3 = -1.0;
_3_4 = -1.0;
_3_5 = -1.0;
_3_6 = -1.0;
_3_7 = -1.0;
_3_8 = -1.0;
_3_9 = -1.0;
_3_10 = -1.0;
end;
 else if _DM12 = '13' then do ;
_3_9 = 1.0;
end;
 else if _DM12 = '7' then do ;
_3_4 = 1.0;
end;
 else do ;
_3_0 = .;
_3_1 = .;
_3_2 = .;
_3_3 = .;
_3_4 = .;
_3_5 = .;
_3_6 = .;
_3_7 = .;
_3_8 = .;
```

```
_3_9 = .;
_3_10 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
_5_0 = 0.0;
_5_1 = 0.0;
_5_2 = 0.0;
_5_3 = 0.0;
_5_4 = 0.0;
_5_5 = 0.0;
_5_6 = 0.0;
_5_7 = 0.0;
_5_8 = 0.0;
_5_9 = 0.0;
_5_10 = 0.0;
_5_11 = 0.0;
_5_12 = 0.0;
_5_13 = 0.0;
_5_14 = 0.0;
if MISSING(AOV16_SRV_CNT) then do ;
_5_0 = .;
_5_1 = .;
_5_2 = .;
_5_3 = .;
_5_4 = .;
_5_5 = .;
_5_6 = .;
_5_7 = .;
_5_8 = .;
_5_9 = .;
_5_10 = .;
_5_11 = .;
_5_12 = .;
_5_13 = .;
_5_14 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(AOV16_SRV_CNT, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_5_0 = 1.0;
end;
 else if _DM12 = '16' then do ;
_5_0 = -1.0;
_5_1 = -1.0;
_5_2 = -1.0;
_5_3 = -1.0;
_5_4 = -1.0;
_5_5 = -1.0;
_5_6 = -1.0;
_5_7 = -1.0;
_5_8 = -1.0;
```

```
_5_9 = -1.0;
_5_10 = -1.0;
_5_11 = -1.0;
_5_12 = -1.0;
_5_13 = -1.0;
_5_14 = -1.0;
end;
 else if _DM12 = '2' then do ;
_5_1 = 1.0;
end;
 else if _DM12 = '15' then do ;
_5_14 = 1.0;
end;
 else if _DM12 = '14' then do ;
_5_13 = 1.0;
end;
 else if _DM12 = '3' then do ;
_5_2 = 1.0;
end;
 else if _DM12 = '4' then do ;
_5_3 = 1.0;
end;
 else if _DM12 = '5' then do ;
_5_4 = 1.0;
end;
 else if _DM12 = '6' then do ;
_5_5 = 1.0;
end;
 else if _DM12 = '8' then do ;
_5_7 = 1.0;
end;
 else if _DM12 = '7' then do ;
_5_6 = 1.0;
end;
 else if _DM12 = '9' then do ;
_5_8 = 1.0;
end;
 else if _DM12 = '10' then do ;
_5_9 = 1.0;
end;
 else if _DM12 = '12' then do ;
_5_11 = 1.0;
end;
 else if _DM12 = '11' then do ;
_5_10 = 1.0;
end;
 else if _DM12 = '13' then do ;
_5_12 = 1.0;
end;
 else do ;
_5_0 = .;
_5_1 = .;
_5_2 = .;
_5_3 = .;
_5_4 = .;
_5_5 = .;
```

```
_5_6 = .;
_5_7 = .;
_5_8 = .;
_5_9 = .;
_5_10 = .;
_5_11 = .;
_5_12 = .;
_5_13 = .;
_5_14 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if MISSING(G_FLAG) then do ;
_6_0 = .;
_6_1 = .;
_6_2 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
end;
 else do ;
_DM12 = put(G_FLAG, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '1' then do ;
_6_0 = 0.0;
_6_1 = 1.0;
_6_2 = 0.0;
end;
 else if _DM12 = '0' then do ;
_6_0 = 1.0;
_6_1 = 0.0;
_6_2 = 0.0;
end;
 else if _DM12 = '2' then do ;
_6_0 = 0.0;
_6_1 = 0.0;
_6_2 = 1.0;
end;
 else if _DM12 = '3' then do ;
_6_0 = -1.0;
_6_1 = -1.0;
_6_2 = -1.0;
end;
 else do ;
_6_0 = .;
_6_1 = .;
_6_2 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if MISSING(G_SERVICE) then do ;
_7_0 = .;
_7_1 = .;
substr(_WARN_, 1.0, 1.0) = 'M';
_DM_BAD = 1.0;
```

```
end;
 else do ;
_DM12 = put(G_SERVICE, BEST12.);
_DM12 = DMNORM(_DM12, 32.0);
if _DM12 = '2' then do ;
_7_0 = -1.0;
_7_1 = -1.0;
end;
 else if _DM12 = '0' then do ;
_7_0 = 1.0;
_7_1 = 0.0;
end;
 else if _DM12 = '1' then do ;
_7_0 = 0.0;
_7_1 = 1.0;
end;
 else do ;
_7_0 = .;
_7_1 = .;
substr(_WARN_, 2.0, 1.0) = 'U';
_DM_BAD = 1.0;
end;
end;
if _DM_BAD > 0.0 then do ;
_P0 = 0.0006798097;
_P1 = 0.0153183775;
_P2 = 0.0558123725;
_P3 = 0.3941083163;
_P4 = 0.534081124;
goto REGDR1;
end;
_LP0 = 0.0;
_LP1 = 0.0;
_LP2 = 0.0;
_LP3 = 0.0;
_TEMP = 1.0;
_LP0 = _LP0 + (8.97309749884509) * _TEMP * _1_0;
_LP1 = _LP1 + (9.475456450304) * _TEMP * _1_0;
_LP2 = _LP2 + (0.08183779939133) * _TEMP * _1_0;
_LP3 = _LP3 + (7.91547642280949) * _TEMP * _1_0;
_LP0 = _LP0 + (-7.09311218652648) * _TEMP * _1_1;
_LP1 = _LP1 + (3.42946756538907) * _TEMP * _1_1;
_LP2 = _LP2 + (-1.63736222687037) * _TEMP * _1_1;
_LP3 = _LP3 + (8.60035492871607) * _TEMP * _1_1;
_LP0 = _LP0 + (16.3315840253036) * _TEMP * _1_2;
_LP1 = _LP1 + (-5.85959164693143) * _TEMP * _1_2;
_LP2 = _LP2 + (-2.53740928241609) * _TEMP * _1_2;
_LP3 = _LP3 + (2.62120809028614) * _TEMP * _1_2;
_LP0 = _LP0 + (-22.5615273556858) * _TEMP * _1_3;
_LP1 = _LP1 + (-5.52330111707437) * _TEMP * _1_3;
_LP2 = _LP2 + (-5.33919133360776) * _TEMP * _1_3;
_LP3 = _LP3 + (0.11884727866076) * _TEMP * _1_3;
_LP0 = _LP0 + (-30.2554906468364) * _TEMP * _1_4;
_LP1 = _LP1 + (0.64526397467362) * _TEMP * _1_4;
_LP2 = _LP2 + (-4.40987507627988) * _TEMP * _1_4;
_LP3 = _LP3 + (-1.46254346452609) * _TEMP * _1_4;
```

```
_LP0 = _LP0 + (13.4444067104834) * _TEMP * _1_5;
_LP1 = _LP1 + (-15.4359581659106) * _TEMP * _1_5;
_LP2 = _LP2 + (-3.78315830765155) * _TEMP * _1_5;
_LP3 = _LP3 + (-4.74730533646477) * _TEMP * _1_5;
_LP0 = _LP0 + (5.99426137980241) * _TEMP * _1_6;
_LP1 = _LP1 + (3.34304711000097) * _TEMP * _1_6;
_LP2 = _LP2 + (-4.49993737709991) * _TEMP * _1_6;
_LP3 = _LP3 + (0.39149662840319) * _TEMP * _1_6;
_LP0 = _LP0 + (8.00404660871621) * _TEMP * _1_7;
_LP1 = _LP1 + (3.87729351931859) * _TEMP * _1_7;
_LP2 = _LP2 + (-5.662863418933) * _TEMP * _1_7;
_LP3 = _LP3 + (0.92431512613497) * _TEMP * _1_7;
_LP0 = _LP0 + (9.73639514490121) * _TEMP * _1_8;
_LP1 = _LP1 + (1.66486268124882) * _TEMP * _1_8;
_LP2 = _LP2 + (-5.34790399310294) * _TEMP * _1_8;
_LP3 = _LP3 + (-0.80452936208339) * _TEMP * _1_8;
_LP0 = _LP0 + (-1.18886754533908) * _TEMP * _1_9;
_LP1 = _LP1 + (0.98108751722337) * _TEMP * _1_9;
_LP2 = _LP2 + (-5.09756573837529) * _TEMP * _1_9;
_LP3 = _LP3 + (0.55035390990751) * _TEMP * _1_9;
_LP0 = _LP0 + (3.33003316374041) * _TEMP * _1_10;
_LP1 = _LP1 + (1.28863079547562) * _TEMP * _1_10;
_LP2 = _LP2 + (4.52005620947533) * _TEMP * _1_10;
_LP3 = _LP3 + (-1.88185495205653) * _TEMP * _1_10;
_LP0 = _LP0 + (-1.23514061750629) * _TEMP * _1_11;
_LP1 = _LP1 + (-0.63165164315095) * _TEMP * _1_11;
_LP2 = _LP2 + (4.47980876228159) * _TEMP * _1_11;
_LP3 = _LP3 + (-2.28762571372038) * _TEMP * _1_11;
_LP0 = _LP0 + (3.45175998109795) * _TEMP * _1_12;
_LP1 = _LP1 + (-0.05911640263949) * _TEMP * _1_12;
_LP2 = _LP2 + (3.7133976012504) * _TEMP * _1_12;
_LP3 = _LP3 + (-3.40533163917284) * _TEMP * _1_12;
_LP0 = _LP0 + (-1.79579379752335) * _TEMP * _1_13;
_LP1 = _LP1 + (-0.66575638518718) * _TEMP * _1_13;
_LP2 = _LP2 + (2.46190197688312) * _TEMP * _1_13;
_LP3 = _LP3 + (-3.86144993561858) * _TEMP * _1_13;
_LP0 = _LP0 + (16.2289623285747) * _TEMP * _1_14;
_LP1 = _LP1 + (3.87844062530087) * _TEMP * _1_14;
_LP2 = _LP2 + (13.5342255495752) * _TEMP * _1_14;
_LP3 = _LP3 + (0.11787447033604) * _TEMP * _1_14;
_TEMP = 1.0;
_LP0 = _LP0 + (4.96178727582277) * _TEMP * _2_0;
_LP1 = _LP1 + (7.19423934264755) * _TEMP * _2_0;
_LP2 = _LP2 + (-2.57814751000107) * _TEMP * _2_0;
_LP3 = _LP3 + (-0.41318251862093) * _TEMP * _2_0;
_LP0 = _LP0 + (2.53606187215301) * _TEMP * _2_1;
_LP1 = _LP1 + (1.02456723195019) * _TEMP * _2_1;
_LP2 = _LP2 + (-3.01518942636817) * _TEMP * _2_1;
_LP3 = _LP3 + (-6.42999803474578) * _TEMP * _2_1;
_LP0 = _LP0 + (-17.0716556901489) * _TEMP * _2_2;
_LP1 = _LP1 + (-2.55836176487159) * _TEMP * _2_2;
_LP2 = _LP2 + (-2.66986765613004) * _TEMP * _2_2;
_LP3 = _LP3 + (-3.77427590976266) * _TEMP * _2_2;
_LP0 = _LP0 + (-11.6228431594003) * _TEMP * _2_3;
_LP1 = _LP1 + (-4.42118648129498) * _TEMP * _2_3;
_LP2 = _LP2 + (-2.41006554535669) * _TEMP * _2_3;
```

```
_LP3 = _LP3 + (-2.47998713977501) * _TEMP * _2_3;
_LP0 = _LP0 + (-6.65446334079067) * _TEMP * _2_4;
_LP1 = _LP1 + (-4.21089586391698) * _TEMP * _2_4;
_LP2 = _LP2 + (-2.40850931862971) * _TEMP * _2_4;
_LP3 = _LP3 + (-2.46504190674716) * _TEMP * _2_4;
_LP0 = _LP0 + (-3.17136687316047) * _TEMP * _2_5;
_LP1 = _LP1 + (-1.69998112881134) * _TEMP * _2_5;
_LP2 = _LP2 + (-2.27711189809608) * _TEMP * _2_5;
_LP3 = _LP3 + (-0.56541361679043) * _TEMP * _2_5;
_LP0 = _LP0 + (0.06485838750697) * _TEMP * _2_6;
_LP1 = _LP1 + (2.083825423476) * _TEMP * _2_6;
_LP2 = _LP2 + (4.31755671819224) * _TEMP * _2_6;
_LP3 = _LP3 + (4.36618153369848) * _TEMP * _2_6;
_LP0 = _LP0 + (-3.15969642288067) * _TEMP * _2_7;
_LP1 = _LP1 + (-3.11663563731206) * _TEMP * _2_7;
_LP2 = _LP2 + (-1.93101189518423) * _TEMP * _2_7;
_LP3 = _LP3 + (-0.66813727595772) * _TEMP * _2_7;
_LP0 = _LP0 + (23.3492198306386) * _TEMP * _2_8;
_LP1 = _LP1 + (15.3692429684277) * _TEMP * _2_8;
_LP2 = _LP2 + (19.6299281653522) * _TEMP * _2_8;
_LP3 = _LP3 + (3.7767067535256) * _TEMP * _2_8;
_LP0 = _LP0 + (0.6888846491937) * _TEMP * _2_9;
_LP1 = _LP1 + (0.26881516596812) * _TEMP * _2_9;
_LP2 = _LP2 + (3.49366097063402) * _TEMP * _2_9;
_LP3 = _LP3 + (4.77521196924485) * _TEMP * _2_9;
_LP0 = _LP0 + (6.93832447370645) * _TEMP * _2_10;
_LP1 = _LP1 + (-14.7995817386477) * _TEMP * _2_10;
_LP2 = _LP2 + (-3.17802481741923) * _TEMP * _2_10;
_LP3 = _LP3 + (-0.75953335528334) * _TEMP * _2_10;
_LP0 = _LP0 + (-5.17421740905568) * _TEMP * _2_11;
_LP1 = _LP1 + (-3.50927803184578) * _TEMP * _2_11;
_LP2 = _LP2 + (-0.93991965967767) * _TEMP * _2_11;
_LP3 = _LP3 + (-0.57578867183536) * _TEMP * _2_11;
_LP0 = _LP0 + (-5.40485675039647) * _TEMP * _2_12;
_LP1 = _LP1 + (-3.43007109867235) * _TEMP * _2_12;
_LP2 = _LP2 + (-11.8686117799293) * _TEMP * _2_12;
_LP3 = _LP3 + (-0.57409656273319) * _TEMP * _2_12;
_TEMP = 1.0;
_LP0 = _LP0 + (42.0263556916437) * _TEMP * _3_0;
_LP1 = _LP1 + (1.55172177304255) * _TEMP * _3_0;
_LP2 = _LP2 + (10.9123737543277) * _TEMP * _3_0;
_LP3 = _LP3 + (2.20643367366059) * _TEMP * _3_0;
_LP0 = _LP0 + (35.8542164366111) * _TEMP * _3_1;
_LP1 = _LP1 + (-7.03832251333459) * _TEMP * _3_1;
_LP2 = _LP2 + (-13.5692536842049) * _TEMP * _3_1;
_LP3 = _LP3 + (-11.6512021486838) * _TEMP * _3_1;
_LP0 = _LP0 + (47.2300160154457) * _TEMP * _3_2;
_LP1 = _LP1 + (5.43079775823532) * _TEMP * _3_2;
_LP2 = _LP2 + (-1.76238042211005) * _TEMP * _3_2;
_LP3 = _LP3 + (2.88051687962657) * _TEMP * _3_2;
_LP0 = _LP0 + (32.2801944616028) * _TEMP * _3_3;
_LP1 = _LP1 + (5.10935792540826) * _TEMP * _3_3;
_LP2 = _LP2 + (2.52744460733309) * _TEMP * _3_3;
_LP3 = _LP3 + (2.95088205442946) * _TEMP * _3_3;
_LP0 = _LP0 + (31.6597113950015) * _TEMP * _3_4;
_LP1 = _LP1 + (-9.07258866978128) * _TEMP * _3_4;
```

```
_LP2 = _LP2 + (1.62190948241675) * _TEMP * _3_4;
_LP3 = _LP3 + (2.04551962977074) * _TEMP * _3_4;
_LP0 = _LP0 + (31.6597116255105) * _TEMP * _3_5;
_LP1 = _LP1 + (2.67824698076013) * _TEMP * _3_5;
_LP2 = _LP2 + (1.62190948383178) * _TEMP * _3_5;
_LP3 = _LP3 + (1.19293530007666) * _TEMP * _3_5;
_LP0 = _LP0 + (31.6597116340262) * _TEMP * _3_6;
_LP1 = _LP1 + (2.54298742758522) * _TEMP * _3_6;
_LP2 = _LP2 + (1.62190948388826) * _TEMP * _3_6;
_LP3 = _LP3 + (1.30825513024406) * _TEMP * _3_6;
_LP0 = _LP0 + (-362.950916427088) * _TEMP * _3_7;
_LP1 = _LP1 + (6.17176825281735) * _TEMP * _3_7;
_LP2 = _LP2 + (2.29729057331607) * _TEMP * _3_7;
_LP3 = _LP3 + (1.72970564346861) * _TEMP * _3_7;
_LP0 = _LP0 + (15.9700734501859) * _TEMP * _3_8;
_LP1 = _LP1 + (-9.54799929498259) * _TEMP * _3_8;
_LP2 = _LP2 + (-9.74287510861865) * _TEMP * _3_8;
_LP3 = _LP3 + (1.95662231341111) * _TEMP * _3_8;
_LP0 = _LP0 + (31.6597115840211) * _TEMP * _3_9;
_LP1 = _LP1 + (-9.0725886711641) * _TEMP * _3_9;
_LP2 = _LP2 + (1.62190948358845) * _TEMP * _3_9;
_LP3 = _LP3 + (2.04551963042141) * _TEMP * _3_9;
_LP0 = _LP0 + (31.291502511214) * _TEMP * _3_10;
_LP1 = _LP1 + (20.319207702854) * _TEMP * _3_10;
_LP2 = _LP2 + (1.22785286240863) * _TEMP * _3_10;
_LP3 = _LP3 + (-8.71070773697709) * _TEMP * _3_10;
_TEMP = 1.0;
_LP0 = _LP0 + (39.0432493014866) * _TEMP * _5_0;
_LP1 = _LP1 + (2.41556930669061) * _TEMP * _5_0;
_LP2 = _LP2 + (10.9819053439207) * _TEMP * _5_0;
_LP3 = _LP3 + (-2.4193090445841) * _TEMP * _5_0;
_LP0 = _LP0 + (26.0525989318919) * _TEMP * _5_1;
_LP1 = _LP1 + (-10.8013995852177) * _TEMP * _5_1;
_LP2 = _LP2 + (7.80802468659326) * _TEMP * _5_1;
_LP3 = _LP3 + (-8.37335359162762) * _TEMP * _5_1;
_LP0 = _LP0 + (-91.7996367657177) * _TEMP * _5_2;
_LP1 = _LP1 + (-7.20941847531768) * _TEMP * _5_2;
_LP2 = _LP2 + (6.37205506985912) * _TEMP * _5_2;
_LP3 = _LP3 + (-4.13523264892108) * _TEMP * _5_2;
_LP0 = _LP0 + (-43.2987854849329) * _TEMP * _5_3;
_LP1 = _LP1 + (9.63628678654799) * _TEMP * _5_3;
_LP2 = _LP2 + (15.2260866612625) * _TEMP * _5_3;
_LP3 = _LP3 + (3.41098536758909) * _TEMP * _5_3;
_LP0 = _LP0 + (-92.5078418147566) * _TEMP * _5_4;
_LP1 = _LP1 + (0.92035946274589) * _TEMP * _5_4;
_LP2 = _LP2 + (14.6028124613418) * _TEMP * _5_4;
_LP3 = _LP3 + (4.74556696940043) * _TEMP * _5_4;
_LP0 = _LP0 + (-169.198537792928) * _TEMP * _5_5;
_LP1 = _LP1 + (17.5135430652249) * _TEMP * _5_5;
_LP2 = _LP2 + (-27.5413368656283) * _TEMP * _5_5;
_LP3 = _LP3 + (5.71011491340335) * _TEMP * _5_5;
_LP0 = _LP0 + (29.0429678675398) * _TEMP * _5_6;
_LP1 = _LP1 + (-4.70698581451379) * _TEMP * _5_6;
_LP2 = _LP2 + (2.19747568966552) * _TEMP * _5_6;
_LP3 = _LP3 + (0.25036394861618) * _TEMP * _5_6;
_LP0 = _LP0 + (27.4220001532713) * _TEMP * _5_7;
```

```
_LP1 = _LP1 + (-5.62951270960282) * _TEMP * _5_7;
_LP2 = _LP2 + (2.97946845585617) * _TEMP * _5_7;
_LP3 = _LP3 + (0.07300025078033) * _TEMP * _5_7;
_LP0 = _LP0 + (24.9838671156593) * _TEMP * _5_8;
_LP1 = _LP1 + (-4.23916148505361) * _TEMP * _5_8;
_LP2 = _LP2 + (3.42557523365742) * _TEMP * _5_8;
_LP3 = _LP3 + (1.46388562797025) * _TEMP * _5_8;
_LP0 = _LP0 + (22.8194752422965) * _TEMP * _5_9;
_LP1 = _LP1 + (-4.25224375283395) * _TEMP * _5_9;
_LP2 = _LP2 + (2.49905210556025) * _TEMP * _5_9;
_LP3 = _LP3 + (-0.01709833699071) * _TEMP * _5_9;
_LP0 = _LP0 + (37.114213383863) * _TEMP * _5_10;
_LP1 = _LP1 + (2.9953971574379) * _TEMP * _5_10;
_LP2 = _LP2 + (-4.63754693643679) * _TEMP * _5_10;
_LP3 = _LP3 + (-4.36468726526216) * _TEMP * _5_10;
_LP0 = _LP0 + (34.2320056651284) * _TEMP * _5_11;
_LP1 = _LP1 + (-2.48152127510367) * _TEMP * _5_11;
_LP2 = _LP2 + (-7.20881969172312) * _TEMP * _5_11;
_LP3 = _LP3 + (2.05199646600986) * _TEMP * _5_11;
_LP0 = _LP0 + (34.1979425371632) * _TEMP * _5_12;
_LP1 = _LP1 + (1.32583179116639) * _TEMP * _5_12;
_LP2 = _LP2 + (-1.94011877303868) * _TEMP * _5_12;
_LP3 = _LP3 + (8.74058490108554) * _TEMP * _5_12;
_LP0 = _LP0 + (39.1512435469843) * _TEMP * _5_13;
_LP1 = _LP1 + (1.88577792759584) * _TEMP * _5_13;
_LP2 = _LP2 + (-1.93386166738385) * _TEMP * _5_13;
_LP3 = _LP3 + (0.84886002004651) * _TEMP * _5_13;
_LP0 = _LP0 + (20.9363766085136) * _TEMP * _5_14;
_LP1 = _LP1 + (-2.0647251475618) * _TEMP * _5_14;
_LP2 = _LP2 + (-13.1892422255085) * _TEMP * _5_14;
_LP3 = _LP3 + (-4.52842188369726) * _TEMP * _5_14;
_TEMP = 1.0;
_LP0 = _LP0 + (1.76663561037174) * _TEMP * _6_0;
_LP1 = _LP1 + (-5.40874215787948) * _TEMP * _6_0;
_LP2 = _LP2 + (-6.87281360284862) * _TEMP * _6_0;
_LP3 = _LP3 + (-6.22229997982126) * _TEMP * _6_0;
_LP0 = _LP0 + (21.8797726373068) * _TEMP * _6_1;
_LP1 = _LP1 + (2.87906958740983) * _TEMP * _6_1;
_LP2 = _LP2 + (1.83666665646742) * _TEMP * _6_1;
_LP3 = _LP3 + (4.13135987011355) * _TEMP * _6_1;
_LP0 = _LP0 + (1.73459041116589) * _TEMP * _6_2;
_LP1 = _LP1 + (-0.75352434519744) * _TEMP * _6_2;
_LP2 = _LP2 + (-0.62400019216188) * _TEMP * _6_2;
_LP3 = _LP3 + (0.53569098310408) * _TEMP * _6_2;
_TEMP = 1.0;
_LP0 = _LP0 + (-3.44927846183227) * _TEMP * _7_0;
_LP1 = _LP1 + (-6.37652016665453) * _TEMP * _7_0;
_LP2 = _LP2 + (-4.25904939215537) * _TEMP * _7_0;
_LP3 = _LP3 + (-4.51685639332432) * _TEMP * _7_0;
_LP0 = _LP0 + (-6.43408008433648) * _TEMP * _7_1;
_LP1 = _LP1 + (-0.80236520705753) * _TEMP * _7_1;
_LP2 = _LP2 + (-0.12922463272966) * _TEMP * _7_1;
_LP3 = _LP3 + (-0.63228249961139) * _TEMP * _7_1;
_LPMAX = 0.0;
_LP0 = -123.067467124716 + _LP0;
if _LPMAX < _LP0 then _LPMAX = _LP0;
```

```
_LP1 = -23.6221258810818 + _LP1;
if _LPMAX < _LP1 then _LPMAX = _LP1;
_LP2 = -18.5909979689337 + _LP2;
if _LPMAX < _LP2 then _LPMAX = _LP2;
_LP3 = -6.00322742797283 + _LP3;
if _LPMAX < _LP3 then _LPMAX = _LP3;
_LP0 = EXP(_LP0 - _LPMAX);
_LP1 = EXP(_LP1 - _LPMAX);
_LP2 = EXP(_LP2 - _LPMAX);
_LP3 = EXP(_LP3 - _LPMAX);
_LPMAX = EXP(-_LPMAX);
_P4 = 1.0 / (_LPMAX + _LP0 + _LP1 + _LP2 + _LP3);
_P0 = _LP0 * _P4;
_P1 = _LP1 * _P4;
_P2 = _LP2 * _P4;
_P3 = _LP3 * _P4;
_P4 = _LPMAX * _P4;
REGDR1: P_ATTACKU2R = _P0;
_MAXP = _P0;
_IY = 1.0;
P_ATTACKR2L = _P1;
if (_P1 - _MAXP > 1E-8) then do ;
_MAXP = _P1;
_IY = 2.0;
end;
P_ATTACKPROBE = _P2;
if (_P2 - _MAXP > 1E-8) then do ;
_MAXP = _P2;
_IY = 3.0;
end;
P_ATTACKNORMAL = _P3;
if (_P3 - _MAXP > 1E-8) then do ;
_MAXP = _P3;
_IY = 4.0;
end;
P_ATTACKDOS = _P4;
if (_P4 - _MAXP > 1E-8) then do ;
_MAXP = _P4;
_IY = 5.0;
end;
I_ATTACK = REGDRF[_IY];
U_ATTACK = REGDRU[_IY];
EM_EVENTPROBABILITY = P_ATTACKU2R;
EM_PROBABILITY = MAX(P_ATTACKU2R, P_ATTACKR2L, P_ATTACKPROBE, P_ATTACKNORMAL,
P_ATTACKDOS);
EM_CLASSIFICATION = I_ATTACK;
_return: ;
end;
 enddata;
```

Example of an Input and Output Variables Scoring File

Here is an example of an XML file that contains information about input and output variables. For more information, see "Files Used in Model Publishing" on page 55.

```
<?xml version="1.0" encoding="utf-8"?>
<Score>
   <Producer>
      <Name> SAS Enterprise Miner </Name>
      <Version> 1.0 </Version>
   </Producer>
   <TargetList>
   </TargetList>
   <Input>
      <Variable>
         <Name> COUNT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> DIF_SRVR </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[diff_srv_rate]]>
         </Description>
      </Variable>
      <Variable>
         <Name> FLAG </Name>
         <Type> character </Type>
      </Variable>
      <Variable>
         <Name> HOT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> SAM_SRAT </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[same_srv_rate]]>
         </Description>
      </Variable>
      <Variable>
         <Name> SERVICE </Name>
         <Type> character </Type>
      </Variable>
      <Variable>
         <Name> SRV_CNT </Name>
         <Type> numeric </Type>
         <Description>
```

```
            <![CDATA[srv_count]]>
         </Description>
      </Variable>
   </Input>
   <Output>
      <Variable>
         <Name> AOV16_COUNT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_DIF_SRVR </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_HOT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_SAM_SRAT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> AOV16_SRV_CNT </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> EM_CLASSIFICATION </Name>
         <Type> character </Type>
         <Description>
            <![CDATA[Prediction for ATTACK]]>
         </Description>
      </Variable>
      <Variable>
         <Name> EM_EVENTPROBABILITY </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[Probability for level U2R of ATTACK]]>
         </Description>
      </Variable>
      <Variable>
         <Name> EM_PROBABILITY </Name>
         <Type> numeric </Type>
         <Description>
            <![CDATA[Probability of Classification]]>
         </Description>
      </Variable>
      <Variable>
         <Name> G_FLAG </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> G_SERVICE </Name>
         <Type> numeric </Type>
      </Variable>
      <Variable>
         <Name> I_ATTACK </Name>
```

```
            <Type> character </Type>
            <Description>
               <![CDATA[Into: ATTACK]]>
            </Description>
         </Variable>
         <Variable>
            <Name> P_ATTACKDOS </Name>
            <Type> numeric </Type>
            <Description>
               <![CDATA[Predicted: ATTACK=dos]]>
            </Description>
         </Variable>
         <Variable>
            <Name> P_ATTACKNORMAL </Name>
            <Type> numeric </Type>
            <Description>
               <![CDATA[Predicted: ATTACK=normal]]>
            </Description>
         </Variable>
         <Variable>
            <Name> P_ATTACKPROBE </Name>
            <Type> numeric </Type>
            <Description>
               <![CDATA[Predicted: ATTACK=probe]]>
            </Description>
         </Variable>
         <Variable>
            <Name> P_ATTACKR2L </Name>
            <Type> numeric </Type>
            <Description>
               <![CDATA[Predicted: ATTACK=r2l]]>
            </Description>
         </Variable>
         <Variable>
            <Name> P_ATTACKU2R </Name>
            <Type> numeric </Type>
            <Description>
               <![CDATA[Predicted: ATTACK=u2r]]>
            </Description>
         </Variable>
         <Variable>
            <Name> U_ATTACK </Name>
            <Type> character </Type>
            <Description>
               <![CDATA[Unnormalized Into: ATTACK]]>
            </Description>
         </Variable>
         <Variable>
            <Name> _WARN_ </Name>
            <Type> character </Type>
            <Description>
               <![CDATA[Warnings]]>
            </Description>
         </Variable>
      </Output>
      <C>
```

```
<Function>
   <Name>
      score
   </Name>
   <ParameterList>
      <Parameter>
         <Array length="7">
            <Type>
               Parm
            </Type>
            <DataMap>
               <Element index="0">
                  <Value>
                     <Origin> COUNT </Origin>
                     <Type> double </Type>
                  </Value>
               </Element>
               <Element index="1">
                  <Value>
                     <Origin> DIF_SRVR </Origin>
                     <Type> double </Type>
                  </Value>
               </Element>
               <Element index="2">
                  <Value>
                     <Origin> FLAG </Origin>
                     <Array length="33">
                       <Type> char </Type>
                     </Array>
                  </Value>
               </Element>
               <Element index="3">
                  <Value>
                     <Origin> HOT </Origin>
                     <Type> double </Type>
                  </Value>
               </Element>
               <Element index="4">
                  <Value>
                     <Origin> SAM_SRAT </Origin>
                     <Type> double </Type>
                  </Value>
               </Element>
               <Element index="5">
                  <Value>
                     <Origin> SERVICE </Origin>
                     <Array length="33">
                       <Type> char </Type>
                     </Array>
                  </Value>
               </Element>
               <Element index="6">
                  <Value>
                     <Origin> SRV_CNT </Origin>
                     <Type> double </Type>
                  </Value>
```

```
            </Element>
         </DataMap>
      </Array>
   </Parameter>

   <Parameter>
      <Array length="18">
         <Type>
            Parm
         </Type>
         <DataMap>
            <Element index="0">
               <Value>
                  <Origin> AOV16_COUNT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="1">
               <Value>
                  <Origin> AOV16_DIF_SRVR </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="2">
               <Value>
                  <Origin> AOV16_HOT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="3">
               <Value>
                  <Origin> AOV16_SAM_SRAT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="4">
               <Value>
                  <Origin> AOV16_SRV_CNT </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="5">
               <Value>
                  <Origin> EM_CLASSIFICATION </Origin>
                  <Array length="33">
                    <Type> char </Type>
                  </Array>
               </Value>
            </Element>
            <Element index="6">
               <Value>
                  <Origin> EM_EVENTPROBABILITY </Origin>
                  <Type> double </Type>
               </Value>
            </Element>
            <Element index="7">
```

```
   <Value>
      <Origin> EM_PROBABILITY </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="8">
   <Value>
      <Origin> G_FLAG </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="9">
   <Value>
      <Origin> G_SERVICE </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="10">
   <Value>
      <Origin> I_ATTACK </Origin>
      <Array length="7">
        <Type> char </Type>
     </Array>
   </Value>
</Element>
<Element index="11">
   <Value>
      <Origin> P_ATTACKDOS </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="12">
   <Value>
      <Origin> P_ATTACKNORMAL </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="13">
   <Value>
      <Origin> P_ATTACKPROBE </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="14">
   <Value>
      <Origin> P_ATTACKR2L </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="15">
   <Value>
      <Origin> P_ATTACKU2R </Origin>
      <Type> double </Type>
   </Value>
</Element>
<Element index="16">
```

```
                        <Value>
                           <Origin> U_ATTACK </Origin>
                           <Array length="7">
                             <Type> char </Type>
                           </Array>
                        </Value>
                     </Element>
                     <Element index="17">
                        <Value>
                           <Origin> _WARN_ </Origin>
                           <Array length="5">
                             <Type> char </Type>
                           </Array>
                        </Value>
                     </Element>
                  </DataMap>
               </Array>
            </Parameter>
         </ParameterList>
      </Function>
   </C>
</Score>
```

Example of a User-Defined Formats Scoring File

Here is an example of an XML file that contains user-defined formats. For more information, see "Files Used in Model Publishing" on page 55.

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="SUVformats.xsl"?>
<LIBRARY type="EXPORT" version="SUV">
   <HEADER>
      <Provider>SAS Institute Inc.</Provider>
      <Version>9.2</Version>
      <VersionLong>9.02.02M2D09012009</VersionLong>
      <CreationDateTime>2009-12-14T12:47:03</CreationDateTime>
   </HEADER>

   <TABLE name="sasscore_score_ufmt">
      <TABLE-HEADER>
         <Provider>SAS Institute Inc.</Provider>
         <Version>9.2</Version>
         <VersionLong>9.02.02M2D09012009</VersionLong>
         <CreationDateTime>2009-12-14T12:47:03</CreationDateTime>
         <ModifiedDateTime>2009-12-14T12:47:03</ModifiedDateTime>

         <Protection />
         <DataSetType />
```

```
   <DataRepresentation />
   <Encoding>utf-8</Encoding>
   <ReleaseCreated />
   <HostCreated />
   <FileName>sasscore_score_ufmt</FileName>

   <Observations />
   <Compression number="1" />
   <Variables number="21" />
</TABLE-HEADER>

<COLUMN name="FMTNAME" label="Format name">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>32</LENGTH>
   <Offset>32</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="START" label="Starting value for format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>16</LENGTH>
   <Offset>16</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="END" label="Ending value for format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>16</LENGTH>
   <Offset>16</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="LABEL" label="Format value label">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MIN" label="Minimum length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MAX" label="Maximum length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
```

```
   <SortedBy />
</COLUMN>

<COLUMN name="DEFAULT" label="Default length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="LENGTH" label="Format length">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="FUZZ" label="Fuzz value">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="PREFIX" label="Prefix characters">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>2</LENGTH>
   <Offset>2</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="MULT" label="Multiplier">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="FILL" label="Fill character">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="NOEDIT" label="Is picture string noedit?">
   <TYPE>numeric</TYPE>
   <DATATYPE>double</DATATYPE>
   <LENGTH>3</LENGTH>
   <Offset>3</Offset>
```

```
   <SortedBy />
</COLUMN>

<COLUMN name="TYPE" label="Type of format">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="SEXCL" label="Start exclusion">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="EEXCL" label="End exclusion">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="HLO" label="Additional information">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>11</LENGTH>
   <Offset>11</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DECSEP" label="Decimal separator">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DIG3SEP" label="Three-digit separator">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>1</LENGTH>
   <Offset>1</Offset>
   <SortedBy />
</COLUMN>

<COLUMN name="DATATYPE" label="Date/time/datetime?">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
```

```
   <SortedBy />
</COLUMN>

<COLUMN name="LANGUAGE" label="Language for date strings">
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>8</LENGTH>
   <Offset>8</Offset>
   <SortedBy />
</COLUMN>

<ROW>
   <FMTNAME missing=" " />
   <START missing=" " />
   <END missing=" " />
   <LABEL missing=" " />
   <MIN missing=" " />
   <MAX missing=" " />
   <DEFAULT missing=" " />
   <LENGTH missing=" " />
   <FUZZ missing=" " />
   <PREFIX missing=" " />
   <MULT missing=" " />
   <FILL missing=" " />
   <NOEDIT missing=" " />
   <TYPE missing=" " />
   <SEXCL missing=" " />
   <EEXCL missing=" " />
   <HLO missing=" " />
   <DECSEP missing=" " />
   <DIG3SEP missing=" " />
   <DATATYPE missing=" " />
   <LANGUAGE missing=" " />
</ROW>

<ROW>
   <DELTA-RECORD key="ABC" />
   <FMTNAME>ABC</FMTNAME>
   <START>1</START>
   <END>1</END>
   <LABEL>yes</LABEL>
   <MIN>1</MIN>
   <MAX>40</MAX>
   <DEFAULT>3</DEFAULT>
   <LENGTH>3</LENGTH>
   <FUZZ>1E-12</FUZZ>
   <PREFIX missing=" " />
   <MULT>0</MULT>
   <FILL missing=" " />
   <NOEDIT>0</NOEDIT>
   <TYPE>N</TYPE>
   <SEXCL>N</SEXCL>
   <EEXCL>N</EEXCL>
   <HLO missing=" " />
   <DECSEP missing=" " />
   <DIG3SEP missing=" " />
```

```
            <DATATYPE missing=" " />
            <LANGUAGE missing=" " />
         </ROW>

         <ROW>
            <DELTA-RECORD key="YESNO" />
            <FMTNAME>YESNO</FMTNAME>
            <START>0</START>
            <END>0</END>
            <LABEL>NO</LABEL>
            <MIN>1</MIN>
            <MAX>40</MAX>
            <DEFAULT>3</DEFAULT>
            <LENGTH>3</LENGTH>
            <FUZZ>0</FUZZ>
            <PREFIX missing=" " />
            <MULT>0</MULT>
            <FILL missing=" " />
            <NOEDIT>0</NOEDIT>
            <TYPE>C</TYPE>
            <SEXCL>N</SEXCL>
            <EEXCL>N</EEXCL>
            <HLO missing=" " />
            <DECSEP missing=" " />
            <DIG3SEP missing=" " />
            <DATATYPE missing=" " />
            <LANGUAGE missing=" " />
         </ROW>

         <ROW>
            <FMTNAME>YESNO</FMTNAME>
            <START>1</START>
            <END>1</END>
            <LABEL>YES</LABEL>
            <MIN>1</MIN>
            <MAX>40</MAX>
            <DEFAULT>3</DEFAULT>
            <LENGTH>3</LENGTH>
            <FUZZ>0</FUZZ>
            <PREFIX missing=" " />
            <MULT>0</MULT>
            <FILL missing=" " />
            <NOEDIT>0</NOEDIT>
            <TYPE>C</TYPE>
            <SEXCL>N</SEXCL>
            <EEXCL>N</EEXCL>
            <HLO missing=" " />
            <DECSEP missing=" " />
            <DIG3SEP missing=" " />
            <DATATYPE missing=" " />
            <LANGUAGE missing=" " />
         </ROW>
      </TABLE>
   </LIBRARY>
```