Contents

PART 1  General Reference  1

Chapter 1  /  LIBNAME Options for Nonrelational Databases  3
   LIBNAME Options for Nonrelational Database Interfaces  4
   Dictionary  4

Chapter 2  /  Data Set Options for Nonrelational Databases  41
   Data Set Options for Nonrelational Database Interfaces  42
   Dictionary  42

Chapter 3  /  System Options for Nonrelational Databases  79
   Introduction to System Options for Nonrelational Databases  79
   System Options for Nonrelational Databases  79
   Using the TIMEZONE= System Option  80

PART 2  Nonrelational DBMS Interfaces for SAS/ACCESS  83

Chapter 4  /  SAS/ACCESS Interface to MongoDB  85
   About SAS/ACCESS Interface to MongoDB  86
   Supported Features for SAS/ACCESS Interface to MongoDB  86
   LIBNAME Statement Specifics for SAS/ACCESS Interface to MongoDB  87
   Data Set Options for SAS/ACCESS Interface to MongoDB  92
   Working with Schemas for MongoDB Data  92
   Using the SAS Client to Query, Create, Update, and Delete Data in MongoDB  97
   SQL Pass-Through Specifics for MongoDB  106
   Using MongoDB Commands to Query, Create, Update, and Delete Tables in MongoDB  107
   Passing SAS Functions to MongoDB  111
   Data Types for MongoDB  111

Chapter 5  /  SAS/ACCESS Interface to the PI System  115
   PI System  115
   Using the PI System Data Archive  125
   Using the PI System Asset Framework  146

Chapter 6  /  SAS/ACCESS Interface to Salesforce  163
   About SAS/ACCESS Interface to Salesforce  163
   Supported Features for SAS/ACCESS Interface to Salesforce  164
   LIBNAME Statement for the Salesforce Engine  165
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Options for Salesforce</td>
<td>167</td>
</tr>
<tr>
<td>Known Limitation When Working with SAS/ACCESS Interface to Salesforce</td>
<td>168</td>
</tr>
<tr>
<td>SQL Pass-Through Facility Specifics for Salesforce</td>
<td>168</td>
</tr>
<tr>
<td>Passing SAS Functions to Salesforce</td>
<td>168</td>
</tr>
<tr>
<td>Data Types for Salesforce</td>
<td>169</td>
</tr>
</tbody>
</table>
What’s New in SAS/ACCESS for Nonrelational Databases

Overview

In the April 2019 release of SAS/ACCESS, this book (SAS/ACCESS for Nonrelational Databases: Reference) replaces the book that contained information for the PI System Interface only (SAS/ACCESS Interface to the PI System: Reference). This book contains information for the following interfaces:

- MongoDB
- PI System
- Salesforce

SAS/ACCESS Interface to MongoDB

In the April 2020 release of SAS/ACCESS on SAS 9.4 M6 and SAS Viya 3.5, support was added to the following features or enhancements:

- Schemas are now stored to schema files by default. If you do not specify the SCHEMA_COLLECTION= LIBNAME option, the default value that is used is sas-default-schema. To work in a temporary (in-memory) schema, specify SCHEMA_COLLECTION='$temp$'. Specifying a descriptive name using SCHEMA_COLLECTION= is still recommended.

- String values that are formatted as dates or dates and times are now read into SAS as DATETIME values. Several common date-time formats and variations are supported.

- Documentation updates. Clarification or additional information was added to the following topics:
  - Clarification was added about how to update stored schemas and when SAS automatically updates stored schemas. See “Update a Stored Schema” on page 96.
  - Information was added about data constraints that are not supported for columns when you create tables in MongoDB. See “Creating Tables” on page 98.
Information was added about special columns for 1:1 and 1:n child tables. See “Special Columns for 1:1 Child Tables” on page 100 and “Special Columns for 1:n Child Tables” on page 100.

Support for SAS/ACCESS Interface to MongoDB was added in the April 2019 release for SAS/ACCESS.

In SAS Viya 3.5, support was added for the following features and enhancements:

- a data connector that transfers data between MongoDB and CAS
- ability to Write, Update, and Delete data from tables in MongoDB

---

**SAS/ACCESS Interface to the PI System**

SAS/ACCESS Interface to the PI System was originally released with SAS 9.4M2. This release included support for the Data Archive in the PI System.

The February 2016 release for SAS/ACCESS includes the following features and enhancements:

- Support is added for the Asset Framework component of the PI System.
- Support for event frames. Event frames are supported as part of the PI System Asset Framework.
- Support on UNIX platforms is added. Supported UNIX platforms include AIX, HP-UX for Itanium, Linux for x64, Solaris for SPARC, and Solaris for x64. For more information, see “Supported Features by Host for SAS/ACCESS Interface to the PI System” on page 117.

In the SAS 9.4M4 release for SAS/ACCESS, support is added for the following features and enhancements:

- Picomp_Summary virtual table for Microsoft Windows environments. This table enables calculation of summary statistics over a specified time span. This table is part of the PI System Data Archive.
- New data set options:
  - CALCULATION_BASIS= This option is used with the Picomp_Summary table.
  - LABEL= This data set option is an alias for the DESCRIPTION= data set option.
  - MEDIAN= This option is used with the Picomp_Summary virtual table.
  - MIXED= This option is used with the Picomp virtual table.
  - SHOWINDEX= This option is used with the Picomp virtual table.
  - TAGLIST= This option is used with the Picomp and Picomp_Summary virtual tables.

Note: Calculation of median statistics is not supported when you access the PI System using the PI Web API.
In the SAS 9.4M6 release for SAS/ACCESS, support was added for the following features and enhancements:

- SAS_PI_WEB_AUTH= environment variable for use with UNIX platforms. This environment variable specifies the authentication method that should be used when you connect to the PI System from a UNIX platform. Specifying the authentication method can reduce the amount of time it takes to connect to the PI System.

- Picomp_Summary table on UNIX environments. Calculation of median statistics is not supported on UNIX environments, because these environments use the PI Web API to access PI System data.

In the SAS 9.4M7 release for SAS/ACCESS, the following changes and enhancements were made:

- Attribute values up to 32767 bytes are supported.

- The SAS_PI_WEB_AUTH= environment variable for UNIX now supports the BASIC value, which enables authentication using the user name and password.

- The syntax to access PI System tags in the Asset Framework has changed.

---

SAS/ACCESS Interface to Salesforce

Support for SAS/ACCESS Interface to Salesforce is added in the April 2019 release of SAS/ACCESS.

In SAS Viya 3.5, support for these features and enhancements is added.

- a data connector that transfers data between Salesforce and CAS

- ability to Write and Delete data in Salesforce

In the May 2020 release of SAS/ACCESS for SAS 9.4M6 and SAS Viya 3.5, support for these features and enhancements has been added.

- new proxy options to connect and allow access to Salesforce using a proxy server: PROXY_HOST=, PROXY_PASS=, PROXY_PORT=, PROXY_USER=

- support for the USE_NATIVE_NAMES= LIBNAME option.
LIBNAME Options for Nonrelational Databases

**Dictionary**

- API_TRACE= LIBNAME Statement Option .............................................. 4
- AUTHDOMAIN= LIBNAME Statement Option .......................................... 5
- AUTHENDPOINT= LIBNAME Statement Option ....................................... 6
- CONNECTION= LIBNAME Statement Option .......................................... 7
- DBGEN_NAME= LIBNAME Statement Option ......................................... 10
- DBMAX_TEXT= LIBNAME Statement Option .......................................... 11
- DBPROMPT= LIBNAME Statement Option ........................................... 12
- DBSASLABEL= LIBNAME Statement Option .......................................... 13
- DEFER= LIBNAME Statement Option ................................................ 14
- HIDEFLAGS= LIBNAME Statement Option .......................................... 15
- HIDEOPTVARS= LIBNAME Statement Option ....................................... 15
- IGNORE_READ_ONLY_COLUMNS= LIBNAME Statement Option ................... 16
- MAX_CHAR_LEN= LIBNAME Statement Option ...................................... 16
- MUTUAL_AUTH= LIBNAME Statement Option ....................................... 17
- PRESERVE_COL_NAMES= LIBNAME Statement Option ............................ 18
- PRESERVE_TAB_NAMES= LIBNAME Statement Option ............................ 19
- PROC_DATASETS_TAGFILTER= LIBNAME Statement Option ...................... 20
- PROXY_HOST= LIBNAME Statement Option .......................................... 21
- PROXY_PASS= LIBNAME Statement Option .......................................... 21
- PROXY_PORT= LIBNAME Statement Option .......................................... 22
- PROXY_USER= LIBNAME Statement Option .......................................... 22
- QUOTE_CHAR= LIBNAME Statement Option ......................................... 23
- ROWSET_SIZE= LIBNAME Statement Option ........................................ 24
- SCHEMA_COLLECTION= LIBNAME Statement Option .............................. 25
- SCHEMA_DB= LIBNAME Statement Option .......................................... 25
- SCHEMA_PASSWORD= LIBNAME Statement Option .................................. 26
- SCHEMA_PORT= LIBNAME Statement Option ....................................... 27
- SCHEMA_SERVER= LIBNAME Statement Option ..................................... 27
- SCHEMA_SSL_CA= LIBNAME Statement Option .................................... 28
- SCHEMA_SSL_CERT= LIBNAME Statement Option ................................ 28
- SCHEMA_USER= LIBNAME Statement Option ....................................... 29
- SHOW_METADATA= LIBNAME Statement Option .................................... 29
- SHOW_RECYCLED= LIBNAME Statement Option ................................... 30
- SHOWFLAGS= LIBNAME Statement Option ......................................... 31
- SPOOL= LIBNAME Statement Option ................................................ 32
- SQL_FUNCTIONS_COPY= LIBNAME Statement Option ............................ 32
- SSL_CA= LIBNAME Statement Option ............................................... 33
LIBNAME Options for Nonrelational Database Interfaces

This documentation contains the details about the LIBNAME options for SAS/ACCESS interfaces to nonrelational databases.

Dictionary

API_TRACE= LIBNAME Statement Option

Specifies whether to display the number of API calls that are made during a SAS session while connected to Salesforce.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: NO
Restriction: Only user name and password are allowed.
Data source: Salesforce
Note: Tracing includes the precise SOQL statements to be executed with Salesforce.
Example: libname x sforce user="myuser1" pass="mypwd1" API_TRACE=yes;
proc datasets library=x;quit;

proc sql;
select * from x.TEST001;quit;

Syntax

API_TRACE=YES | NO
Syntax Description

**YES**
specifies to display the number of API calls that are made in general during any operations that connect to Salesforce. It displays any SOQL statements that are executed in the database. As shown in the example below, it lets you view diagnostics on your SAS program in the SAS log.

\[ \text{NOTE: API Trace: Prepared query. Usage: ##.##, ###### calls used out of ###### allocated.} \]

**NO**
specifies that the number of API calls is not displayed.

---

**AUTHDOMAIN= LIBNAME Statement Option**

Allows connection to a server by specifying the name of an authentication domain metadata object.

Valid in: SAS/ACCESS LIBNAME statement

Category: Data Access

Default: none

Requirement: For data sources with the SERVER= connection option, if you specify AUTHDOMAIN=, you must also specify the METASERVER= system option. See Details for more information.

Interaction: To specify AUTHDOMAIN=, you must also specify the SERVER= connection option.

Data source: MongoDB, Salesforce

See: For complete information about creating and using authentication domains, see the credential management information in [SAS Intelligence Platform: Security Administration Guide](http://example.com).

**Syntax**

```
AUTHDOMAIN= authentication-domain
```

**Syntax Description**

**authentication-domain**
specifies the name of an authentication domain metadata object.

\[ \text{Note: Enclose the authentication domain value in quotation marks if the value is specified in lower case or in mixed case.} \]

**Details**

An administrator creates authentication domain definitions while creating a user definition with the User Manager in SAS Management Console. The authentication domain is associated with one or more login metadata objects, which provide
access to the server. The authentication domain is resolved when the DBMS engine calls the SAS Metadata Server and returns the authentication credentials.

For data sources with the SERVER= connection option, if you specify AUTHDOMAIN=, you must also specify the METASERVER= system option. However, the authentication domain references credentials, so you do not need to explicitly specify database USER= and PASSWORD=. Here is an example:

```plaintext
options metauser="metadata-userid" metapass="metadata-password"
    metaport=port metaprotocol=bridge
    metarepository="metadata-repository"
    metaserver="server-name";
```

```plaintext
libname mydb mongo server=mysrv1 port=27017 authodomain="mngoauth";
```

The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running to resolve the metadata object specification.

### AUTHENDPOINT= LIBNAME Statement Option

- Specifies the authorization endpoint to use.
- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Access
- **Default:** none
- **Data source:** Salesforce
- **Note:** If you do not specify this option, SAS uses https://login.salesforce.com to authenticate users.
- **Tip:** Use this option only if your site uses a custom URL as the authorization endpoint or if you connect to a sandbox.

**Syntax**

```
AUTHENDPOINT=authorization-endpoint
```

**Syntax Description**

`AUTHENDPOINT=authorization-endpoint`

specifies a URL that represents the authorization endpoint to use to authenticate a user’s Salesforce account. If you are logging in to a production Salesforce instance, you do not need to specify this option. The default value is used.

If you are logging in to a sandbox, set `AUTHENDPOINT="test.salesforce.com"` and add the suffix that Salesforce provides to your user name.

**Example: Specify an Authorization Endpoint**

```plaintext
libname x sforce user='myusr1' pwd='mypwd1'
    authendpoint='test.salesforce.com';
```

```plaintext
proc datasets lib=x;quit;
```
### CONNECTION= LIBNAME Statement Option

Specifies whether operations on single or multiple librefs can share a connection to the DBMS.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Access
- **Defaults:**
  - SHARED [MongoDB]
  - SHAREDREAD [Salesforce]
- **Data source:** MongoDB, Salesforce
- **See:** DEFER= LIBNAME option

#### Syntax

**CONNECTION=**GLOBAL | GLOBALREAD | SHARED | SHAREDREAD | UNIQUE

#### Syntax Description

**GLOBAL**

specifies that all operations that access DBMS tables in multiple librefs share a single connection if the conditions for sharing a connection are met. For more information, see "Conditions for a Shared DBMS Connection" on page 8.

**Restriction**

GLOBAL is not a supported value for MongoDB.

**Interactions**

One connection is shared for all tables in a location that is referenced by a libref for which you specify CONNECTION=GLOBAL.

When you set CONNECTION=GLOBAL, any pass-through code that you include after the LIBNAME statement can share the connection.

**GLOBALREAD**

specifies that all READ operations that access DBMS tables in multiple librefs share a single connection if the conditions for sharing a connection are met. For more information, see "Conditions for a Shared DBMS Connection" on page 8.

A separate connection is established for each table that is opened for update or output operations.

**Restriction**

GLOBALREAD is not a supported value for MongoDB.

**SHARED**

specifies that all operations that access DBMS tables in a single libref share a single connection if the conditions for sharing a connection are met. For more information, see "Conditions for a Shared DBMS Connection" on page 8.

**Restriction**

The CONNECTION= option controls only connections that you use to open tables with a libref. When you set
CONNECTION=SHARED, it has no effect on utility connections or explicit pass-through connections.

**Tip** Use SHARED to eliminate the deadlock that can occur when you create and load a DBMS table from an existing table that exists in the same database or tablespace. This happens only in certain output processing situations and is the only recommended use for CONNECTION=SHARED.

**SHAREDREAD** specifies that all READ operations that access DBMS tables in a single libref share a single connection if the conditions for sharing a connection are met. For more information, see “Conditions for a Shared DBMS Connection” on page 8.

A separate connection is established for every table that is opened for update or output operations.

**TIP** Typically, this value offers the best performance and guarantees data integrity.

**UNIQUE** specifies that a separate connection is established every time a DBMS table is accessed by your SAS application.

Use UNIQUE if you want each use of a table to have its own connection.

**Details**

**Overview of the CONNECTION= LIBNAME Option**

The main reason for using the CONNECTION= LIBNAME option is to control the number of physical connections to your DBMS. When you specify that you want to share DBMS connections, you enable SAS to use one physical connection across multiple DATA steps and procedure calls. In this way, you limit the number of physical connections for your SAS session.

For most SAS/ACCESS interfaces, there must be a connection, also known as an attach, to the DBMS server before a user can access any data. Typically, a DBMS connection has one transaction, or work unit, that is active in the connection. This transaction is affected by any SQL commits or rollbacks that the engine performs within the connection while executing the SAS application.

The CONNECTION= option lets you control the number of connections, and therefore transactions, that your SAS/ACCESS interface executes and supports for each LIBNAME statement, SQL pass-through CONNECT statement, or both.

When you connect to the DBMS server via PROC SQL, only the GLOBAL and SHARED values from a LIBNAME statement are passed down.

**Conditions for a Shared DBMS Connection**

If you want to share a connection across librefs, the critical connection options that you specify must be the same. You can specify the connection options in a LIBNAME statement or in the CONNECT statement in an SQL procedure call. When SAS/ACCESS compares connection option values, it does not matter whether you use optional quotation marks across libref declarations.
Here are the conditions that must be met to share a physical connection to your DBMS:

- These connection options must have the same value for each libref declaration:
  - USER=
  - PWD=
  - SERVER=
  - DATABASE=
  - SCHEMA_COLLECTION= (MongoDB)

- The CONNECTION= LIBNAME option must have the same value for each libref declaration.

If these conditions are not met, SAS/ACCESS automatically creates additional physical connections to the DBMS.

Examples

Example 1: Use SHAREDREAD

In this example, MYDBLIB makes the first connection to the DBMS. This connection is used to print the data from MYDBLIB.TAB. MYDBLIB2 makes the second connection to the DBMS. These connections are closed with the CLEAR option.

```plaintext
/* connection 1 */
libname mydblib mongo user=myusr1 pwd=mypwd1 server='mysrv1' connection=sharedread;
/* connection 2 */
libname mydblib2 mongo user=myusr1 pwd=mypwd1 server='mysrv1' connection=sharedread;
proc print data=mydblib.tab;
run;
libname mydblib clear;
libname mydblib2 clear;
```

Example 2: Use GLOBALREAD

In this example, the two librefs, MYDBLIB and MYDBLIB2, share the same connection for Read access because CONNECTION=GLOBALREAD and the connection options are identical. The first connection prints the data from MYDBLIB.TAB and a second connection updates MYDBLIB.TAB. The second connection is closed at the end of the step. The first connection is closed with the final LIBNAME statement.

```plaintext
/* connection 1 */
libname mydblib mongo user=mysrv1 pw=mypwd1 server='mysrv1' connection=globalread;
libname mydblib2 mongo user=mysrv1 pw=mypwd1 server='mysrv1' connection=globalread;
/* connection 2 */
proc sql;
update mydblib.tab <additional code...>;
/* does not close connection 1 */
libname mydblib clear;
/* closes connection 1 */
libname mydblib2 clear;
```
data mydata;
   set mydblib.tab mydblib2.tab2;
   if myvar > 1000 and var2 contains "Helsinki";
run;

---

**DBGEN_NAME= LIBNAME Statement Option**

Specifies how columns with invalid characters are automatically renamed to valid SAS variable names.

**Valid in:** SAS/ACCESS LIBNAME statement

**Category:** Data Set Control

**Default:** DBMS

**Data source:** MongoDB, Salesforce

**See:** DBGEN_NAME= data set option, VALIDVARNAME= in "System Options for Nonrelational Databases"

### Syntax

**DBGEN_NAME=** DBMS | SAS

### Syntax Description

- **DBMS**
  - specifies that SAS renames DBMS columns to valid SAS variable names. SAS converts to underscores any characters that it does not allow. If it converts a column to a name that already exists, it appends a sequence number at the end of the new name.

- **SAS**
  - specifies that SAS converts DBMS columns that contain characters that SAS does not allow into valid SAS variable names. SAS uses the format _COLn, where n is the column number, starting with 0. If SAS converts a name to a name that already exists, it appends a sequence number at the end of the new name.

### Details

SAS retains column names when it reads data from DBMS tables unless a column name contains characters that SAS does not allow, such as $ or @. SAS allows alphanumeric characters and the underscore (_).

This option is intended primarily for National Language Support, notably for the conversion of kanji to English characters. English characters that are converted from kanji are often those that SAS does not allow. Although this option works for the single-byte character set (SBCS) version of SAS, SAS ignores it in the double-byte character set (DBCS) version. So if you have the DBCS version, you must first set VALIDVARNAME=ANY before using your language characters as column variables.
Example

If you specify DBGEN_NAME=SAS, SAS renames a DBMS column named $Dept_Amt$ to _ColName. If you specify DBGEN_NAME=DBMS, SAS renames the $Dept_Amt$ column to Dept_Amt.

---

**DBMAX_TEXT= LIBNAME Statement Option**

Specifies the maximum number of bytes to allocate for character columns.

**Valid in:** SAS/ACCESS LIBNAME statement

**Category:** Data Set Control

**Alias:** TEXTSIZE= [PI System Asset Framework]

**Defaults:**
- 32 [PI System Data Archive]
- 256 [PI System Asset Framework]
- 1024 [MongoDB, Salesforce]

**Range:** 1–976 [PI System Data Archive]

**Restriction:** PI System Data Archive: DBMAX_TEXT= does not apply to the Tag column or to any column when reading from Pipoint. These maximum lengths are determined dynamically at run time.

**Data source:** MongoDB, PI System Asset Framework, PI System Data Archive, Salesforce

**See:** DBMAX_TEXT= data set option

---

**Syntax**

```
DBMAX_TEXT=number-of-bytes
```

**Syntax Description**

`number-of-bytes`

specifies the maximum number of bytes to allocate for character columns. Data with a string value that is longer than the allocated number of bytes is truncated. If a truncation occurs when SAS reads the data, SAS issues a warning and indicates a sufficient value for DBMAX_TEXT=.

The number of bytes that are allocated for all character strings is based on the minimum of these values: the default column size or the specified value of DBMAX_TEXT.  

**Note:** The length of a GUID column for the PI System is always 36 bytes.

---

**Details**

*PI System Data Archive:* This option controls the length of the Value and Status columns only.
**DBPROMPT= LIBNAME Statement Option**

Specifies whether SAS displays a dialog box that prompts the user to enter DBMS connection information before connecting to the DBMS in interactive mode.

**Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement

**Category:** Data Set Control

**Default:** NO

**Restriction:** The maximum password length for most of the SAS/ACCESS LIBNAME interfaces is 32 characters.

**Interaction:** The DBPROMPT= option interacts with the DEFER=LIBNAME option to determine when the prompt dialog box appears. If DEFER=NO, the DBPROMPT dialog box appears when the LIBNAME statement is executed. If DEFER=YES, the DBPROMPT dialog box appears when you first open a table or view. The DEFER= option normally defaults to NO, but it defaults to YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

**Data source:** MongoDB, Salesforce

**See:** DBPROMPT= data set option, DEFER= LIBNAME option

**Syntax**

```
DBPROMPT=YES | NO
```

**Syntax Description**

- **YES** specifies that SAS displays a dialog box that interactively prompts you for the DBMS connection options the first time the libref is used.

- **NO** specifies that SAS does not display the prompting dialog box.

**Details**

If you specify DBPROMPT=Yes, it is not necessary to provide connection options with the LIBNAME statement. If you use the LIBNAME statement to specify connection options and DBPROMPT=YES, connection option values are displayed in the dialog box. The value of the password appears as a series of asterisks. You can override all of these values interactively.

The DBPROMPT dialog box usually opens only once for each time that the LIBNAME statement is specified. It might open multiple times if DEFER=YES and the connection fails when SAS tries to open a table. In such cases, the DBPROMPT dialog box appears until a successful connection occurs or you click **Cancel**.

**Note:** For Salesforce, you can enter 30 characters for the user name and password and up to 70 characters for the path, depending on your platform.
Examples:

Example 1: Preventing a Prompt Dialog Box from Opening

In this example, the DBPROMPT dialog box does not open when the LIBNAME statement is submitted because DEFER=YES. The DBPROMPT dialog box appears when the PRINT procedure is processed, a connection is made, and the table is opened.

```sas
libname mydblib sforce dbprompt=yes defer=yes;
proc print data=mydblib.staff;
run;
```

Example 2: Allow a Prompt Dialog Box to Open Only Once

In this example, the DBPROMPT dialog box appears while the LIBNAME statement is processing. The DBPROMPT dialog box does not appear in subsequent statements because the DBPROMPT dialog box appears only once per LIBNAME statement.

```sas
libname mydblib sforce dbprompt=yes defer=no;
```

Example 3: Allow Values to Appear in a Prompt Dialog Box

In this example, values provided in the LIBNAME statement are pulled into the DBPROMPT dialog box. The values myusr1 and mysrv1 appear in the DBPROMPT dialog box, and the user can edit and confirm them. The password value appears in the DBPROMPT dialog box as a series of asterisks, so the user can also edit it.

```sas
libname mydblib sforce user=myusr1 pw=mypwd1 path='mysrv1' dbprompt=yes defer=no;
```

---

**DBSASLABEL= LIBNAME Statement Option**

Specifies the column labels that an engine uses.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Default:** COMPAT
- **Data source:** MongoDB, Salesforce
- **Tip:** This option is useful when PROC SQL uses column labels as headers instead of column aliases.
- **See:** `DBSASLABEL=` data set option

**Syntax**

`DBSASLABEL=COMPAT | DBMS | NONE`
Syntax Description

**COMPAT**
 specifies that the labels returned should be compatible with what the application normally receives. In other words, engines exhibit their normal behavior.

**DBMS**
specifies that the engine returns a label exactly as it is stored in the database.

**NONE**
specifies that the engine does not return a column label. The engine returns blanks for the column labels.

Details

By default, the SAS/ACCESS interface for your DBMS generates column labels from the column names instead of from the real column labels.

Example: Return Blank Labels for Aliases in Headings

This example shows how to use DBSASLABEL= as a LIBNAME option to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```sas
libname sforce mongo user=myusr1 pwd=mypwd1 dbsaslabel=none;
proc sql;
select deptno as Department ID, loc as Location from mylib.dept;
```

Without DBSASLABEL=NONE, aliases are ignored, and DEPTNO and LOC are used as column headings in the result set.

---

**DEFER= LIBNAME Statement Option**

Specifies whether to wait until a table is accessed before connecting to the data source.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Access
- **Defaults:** NO [MongoDB, PI System Data Archive, Salesforce]
  YES [PI System Asset Framework]
- **Restriction:** Do not set DEFER= to YES when you are using PROC DS2 to interact with your data. PROC DS2 assumes that a connection to your database is in place when the procedure is invoked.
- **Interaction:** If the connection option DBPROMPT=YES, then DEFER=YES is the default.
- **Data source:** MongoDB, PI System Asset Framework, PI System Data Archive, Salesforce

**Syntax**

```
DEFER=YES | NO
```
Syntax Description

YES
specifies to wait until a table is accessed before connecting to the data source.

NO
specifies to connect to the external data source when a libref is assigned by a LIBNAME statement.

Details
For the PI System, the connection is made to the PI System server.

HIDEFLAGS= LIBNAME Statement Option
Specifies whether the columns Questionable, Annotated, and Substituted are hidden.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: NO
Data source: PI System Data Archive
See: HIDEFLAGS= data set option, SHOWFLAGS= LIBNAME option

Syntax

**HIDEFLAGS=** YES | NO

Syntax Description

YES
hides the Questionable, Annotated, and Substituted columns.

NO
displays all columns.

HIDEOPTVARS= LIBNAME Statement Option
Specifies whether to hide the secondary variables of a data set.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: NO
Data source: PI System Asset Framework
See: HIDEOPTVARS= data set option
Syntax

HIDEOPTVARS=YES | NO

Syntax Description

YES
specifies to exclude nonessential variables in the data set. For more information about which variables are hidden when HIDEOPTVARS=YES, see “About the Asset Framework Virtual Tables” on page 150.

NO
specifies not to exclude the secondary variables of a data set. To display all of the variables in a data set, specify HIDEOPTVARS=NO.

IGNORE_READ_ONLY_COLUMNS= LIBNAME Statement Option

Specifies whether to ignore read-only columns.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: NO
Data source: Salesforce

Syntax

IGNORE_READONLY_COLUMNS=YES | NO

Syntax Description

YES
specifies that the SAS/ACCESS engine ignores columns where data types are read-only when you are generating insert and update SQL statements.

NO
specifies that the SAS/ACCESS engine does not ignore columns where data types are read-only when you are generating insert and update SQL statements.

Details

Use this option for data types that are read-only or when Salesforce has properties that allow certain data types to be read-only.

MAX_CHAR_LEN= LIBNAME Statement Option

Specifies the maximum column width for results that are returned from an external database.

Valid in: SAS/ACCESS LIBNAME statement
Syntax

**MAX_CHAR_LEN=value**

**Syntax Description**

*value*
- specifies the maximum column width for results that are returned by pass-through queries.

**MUTUAL_AUTH= LIBNAME Statement Option**

Specifies whether to enforce mutual authentication.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Default:** NO
- **Data source:** Salesforce

**Syntax**

**MUTUAL_AUTH=YES | NO**

**Syntax Description**

**YES**
- Specifies that mutual authentication is enabled. This requires that both the SAS client and the Salesforce server must authenticate each other's identities using a mutual authentication certificate. To do this, you must set the path to your local certificate (CERT_PATH=) and a password (CERT_PASS).

**NO**
- Specifies that mutual authentication is disabled. By default, the Salesforce engine connects to port 4443 through a secured connection over HTTPS.

**Example: Enable Mutual Authentication**

```bash
libname x sforce user="myusr1" pass="mypwd1" mutual_auth=yes
cert_path="c:\mycertpath\tls\cert.pem" cert_pass="mycertpwd"
```
PRESERVE_COL_NAMES= LIBNAME Statement Option

Preserves spaces, special characters, and case sensitivity in DBMS column names when you create DBMS tables.

- **Valid in:** SAS/ACCESS LIBNAME statement (when you create DBMS tables)
- **Category:** Data Set Control
- **Alias:** PRESERVE_NAMES= (see “Details”)
- **Default:** YES
- **Restrictions:** This option applies only when you use SAS/ACCESS to create a new DBMS table. PRESERVE_COL_NAMES= does not apply to the SQL pass-through facility.
- **Interaction:** If you use the DS2 or FedSQL language, quoting and casing of names is different. For more information, see the identifiers topic in SAS DS2 Language Reference or SAS FedSQL Language Reference.
- **Data source:** Salesforce
- **See:** PRESERVE_COL_NAMES= data set option, PRESERVE_TAB_NAMES= LIBNAME option, SAS Names and Support for DBMS Names, , VALIDVARNAME= in “System Options for Nonrelational Databases”

**Syntax**

PRESERVE_COL_NAMES=Yes | No

**Syntax Description**

**YES**

specifies that column names that are used in table creation are passed to the DBMS with special characters and the exact, case-sensitive spelling of the name is preserved.

**NO**

specifies that column names that are used to create DBMS tables are derived from SAS variable names (VALIDVARNAME= system option) by using the SAS variable name normalization rules. However, the database applies its DBMS-specific normalization rules to the SAS variable names when creating the DBMS column names.

The use of N-literals to create column names that use database keywords or special symbols other than the underscore character might be invalid when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

**Details**

When you create a table, you assign the column names by using one of these methods.

- To control the case of the DBMS column names, specify variables using the case that you want and set PRESERVE_COL_NAMES=YES. If you use special
symbols or blanks, you must set VALIDVARNAME= to ANY and use N-literals. For more information, see the SAS/ACCESS naming topic in the DBMS-specific reference section for your interface in this document and also SAS Data Set Options: Reference.

- To enable the DBMS to normalize the column names according to its naming conventions, specify variables using any case and set PRESERVE_COL_NAMES= NO.

The Salesforce engine automatically converts all schema, table, and column names to lowercase.

To save some time when coding, specify the PRESERVE_NAMES= alias if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement.

To use column names that are not valid SAS names in your SAS program, you must use one of these techniques.

- Use the DQUOTE= option in PROC SQL and reference your columns using double quotation marks, as shown in this example.

  ```sas
  proc sql dquote=ansi;
  select "Total$Cost" from mydblib.mytable;
  ```

- Specify the global system option VALIDVARNAME=ANY and use name literals in the SAS language, as shown in this example.

  ```sas
  proc print data=mydblib.mytable;
  format 'Total$Cost'n 22.2;
  ```

If you are creating a table in PROC SQL, you must also include the PRESERVE_COL_NAMES=YES option in your LIBNAME statement, as shown in this example.

```sas
libname mydblib sforce user=myusr1 password=mypwd1
  preserve_col_names=yes;
proc sql dquote=yes;
  create table mydblib.mytable ("my$column" int);
```
Syntax

**PRESERVE_TAB_NAMES=**YES | NO

Syntax Description

**YES** specifies that table names are read from the DBMS with special characters, and the exact, case-sensitive spelling of the name is preserved.

**NO** specifies that when you refer to an existing table, the table names are derived from SAS member names by using SAS member name normalization.

When you use SAS to read a list of table names, tables with names that do not conform to SAS member name normalization rules do not appear in output. In SAS line mode, here is how SAS indicates the number of tables that are not displayed from PROC DATASETS because of this restriction:

Due to the PRESERVE_TAB_NAMES=NO LIBNAME option setting, 12 table(s) have not been displayed.

You do not receive this warning when you use SAS Explorer. SAS Explorer displays DBMS table names in capitalized form when PRESERVE_TAB_NAMES=NO. This is not how the tables are represented in the DBMS.

Details

To use table names in your SAS program that are not valid SAS names, use one of these techniques.

- Use the PROC SQL option DQUOTE= and place double quotation marks around the table name. The libref must specify PRESERVE_TAB_NAMES=YES. Here is an example.

```sas
libname mydb sforce user=myusr1 pwd=mypwd1 preserve_tab_names=yes;
proc sql dquote=ansi;
select * from mydb."my table"; quit;
```

- Use name literals in the SAS language. The libref must specify PRESERVE_TAB_NAMES=YES. Here is an example.

```sas
libname mydb sforce user=myusr1 pwd=mypwd1 preserve_tab_names=yes;
proc print data=mydb.'my table'n; run;
```

### PROC_DATASETS_TAGFILTER= LIBNAME Statement Option

Specifies a character string, which can include the wildcard characters ‘*’ and ‘?’, that is used to limit the output from the DATASETS procedure.

**Valid in:** SAS/ACCESS LIBNAME statement

**Category:** Data Access

**Default:** *
Syntax

PROC_DATASETS_TAGFILTER="filter-string"

Syntax Description

filter-string
specifies a character string that is used to limit the output from the DATASETS procedure. The character string can contain these wildcard characters:

* represents zero or more of any character.
? represents any single character.

---

PROXY_HOST= LIBNAME Statement Option

Specifies the host name or IP address of a proxy server.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Aliases: PROXY_SERVER=
PROXYHOST=
PROXYSERVER=
Default: none
Data source: Salesforce
Note: This option is available on SAS Viya 3.5 and 4.0 and also SAS 9.4M6, beginning in April 2020.
Tip: You might use the (optional) proxy options to request a connection from and allow login access to Salesforce through a specific IP address using a proxy server.
See: PROXY_PASS=, PROXY_PORT=, PROXY_USER=
Example: libname x sasioslf AUTHENDPOINT="test.salesforce.com" user="user-id" password="password";

Syntax

PROXY_HOST="host-name";

---

PROXY_PASS= LIBNAME Statement Option

Specifies the password that is associated with PROXY_USER=.
Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Alias: PROXYPASS=
Default: none
Data source: Salesforce
Note: This option is available on SAS Viya 3.5 and 4.0 and also SAS 9.4M6, beginning in April 2020.
Tip: You might use the (optional) proxy options to request a connection from and allow login access to Salesforce through a specific IP address using a proxy server.
See: PROXY_HOST=, PROXY_PORT=, PROXY_USER=
Example: libname x sasioslf AUTHENDPOINT="test.salesforce.com" user="user-id" password="password";

Syntax

`PROXY_PASS="password";`

PROXY_PORT= LIBNAME Statement Option

Specifies the port number of the proxy server.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Alias: PROXYPORT=
Default: 8080
Data source: Salesforce
Note: This option is available on SAS Viya 3.5 and 4.0 and also SAS 9.4M6, beginning in April 2020.
Tip: You might use the (optional) proxy options to request a connection from and allow login access to Salesforce through a specific IP address using a proxy server.
See: PROXY_HOST=, PROXY_PASS=, PROXY_USER=
Example: libname x sasioslf AUTHENDPOINT="test.salesforce.com" user="user-id" password="password";

Syntax

`PROXY_PORT="port-number";`

PROXY_USER= LIBNAME Statement Option

Specifies the user ID that is used to access the proxy server.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Alias: PROXYUSER=
Default: none
Data source: Salesforce
Note: This option is available on SAS Viya 3.5 and 4.0 and also SAS 9.4M6, beginning in April 2020.
Tip: You might use the (optional) proxy options to request a connection from and allow login access to Salesforce through a specific IP address using a proxy server.
See: PROXY_HOST=, PROXY_PASS=, PROXY_PORT=
Example: libname x sasioslf AUTHENDPOINT="test.salesforce.com" user="user-id" password="password";

Syntax

`PROXY_USER="user-id";`
QUOTE_CHAR= LIBNAME Statement Option

Specifies which quotation mark character to use when delimiting identifiers.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: none

Syntax

QUOTE_CHAR= "character"

Syntax Description

*character*

specifies the quotation mark character to use when delimiting identifiers, such as the double quotation mark (".

Enclose the character in single or double quotation marks, as appropriate. For more information, see the examples below.

Details

The provider usually specifies the delimiting character. However, when there is a difference between what the provider and the DBMS allow for this character, the QUOTE_CHAR= option overrides the character that the provider returns.
Examples:

Example 1: Specify a Single Quotation Mark
Here is what to specify if you want your quotation character to be a single quotation mark.

`libname mydb mongo server=mysrv1 pwd=mypassword quote_char='';`

Example 2: Specify a Double Quotation Mark
Here is what to specify if you want your quotation character to be a double quotation mark.

`libname mydb sforce server=mysrv1 pwd=mypassword quote_char='';`

---

**ROWSET_SIZE= LIBNAME Statement Option**

Specifies the number of rows of DBMS data to read into the buffer.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Set Control
- **Alias:** READBUFF=
- **Default:** 0
- **Restriction:** When ROWSET_SIZE=1, only one row is retrieved at a time.
- **Interaction:** Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.
- **Data source:** MongoDB, Salesforce
- **Tip:** This option improves performance by specifying a number of rows that can be held in memory for input into SAS.
- **See:** `IGNORE_READ_ONLY_COLUMNS= LIBNAME option, ROWSET_SIZE= data set option`

---

**Syntax**

`ROWSET_SIZE=integer`

**Syntax Description**

`integer` specifies the number of rows to hold in memory. SAS allows the maximum value that is supported by the DBMS.
SCHEMA_COLLECTION= LIBNAME Statement Option

Specifies the MongoDB collection where the schema file is stored.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: sas-default-schema
Restriction: There can be only one schema per collection. However, there can be multiple collections (and therefore multiple schemas) in a database.
Requirements: To specify a new collection name, which results in the creation of a new schema file, a user must have Write permission to the MongoDB database. Similarly, to perform any action that alters a schema, a user must have Write permission to the MongoDB database.
Journaling must be enabled for MongoDB before you can save a schema file.
Schema names must be unique across MongoDB collections.

Data source: MongoDB
See: “Working with Schemas for MongoDB Data”

Syntax

SCHEMA_COLLECTION='collection-name'

Syntax Description

collection-name

specifies the MongoDB collection where the schema file is stored. If the schema file does not already exist, SAS/ACCESS scans the specified collection and stores the generated schema with the collection.

Details

Beginning in the April 2020 release of SAS/ACCESS Interface to MongoDB, if you do not specify the SCHEMA_COLLECTION= option when you connect to MongoDB, a stored schema is created with the name sas-default-schema. As a best practice, provide a descriptive name for your schema. Schema names are case-sensitive. Multiple users can access the same stored schema by specifying the same schema name for SCHEMA_COLLECTION=.

To create a temporary schema, specify SCHEMA_COLLECTION='$temp$'. Temporary schemas are held in memory for the duration of your SAS session. For more information, see “Working with Temporary Schemas” on page 95.

SCHEMA_DB= LIBNAME Statement Option

Specifies the MongoDB database where a schema is stored.
Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: DATABASE= value
Requirement: To create a stored schema file or modify an existing stored schema file, a user must have Write permission to the specified database.
Data source: MongoDB
See: “Working with Schemas for MongoDB Data”

Syntax

SCHEMA_DB='database-name'

Syntax Description
database-name
specifies the MongoDB database where a schema file is stored.
There can be multiple schemas in a database, but no more than one schema per database collection.

SCHEMA_PASSWORD= LIBNAME Statement Option

Specifies the password that corresponds to a schema.
Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Alias: SCHEMA_PWD=
Default: PWD= option value
Requirement: You must specify the alias SCHEMA_PWD= (rather than SCHEMA_PASSWORD=) when you provide this option in a connection string, such as with the NOPROMPT= connection option.
Data source: MongoDB
See: NOPROMPT= connection option, “Working with Schemas for MongoDB Data”

Syntax

SCHEMA_PASSWORD='string'

Required Argument
string
specifies the password that corresponds to a schema.
SCHEMA_PORT= LIBNAME Statement Option

Specifies the port that corresponds to a schema.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: PORT= option value
Data source: MongoDB
See: "Working with Schemas for MongoDB Data"

Syntax

```plaintext
SCHEMA_PORT=value
```

Syntax Description

`value`

specifies the port that corresponds to a schema.

SCHEMA_SERVER= LIBNAME Statement Option

Specifies the server name or IP address of the server that stores the schema mapping.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: SERVER= option value
Data source: MongoDB
See: "LIBNAME Options for MongoDB", "Working with Schemas for MongoDB Data"

Syntax

```plaintext
SCHEMA_SERVER='<server-name>'
```

Syntax Description

`server-name`

specifies the server that stores the schema definition for a collection.

Requirement

Use quotation marks if `server-name` contains spaces or nonalphanumeric characters or if the server name is an IP address.
SCHEMA_SSL_CA= LIBNAME Statement Option
Specifies the full path name to the certificate authority file for the MongoDB schema location.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: SSL_CA=value
Interaction: If you use the SCHEMA_SSL_CA= option to connect to the MongoDB schema file, then you must also specify the SSL_CA= option to connect to the MongoDB database.
Data source: MongoDB
See: SSL_CA= LIBNAME option, “Working with Schemas for MongoDB Data”

Syntax
SCHEMA_SSL_CA="path-and-file-name"

Syntax Description
path-and-file-name specifies the full path, including file name, to the certificate authority (CA) file to use to establish a secure connection to the MongoDB schema location.

Details
Certificate files enable MongoDB to support secure connections using TLS encryption and validation. A CA file contains a list of trusted certificate authorities. A CA file is configured as part of configuring your MongoDB server to use secure connections. The CA file must be specified in order to establish a secure connection to the server. For more information about configuring MongoDB and creating CA files, see the security information in your MongoDB documentation.

SCHEMA_SSL_CERT=_LIBNAME Statement Option
Specifies the full path name to the certificate file for the MongoDB schema location.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Access
Default: SSL_CERT=value
Interaction: If you use the SCHEMA_SSL_CERT= option to connect to your MongoDB schema file, then you must also specify the SSL_CERT= option to connect to the MongoDB server.
Data source: MongoDB
See: SSL_CERT= LIBNAME option, “Working with Schemas for MongoDB Data”
Syntax

`SCHEMA_SSL_CERT="path-name"`

Syntax Description

`path-name`

specifies the full path name to the certificate file to use to establish a secure connection to the MongoDB schema location.

Details

Certificate files enable MongoDB to support secure connections using TLS encryption and validation. A certificate file identifies the server public key certificate. A certificate file is configured as part of configuring your MongoDB server to use secure connections. The certificate file must be specified in order to establish a secure connection to the server. For more information about configuring MongoDB and creating certificates, see the security information in your MongoDB documentation.

---

**SCHEMA_USER= LIBNAME Statement Option**

Specifies the user ID that corresponds to a schema.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Access
- **Alias:** SCHEMA_UID=
- **Default:** USER= option value
- **Requirement:** You must specify the alias SCHEMA_UID= (rather than SCHEMA_USER=) when you provide this option in a connection string, such as with the NOPROMPT= connection option.

**Data source:** MongoDB

**See:** NOPROMPT= connection option, “Working with Schemas for MongoDB Data”

Syntax

`SCHEMA_USER='user-ID'`

Syntax Description

`user-ID`

specifies the user ID that corresponds to a schema.

---

**SHOW_METADATA= LIBNAME Statement Option**

Specifies whether to display metadata tables and columns.
SHOW_METADATA= LIBNAME Statement Option

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: NO
Data source: Salesforce
See: SHOW_RECYCLED= LIBNAME option

Syntax
SHOW_METADATA=YES | NO

Syntax Description
YES
specifies that the SAS/ACCESS engine displays metadata tables and columns.

NO
specifies that SAS/ACCESS engine does not display metadata tables and columns.

Details
In Salesforce, some system fields that are on most Tooling API objects are automatically updated during API operations, are system-generated, and are read-only. Unless you set this option to YES, these fields are hidden by default.

<table>
<thead>
<tr>
<th>CreatedBy</th>
<th>LastModifiedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreatedById</td>
<td>LastReferencedBy</td>
</tr>
<tr>
<td>CreatedDate</td>
<td>LastViewedBy</td>
</tr>
<tr>
<td>CurrencyIsoCode</td>
<td>MasterRecordId</td>
</tr>
<tr>
<td>Id</td>
<td>OwnerId</td>
</tr>
<tr>
<td>IsDeleted</td>
<td>RecordTypeId</td>
</tr>
<tr>
<td>LastActivityDate</td>
<td>SystemModstamp</td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td></td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td></td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td></td>
</tr>
</tbody>
</table>

For more information, see the Salesforce documentation.

Example: Show Metadata Columns
libname x sforce user="myusr1" pass="mypwd1"
  show_metadata=yes show_recyced=yes;
  proc print data=x.Account(where=(IsDeleted=1));run;

SHOW_RECYCLED= LIBNAME Statement Option

Specifies whether to display deleted table rows in the recycle bin.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
SHOW_DELETED= LIBNAME Statement Option

Alias: SHOW_DELETED=
Default: NO
Requirement: To use this option to see the IsDeleted column, you must first set SHOW_METADATA=YES.
Data source: Salesforce
Note: By default, all table rows are displayed, whether or not they were recently deleted. Rows that were recently deleted have a value of 1 in the IsDeleted column.
See: SHOW_METADATA= LIBNAME option

Syntax

SHOW_DELETED=YES | NO

Syntax Description

YES
specifies that recently deleted table rows are displayed in the recycle bin.

NO
specifies that recently deleted table rows are not displayed in the recycle bin.

Example: Display Only Deleted Table Rows

libname x sforce user="myusr1" pass="mypwd1"
show_metadata=yes show_recycled=yes;
proc print data=x.mytable (where=(IsDeleted=1));run;

SHOWFLAGS= LIBNAME Statement Option

Specifies whether the columns Questionable, Annotated, and Substituted display.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Default: YES
Data source: PI System Data Archive
See: HIDEFLAGS= LIBNAME option, SHOWFLAGS= data set option

Syntax

SHOWFLAGS=YES | NO

Syntax Description

YES

displays the Questionable, Annotated, and Substituted columns.
SPOOL= LIBNAME Statement Option

Specifies whether SAS creates a utility spool file during Read transactions that read data more than once.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Alias: READBUFF= [Salesforce]
Default: YES
Data source: MongoDB, Salesforce
See: CONNECTION= LIBNAME option

Syntax

SPOOL= YES | NO

Syntax Description

YES specifies that SAS creates a utility spool file into which it writes the rows that are read the first time. For subsequent passes through the data, the rows are read from the utility spool file rather than being reread from the DBMS table. This guarantees that the rowset is the same for every pass through the data.

NO specifies that the required rows for all passes of the data are read from the DBMS table. No spool file is written. There is no guarantee that the rowset is the same for each pass through the data.

Details

In some cases, SAS processes data in more than one pass through the same set of rows. Spooling is the process of writing rows that have been retrieved during the first pass of a data Read to a spool file. In the second pass, rows can be reread without performing input and output to the DBMS a second time. When data must be read more than once, spooling improves performance. Spooling also guarantees that the data remains the same between passes, as most SAS/ACCESS interfaces do not support member-level locking.

SQL_FUNCTIONS_COPY= LIBNAME Statement Option

Writes the function associated with a particular LIBNAME statement to a SAS data set or to the SAS log.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Requirements:
When you specify a table, you must specify a two-part table name (libref.member), or an error results.

The value of libref.member must be a SAS data set. It is not checked to make sure that it is assigned to the default Base SAS engine.

Data source: MongoDB, Salesforce

Syntax

**SQL_FUNCTIONS_COPY=libref.member | SASLOG**

Syntax Description

- **libref.member**
  writes the current in-memory function list to a user-specified SAS data set (member) in the specified libref.

- **SASLOG**
  writes the current in-memory function list to the SAS log for a LIBNAME statement.

**SSL_CA= LIBNAME Statement Option**

Specifies the full path name to the certificate authority file.

- Valid in: SAS/ACCESS LIBNAME statement
- Category: Data Access
- Default: none
- Interaction: If you use the SSL_CA= option to connect to your MongoDB database, then you must also specify the SCHEMA_SSL_CA= option to connect to a schema file.

Data source: MongoDB

See: SCHEMA_SSL_CA= LIBNAME option, SSL_CERT= LIBNAME option

Syntax

**SSL_CA="path-and-file-name"**

Syntax Description

- **path-and-file-name**
  specifies the full path, including file name, to the certificate authority (CA) file to use to establish a secure connection to the MongoDB server.

Details

Certificate files enable MongoDB to support secure connections using TLS encryption and validation. A CA file contains a list of trusted certificate authorities. A
CA file is configured as part of configuring your MongoDB server to use secure connections. The CA file must be specified in order to establish a secure connection to the server. For more information about configuring MongoDB and creating CA files, see the security information in your MongoDB documentation.

### SSL_CERT= LIBNAME Statement Option

Specifies the full path name to the certificate file.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Access
- **Default:** none
- **Interaction:** If you use the SSL_CERT= option to connect to your MongoDB database, then you must also specify the SCHEMA_SSL_CERT= option to connect to a schema file.
- **Data source:** MongoDB
- **See:** SCHEMA_SSL_CERT= LIBNAME option, SSL_CA= LIBNAME option

**Syntax**

```
SSL_CERT="path-name"
```

**Syntax Description**

`path-name`

specifies the full path name to the certificate file to use to establish a secure connection to the MongoDB server.

**Details**

Certificate files enable MongoDB to support secure connections using TLS encryption and validation. A certificate file identifies the server public key certificate. A certificate file is configured as part of configuring your MongoDB server to use secure connections. The certificate file must be specified in order to establish a secure connection to the server. For more information about configuring MongoDB and creating certificates, see the security information in your MongoDB documentation.

### STRINGDATES= LIBNAME Statement Option

Specifies whether to read date and time values from the database as character strings or as numeric date values.

- **Valid in:** SAS/ACCESS LIBNAME statement
- **Category:** Data Set Control
- **Default:** NO
- **Data source:** MongoDB, Salesforce
Syntax

**STRINGDATES=** YES | NO

**Syntax Description**

**YES**

specifies that SAS reads date and time values as character strings.

**NO**

specifies that SAS reads date and time values as numeric date values.

---

**TESTDATE= LIBNAME Statement Option**

In a test environment, specifies a PI System datetime string that is used to test how a value appears as a SAS datetime value.

**Valid in:**

SAS/ACCESS LIBNAME statement

**Category:**

Data Set Control

**Alias:**

TESTTIME=

**Restrictions:**

When you use this option, you do not assign an actual libref.

This option is not supported when you use the PI Web API to access PI System data from a UNIX environment.

**Data source:**

PI System Data Archive

**Examples:**

libname pidata pisystem testdate="**";

libname pidata pisystem testdate="*-1h";

libname pidata pisystem

---

**Syntax**

**TESTDATE=** "PI-System-datetime-string"

**Syntax Description**

**PI-System-datetime-string**

in a test environment, specifies a PI System datetime string that is used to test how the value appears as a SAS datetime value. The PI System allows for calculations in dates and relative references. For example, "*-1d+2h" is the current time, minus 1 day, plus 2 hours. With this option users can test whether the date strings that are specified in START, END, or INTERVAL options are what is expected.
Details

When you use this option, you do not assign an actual libref. A LIBNAME statement with using the TESTDATE= option clears any existing libref of the same name. Ignore the “libref assign failed” error message when you use this option. Use this option only for testing SAS commands. Do not use it in production environments.

TIMEZONE= LIBNAME Statement Option

Specifies whether SAS reads and writes PI System timestamps using local time or GMT.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Alias: TZ=
Default: LOCAL
Data source: PI System Asset Framework, PI System Data Archive
See: TIMEZONE= LIBNAME option

Syntax

TIMEZONE=LOCAL | GMT

Syntax Description

LOCAL

specifies to use the local time to read and write timestamps in SAS.

GMT

specifies to use GMT to read and write timestamps in SAS.

Note: For SAS/ACCESS Interface to the PI System, GMT and UTC are equivalent and can be used as alias values for each other.

Alias UTC

Details

The Data Archive internally stores timestamps using GMT. For example, in the Data Archive, a numeric value of 0 in GMT is 01JAN1970:00:00:00. In Eastern Time, that GMT value is 31DEC1969:19:00:00 (GMT–5 hrs).

CAUTION

Data might be sequenced incorrectly for PI tags that are read between 1 A.M. and 2 A.M. when changing from daylight to standard time. This can cause differences when importing and exporting tag data. Timestamps in the range between 1 A.M. and 2 A.M. cannot be properly converted back to UTC, which might result in the data values in that time range being sequenced differently.
TRACE= LIBNAME Statement Option

Specifies whether to turn on tracing information for use in debugging.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Set Control
- **Default:** NO
- **Restriction:** This option is not supported on UNIX platforms.
- **Data source:** MongoDB, Salesforce
- **See:** TRACEFILE= LIBNAME option, TRACEFLAGS= LIBNAME Option

**Syntax**

```
TRACE= YES | NO
```

**Syntax Description**

- **YES**
  - specifies that tracing is enabled, and the DBMS driver manager writes each function call to the trace file that TRACEFILE= specifies.
- **NO**
  - specifies that tracing is enabled.

TRACEFILE= LIBNAME Statement Option

Specifies the file name to which the DBMS driver manager writes trace information.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Set Control
- **Default:** none
- **Restriction:** This option is not supported on UNIX platforms.
- **Interaction:** TRACEFILE= is used only when TRACE=YES.
- **Data source:** MongoDB, Salesforce
- **Tip:** If you do not specify the TRACEFILE= option when TRACE=YES, trace information is written to the default output file, ttrace.txt, in the current working directory. When you specify a different path and name using TRACEFILE=, you do not need to set TRACE=YES again.
- **See:** TRACE= LIBNAME option, TRACEFLAGS= LIBNAME Option
Syntax

\texttt{TRACEFILE=} file-name \mid 'path-and-file-name'

Syntax Description

\textit{file-name}

specifies the name of the file in which to store trace information. When no path is provided, the SAS trace file is stored with your data files.

\textit{path-and-file-name}

specifies the name of the file in which to store trace information and the directory where the trace file is stored. Enclose the fully qualified path name in single quotation marks.

---

**TRACEFLAGS= LIBNAME Statement Option**

Enables various tracing operations.

- **Valid in:** SAS/ACCESS LIBNAME statement, CONNECT statement
- **Category:** Data Set Control
- **Default:** none
- **Data source:** Salesforce
- **Tip:** If you do not specify the \texttt{TRACEFILE=} option when \texttt{TRACE=YES}, trace information is written to the default output file, tstrace.txt, in the current working directory. When you specify a different path and name using \texttt{TRACEFILE=} , you do not need to set \texttt{TRACE=YES} again.

**See:** \texttt{TRACE=} LIBNAME option, \texttt{TRACEFILE=} LIBNAME option

Syntax

\texttt{TRACEFLAGS=} ALL \mid COLBINDINGS \mid GETDATA \mid PUTDATA \mid TIMESTAMP

Syntax Description

\texttt{ALL}

enables tracing for all operations that are available for this option.

\texttt{COLBINDINGS}

enables tracing of column information on Read and Write.

\texttt{GETDATA}

enables tracing of operations that are related to Read.

\texttt{PUTDATA}

enables tracing of operations that are related to Write.

\texttt{TIMESTAMP}

enables tracing of operations that are related to TIMESTAMP.
USE_NATIVE_NAMES= LIBNAME Statement Option

Specifies whether to restore native naming conventions for custom tables and columns.

Valid in: SAS/ACCESS LIBNAME statement
Category: Data Set Control
Alias: NATIVE_NAMES=
Default: NO
Restriction: The __c suffix must not be used in queries.
Requirement: When USE_NATIVE_NAMES=YES, the __s suffix is required on component fields to update them.
Data source: Salesforce

Syntax

USE_NATIVE_NAMES=\ YES | NO

Syntax Description

YES
restores the native naming convention of using the __c suffix with custom tables and columns.

NO
retains the current names for custom tables and columns. By default, the __c suffix at the end of custom table and column names is not displayed, and no suffix is needed to update component fields.

Example: Restore Native Names

libname x sforce user="myusr1" pass="mypwd1"
    use_native_names=yes;
# Data Set Options for Nonrelational Databases

## Data Set Options for Nonrelational Database Interfaces

- Overview of Nonrelational Data Set Options .................................................. 42
- Precedence of Nonrelational Data Set Options ............................................. 42

## Dictionary

- ATTRIBUTE= Data Set Option ............................................................................. 42
- BOUNDARY= Data Set Option ........................................................................... 43
- CALCULATION_BASIS= Data Set Option ......................................................... 43
- CATEGORY= Data Set Option ............................................................................. 45
- CHILD= Data Set Option .................................................................................. 46
- COUNT= Data Set Option .................................................................................... 47
- DBCONDITION= Data Set Option ...................................................................... 48
- DBGEN_NAME= Data Set Option ......................................................................... 49
- DBMASTER= Data Set Option ............................................................................. 50
- DBMAX_TEXT= Data Set Option ......................................................................... 51
- DBNULL= Data Set Option LIBNAME Statement Option ...................................... 52
- DBPROMPT= Data Set Option LIBNAME Statement Option .................................. 53
- DBSASLABEL= Data Set Option ......................................................................... 53
- DBTYPE= Data Set Option .................................................................................. 54
- DELAY= Data Set Option .................................................................................... 56
- END= Data Set Option ........................................................................................ 56
- FILTER= Data Set Option ................................................................................... 57
- GUID= Data Set Option ....................................................................................... 58
- HIDEFLAGS= Data Set Option ............................................................................. 58
- HIDEOPTVARS= Data Set Option ....................................................................... 59
- INTERVAL= Data Set Option .............................................................................. 59
- LABEL= Data Set Option: PI System ................................................................. 61
- LABEL= Data Set Option: Salesforce ................................................................. 62
- MAX= Data Set Option ......................................................................................... 62
- MEDIAN= Data Set Option .................................................................................. 63
- MIN= Data Set Option ........................................................................................ 63
- MIXED= Data Set Option ..................................................................................... 64
- MODEL= Data Set Option ..................................................................................... 65
- NAME= Data Set Option ....................................................................................... 66
- PATH= Data Set Option ....................................................................................... 67
- PRESERVE_COL_NAMES= Data Set Option ....................................................... 67
- ROWSET_SIZE= Data Set Option ....................................................................... 69
- SASDATEFMT= Data Set Option ........................................................................ 69
- SEARCHBY= Data Set Option ............................................................................. 71
- SELECT= Data Set Option .................................................................................... 72
Data Set Options for Nonrelational Database Interfaces

Overview of Nonrelational Data Set Options

This documentation contains the details about the data set options for SAS/ACCESS interfaces to nonrelational databases.

Precedence of Nonrelational Data Set Options

For data set options with a corresponding LIBNAME option, when both are specified the data set option value overrides the LIBNAME option value.

Dictionary

ATTRIBUTE= Data Set Option

Specifies the attribute to use for retrieving elements.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none
Applies to: Asset Framework tables Element_Attribute, Eventframe_Attribute, Eventframe_Tsdata, Member, Tdata_Attribute, Tdata_Pipoint, Tdata_Summary
Requirement: This option is required when you specify SEARCHBY=ATTRIBUTE.
Data source: PI System Asset Framework
See: SEARCHBY= data set option
**ATTRIBUTE=** attribute-name

**Required Argument**

*attribute-name* specifies the attribute to use for retrieving elements.

---

**BOUNDARY= Data Set Option**

Specifies the boundary type to use to retrieve PI Point time series data values.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Default:** none
- **Applies to:** Asset Framework tables Eventframe_Tsdata, Tsdale_Attribute, Tsdale_Pipoint
- **Requirement:** The BOUNDARY= option requires the START= and END= data set options.
- **Interaction:** The BOUNDARY= option is applicable only when MODEL=ARCHIVE.
- **Data source:** PI System Asset Framework
- **See:** END= data set option, MODEL= data set option, START= data set option

**Syntax**

**BOUNDARY=** INSIDE | OUTSIDE | INTERPOLATED

**Required Arguments**

**INSIDE**

specifies to return recorded values from within the requested time period as the first and last values. That is, the first value is on or after the START= time, and the last value is on or before the END= time.

**OUTSIDE**

specifies to return recorded values outside of the requested time period as the first and last values. That is, the first value is on or before the START= time, and the last value is on or after the END= time.

**INTERPOLATED**

specifies to create an interpolated value at the end points of the requested time period if a recorded value does not appear at the START= or END= times.

---

**CALCULATION_BASIS= Data Set Option**

Specifies the method to determine the number of data points to use when calculating summary statistics.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Syntax

\[
\text{CALCULATION}_\text{BASIS} = \text{method}
\]

Required Argument

\textit{method}

specifies the method to determine the number of data points to use when calculating summary statistics. Some methods use the existing data points and some methods require interpolating data points across the summary interval.

Here are the possible values for \textit{method}:

\textbf{EventWeighted}

specifies that summary statistics evaluate each event (data point) equally. No interpolation of data over the summary interval is done.

The following rules are implemented with respect to summary interval boundaries:

- Use events at both boundaries when there is only one summary interval.
- Include events at the start time of a summary interval when there are multiple summary intervals and the intervals are in ascending time order.
- Include events at the end time of a summary interval when there are multiple summary intervals and the intervals are in descending time order.

\textbf{EventWeightedExcludeMostRecentEvent}

specifies that summary calculation behaves as EventWeighted, except in the handling of events at the boundaries of multiple summary intervals. Use this option to prevent events at the interval boundary from being counted in two intervals. With this option value, events at the end time (most recent time) of an interval are not used in that interval.
**EventWeightedExcludeEarliestEvent**

specifies that summary calculation is similar to EventWeightedExcludeMostRecentEvent, except that events at the start time (earliest time) for an interval are not used in that interval.

Aliases EWEEE

**EventWeightedIncludeBothEnds**

specifies that events on the boundary at either end of an interval are included in event-weighted calculations.

Aliases EWIBE

**TimeWeighted**

specifies to weight the values in the calculation by the time over which they apply. Interpolation is based on whether the attribute is stepped. Interpolated events are generated at the interval boundaries if necessary.

For non-median statistics, TimeWeighted specifies that the PI system should interpolate one data point per second over a summary interval.

For median statistics, SAS/ACCESS interpolates one data point per second when MEDIAN=YES. However, SAS/ACCESS does not generate more than 1000 data points over the time span indicated by the INTERVAL= option. When MEDIAN=count, count data points are interpolated.

Alias TW

**TimeWeightedContinuous**

specifies to apply weighting as you do for TimeWeighted, but perform interpolation between values as if they represent continuous data (standard interpolation), whether the value is continuous or discrete.

Aliases TWC

**TimeWeightedDiscrete**

specifies to apply weighting as in TimeWeighted, but perform interpolation as if values represent discrete values (stair-step plot), whether the value is continuous or discrete.

Aliases TWD

---

**CATEGORY= Data Set Option**

Specifies the category name to use as a filter when you retrieve observations.
**CATEGORY= Data Set Option**

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Default: none

Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary

Restriction: The CATEGORY= data set option pertains to element categories only. Matching attribute categories are ignored.

Requirement: This option is required when you specify SEARCHBY=CATEGORY.

Data source: PI System Asset Framework

See: SEARCHBY= data set option

**Syntax**

`CATEGORY=category-name`

Required Argument

*category-name* specifies the category name to use as a filter when you retrieve observations.

**CHILD= Data Set Option**

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Default: ALL

Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary

Interaction: This option is used only when you also specify SEARCHBY=PATH and the PATH= option.

Data source: PI System Asset Framework

See: SEARCHBY= data set option

**Syntax**

`CHILD=ALL | NO | IMMEDIATE | LEAF`

Required Arguments

*ALL* recursively retrieves all child elements. That is, the system retrieves all children, all children of those children, and so on.
NO
does not retrieve child elements.

IMMEDIATE
retrieves only the immediate child elements.

LEAF
retrieves only the children that have no children of their own.

Alias LEAVES

COUNT= Data Set Option

For PI System Data Archive, specifies the number of intervals to display in your data. For PI System Asset Framework, specifies the number of data values to retrieve.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: See Details.
Applies to: Asset Framework tables Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary
Data source: PI System Asset Framework, PI System Data Archive
Note: For PI System restrictions, requirements, and interactions, see Details.
See: BOUNDARY= data set option, INTERVAL= data set option, MAX= data set option, MIN= data set option, MODEL= data set option, START= data set option

Syntax

COUNT=integer

Required Argument

integer

for the Data Archive Picomp table, specifies the number of evenly spaced observations to interpolate between the START= and END= times.
For the Data Archive Picomp_Summary table, this option specifies the number of sample intervals to display between the START= and END= times. For example, if START= and END= span 24 hours, then COUNT=24 creates 24 one-hour intervals between START= and END=.
For the Asset Framework, specifies the number of data values to retrieve.

Details

Processing for this option is performed in the PI System.
Table 2.1 Default Values for PI System Interface

<table>
<thead>
<tr>
<th>PI System Interface</th>
<th>Table</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Archive</td>
<td>Picomp</td>
<td>Do not interpolate. Display all recorded events between the START= and END= times.</td>
</tr>
<tr>
<td></td>
<td>Picomp_Summary</td>
<td>Create only one interval that covers the time spanned by START= and END= values.</td>
</tr>
<tr>
<td>Asset Framework</td>
<td>any</td>
<td>none</td>
</tr>
</tbody>
</table>

The following restrictions, requirements, and interactions apply to the PI System Data Archive:

- **Restriction**: This data set option is not valid for reading from Pipoint.
- **Restriction**: When you access PI System data from a UNIX environment (using the PI Web API), you can use the COUNT= option only for the Picomp_Summary table.
- **Requirement**: The START= and END= options are required when you use this data set option.
- **Interaction**: If you specify COUNT=, do not specify INTERVAL=, MIN=, or MAX= data set options.

The following restriction and interactions apply to the PI System Asset Framework:

- **Restriction**: This option does not apply when you access PI System data by using the PI Web API.
- **Interaction**: This option can be used with the MODEL=, START=, END=, and BOUNDARY= options.
- **Interaction**: This option applies when you specify either MODEL=COUNT or MODEL=PLOT.

**Note**: When you specify MODEL=PLOT and a small value for COUNT=, you might receive more records than expected. The PI System supplies enough values to reproduce the shape of the underlying data in a plot. For each interval, the data is examined, and notable values are returned. These notable values include up to five unique values—the first, last, maximum, minimum, and exceptional (bad status or bad digital state) values.

---

**DBCONDITION= Data Set Option**

Specifies criteria for subsetting and ordering DBMS data.

**Valid in**: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
**Syntax**

**DBCONDITION=**"DBMS-SQL-query-clause"

**Required Argument**

**DBMS-SQL-query-clause**

specifies a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

**Details**

When you use this option to specify selection criteria in the form of DBMS-specific SQL query clauses, performance is often enhanced because the SAS/ACCESS engine passes these clauses directly to the DBMS for processing. The DBMS checks the criteria for syntax errors when it receives the SQL query.

**Example**

In this example, the function that is passed to the DBMS with the DBCONDITION= option causes the DBMS to return to SAS only those rows that satisfy the condition.

```sql
proc sql;
  create view educate as
    select name from mydb.companies
    (dbcondition="where category_code = 'education'"
    )
    using libname mydb mongo user=myusr1 pwd=mypwd1
    server=mysrv1
    db=myDB;
  select name from educate;
quit;
```

---

**DBGEN_NAME= Data Set Option**

Specifies how SAS automatically renames columns to valid SAS variable names when those column names contain characters that SAS does not allow.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Category:** Data Set Control

**Default:** LIBNAME option value

**Data source:** MongoDB, Salesforce

**See:** `DBGEN_NAME= LIBNAME option`, `VALIDVARNAME= in “System Options for Nonrelational Databases”`
Syntax

\textbf{DBGEN\_NAME=\texttt{DBMS | SAS}}

Required Arguments

\textbf{DBMS}

specifies that SAS renames DBMS columns to valid SAS variable names. SAS converts any disallowed characters to underscores. If it converts a column to a name that already exists, it appends a sequence number at the end of the new name.

\textbf{SAS}

specifies that SAS converts DBMS columns with disallowed characters into valid SAS variable names. SAS uses the format \_COLn, where \( n \) is the column number, starting with 0. If SAS converts a name to a name that already exists, it appends a sequence number at the end of the new name.

Details

SAS retains column names when it reads data from DBMS tables unless a column name contains characters that SAS does not allow, such as \$ or @. SAS allows alphanumeric characters and the underscore (\_).

This option is intended primarily for National Language Support, notably converting kanji to English characters. English characters that are converted from kanji are often those that SAS does not allow. Although this option works for the single-byte character set (SBCS) version of SAS, SAS ignores it in the double-byte character set (DBCS) version. So if you have the DBCS version, you must first set VALIDVARNAME=ANY before using your language characters as column variables.

Example

If you specify DBGEN\_NAME=SAS, SAS renames a DBMS column named Dept\$Amt to \_COLn. If you specify DBGEN\_NAME=DBMS, SAS renames the Dept\$Amt column to Dept\_Amt.

\textbf{DBMASTER= Data Set Option}

Designates which table is the larger table when you are processing a join that involves tables from two different types of databases.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Default: none

Restriction: This option is ignored when outer joins are processed.

Data source: MongoDB, Salesforce

Syntax

\textbf{DBMASTER=YES}
Required Argument

**YES**

identifies which of two tables in a join operation is the larger table.

Details

You can use this option to specify which table reference in a join is the larger table. Identifying the larger table can improve performance by eliminating the processing that is normally performed to determine this information.

Example

In this example, a table from MongoDB database and a table from a Salesforce database are joined. **DBMASTER=** is set to **YES** to indicate that the MongoDB table is the larger table. The Salesforce table is the smaller table.

```sql
libname mydb mongo user=myusr1 /*database 1 */
  pwd=mypwd1 server='mysvr';
libname mydb2 sforce user=myusr1 /*database 2 */
  pwd=mypwd1;
proc sql;
  select * from mydb.bigtab(dbmaster=yes), mydb2.smalltab
  where bigtab.x=smalltab.x;
quit;
```

---

**DBMAX_TEXT= Data Set Option**

Specifies the maximum number of bytes to allocate for character columns.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Alias:** TEXTSIZE= [PI System Asset Framework]
- **Defaults:**
  - 32 [PI System Data Archive]
  - 256 [PI System Data Asset Framework]
  - 1024 [MongoDB, Salesforce]
- **Range:** 1–976 [PI System Data Archive]
- **Applies to:**
  - PI System Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Member, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary
- **Restriction:** PI System Data Archive: This data set option is not valid for reading from Pipoint.
- **Data source:** MongoDB, PI System Asset Framework, PI System Data Archive, Salesforce
- **See:** DBMAX_TEXT= LIBNAME option

**Syntax**

`DBMAX_TEXT=number-of-bytes`
Required Argument

number-of-bytes

specifies the maximum number of bytes to allocate for character columns. Data with a string value that is longer than the allocated number of bytes is truncated. If a truncation occurs when SAS reads the data, SAS issues a warning and indicates a sufficient value for DBMAX_TEXT=.

The number of bytes that are allocated for all character strings is based on the minimum of these values: the default column size or the specified value of DBMAX_TEXT=.

Note: The length of a GUID column for PI System is always 36 bytes.

Details

PI System Data Archive: This option controls the length of the Value and Status columns only.

DBNULL= Data Set Option LIBNAME Statement Option

Specifies whether NULL is a valid value for the specified columns when a table is created.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: YES
Data source: Salesforce

Syntax

DBNULL= YES | NO

Syntax Description

YES

specifies that the NULL value is valid for the specified columns in the DBMS table.

NO

specifies that the NULL value is not valid for the specified columns in the DBMS table.

Details

This option is valid only for creating DBMS tables. If you specify more than one column name, you must separate them with spaces.

The DBNULL= option processes values from left to right. If you specify a column name twice, the last value overrides the first value that you specified for the column.
DBPROMPT= Data Set Option LIBNAME Statement Option

Specifies whether SAS displays a dialog box that prompts the user to enter DBMS connection information before connecting to the DBMS in interactive mode.

Valid in: DATA and PROC steps
Category: Data Set Control
Default: LIBNAME option setting
Restriction: The maximum password length for most of the SAS/ACCESS LIBNAME interfaces is 32 characters.
Interaction: The DBPROMPT= option interacts with the DEFER= LIBNAME option to determine when the prompt dialog box appears. If DEFER=NO, the DBPROMPT dialog box appears when the LIBNAME statement is executed. If DEFER=YES, the DBPROMPT dialog box appears when you first open a table or view. The DEFER= option normally defaults to NO, but it defaults to YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

Data source: Salesforce
See: DBPROMPT= LIBNAME option, DEFER= LIBNAME option

Syntax

DBPROMPT=YES | NO

Syntax Description

YES
 specifies that SAS displays a dialog box that interactively prompts you for the DBMS connection options the first time the libref is used.

NO
 specifies that SAS does not display the prompting dialog box.

Details

This data set option is supported only for view descriptors.

DBSASLABEL= Data Set Option

Specifies how the engine returns column labels.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: COMPAT
Data source: MongoDB, Salesforce
See: DBSASLABEL= LIBNAME option
Syntax

**DBSASLABEL**=COMPAT | DBMS | NONE

**Required Arguments**

**COMPAT**

specifies that the labels returned should be compatible with what the application normally receives. In other words, engines exhibit their normal behavior.

**DBMS**

specifies that the engine returns a label exactly as it is stored in the database.

**NONE**

specifies that the engine does not return a column label. The engine returns blanks for the column labels.

**Details**

By default, the SAS/ACCESS interface for your DBMS generates column labels from column names instead of from the real column labels.

You can use this option to override the default behavior. It is useful for when PROC SQL uses column labels as headings instead of column aliases.

**Example: Return Blank Labels for Aliases in Headings**

This example shows how to use **DBSASLABEL**= to return blank column labels so that PROC SQL can use the column aliases as the column headings.

```sql
proc sql;
    select deptno as Department ID, loc as Location
    from mylib.dept(dbsaslabel=none);
    <additional code...>
quit;
```

When **DBSASLABEL**=NONE, PROC SQL ignores the aliases, and it uses DEPTNO and LOC as column headings in the result set.

---

**DBSASTYPE**= Data Set Option

Specifies data types that override the default SAS data types during input processing.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Set Control</td>
</tr>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>Data source:</td>
<td>MongoDB, Salesforce</td>
</tr>
</tbody>
</table>
Syntax

\texttt{DBSASTYPE=} (\texttt{column-name-1} = \langle > \texttt{SAS-data-type} \langle > \ldots \texttt{column-name-n} = \langle > \texttt{SAS-data-type} \langle > )

Required Arguments

\textit{column-name-1}  
- specifies a DBMS column name.

\textit{SAS-data-type}  
- specifies a SAS data type, which can be \texttt{CHAR(n)}, \texttt{NUMERIC}, \texttt{DATETIME}, \texttt{DATE}, \texttt{TIME}. See the DBMS-specific reference section for your \texttt{SAS/ACCESS} interface for details.

Details

By default, the \texttt{SAS/ACCESS} interface for your DBMS converts each DBMS data type to a SAS data type during input processing. When you need a different data type, you can use this option to override the default and assign a SAS data type to each specified DBMS column. Some conversions might not be supported. In that case, SAS prints an error to the log.

If you convert a long string value to the \texttt{NUMERIC} type, the numeric value that is stored in SAS might not exactly match the original character value. This happens with long strings that contain more than 15 significant digits. For example, if SAS reads in a character value of ‘123456789012345678901234567890’ and converts that to type \texttt{NUMERIC}, then the numeric value that SAS stores is 12345678901234600000000000000. For more information, see “Choosing Your Degree of Numeric Precision” in \texttt{SAS/ACCESS for Relational Databases: Reference}.

Examples:

Example 1: Override the Default Data Type

In this example, \texttt{DBSASTYPE=} specifies a data type to use for the \texttt{Mycolumn} column when SAS prints DBMS data. SAS can print the values if the data in this DBMS column is stored in a format that SAS does not support, such as \texttt{SQL\_DOUBLE(20)}.

```sas
proc print data=mylib.mytable
dbsastype=(mycolumn='CHAR(20)');
run;
```

Example 2: Convert Column Length

In the next example, data that is stored in the DBMS \texttt{Fibersize} column has a data type that provides more precision than SAS can accurately support, such as \texttt{DECIMAL(20)}. If you use only \texttt{PROC PRINT} on the DBMS table, the data might be rounded or displayed as a missing value. So you could use \texttt{DBSASTYPE=} instead to convert the column so that the length of the character field is 21. The DBMS performs the conversion before the data is brought into SAS, so precision is preserved.

```sas
proc print data=mylib.specprod
dbsastype=(fibersize='CHAR(21)');
```
Example 3: Append Tables to Match Data Types

The next example uses DBSASTYPE= to append one table to another when the data types cannot be compared. If the Empid variable in the SAS data set is defined as CHAR(20) and the Empid column in the DBMS table is defined as DECIMAL(20), you can use DBSASTYPE= to make them match:

```sas
proc append base=dblib.hrdata (dbsastype=(empid='CHAR(20)'))
data=saslib.personnel;
run;
```

DBSASTYPE= specifies to SAS that the Empid is defined as a character field of length 20. When a row is inserted from the SAS data set into a DBMS table, the DBMS performs a conversion of the character field to the DBMS data type DECIMAL(20).

---

**DELAY= Data Set Option**

Specifies the number of seconds to wait before SAS begins to read data.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Access
- **Default:** 0
- **Data source:** PI System Data Archive

**Syntax**

```
DELAY=seconds
```

**Required Argument**

*seconds*

specifies the number of seconds to wait before SAS begins to read the data. When data is added to a tag, it often takes up to a second before data is available to read. This option facilitates cases where data is added to a tag and is immediately read back.

---

**END= Data Set Option**

Specifies the timestamp of the last tag value to read in the form of a PI System timestamp string.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Alias:** END_TIME=
- **Default:** 12/31/9999 11:59:59 PM
- **Applies to:** Asset Framework tables Eventframe, Eventframe_Attribute, Tsdata_Summary
Data source: PI System Asset Framework, PI System Data Archive

See: START= data set option

Syntax

END="end-time"

Required Argument

end-time

specifies the timestamp of the last tag value to read in the form of a PI System
timestamp string. Enclose the value in quotation marks.

Details

Processing for this option is performed in the PI System.

Specifying END="12/23/2014" implies a time of "00:00:00". Specifying a date with
no time component for the END= data set option includes records up to and
including midnight the day before. To include a given day, specify either the next day
with the END data set option or explicitly use END="12/23/2014 11:59:59 PM".

To ensure a correct end time, specify end-time using the same date format as your
system locale setting. For example, in the United States, you would specify
'7/1/2014' for July 1, 2014. In France, you would specify '1.7.2014'.

PI System Asset Framework: When you use START= and END= with event frames,
SAS returns all of the records from event frames that overlap the START= and
END= time period. This means that you are likely to receive records outside of the
specified time period.

FILTER= Data Set Option

Specifies filter expression that is used to limit the records that are read.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Default: none

Restriction: PI System Data Archive: This data set option is not valid for reading from Pipoint.

Interaction: PI System Data Archive: If you specify FILTER=, do not specify MAX=, MIN=, and
SELECT= data set options.

Data source: PI System Data Archive

Tip: To create more complex filters, refer to the Performance Equations Reference Manual
from OSIsoft.

Example: FILTER= (*.' >= 10) AND (*.' <= 50)*
Syntax

FILTER="filter-expression"

Required Argument

/filter-expression

specifies a PI System filter expression that is used to limit the records that are read.

GUID= Data Set Option

Specifies the element ID to use for retrieving an element.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none
Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tdata_Attribute, Tdata_Pipoint
Requirement: This option is required when you specify SEARCHBY=GUID.
Data source: PI System Asset Framework
See: SEARCHBY= data set option

Syntax

GUID=element-ID

Required Argument

element-ID

specifies the element ID to use for retrieving an element.

HIDEFLAGS= Data Set Option

Specifies whether the columns Questionable, Annotated, and Substituted are hidden.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: NO
Restriction: This data set option is not valid for reading from Pipoint.
Data source: PI System Data Archive
See: HIDEFLAGS= LIBNAME option, SHOWFLAGS= data set option
Syntax

**HIDEFLAGS=** YES | NO

Required Arguments

**YES**

does not display the Questionable, Annotated, and Substituted columns.

**NO**
displays all columns.

---

**HIDEOPTVARS= Data Set Option**

Specifies whether to exclude nonessential variables in the data set.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Default: NO

Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Member, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary

Data source: PI System Asset Framework

See: HIDEOPTVARS= LIBNAME option

---

Syntax

**HIDEOPTVARS=** YES | NO

Required Arguments

**YES**

specifies to exclude nonessential variables in the data set. For more information about which variables are hidden when HIDEOPTVARS=YES, see “About the Asset Framework Virtual Tables” on page 150.

**NO**

specifies not to exclude optional variables. To include all of the variables in a data set, specify HIDEOPTVARS=NO.

---

**INTERVAL= Data Set Option**

Specifies the length of the time interval to use when displaying data.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

Category: Data Set Control

Alias: INT=

Default: See Details.
Applies to: Asset Framework tables Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary

Restriction: Data Archive: This data set option is not valid for reading from Pipoint.

Interactions: Data Archive: If you specify INTERVAL=, do not specify COUNT=, MIN=, or MAX= data set options.
Asset Framework: This data set option can be used with the MODEL=, START=, END=, and BOUNDARY= options.

Windows specifics: When you access PI System data from a Microsoft Windows environment, the limit for the number of returned records is 1,500,000.

UNIX specifics: When you use the PI Web API to access PI System data from a UNIX environment, the limit for the number of returned records is 150,000.

Data source: PI System Asset Framework, PI System Data Archive

See: BOUNDARY= data set option, COUNT= data set option, END= data set option, MAX= data set option, MIN= data set option, MODEL= data set option, START= data set option

Example: "1:00:00.000" is one hour.

Syntax

**INTERVAL=**"interval-length"

Required Argument

**interval-length**

specifies the time interval to use when retrieving Pipoint values. You can specify days, hours, minutes, seconds, and milliseconds, and the resulting time intervals are evenly spaced. The value that you specify must be a PI System time interval.

Typically, you use this data set option with the START= and END= data set options. If you do not specify values for START= and END=, the PI System generates results from the earliest recorded record. This could result in a very large number of records.

When you specify a positive interval, interval calculation begins at the earliest point in the specified time period and intervals are calculated moving forward in time. If you specify a negative interval, then interval calculation begins at the end of the time period and intervals are calculated moving backward in time. The order of returned values still begins at or near the start time and moves forward in time, whether you specify a positive or negative interval.

Details

Processing for this option is performed in the PI System.
Table 2.2  Default Values for the INTERVAL= Data Set Option

<table>
<thead>
<tr>
<th>PI System Interface</th>
<th>Table</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Archive</td>
<td>Picomp</td>
<td>Do not interpolate. Display all recorded events between the START= and END= times.</td>
</tr>
<tr>
<td></td>
<td>Picomp_Summary</td>
<td>Create only one interval that covers the time spanned by START= and END= values.</td>
</tr>
<tr>
<td>Asset Framework</td>
<td>any</td>
<td>Do not interpolate. Display all recorded events between the START= and END= times.</td>
</tr>
</tbody>
</table>

PI System Data Archive: For Picomp, interval-length specifies to interpolate data using the specified time length intervals. For Picomp_Summary, interval-length specifies the sample interval for display between the START= and END= times. For example, if START= and END= span 24 hours, then INTERVAL='1:00:00' results in 24 one-hour intervals.

**LABEL= Data Set Option: PI System**

Specifies a description of a tag that is created in SAS.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Aliases:** DESC=
- **Default:** none
- **Data source:** PI System Data Archive

**Syntax**

```
LABEL="string"
```

**Required Argument**

**string**

specifies a description of a tag that is created in SAS. This value is used in the descriptor string for a tag when you write to the PI System. Use this option only when you create a new tag or when you modify an existing tag that you created by using SAS. Otherwise, the description is ignored.
**LABEL= Data Set Option: Salesforce**

Specifies that a SAS variable label is used as the output DBMS column name.

- **Valid in:** DATA and PROC steps
- **Category:** Data Set Control
- **Default:** none
- **Data source:** Salesforce

**Syntax**

```
LABEL='label'
```

**Syntax Description**

*label*

Use a SAS variable label as the output DBMS column name.

---

**MAX= Data Set Option**

Specifies the maximum value to read from the Value column.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Alias:** MAXVAL=
- **Default:** none (Do not filter.)
- **Restriction:** This data set option is not valid for reading from Pipoint.
- **Interaction:** If you specify MAX=, do not specify the COUNT=, FILTER, or INTERVAL= data set options.
- **Data source:** PI System Data Archive
- **See:** COUNT= data set option, INTERVAL= data set option, MIN= data set option

**Syntax**

```
MAX=value
```

**Required Argument**

*value*

specifies the maximum value to read from the Value column in PI System data.
**MEDIAN= Data Set Option**

For the PI System, specifies whether to calculate median statistics (Median and MedianCount) in the Picomp_Summary table.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Default:** NO
- **Restriction:** Median statistics are not supported when you use the PI Web API to access the PI System.
- **Interaction:** When you calculate median statistics, the only valid values for CALCULATION_BASIS are EventWeighted or TimeWeighted.
- **Data source:** PI System Data Archive
- **See:** CALCULATION_BASIS= data set option

**Syntax**

 MEDIAN= YES | NO | count

**Required Arguments**

- **YES**
  - specifies to read required data into SAS and calculate median statistics. SAS interpolates 1000 data points over the summary interval or one data point per second over the specified summary interval, whichever is smaller.
  - Calculation of median statistics requires reading all required data records into SAS/ACCESS, which can significantly increase computation time.

- **NO**
  - specifies that SAS should not calculate median statistics.

- **count**
  - specifies to read required data into SAS and calculate median statistics. SAS interpolates count data points over the summary interval and then calculates the median per interval.
  - Setting MEDIAN=1 is equivalent to setting MEDIAN=YES. Setting MEDIAN=0 is equivalent to setting MEDIAN=NO.

---

**MIN= Data Set Option**

Specifies the minimal value to read from the Value column.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Alias:** MINVAL=
Default: none (Do not filter.)
Restriction: This data set option is not valid for reading from Pipoint.
Interaction: If you specify MIN=, do not specify the COUNT=, FILTER=, and INTERVAL= data set options.
Data source: PI System Data Archive
See: COUNT= data set option, INTERVAL= data set option, MAX= data set option

Syntax
**MIN=value**

Required Argument

**value**

specifies the minimal value to read from the Value column.

---

**MIXED= Data Set Option**

Specifies whether to merge data sets with different data types for Value with the Picomp virtual table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: NO
Data source: PI System Data Archive

Syntax
**MIXED=YES | NO**

Required Arguments

**YES**

specifies whether to merge data sets with different data types for Value in the Picomp virtual table. The data sets being merged with Picomp are listed in TAGLIST. If the data sets in TAGLIST have different data types for Value, then you must set MIXED=YES. Setting MIXED=YES converts all numeric values into strings if mixed numeric and string values are detected.

**NO**

do not merge data sets with different data types for Value in the Picomp virtual table.
MODEL= Data Set Option

Specifies the data model to use to retrieve time series data values. This setting applies to the attribute of the PI Point value that is retrieved.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Alias: DATAMODEL=
Default: none
Applies to: Asset Framework tables Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint
Data source: PI System Asset Framework
See: BOUNDARY= data set option, COUNT= data set option, END= data set option, INTERVAL= data set option, START= data set option

Syntax

MODEL=ARCHIVE | COUNT | INTERPOLATE | PLOT

Required Arguments

ARCHIVE
returns a list of compressed values for the requested time range.

Alias ARCHIVE_RANGE
Requirement This value requires use of the START= and END= options.
Interaction When MODEL=ARCHIVE, do not use the INTERVAL= or COUNT= options.

COUNT
returns a specified number of compressed values from the time period that you indicate with the START= option.

Alias ARCHIVE_COUNT
Requirement This value requires the START= and COUNT= options.
Interaction When MODEL=COUNT, do not use the END= or INTERVAL= data set options.

INTERPOLATE
retrieves interpolated values over the specified time period, based on the sampling interval that you specify.

Alias SAMPLED
Requirement This value requires the START=, END=, and INTERVAL= options.
Interaction

When MODEL=INTERPOLATE, do not use the COUNT= or BOUNDARY= options.

**PLOT**
returns the specified number of interpolated values, evenly spaced throughout the requested time period.

**Alias**
PLOT_COUNT

**Requirement**
This value requires the START=, END=, and COUNT= options.

Interaction

When MODEL=PLOT, do not use the INTERVAL= or BOUNDARY= options.

---

**NAME= Data Set Option**

Specifies the element name to use for retrieving elements.

**Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)

**Category:** Data Set Control

**Default:** none

**Applies to:** Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint

**Interaction:** This option is required when you specify SEARCHBY=NAME.

**Data source:** PI System Asset Framework

**See:** SEARCHBY= data set option

---

**Syntax**

**NAME=</element-name>**

**Required Argument**

**element-name**

specifies the element name to use for retrieving elements. Enclose the element-name value in quotation marks if the name contains characters other than alphanumeric characters or underscores ('_').

You can specify a character pattern for the element-name value that includes wildcard characters. You can use the backslash ('\') to escape a wildcard character, or use a double backslash ('\\') to match a single backslash character.

Character pattern matching follows these rules:

- An empty string matches all element names.
- Characters and numbers must match with an element name exactly.
- The * wildcard character can be placed anywhere in a pattern, and it corresponds to zero or more characters in a pattern.
- The ? wildcard character can be placed anywhere in a pattern, and it corresponds to exactly one character.
You can specify multiple characters at a single position by placing the possible characters within [ and ] symbols. For example, a[bc] matches either ab or ac. This pattern does not match ad or abd.

You can exclude multiple characters at a single position by placing them within [ and ] symbols and by preceding them with the ! character. For example, a[!bc] matches ad, but it would not match ab, ac, or abd.

**PATH= Data Set Option**

Specifies the full path to use for retrieving an element and any children.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Default:** none
- **Applies to:** Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint
- **Requirement:** This option is required when you specify SEARCHBY=PATH.
- **Interaction:** When you specify the PATH= option, you can also specify the CHILD= option. The CHILD= option is not required.
- **Data source:** PI System Asset Framework
- **See:** SEARCHBY= data set option
- **Example:** Specify the PATH= option similar to this example:

  ```
  PATH="\\myServer\myDB\MainProcess\myState\Equipment"
  ```

**Syntax**

```
PATH="path-value"
```

**Required Argument**

`path-value`

specifies the full path to use for retrieving an element and any children. The value for `path-value` must end in a valid element name.

**PRESERVE_COL_NAMES= Data Set Option**

Preserves spaces, special characters, and case sensitivity in DBMS column names when you create DBMS tables.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Default:** LIBNAME option setting
- **Data source:** Salesforce
See: PRESERVE_COL_NAMES=LIBNAME option, PRESERVE_TAB_NAMES= LIBNAME option, SAS Names and Support for DBMS Names, VALIDVARNAME= in “System Options for Nonrelational Databases”

Syntax

PRESERVE_COL_NAMES=YES | NO

Syntax Description

YES

specifies that column names that are used in table creation are passed to the DBMS with special characters and the exact, case-sensitive spelling of the name is preserved.

NO

specifies that column names that are used to create DBMS tables are derived from SAS variable names (VALIDVARNAME= system option) by using the SAS variable name normalization rules. However, the database applies its DBMS-specific normalization rules to the SAS variable names when creating the DBMS column names.

The use of N-literals to create column names that use database keywords or special symbols other than the underscore character might be invalid when DBMS normalization rules are applied. To include nonstandard SAS symbols or database keywords, specify PRESERVE_COL_NAMES=YES.

Details

When you create a table, you assign the column names by using one of these methods.

- To control the case of the DBMS column names, specify variables using the case that you want and set PRESERVE_COL_NAMES=YES. If you use special symbols or blanks, you must set VALIDVARNAME= to ANY and use N-literals. For more information, see the SAS/ACCESS naming topic in the DBMS-specific reference section for your interface in this document and also SAS Data Set Options: Reference.

- To enable the DBMS to normalize the column names according to its naming conventions, specify variables using any case and set PRESERVE_COL_NAMES= NO.

The Salesforce engine automatically converts all schema, table, and column names to lowercase.

To save some time when coding, specify the PRESERVE_NAMES= alias if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement.

To use column names that are not valid SAS names in your SAS program, you must use one of these techniques.

- Use the DQUOTE= option in PROC SQL and reference your columns using double quotation marks, as shown in this example.

```sql
proc sql dquote=ansi;
    select "Total\$Cost" from mydblib.mytable;
```
Specify the global system option VALIDVARNAME=ANY and use name literals in the SAS language, as shown in this example.

```sas
proc print data=mydblib.mytable;
  format 'Total$Cost'n 22.2;
```

If you are creating a table in PROC SQL, you must also include the PRESERVE_COL NAMES=YES option in your LIBNAME statement, as shown in this example.

```sas
libname mydblib oracle user=myusr1 password=mypwd1 
  preserve_col_names=yes;
```

**ROWSET_SIZE= Data Set Option**

Specifies the number of rows of DBMS data to read into the buffer.

- **Valid in:** DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
- **Category:** Data Set Control
- **Alias:** READBUFF=
- **Default:** LIBNAME option value
- **Restrictions:** When ROWSET_SIZE=1, only one row is retrieved at a time.

Buffering data reads can decrease network activities and increase performance. However, because SAS stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, rows that are returned to the SAS application might be out of date. For example, if someone else modifies the rows, you do not see the changes.

- **Data source:** MongoDB, Salesforce
- **Tip:** This option improves performance by specifying a number of rows that can be held in memory for input into SAS.
- **See:** ROWSET_SIZE= LIBNAME option

**Syntax**

```
ROWSET_SIZE= integer
```

**Required Argument**

*integer*

specifies the number of rows to hold in memory. SAS allows the maximum number that the DBMS allows.

**SASDATEFMT= Data Set Option**

Changes the SAS date format of a DBMS column.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none
Restriction: The column names specified in this option must be DATE, DATETIME, or TIME columns. Columns of any other type are ignored.
Requirement: The SAS date format specified must be a valid date format.
Data source: MongoDB, Salesforce
See: DBSASTYPE= data set option

Syntax

\[ \text{SASDATEFMT=} (DBMS-date-column-1=\langle SAS-date-format \rangle < DBMS-date-column-n=\langle SAS-date-format \rangle >) \]

Required Arguments

- **DBMS-date-column**
  specifies the name of a date column in a DBMS table.

- **SAS-date-format**
  specifies a SAS date format that has an equivalent (like-named) informat. For example, DATETIME21.2 is both a SAS format and a SAS informat, so it is a valid value for the **SAS-date-format** argument.

Details

If the SAS column date format does not match the date format of the corresponding DBMS column, convert the SAS date values to the appropriate DBMS date values. Use the **SASDATEFMT**= option to convert date values from the default SAS date format to another SAS date format that you specify.

Use the **SASDATEFMT**= option to prevent date type mismatches during input operations to convert DBMS date values to the correct SAS DATE, TIME, or DATETIME values.

- during input operations to convert DBMS date values to the correct SAS DATE, TIME, or DATETIME values.
- during output operations to convert SAS DATE, TIME, or DATETIME values to the correct DBMS date values.

If the SAS date format and the DBMS date format match, this option is not needed.

Note: For non-English date types, SAS automatically converts the data to the SAS type of NUMBER. **SASDATEFMT**= does not currently handle these date types. However, you can use a PROC SQL view to convert the DBMS data to a SAS date format as you retrieve the data, or use a format statement in other contexts.
Example: Change a DBMS Date Format to a SAS Date Format

In this example, SASDATEFMT= converts Date1, a MongoDB date column, to a column named Date1 that has a SAS DATETIME format.

```sas
libname x mongo user=myusr1 psd=mypwd1;
data sas_local;
format date1 datetime21.;
set x.dateinfo( sasdatefmt=( date1='datetime21.' ) );
run;
```

SEARCHBY= Data Set Option

Specifies the search method that locates element objects that are defined in the target server and database.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none
Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint
Restriction: Do not specify any other search options when you use the SEARCHBY= data set option. The only exception is the ATTRIBUTE= option, if this is allowed for the virtual table that you specify.
Requirement: This option is a required option for virtual tables.
Data source: PI System Asset Framework
See: ATTRIBUTE= data set option, CATEGORY= data set option, GUID= data set option, NAME= data set option, PATH= data set option, TEMPLATE= data set option

Syntax

```
SEARCHBY=ATTRIBUTE | CATEGORY | GUID | NAME | PATH | TEMPLATE
```

Required Arguments

**ATTRIBUTE**
specifies to search for an element based on attributes.

Requirement: This value requires that you also specify the ATTRIBUTE= data set option.

**CATEGORY**
specifies to search for an element object based on a category.

Requirement: This value requires that you also specify the CATEGORY= data set option.

**GUID**
specifies to search for an element object by its unique ID.
**SELECT= Data Set Option**

Specifies which values are displayed.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Set Control</td>
</tr>
<tr>
<td>Default:</td>
<td>ALL</td>
</tr>
<tr>
<td>Restriction:</td>
<td>This data set option is not valid for reading from Pipoint.</td>
</tr>
<tr>
<td>Interaction:</td>
<td>If you specify SNAP or SNAPSHOT, do not specify the COUNT=, END=, FILTER=, INTERVAL=, MIN=, MAX=, or START= data set options.</td>
</tr>
<tr>
<td>Data source:</td>
<td>PI System Data Archive</td>
</tr>
<tr>
<td>See:</td>
<td>COUNT= data set option, END= data set option, INTERVAL= data set option, MAX= data set option, MIN= data set option, START= data set option</td>
</tr>
</tbody>
</table>

**Syntax**

`SELECT=ALL | GOOD | BAD | SNAPSHOT`

**Required Arguments**

**ALL**

displays all records.

**GOOD**

displays only records with a Status column value of OK.

**BAD**

**SNAPSHOT**

**Syntax**

`SELECT=ALL | GOOD | BAD | SNAPSHOT`

**Required Arguments**

**ALL**

displays all records.

**GOOD**

displays only records with a Status column value of OK.

**BAD**

**SNAPSHOT**

**Alias**

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This value requires that you also specify the GUID= data set option.</td>
</tr>
</tbody>
</table>

**NAME**

specifies to search for an element object by name.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>This value requires that you also specify the NAME= data set option.</td>
</tr>
</tbody>
</table>

**PATH**

specifies to search for an element object based on its path.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>This value requires that you also specify the PATH= data set option.</td>
</tr>
</tbody>
</table>

**TEMPLATE**

specifies to search for an element object based on a template.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>This value requires that you also specify the TEMPLATE= data set option.</td>
</tr>
</tbody>
</table>
BAD
  displays only records that do not have a Status column value of OK.
  Alias  ERROR

SNAPSHOT
  displays the last snapshot record.
  Alias  SNAP

SHOWFLAGS= Data Set Option
Specifies whether the columns Questionable, Annotated, and Substituted display.
Valid in:  DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category:  Data Set Control
Default:  YES
Restriction:  This data set option is not valid for reading from Pipoint.
Data source:  PI System Data Archive
See:  HIDEFLAGS= data set option, SHOWFLAGS= LIBNAME option

Syntax
SHOWFLAGS=YES | NO

Required Arguments
YES
  displays the Questionable, Annotated, and Substituted columns and other columns.

NO
  displays only the Tag, Timestamp, Value, and Status columns.

SHOWINDEX= Data Set Option
Specifies whether to include column _index from the Picomp table.
Valid in:  DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category:  Data Set Control
Default:  NO
Data source:  PI System Data Archive
Syntax
SHOWINDEX=YES | NO

Required Arguments
YES
specifies whether to include column _index from the Picomp table that indicates the index that corresponds to a timestamp. The index value is 1 when there is only one observation for a given timestamp. The index value increments by one for each additional observation that is recorded with the same timestamp.

NO
specifies not to include the column _index from the Picomp table.

Details
The _index column enables you to create a unique key for each observation.
Two observations are considered to occur at the same time when their timestamps differ by less than 0.001 seconds (1 ms).

START= Data Set Option
Specifies the timestamp of the earliest tag value to read in the form of a PI System TIMESTAMP string.
Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Alias: START_TIME= [Data Archive]
Default: 1/1/1970 12:00:00.001 AM GMT
Applies to: Asset Framework tables Eventframe, Eventframe_Attribute, Tsdata_Summary
Restrictions:
Data Archive: If SAS is running on a machine with a positive UTC time zone offset (east of London), do not specify a start time that would convert to a time prior to 1/1/1970 12:00:00.001 AM GMT. As a rule, do not specify a start time before 1/2/1970.
Data Archive: This data set option is not valid for reading from Pipoint.
Data source: PI System Asset Framework, PI System Data Archive
See: END= data set option

Syntax
START="start-time"

Required Argument

start-time
specifies the timestamp of the earliest tag value to read in the form of a PI System TIMESTAMP string. Enclose the value in quotation marks.
Details

Processing for this option is performed in the PI System.

The earliest start time allowed by the PI System is one millisecond after midnight of 1/1/1970 GMT. Time values are interpreted as local time unless a time zone indicator, such as a trailing “Z” or “GMT”, indicates a different time zone. For example, you might supply any of the following values: "17Oct2014", '10/16/2014 12:00:00', "2014-10-17Z", or '2014-10-17 12:00:00gmt'.

You can also specify relative time formats that use "T" (today), “Y” (yesterday), “*” (now), “D” (days), or “H” (hours). You can specify the letters in upper or lowercase. Here are some examples that use relative time formats:

- "T" today (at time 00:00:00.000)
- "Y" yesterday (at time 00:00:00.000)
- "*-7d" Seven days ago at the current time
- "*-1h" One hour before the current time (one hour ago)

If the START time is before the END time, the resulting values are retrieved in time-ascending order. Otherwise, values are retrieved in time-descending (reverse) order.

To ensure a correct start time, specify start-time using the same date format as your Windows system locale setting. For example, in the United States, you would specify '7/1/2014' for July 1, 2014. In France, you would specify ‘1.7.2014’.

PI System Asset Framework: When you use START= and END= with event frames, SAS returns all of the records from event frames that overlap the START= and END= time period. This means that you are likely to receive records outside of the specified time period.

TAGFILTER= Data Set Option

Specifies a character string that is used to filter tags when reading from the Pipoint virtual table.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: *

Data source: PI System Data Archive

Syntax

TAGFILTER="string"

Required Argument

string

specifies a character string that is used to filter tags when reading from the Pipoint virtual table. Wildcard characters * and ? are allowed:

- * represents zero or more of any character.
- ? represents any single character.
TAGLIST= Data Set Option

Specifies a SAS data set name that is used for the selection process when reading from Picomp.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none

Data source: PI System Data Archive

Examples:
proc print data=db.picomp (taglist=mytags); run;
proc print data=db.picomp (taglist=list (select=snap)); run;

Syntax

**TAGLIST=** *data-set-name* <( *data-set-options*)>

Required Arguments

*data-set-name* specifies a SAS data set name that is used for the selection process when reading from Picomp. The specified data set can contain one or more tag names in the Tag column. If there is no Tag column in the data set specified by the TAGLIST= option, the first column is treated as the Tag column and a warning is printed to the log.

*data-set-options* specifies optional data set options to apply to the TAGLIST= data set. Parentheses are not required around data set options, but are recommended for readability.

TAGLIST_JOIN= Data Set Option

Specifies whether to perform a join between the requested table (Picomp or Picomp_Summary) and the table that is listed in TAGLIST.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Alias: TAGLISTJOIN=
Default: NO
Data source: PI System Data Archive
See: **TAGLIST=** data set option
Syntax

TAGLIST_JOIN=YES | NO

Required Arguments

YES
specifies to perform a join between the requested table (Picomp or Picomp_Summary) and the table that is listed for TAGLIST=. The system joins the regular column layout of the requested table with the data columns from the table for TAGLIST=.

NO
specifies not to join the requested table and the table that is listed for TAGLIST=.

TEMPLATE= Data Set Option

Specifies the template to use for retrieving elements.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Default: none
Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary
Requirement: This option is required when you specify SEARCHBY=TEMPLATE.
Data source: PI System Asset Framework
See: SEARCHBY= data set option

Syntax

TEMPLATE=template-name

Required Argument

template-name
specifies the template to use for retrieving elements.

TIMEZONE= Data Set Option

Specifies whether SAS reads and writes PI System timestamps using local time or GMT.

Valid in: DATA and PROC steps (when accessing DBMS data using SAS/ACCESS software)
Category: Data Set Control
Alias: TZ= [Data Archive]
Default: LOCAL
Applies to: Asset Framework tables Element, Element_Attribute, Eventframe, Eventframe_Attribute, Eventframe_Tsdata, Member, Tsdata_Attribute, Tsdata_Pipoint, Tsdata_Summary

Data source: PI System Asset Framework, PI System Data Archive

See: TIMEZONE= LIBNAME option

Syntax

```
TIMEZONE= LOCAL | GMT
```

Required Arguments

- **LOCAL**
  - specifies to use the local time to read and write timestamps in SAS.

- **GMT**
  - specifies to use GMT to read and write timestamps in SAS.

Alias: UTC

Details

The PI System stores timestamps using GMT. For example, in the PI System, a numeric value of 0 in GMT is 01JAN1970:00:00:00. In Eastern Time, that GMT value is 31DEC1969:19:00:00.

**CAUTION**

Data might be sequenced incorrectly for PI tags that are read between 1 A.M. and 2 A.M. when changing from daylight to standard time. This can cause differences when importing and exporting tag data. Timestamps in the range between 1 A.M. and 2 A.M. cannot be properly converted back to UTC, which might result in the data values in that time range being sequenced differently.
This section describes the system options that you can use with SAS/ACCESS interfaces to nonrelational databases. Most SAS system options are available for use with SAS/ACCESS interfaces. The system options that are included in this chapter have behaviors that are specific to one or more of the nonrelational databases for SAS/ACCESS.

For more information about all SAS system options, see SAS System Options: Reference. System options that are discussed in this chapter include links to more information about each option.

This table describes the system options that behave uniquely for SAS/ACCESS interfaces to nonrelational databases.
Table 3.1 System Options for Nonrelational Databases

<table>
<thead>
<tr>
<th>System Option</th>
<th>Notes for Usage with Nonrelational Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEZONE=</td>
<td>Using the TIMEZONE= system option is important when you work with UTC datetime values. For Salesforce and the PI System, specifying the appropriate TIMEZONE= value is key when you convert historical UTC datetime values to local datetime values. For more information, see “Using the TIMEZONE= System Option”.</td>
</tr>
<tr>
<td>VALIDVARNAME=</td>
<td>Controls the type of SAS variable names that can be used or created during a SAS session. For MongoDB, when VALIDVARNAME=V7, the value of Label is set to the variable name after the variable name is changed to a valid SAS name. For more information, see “VALIDVARNAME= SAS System Option” in SAS/ACCESS for Relational Databases: Reference.</td>
</tr>
</tbody>
</table>

Using the TIMEZONE= System Option

Datetime values for the PI System and for Salesforce are stored as Coordinated Universal Time (UTC) values. For Salesforce, datetime values are always represented as local datetime values. For the PI System, you can choose to represent datetime values as local datetime values. In countries that change between standard time and daylight saving time during the year, a discrepancy in datetime values can occur when historical data is converted from UTC to local time values. The time can be off by an hour, depending on the time zone that is in effect when you are working with the data.

Note: UTC time values are functionally equivalent to GMT time values, meaning that the terms ‘UTC’ and ‘GMT’ are synonymous.

To illustrate the problem, suppose that today is January 22 and you are in the eastern region of the US. Because Eastern Standard Time is in effect, the conversion from a UTC time to a local time subtracts five hours from the stored time. You read in a UTC timestamp value of ‘2019-01-20 15:00:00’ from the database. SAS subtracts five hours to convert this value to a local time of ‘2019-01-20 10:00:00’.

Now, suppose that it’s June. Daylight Saving Time is in effect, which means that the conversion from a UTC time to a local time subtracts four hours. You read in the same datetime value of ‘2019-01-20 15:00:00’. Because the conversion to local time has changed, this value is now converted to ‘2019-01-20 11:00:00’. This data point
is read as occurring one hour later than the actual time when the observation was recorded.

To correct this situation, use the "TIMEZONE= System Option" in SAS System Options: Reference. Set the TIMEZONE= system option to the time-zone-ID (specified as region and area) that corresponds to your local time zone. For our example, you might invoke SAS with this command:

```
sas -timezone "America/New_York"
```

When TIMEZONE= is set to your local time zone, UTC datetime values are converted to the appropriate date and time based on the time zone that was in effect when the observation was recorded. Therefore, our example observation is always represented as '2019-1-20 10:00:00'.

For the list of valid region and area values for the TIMEZONE= system option, see "Time Zone Information and Time Zone Names" in SAS System Options: Reference.

For the PI System, do not confuse the TIMEZONE= system option with the TIMEZONE= LIBNAME and data set options. You use the TIMEZONE= LIBNAME or data set option to indicate whether datetime values are represented as local time values or as UTC values. If you are converting UTC datetime values to local datetime values, then you also need to set the TIMEZONE= system option to the appropriate value for your local time zone.
PART 2

Nonrelational DBMS Interfaces for SAS/ACCESS

Chapter 4
SAS/ACCESS Interface to MongoDB ........................................ 85

Chapter 5
SAS/ACCESS Interface to the PI System .................................. 115

Chapter 6
SAS/ACCESS Interface to Salesforce ...................................... 163
# SAS/ACCESS Interface to MongoDB

## About SAS/ACCESS Interface to MongoDB

**Supported Features for SAS/ACCESS Interface to MongoDB**  
86

**LIBNAME Statement Specifics for SAS/ACCESS Interface to MongoDB**  
87
- Syntax  
87
- Required Arguments for MongoDB  
87
- Connection Options for MongoDB  
87
- LIBNAME Options for MongoDB  
88
- MongoDB LIBNAME Examples  
89
- LIBNAME Options and Usage for Schemas  
90

**Data Set Options for SAS/ACCESS Interface to MongoDB**  
92

**Working with Schemas for MongoDB Data**  
92
- How Does SAS/ACCESS Derive a Schema?  
92
- Importance of Specifying the SCHEMA_COLLECTION= LIBNAME Option  
94
- Working with Temporary Schemas  
95
- How Does SAS Derive Data Types?  
95
- Update a Stored Schema  
96

**Using the SAS Client to Query, Create, Update, and Delete Data in MongoDB**  
97
- Overview  
97
- Querying MongoDB Data  
98
- Creating Tables  
98
- Creating Parent and Child Tables  
99
- Special Columns for 1:n Child Tables  
100
- Special Columns for 1:1 Child Tables  
100
- Example : Creating a 1:n Child Table  
101
- Example: Creating a 1:1 Child Table  
103
- Inserting Additional Values into Parent and Child Tables  
104
- Updating MongoDB Data  
105
- Deleting MongoDB Data  
106

**SQL Pass-Through Specifics for MongoDB**  
106
- Key Information  
106

**Using MongoDB Commands to Query, Create, Update, and Delete Tables in MongoDB**  
107
- Overview of Passing Commands to MongoDB  
107
- Find and Aggregate Commands  
107
- Create a MongoDB Table  
108
- Update Data in a MongoDB Table  
109
- Delete MongoDB Tables  
109
- Example: Modifying Data in MongoDB with SQL Pass-Through  
110

**Passing SAS Functions to MongoDB**  
111
About SAS/ACCESS Interface to MongoDB

SAS/ACCESS Interface to MongoDB enables you to transfer data between your MongoDB document database and the SAS client. To access data from MongoDB, SAS/ACCESS builds a relational schema that is based on a scan of the collections and documents in MongoDB. The generated schema enables the SAS client to treat MongoDB data as a collection of tables.

Support for SAS/ACCESS Interface to MongoDB was added in April 2019. Beginning in SAS Viya 3.5, SAS/ACCESS Interface to MongoDB includes SAS Data Connector to MongoDB. The data connector enables you to transfer large amounts of data between MongoDB and the CAS server for parallel processing. For more information, see these topics:


Supported Features for SAS/ACCESS Interface to MongoDB

SAS/ACCESS Interface to MongoDB is available on Linux 64 platforms. SAS/ACCESS Interface to MongoDB enables you to perform the following tasks:

- Generate a schema that describes a relational structure for MongoDB collections. Beginning in April 2020, a schema is automatically saved for use in subsequent SAS sessions.
- Query MongoDB data.
- Create, update, and delete MongoDB collections.
- Refresh a schema to include changes made to the MongoDB database by a means other than SAS.
LIBNAME Statement Specifics for SAS/ACCESS Interface to MongoDB

Syntax

This LIBNAME statement associates a libref with a MongoDB data source and enables you to read from and write to MongoDB.

LIBNAME  libref MONGO <connection-options> <LIBNAME-options>;

Required Arguments for MongoDB

libref
specifies any SAS name that serves as an alias to associate SAS with a database or group of tables and views.

MONGO
specifies the SAS/ACCESS engine name for the MongoDB interface.

Connection Options for MongoDB

Here are the connection options that are available for the MongoDB interface:

CATALOG="catalog-name"
specifies a logical catalog name. The logical name can be any user-defined name.

  Default  none

DATABASE="database-name"
specifies the name of a MongoDB database.

  Alias  DB=
  Default  none

DRIVER="driver"
specifies the driver to use to connect to the MongoDB database.

  Default  none
  Requirement  Use quotation marks if driver-name contains spaces or nonalphanumeric characters or if the server name is an IP address.
NOPROMPT="MongoDB-connection-options"

specifies connection options for your database. Separate multiple options with a semicolon (";"). If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to prompt you for the connection string.

Default  none

Restriction  This option is not applicable in SAS Viya.

Requirement  You must specify the aliases SCHEMA_UID= (rather than SCHEMA_USER=) and SCHEMA_PWD= (rather than SCHEMA_PASSWORD=) when you provide these options in a connection string.

PORT=port-ID

specifies the port ID for your server.

Default 27017

PWD='password'

specifies the password for authentication.

SERVER="server-name"

specifies the MongoDB server to connect to.

Requirement  Use quotation marks if server-name contains spaces or nonalphanumeric characters or if the server name is an IP address.

USER="user-ID"

specifies the user ID to use for authentication.

Alias  UID=

LIBNAME Options for MongoDB

Here are the LIBNAME options that are available for the MongoDB interface.

Table 4.1  LIBNAME Options for SAS/ACCESS Interface to MongoDB

<table>
<thead>
<tr>
<th>LIBNAME Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHARED</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>LIBNAME Option</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>MAX_CHAR_LEN=</td>
<td>32767</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
</tr>
<tr>
<td>ROWSET_SIZE=</td>
<td>0</td>
</tr>
<tr>
<td>SCHEMA_COLLECTION=</td>
<td>sas-default-schema</td>
</tr>
<tr>
<td>SCHEMA_DB=</td>
<td>DATABASE= value</td>
</tr>
<tr>
<td>SCHEMA_PASSWORD=</td>
<td>PWD= value</td>
</tr>
<tr>
<td>SCHEMA_PORT=</td>
<td>PORT= value</td>
</tr>
<tr>
<td>SCHEMA_SERVER=</td>
<td>SERVER= value</td>
</tr>
<tr>
<td>SCHEMA_SSL_CA=</td>
<td>SSL_CA= value</td>
</tr>
<tr>
<td>SCHEMA_SSL_CERT=</td>
<td>SSL_CERT= value</td>
</tr>
<tr>
<td>SCHEMA_USER=</td>
<td>USER= value</td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
</tr>
<tr>
<td>SSL_CA=</td>
<td>none</td>
</tr>
<tr>
<td>SSL_CERT=</td>
<td>none</td>
</tr>
<tr>
<td>STRINGDATES=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACE=</td>
<td>NO</td>
</tr>
<tr>
<td>TRACEFILE=</td>
<td>none</td>
</tr>
</tbody>
</table>

### MongoDB LIBNAME Examples

Here is a sample LIBNAME statement that connects to a MongoDB database called Production.

```
libname mydb mongo user=myuser1 pwd=passwd1
```
Here is a sample LIBNAME statement that specifies connection options for a SAS derived schema.

```sas
libname myschema mongo uid=myuser1 pwd=passwd1
server="mongodb-production-server.unx.com"
port=27017 db=productiondb
schema_server="mongodb-schema-server.unx.com"
schema_collection="production-schema"
schema_user=username schema_password=password
schema_db=schemadb;
```

LIBNAME Options and Usage for Schemas

Here are the LIBNAME options that you can use to specify schema details when you connect to MongoDB:

- `SCHEMA_COLLECTION=`
- `SCHEMA_DB=`
- `SCHEMA_PASSWORD=`
- `SCHEMA_PORT=`
- `SCHEMA_SERVER=`
- `SCHEMA_SSL_CA=`
- `SCHEMA_SSL_CERT=`
- `SCHEMA_USER=`

These LIBNAME options are optional. Beginning in April 2020, if none of these are specified, a default schema, `sas-default-schema`, is generated and stored in the same MongoDB location as your data. The following table shows some use cases and indicates the LIBNAME options that you would use in each case.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Use Case Description</th>
<th>Applicable LIBNAME Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scan documents and generate a new schema each time you connect.</td>
<td><code>SCHEMA_COLLECTION=&quot;$temp$&quot;</code></td>
</tr>
<tr>
<td>Use Case</td>
<td>Use Case Description</td>
<td>Applicable LIBNAME Options</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Store a schema in the same location as your data in the same database. Saving a schema saves time and enables multiple users to access the same schema information. <strong>Note:</strong> Creating a schema in this way creates a MongoDB collection with the same name as the schema alongside your operational data collections. Consider using the <code>SCHEMA_DB=</code> option to keep the schema collection separate from your operational data.</td>
<td><code>SCHEMA_COLLECTION= &quot;schema-name&quot;</code> <strong>Note:</strong> If you do not specify a schema name, the default schema name, sas-default-schema, is used.</td>
</tr>
<tr>
<td>3</td>
<td>Store a schema on the same MongoDB server as the data, but in a different database. <strong>Note:</strong> The user name specified in the LIBNAME statement must have Write access to the database that is specified in the <code>SCHEMA_DB=</code> option (unless you use the <code>SCHEMA_USER=</code> and the <code>SCHEMA_PASSWORD=</code> options).</td>
<td><code>SCHEMA_COLLECTION=</code> <code>SCHEMA_DB=</code></td>
</tr>
<tr>
<td>4</td>
<td>Store a schema on a different MongoDB server.</td>
<td><code>SCHEMA_COLLECTION=</code> <code>SCHEMA_SERVER=</code> <code>SCHEMA_DB=</code> <code>SCHEMA_PORT=</code> (if the port value differs from the port value that you use for your data)</td>
</tr>
<tr>
<td>5</td>
<td>Your user and password information to access a schema differs from the <code>USER=</code> and <code>PWD=</code> connection option values. <strong>Note:</strong> This use case can coincide with any of the preceding use cases.</td>
<td><code>SCHEMA_USER=</code> <code>SCHEMA_PASSWORD=</code></td>
</tr>
<tr>
<td>6</td>
<td>You need to specify a certificate authority file or a certificate file that is specific to your schema. <strong>Note:</strong> This use case can coincide with any of the preceding use cases.</td>
<td><code>SCHEMA_SSL_CA=</code> <code>SCHEMA_SSL_CERT=</code></td>
</tr>
</tbody>
</table>
Data Set Options for SAS/ACCESS Interface to MongoDB

Here are the data set options that are available for the MongoDB interface. In cases where you specify both a LIBNAME option and its corresponding data set option, the value of the data set option takes precedence.

Table 4.3  Data Set Options for MongoDB

<table>
<thead>
<tr>
<th>Data Set Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>LIBNAME option value</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>none</td>
</tr>
<tr>
<td>ROWSET_SIZE=</td>
<td>LIBNAME option value</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
</tbody>
</table>

Working with Schemas for MongoDB Data

How Does SAS/ACCESS Derive a Schema?

Overview of Schema Creation

MongoDB structures data as a collection of JSON documents. Data in the documents is stored as key-value pairs. A collection can include both structured and unstructured data.
SAS/ACCESS software creates a schema to enable you to read the data in a MongoDB database. The schema maps MongoDB concepts to relational concepts as shown in Table 4.4.

**Table 4.4 Mapping of JSON and SQL Structures**

<table>
<thead>
<tr>
<th>JSON</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Collection</td>
<td>Table</td>
</tr>
<tr>
<td>Document</td>
<td>Row</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
</tr>
</tbody>
</table>

1. Every collection in a database becomes a table.
2. Simple types within the collection become columns within that table.
3. Array fields and sub-collections become “child” tables.
4. Primary keys for child tables are calculated based on their parent’s document ID, and their position within the array.
5. If the documents within a collection do not have the same structure, then the table structure represents a superset of the structure of all of the documents.

SAS/ACCESS uses the resulting schema to present MongoDB data as relational-like tables that can be used by SAS. If there is not a predefined schema in place when you connect to MongoDB, SAS/ACCESS automatically scans the MongoDB data and generates a schema. By default, the schema is stored as a SAS/ACCESS schema called sas-default-schema for later use. As a best practice, use the `SCHEMA_COLLECTION=LIBNAME` option to provide a descriptive name for the stored schema. You can specify only one schema per connection to MongoDB. However, there can be more than one schema defined for a MongoDB database.

**Note:** In order to store a schema, journaling must be enabled for MongoDB.

By default, a schema is stored for access in later SAS sessions or by other SAS users. If you do not want to store a schema from your SAS session, then specify `SCHEMA_COLLECTION=\"$temp\"`. When you do this, the generated SAS schema remains in memory only and is discarded at the end of the SAS session.

Use PROC DATASETS and PROC CONTENTS to get information about the tables in the schema. The procedures read metadata from the generated schema to create table and column listings.

**Generated Schema Tables**

SAS/ACCESS software creates representations of the MongoDB collection within the schema when a connection is made to the database. These representations map the collection onto tables. SAS retrieves the data from the database when data is read. At a minimum, the following tables are created for each MongoDB collection in the connected database:
- an Alldata table (table-name_alldata) — a read-only representation of all of the data for a collection

- a table named table-name that represents the root table. This table includes simple types from the collection. If the collection has array fields or sub-collections, this table is effectively the "parent" table.

For collections that have array fields or sub-collections, the following additional tables are created:

- table-name_flat — a read-only table that contains the currently observed value for each document in the specified collection. Each document is shown as a row in the table. The existence of this table indicates there is array data or a sub-collection in the collection. Documents that do not have array data or a sub-collection also appear in this table, but the columns derived from the schema for the array or sub-collection are empty.

- one or more tables loosely named table-name_column-name, where column-name is the name of the array field or sub-collection. These are effectively child tables.

Naming Modifications for SAS

MongoDB collection and field names might be modified for length or acceptability. The SAS modifies the MongoDB names as follows:

- invalid characters (anything other than numbers, letters, and underscores) are removed

- names longer than 32 characters are shortened, first by removing latin vowels, then by truncation, as necessary

- if a name becomes zero length, then a default name is used

- a numeric value might be appended to ensure uniqueness.

Use PROC DATASETS to determine the names of the child tables.

Importance of Specifying the SCHEMA_COLLECTION= LIBNAME Option

It is recommended that you always specify the SCHEMA_COLLECTION= option and a descriptive name in your LIBNAME statements. For a new schema collection, SAS/ACCESS scans the MongoDB database and saves the resulting schema in the same location as the MongoDB data. The schema is named according to the value that you provide for SCHEMA_COLLECTION=, and the name is case sensitive. The schema is then available for later SAS sessions or for use by other SAS users. For an existing schema collection, SAS/ACCESS can refer to a stored schema without the need to rescan the database, which is a resource intensive task. Therefore, storing a schema saves time.

Any action that modifies the schema, such as creating or dropping a table in your SAS session, immediately results in an update to the schema. In this way, users who specify the same SCHEMA_COLLECTION= value are accessing data in the same schema configuration.
Working with Schemas for MongoDB Data

Working with Temporary Schemas

You might choose not to work with a stored schema in some situations. In those cases, you can create a temporary schema. If you do not want to store a schema from your SAS session, then specify SCHEMA_COLLECTION="$temp$". Temporary schemas are discarded when your session ends. You might create a temporary schema to make experimental changes to a schema, for debugging, or when you have limited permissions for a database location.

CAUTION

It is possible to change the underlying MongoDB data when you work with a temporary schema. If you have permission to make changes to MongoDB data, then changes that you make with a temporary schema, such as creating a table or deleting records, are made in the underlying data.

How Does SAS Derive Data Types?

Derivation of Character and Numeric Data Types

Documents contain sets of key-value pairs, where the name corresponds to a variable name. The data type of a variable is inferred from the values that are provided across documents. When the data types for a key-value pair differ across documents, SAS/ACCESS uses rules to determine the overriding data type for a variable.

- A variable with integer and string values is represented as a string type.
- A variable with an integer and a double value is represented as a double type.
- A combination of double, integer, and string values is represented by a string, and so on.

Derivation of Date-Based Data Types for Use in SAS

A DATE value in MongoDB is typically mapped to a DATETIME value in the generated relational schema that is used by SAS. However, because MongoDB is not aware of the generated relational schema, it is possible that actions in MongoDB could result in adding values that are not of the DATE type to name-value pairs where SAS expects DATE values. Often, a character string that uses a date format might be inserted instead of a DATE value.

The following table lists several string formats that are successfully read into SAS as DATETIME values. The list of values is not exhaustive. Most combinations that include one or more of the variations listed are read correctly into SAS DATETIME values. String values that do not conform to these formats and variations are read in as NULL values by SAS.
Table 4.5  String Formats That Convert to SAS DATETIME Values from MongoDB

<table>
<thead>
<tr>
<th>String Format</th>
<th>Variation</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-8601 / RFC-3339 datetime</td>
<td>with GMT offset</td>
<td>2020-01-16T13:53:07+03:00</td>
</tr>
<tr>
<td></td>
<td>without a time zone</td>
<td>2020-01-16T13:53:07</td>
</tr>
<tr>
<td></td>
<td>‘Z’ time zone (GMT)</td>
<td>2020-01-16T13:53:07Z</td>
</tr>
<tr>
<td></td>
<td>space delimiter (instead of ‘T’)</td>
<td>2020-01-16 13:53:07+03:00</td>
</tr>
<tr>
<td></td>
<td>date only(^1)</td>
<td>2020-1-16</td>
</tr>
<tr>
<td>RFC-1123 datetime format</td>
<td>with GMT offset</td>
<td>Thu, 16 Jan 2020 13:53:07 +0300</td>
</tr>
<tr>
<td></td>
<td>with time zone code(^2)</td>
<td>Thu, 16 Jan 2020 13:53:07 CST</td>
</tr>
<tr>
<td></td>
<td>omitting the day of the week</td>
<td>16 Jan 2020 13:53:07 +0300</td>
</tr>
</tbody>
</table>

\(^1\) When no time is supplied, a time of 00:00 is used.

\(^2\) Valid time zone codes include: UT, GMT, EST, EDT, CST, CDT, MST, MDT, PST, PDT, Z, A, M, N, and Y.

Update a Stored Schema

In SAS Viya 3.5, the format of a stored schema has changed. If you have an existing schema from a prior SAS/ACCESS release, then the schema should be updated with the REFRESH command.

You might also want to periodically refresh your schema to pick up database changes made with tools other than SAS. This includes changes made using the MongoDB command line interface, MongoDB Compass tool, or MongoDB programming API from applications other than SAS. Some examples of external database operations that would change the database’s schema include the following:

- creation or deletion of any MongoDB collection
- creation, modification, or deletion of an embedded array or structure within an existing MongoDB collection
- inserting or replacing one or more data values in a field (column) that renders that field’s schema specification incorrect. For example, adding a STRING value into a field that is designated as an INTEGER field would change the specifications for that field in the schema. Similarly, adding a STRING value that is 20 characters long into a field that was previously defined with a length of 10 characters would change the specifications for that field.

These changes are accounted for when you rescan the MongoDB database and refresh the SAS schema.
Note: It is not necessary to refresh the schema when you use SAS languages to add and remove collections. When you are working with a stored schema, the schema is automatically updated to reflect any changes made using SAS.

To refresh a stored schema, SAS/ACCESS provides a REFRESH command that updates the generated schema. The REFRESH command is executed with PROC FEDSQL in the EXECUTE statement.

Execution of the REFRESH command is an administrative task and should be performed only by the system administrator. REFRESH forces a scan of the current MongoDB database collections and updates the current schema (whether in memory or stored) accordingly. This overwrites any manual changes that might have been made directly to a stored schema, so this command should be used with care. It is also important to schedule the refresh when people are not using the schema, to avoid lost updates.

Submit the REFRESH command as follows:

```sas
proc fedsql;
   execute (refresh) by libref;
quit;
```

Note: Journaling must be enabled on the MongoDB server in order to read and write a stored schema.

---

Using the SAS Client to Query, Create, Update, and Delete Data in MongoDB

Overview

You can read and write data to MongoDB with the SQL procedure, the FEDSQL procedure, the DS2 procedure, and the DATA step. This section describes how to read and write data in MongoDB with PROC SQL and PROC FEDSQL.

PROC SQL supports use of the CREATE TABLE statement with the AS expression, and the DELETE and DROP TABLE statements with MongoDB.

The FedSQL CREATE TABLE, DELETE, DROP TABLE, INSERT, SELECT, and UPDATE statements are supported with MongoDB.

Note: SAS data set options cannot be used in PROC FEDSQL.
Querying MongoDB Data

You query MongoDB data by querying the tables created by the generated MongoDB schema. For more information about these tables, see “How Does SAS/ACCESS Derive a Schema?” on page 92.

When querying MongoDB:

- Remember that each generated table includes simple types only. Collections that have array fields or sub-collections are represented as parent and child tables. To read a simple table or a parent table, identify the table representing the collection by its name. For example:

  select * from table-name

  To read a child table, you must query the child table directly. To view the content a child table, specify:

  select * from table-name_column-name

  Use PROC DATASETS to obtain table names.

- To retrieve columns from a parent and child table at the same time, you can join the tables. For example:

  select a._id, a.col1, b._parent_id, b.col1, b.col2
  from table-name a, table-name_column-name b
  where a._id=b._parent_id

- You can join arbitrary simple tables as well.

Creating Tables

Here are important points about table creation for MongoDB:

- Table creation supports creation of any of the following types of data: CHAR(n), VARCHAR(n), INTEGER, BIGINT, DOUBLE, or TIMESTAMP.
- You can define a new table or create a new table from an existing table.
- To define a new table with the CREATE TABLE and INSERT statements, you must use PROC FEDSQL. The FedSQL language supports the full range of data types.
- To create a new table from an existing table (CREATE TABLE with the AS expression), you can use PROC SQL or PROC FEDSQL. When PROC SQL is used to create the table, the columns are created as SAS character and numeric data types.
- The following SAS data constraints on column values are not supported for MongoDB: NOT NULL, CHECK, UNIQUE, DEFERRED, and IMMEDIATE.
- A table can include only simple types. Collections that include array data or sub-collections must be created as parent and child tables.
- Parent and child tables must be created with PROC FEDSQL. There are specific requirements for creating parent and child tables. These are described in “Creating Parent and Child Tables” on page 99.
When defining a table with PROC FEDSQL, you must insert at least one row of data in each table that you create in order for the columns to be persisted in MongoDB.

When creating a new table from an existing table, note that you can only copy root tables in MongoDB. You cannot create a new child table from an existing child table. If you use CREATE TABLE AS to copy a root table that is also a parent table, the resulting new root table contains only the top-level data from that parent table. Any child table data that the parent had is not copied.

In order to create a new child table with the data from an existing child table, you must use the FedSQL CREATE TABLE statement to define a table with precisely matching columns and with the appropriate foreign key constraint on the new parent_ID column. You can populate this child table by using a FedSQL INSERT INTO...SELECT FROM statement to copy the data from the target child table.

Table creation automatically updates the MongoDB database.

Table creation updates the active schema, whether or not SCHEMA_COLLECTION= is specified. However, if SCHEMA_COLLECTION= is not specified, new tables are visible only to the SAS/ACCESS user that created the table. The changes to the temporary schema are lost at the end of the SAS session.

Creating Parent and Child Tables

Parent and child tables are simple tables that have additional special columns defined and have a referential relationship defined between the tables. The referential relationship is created by defining PRIMARY KEY and FOREIGN KEY integrity constraints in the parent and child tables.

The PRIMARY KEY integrity constraint is defined in the parent table as follows:

- In addition to columns for the simple types, define a column named _id. A type of CHAR(24) is recommended for this column, even if you intend to use shorter ID values. A length of 24 characters is requested because the software appends information to the identifier in support of the referential relationship.

  Note: The name _id is required and it must be lowercase. In FedSQL, casing is preserved by enclosing a value between double-quotation marks.

- specify the PRIMARY KEY integrity constraint on the _id column.

Here is an example of a statement that creates the _id column and constraint:

```
create table "parent-table" ("_id" CHAR(24) PRIMARY KEY, other-columns...);
```

In addition to establishing an anchor for the referential relationship, the PRIMARY KEY integrity constraint requires that values in column _id are unique.

- Insert at least one row into parent-table and make a note of the _id value elsewhere.

  Note: Do not use a timestamp as the value for the _id column.
Special Columns for 1:n Child Tables

A 1:n (one to many) relationship allows zero or more associated values to be defined for each row in a parent table. Define a 1:n child table to represent array data. In MongoDB, this is equivalent to a JSON array in a parent document. At least three special columns are required for 1:n child tables in addition to the data values in the table:

- **_id.** Define the _id column as CHAR(35) or greater. Specify the PRIMARY KEY integrity constraint for the _id column. The length requirement for _id is discussed further below.

- **_parent_id.** The _parent_id column must be the same data type and length as the _id column in the parent table. Define this column with the FOREIGN KEY integrity constraint, similar to the following code:

  ```
  FOREIGN KEY ("_parent_id") REFERENCES "parent-table"("_id");
  ```

  **Note:** Replace `parent-table` with the name of the actual parent table.

- One or more index columns. For a one-dimensional array, call this column _index and define it as an INTEGER. This column indicates a value’s position in a value listing (array). The first value in a listing occurs at position 0. If the array is multi-dimensional, then you need to define an index column for each dimension. Use the names _dimension_1_index, _dimension_2_index, and so on, to identify the dimensions. Like a one-dimensional array, the first value for multi-dimensional index columns occurs at position 0.

  **Note:** The names _id, _parent_id, and _index (or _dimension_n_index) must be specified in lowercase letters. Be sure to enclose the names in double quotation marks to preserve the letter casing in FedSQL.

The length of the _id column in a 1:n child table requires 24 spaces to store the table’s ID value plus an additional 11 column spaces for each index in a child table. That is, if a child table has one _index column, then the length of the table’s _id column should be CHAR(35). The _id column for a child table that represents a multi-dimensional array should include an additional 11 spaces for each dimension. That is, 24-character spaces + (number-of-dimensions * 11).

If the array to be stored in a 1:n child table consists of a set of discrete values with no column name, then a special column called _value must be used.

Special Columns for 1:1 Child Tables

A 1:1 relationship allows zero or one associated value to be defined for each row in a parent table. This data is often called an object or structure. Only one special column, _id, is required when a child table stores structure data. This column is defined in addition to the columns that contain data values in the child table.

Specify the _id column as follows:
Define the _id column with the same length and data type as the target column in the parent table.

Specify the PRIMARY KEY integrity constraint.

The _id column also requires a FOREIGN KEY integrity constraint, specified as shown in the following code:

```sql
FOREIGN KEY ("_id") REFERENCES "parent-table"("_id");
```

Note: Replace `parent-table` with the name of the actual parent table.

1:1 child tables that contain structure data do not need the _index column that is required for array data.

---

**Example: Creating a 1:n Child Table**

**Details**

The following program creates a parent table named Person and a child table named Credit_Card. The child table can store zero or many credit card numbers for each person in the parent table.

**Program**

```sql
libname a mongo UID=myuser PWD=myPwd SERVER="myserver.com" PORT=27017
   DB=myDB SCHEMA_COLLECTION=sample_schema;

proc fedsql;
  create table a."person"("_id" CHAR(24) PRIMARY KEY,
   "firstname" VARCHAR(20),
   "lastname" VARCHAR(20));

  insert into a."person" ("_id", "firstname", "lastname")
    values ('ident_12345', 'Joe', 'Bloggs');

  create table a."credit_card"("_id" CHAR(35) PRIMARY KEY,
    "_parent_id" CHAR(24),
    "_index" INT,
    "card_number" BIGINT,
    "company" CHAR(10),
    FOREIGN KEY ("_parent_id")
      REFERENCES "person"("_id");

  insert into a."credit_card"("_parent_id", "_index", "card_number", "company")
    values ('ident_12345', 0, 123456789, 'Visa');
  insert into a."credit_card"("_parent_id", "_index", "card_number", "company")
    values ('ident_12345', 1, 987654321, 'AmEx');
quit;
```
Key Ideas

- The primary and foreign keys are mandatory for a parent-child relationship.
- The data type of the _id column in the Person table and the _parent_id in the Credit_Card table must be the same.
- The _index column is necessary to make the relationship 1:n (results in a MongoDB array).
- The length of the child table’s _id column is incremented to account for the _index column.
- Values for _index must start at 0.
- Neither the _id, _index, nor _parent_id columns appear as actual fields in MongoDB. They are special columns that are used by the SAS/ACCESS software.

Entity-Relationship Diagram

![Entity-Relationship Diagram]

Resulting MongoDB Document Collection

The SQL results in the creation of a new MongoDB collection called Person. Inside each document in this collection is a MongoDB array called Credit_Card, forming a 1:n relationship between them.

**Output 4.1  MongoDB Collection Person**

```json
{
    "_id" : "ident_12345",
    "firstname" : "Joe",
    "lastname" : "Bloggs",
    "credit_card" : [
        { "card_number" : 123456789, "company" : "Visa" },
        { "card_number" : 987654321, "company" : "AmEx" }
    ]
}
```
Example: Creating a 1:1 Child Table

Details
The following program creates a parent table named Country and a child table named Capital_City. Each row in the parent table has one associated value in the child table.

Program

```sas
libname a mongo UID=myuser PWD=myPwd SERVER="myserver.com" PORT=27017 DB=myDB SCHEMA_COLLECTION=sample_schema;

proc fedsql;
create table a."country"("_id" CHAR(24) PRIMARY KEY,
    "name" VARCHAR(20),
    "population" BIGINT);
insert into a."country" ("_id", "name", "population")
    values ('Y345', 'Spain', 46700000);
insert into a."country" ("_id", "name", "population")
    values ('X89', 'France', 67000000);
create table a."capital_city"("_id" CHAR(24) PRIMARY KEY,
    "name" VARCHAR(30),
    "population" BIGINT),
FOREIGN KEY ("_id") REFERENCES
    "country"("_id") REFERENCES
"country"("_id")
);
insert into a."capital_city"("_id","name", "population")
    values ('Y345','Madrid', 3100000);
insert into a."capital_city"("_id","name", "population")
    values ('X89','Paris', 2100000);
quit;
```

Key Ideas
- The primary and foreign keys are mandatory for a parent-child relationship.
- The child table’s _id column is both the primary key and foreign key in the 1:1 case.
- The data type of the _id column in the Country table and _id column in the Capital_City table is the same.
- The lack of an _index column and a _parent_id column is what makes the relationship 1:1 (results in a MongoDB object).
- The child table’s _id column never appears as an actual field in MongoDB. It is a special column that is used by the SAS/ACCESS software.
Entity-Relationship Diagram

Resulting MongoDB Document Collection

The SQL results in the creation of a new MongoDB collection called Country. Inside each document in this collection is a MongoDB object called Capital_City, forming a 1:1 relationship between them.

Output 4.2 MongoDB Collection Country

```json
{
  "_id" : "Y345",
  "name" : "Spain",
  "population" : 46700000,
  "capital_city" :
    {
      "name" : "Madrid",
      "population" : 3100000
    }
},
{
  "_id" : "X89",
  "name" : "France",
  "population" : 67000000,
  "capital_city" :
    {
      "name" : "Paris",
      "population" : 2100000
    }
}
```

Inserting Additional Values into Parent and Child Tables

Details

The following code inserts an additional column into table Person, which has a 1:n (one-to-many) relationship with table Credit_Cards.
Program

```
libname a mongo UID=myuser PWD=myPwd SERVER="myserver.com" PORT=27017
   DB=myDB SCHEMA_COLLECTION=sample_schema;

proc fedsql;
   insert into a."person" (_id, "firstname", "lastname")
      VALUES ('ident_12346', 'Samantha', 'Doe');

   insert into a."credit_card" (_parent_id, "_index", "card_number", "company")
      values ('ident_12346', 0, 98765432, 'Discover');
   insert into a."credit_card" (_parent_id, "_index", "card_number", "company")
      values ('ident_12346', 1, 34343434, 'Mastercard');
quit;
```

Key Ideas

- A unique value is inserted into the _id column of the parent table.
- The _id value from the parent table is inserted into the _parent_id column of the child table.
- The numbering in the _index value begins with 0 because this is a new record.

Note: Any insert into a child table where the parent record already has values for that field fails. To modify array field values, you must update the child table.

Updating MongoDB Data

You can update the rows in a table from a MongoDB schema with PROC FEDSQL. When you update the rows in a table from a MongoDB schema, the MongoDB database is updated immediately.

Here is an example of updating a table:

```
libname a mongo UID=myuser PWD=myPwd SERVER="myserver.com" PORT=27017
   DB=myDB SCHEMA_COLLECTION=sample_schema;

proc fedsql;
   update a."credit_card" set "card_number"=123456789
      where "parent_id"='ident_12345' and "company" = 'Visa';
quit;
```

This example updates the card_number value to 123456789 for the observation where _parent_id='ident_12345' and company='Visa'. For more information about the FedSQL UPDATE statement, see SAS FedSQL Language Reference.
Deleting MongoDB Data

You can delete rows from a table in a MongoDB schema by using the DELETE statement in PROC FEDSQL or PROC SQL.

The following statement removes all rows from the Credit_Card table that have ‘Visa’ in the Company column, whether you submit it in PROC FEDSQL or PROC SQL.

```sql
delete from a."credit_card" where company='Visa';
```

You can drop tables using the DROP TABLE statement in PROC FEDSQL or PROC SQL. To drop a table, that table must not have any associated child tables. Therefore, for a table with three child tables, delete the child tables first, and then drop the original parent table (which now has no associated child tables). The following statements delete the Credit_Card and Person tables from the MongoDB database, whether you submit it in PROC FEDSQL or PROC SQL. Here is an example of deleting a table in the MongoDB database with PROC SQL:

```sql
drop table a."credit_card";
  drop table a."person";
```

When you delete rows in a table from a MongoDB schema, the MongoDB database is updated immediately. When you drop a table from a MongoDB schema, the MongoDB database is updated immediately. The in-memory version of the schema in use is also updated immediately. A stored schema is not updated unless you specified SCHEMA_COLLECTION= LIBNAME option in the MongoDB connection.

---

Note:

You cannot drop an in-memory (temporary) table that contains no data.

---

SQL Pass-Through Specifics for MongoDB

Key Information

You can use the SQL procedure or the FEDSQL procedure to submit native commands to MongoDB.

Because MongoDB is a non-SQL database, queries are passed down in the form of JSON queries.

Here are the connection specifics:

- The `dbms-name` is `mongo`.
- For PROC SQL, the CONNECT statement’s `database-connection-arguments` are identical to the “Connection Options for MongoDB” on page 87.
When using PROC FEDSQL, it is not necessary to use CONNECT and DISCONNECT statements to submit native commands. Simply specify a mongoDB libref with the table name. For more information, see “FedSQL Explicit Pass-Through Facility” in SAS FedSQL Language Reference.

Using MongoDB Commands to Query, Create, Update, and Delete Tables in MongoDB

Overview of Passing Commands to MongoDB

MongoDB does not support SQL natively. However, you can use SAS/ACCESS to pass JSON style queries to a MongoDB database. MongoDB commands are supported in the EXECUTE statement of both FedSQL and PROC SQL. The supported syntax is shown in the following sections. For more information, see SAS FedSQL Language Reference or SAS SQL Procedure User’s Guide.

Keep these points in mind when you use the commands in the following sections:

- The parameters in myquery and myoptions must be valid MongoDB Extended JSON strings in strict mode (as opposed to MongoDB’s shell mode). When you use strict mode for JSON, be aware that names and values must be properly quoted or escaped. For more information, see MongoDB Extended JSON documentation.
- Only the commands that are shown are supported for queries that you pass to MongoDB using SAS/ACCESS.
- Anytime that you make changes to the schema using the commands in this section (Insert, Update, Delete, and so on), you must refresh the schema with the REFRESH command in the EXECUTE statement. For more information, see “Update a Stored Schema” on page 96.

The result of the query is a data set that is made of a string column called ROWDATA that contains the JSON document result for each row.

Find and Aggregate Commands

Issue MongoDB Find and Aggregate Commands

The following command returns all documents in the connected MongoDB collection Mycollection that match query Myquery.

mycollection.find(myquery)

This command returns all documents in the connected MongoDB collection Mycollection that match query Myquery using options Myoptions.

mycollection.find(myquery) withoptions (myoptions)
This command returns all documents in the connected MongoDB collection `Mycollection` that match the AGGREGATE query `Myquery`.

```r
mycollection.aggregate(myquery)
```

Unless you specify the Projection option with the FIND command, the FIND command returns the entire document. Be aware that documents might exceed the 32K column size limit for PROC FEDSQL.

**Note:** You can use the `MAX_CHAR_LEN= LIBNAME` option to control the maximum size limit. However, the maximum value allowed for `MAX_CHAR_LEN=` is 32K.

---

**Example: Submitting a Find Withoptions Command to MongoDB**

This query returns the Name and Homepage_url fields from collection `Companies` for companies whose names begin with “Wiz”.

```r
libname mylib mongo PRESERVE_TAB_NAMES=YES UID=myUser PWD=myPwd SERVER="myserver.com" PORT=27017 DB=MongoData;

proc fedsql;
select * from connection to mylib
   (companies.find({"name": { "$regex": "^Wiz","$options": "i"}})
    withoptions ({{"projection" : {{"name" : 1, "homepage_url" : 1,
    "_id" : 0}}}});
execute (companies.find({"name": { "$regex": "^Wiz","$options": "i"}})
    withoptions ({{"projection" : {{"name" : 1, "homepage_url" : 1,
    "_id" : 0}}}}))
   by mylib;
quit;
```

---

**ROWDATA**

```
{ "name" : "Wize", "homepage_url" : "http://wize.com" }
{ "name" : "Wizario", "homepage_url" : "http://www.wzario.com" }
{ "name" : "Wizzgo", "homepage_url" : "http://mutchs.hd.free.fr/312/wizzgo.html" }
{ "name" : "WizeHive", "homepage_url" : "http://www.wizehive.com" }
{ "name" : "wizard websites", "homepage_url" : null }
{ "name" : "Wizzard Software", "homepage_url" : "http://wizzardsoftware.com" }
```

---

**Create a MongoDB Table**

Use the Insert or InsertOne command to create a document in MongoDB. This has the effect of creating a table in the active MongoDB schema.

For the Insert and InsertOne commands, if the specified collection does not exist, it is created.

The following command inserts a document, `Mydocument`, into the `Mycollection` collection.
Update Data in a MongoDB Table

Use the Update, UpdateOne, or UpdateMany commands to update data in a MongoDB table. Use the update commands to modify the values in a name::value pair in a table.

The Update command updates documents in a collection, Mycollection, that are identified in a JSON style query, Myquery. The values to be updated are specified by Mydata.

mycollection.update(myquery) withdata(mydata)

The UpdateOne command updates the first document in collection Mycollection that is identified by query Myquery. Specify the data to update in Mydata, and you can apply additional options in Myoptions.

mycollection.updateOne(myquery) withdata(mydata)

The UpdateMany command updates all of the documents in collection Mycollection that are identified by query Myquery. The data to update is specified in Mydata, and you can provide additional options to apply in Myoptions.

mycollection.updateMany(myquery) withdata(mydata)

Delete MongoDB Tables

Use the DeleteOne or DeleteMany command to delete MongoDB documents.

The DeleteOne command deletes the first document in collection Mycollection that is identified by query Myquery. You can provide additional options to apply in Myoptions.

mycollection.deleteOne(myquery)

mycollection.deleteOne(myquery) withoptions(myoptions)

The DeleteMany command deletes all documents in collection Mycollection that are identified by query Myquery. You can provide additional options to apply in Myoptions.

mycollection.deleteMany(myquery)

mycollection.deleteMany(myquery) withoptions(myoptions)
Example: Modifying Data in MongoDB with SQL Pass-Through

Details
This example creates, updates, and deletes data from a collection using MongoDB commands.

Program

```
libname l mongo UID=myUser PWD=myPwd
     SERVER="myserver.com" PORT=27017 DB=MongoData;

proc fedsql noerrorstop;
execute(people.insert({"firstname" : "John", "lastname" : "Smith"})) by l;
execute(people.insert({"firstname" : "John", "lastname" : "Jones"})) by l;
execute(people.insert({"firstname" : "Joanne", "lastname" : "Jones"})) by l;
execute(people.updateMany({"firstname" : "John"}) withdata ({{"$set" : 
    {"lastname" : "Joe"}}})) by l;
execute(people.deleteOne({"lastname" : "Smith"})) by l;
execute(refresh) by l;
select "firstname", "lastname" from l."people";
quit;
```

<table>
<thead>
<tr>
<th>_id</th>
<th>firstname</th>
<th>lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>5dc025b66f3a635c471ad5c6</td>
<td>Joe</td>
<td>Jones</td>
</tr>
<tr>
<td>5dc025b66f3a635c471ad5c8</td>
<td>Joanne</td>
<td>Jones</td>
</tr>
</tbody>
</table>

Key Ideas

- The syntax of the pass-through commands is always "<collection-name>.<operation>".
- The MongoDB collection, People, is created on the first insert. There is no need to create a collection explicitly when using pass-through.
- Despite the fact that the commands are executed with PROC FEDSQL, there is no need to quote the table names in the execute commands. The MongoDB commands are submitted directly to the database.
- You must quote the table name when using executing the SELECT statement. Otherwise, FedSQL changes the name to uppercase.
- Execute the REFRESH command before the SELECT statement to update the schema and make the table visible to the SELECT statement.
Passing SAS Functions to MongoDB

SAS/ACCESS Interface to MongoDB passes the following SAS functions to MongoDB for processing if the DBMS driver or client that you are using supports this function. The results that are generated when you pass these functions to MongoDB match the output that you would see when calling these functions in SAS. However, there is often not a one-to-one relationship between the SAS function name and the generated JSON code that is passed to MongoDB.

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Generated JSON Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>LOG10</td>
</tr>
<tr>
<td>CEIL</td>
<td>LOWER</td>
</tr>
<tr>
<td>EXP</td>
<td>MOD (see note)</td>
</tr>
<tr>
<td>FLOOR</td>
<td>SQRT</td>
</tr>
<tr>
<td>LENGTH</td>
<td>SUBSTR</td>
</tr>
<tr>
<td>LOG</td>
<td>UPPER</td>
</tr>
</tbody>
</table>

Note: SAS does not modify non-integer arguments to the MOD function. If your DBMS does truncate non-integer arguments to MOD, then DBMS results for this function might vary from SAS results. For more information, see "Functions Where Results Might Vary: MOD Function" in SAS/ACCESS for Relational Databases: Reference.

Data Types for MongoDB

The FedSQL and DS2 languages support different data types than PROC SQL and the DATA step. See "Data Types for MongoDB" in SAS FedSQL Language Reference for information about the data types supported for the MongoDB interface with FedSQL and DS2.

Supported Data Types for MongoDB

The following data types are supported for the MongoDB interface.

- Character
  - JAVASCRIPT
  - REGEX (regular expression string)
  - STRING
- Numeric
  - BOOL (Boolean)
Data Type Conversions When Loading MongoDB Data into SAS

The following table shows the data type conversions for data that is loaded from MongoDB into SAS.

<table>
<thead>
<tr>
<th>MongoDB Data Type</th>
<th>SAS Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVASCRIPT</td>
<td>CHAR</td>
<td>Default format is $w$.²</td>
</tr>
<tr>
<td>REGEX</td>
<td>CHAR</td>
<td>Default format is $w$.²</td>
</tr>
<tr>
<td>STRING</td>
<td>CHAR</td>
<td>Default format is $w$.²</td>
</tr>
<tr>
<td><strong>Numeric Data</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOL</td>
<td>NUMERIC</td>
<td>In SAS the value 0 is false, and any other numeric value is interpreted as true.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>NUMERIC</td>
<td>Default format is $w$.¹</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>NUMERIC</td>
<td>Default format is $w$.¹</td>
</tr>
<tr>
<td>INT</td>
<td>NUMERIC</td>
<td>Default format is $w$.¹</td>
</tr>
<tr>
<td>LONG</td>
<td>NUMERIC</td>
<td>Default format is $w$.¹</td>
</tr>
<tr>
<td><strong>Other Data</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Loading Large Numeric Values for MongoDB**

*Note:* In FedSQL and DS2, the information below applies only to the DOUBLE type.

Be aware that when performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that are read are rounded to meet this condition. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data

<table>
<thead>
<tr>
<th>MongoDB Data Type</th>
<th>SAS Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT</td>
<td>Varies</td>
<td>JSON objects typically become child tables with variables of the appropriate data type. However, if a JSON object that is not accounted for in the current schema map is encountered, then that object is returned as a missing value. This might occur if a JSON object is added after the initial scan to determine the schema for a collection.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>Varies</td>
<td>Converted to a child table in SAS with variables of the appropriate type.</td>
</tr>
<tr>
<td>BINDATA</td>
<td>CHAR</td>
<td>Contains a hexadecimal representation of the binary data. Default format is $w.²</td>
</tr>
<tr>
<td>DATE</td>
<td>NUMERIC</td>
<td>Date values become SAS dates, formatted based on your specifications.</td>
</tr>
<tr>
<td>NULL</td>
<td>CHAR</td>
<td>MongoDB NULL values are read in as missing character values.</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>CHAR</td>
<td>Default format is $w.²</td>
</tr>
</tbody>
</table>

1 The maximum precision for SAS numeric values is approximately 15 digits of precision. For more information, see "Loading Large Numeric Values for MongoDB".
2 The maximum length of a CHAR value for MongoDB data is 32,767 characters.
in as character data. For more information, see “Choosing Your Degree of Numeric Precision” in SAS/ACCESS for Relational Databases: Reference.
About the PI System

The PI System is a repository for time series data. The data is collected from sensors that are attached to various types of monitoring and analysis equipment.
For example, a wind-generation farm might have sensors that report wind speed, temperature, orientation, and kilowatts of electricity attached to each generator. These timed-data values are sent to a server at predefined intervals and are stored in the PI System for later retrieval.

The PI System is designed to continuously collect and store data from a large number of data sources. These data sources are typically sensors that monitor processes, production, or status within different types of systems. The PI System consists of these functional layers:

**Data Archive**

is the actual storage of the PI System time series data. The data is organized as a flat namespace of tags, which are also called PI Points. Each tag consists of a list of events, which is a time series record. Each event stores the tag name, the timestamp of the event, the event value, and status information. Within the context of SAS, a tag is a data set and events are observations.

Support for the Data Archive was provided in the initial release of SAS/ACCESS Interface to the PI System, which was part of SAS 9.4M2.

**Asset Framework**

is a more recent OSIsoft product that sits on top of the Data Archive. It provides a hierarchical view of the flat namespace of tags and other metadata such as attributes and units of measure.

Support for the Asset Framework was added in the February 2016 release of SAS/ACCESS Interface to the PI System. This release also includes support for PI Event Frames. PI Event Frames track important process events and are supported as part of the Asset Framework.

The PI System is not ODBC-based or SQL-based, so it has no pass-through engine. However, after the data is in a SAS data set, you can process the data using PROC SQL. For sample code, see "Sample Code for the Data Archive" on page 138 and "Sample Code for the Asset Framework" on page 158.

**PI System Terminology**

The PI System is not a relational database management system, and there are some terms that are unique to this system.

**element**

is a functional part of the Asset Framework. An element can provide hierarchical structure to the Asset Framework or it can contain data. An element can contain other elements, similar to a directory that contains subdirectories. Alternatively, an element can be a tag (flat data file).

For example, suppose your system contains temperature sensors for multiple buildings in several cities. Your data might be organized as TemperatureSystem → Region → City → Building → Floor → Section. The elements at the Region level might be Northeast, Southeast, Midwest, and Southwest. The element Northeast might contain the elements NewYork, Buffalo, and Boston. At the Section level, the elements are flat files (tags) that contain time series data from temperature sensors in a section of a building.

An element that contains data corresponds to a data set in SAS. An element that contains other elements corresponds to a library in SAS.

**event frame**

tracks process events that are important to your business. Events that you might track include power outages, starting and stopping servers, or readings that are
out of specified tolerances. The attributes about an event include the start and end times and any other information that you choose to track about that event.

PI Event Frames are part of the PI Asset Framework.

tag
is a flat data file with data from a sensor or other device. The Data Archive consists of data stored entirely in tags. The Asset Framework consists of elements, some of which are tags that contain sensor data.

A tag corresponds to a data set in SAS.

PI point
is another name for a tag. Some virtual table names refer to PI points.

Supported Features by Host for SAS/ACCESS Interface to the PI System

Here are the features that SAS/ACCESS Interface to the PI System supports across available platforms. To find out which versions of your DBMS are supported, see your system requirements documentation.

Table 5.1 Supported Features for SAS/ACCESS and the PI System

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>Push Down To the PI Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HP-UX for Itanium</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Linux for x64</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Solaris for SPARC</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Solaris for x64</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows 64-bit</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

To improve performance, the processing of some data set options is pushed down to the PI server.

SAS/ACCESS Interface to the PI System does not support these functions:

- passing SAS functions
- passing joins

In addition, there is no bulk-load functionality for the PI System. For the Data Archive on a Windows environment, you can create new tags and add data by using SAS DATA step syntax.

For UNIX platforms, there is no support for writing to the PI System, which includes creating tags, writing data, and deleting tags.
Methods to Access PI System Data

If you use SAS/ACCESS Interface to the PI System on a Windows environment, including the PI System client, you use the native API to access the PI System data.

In the February 2016 release of SAS/ACCESS Interface to the PI System, support has been added to run SAS/ACCESS Interface to the PI System on a UNIX environment. In this configuration, SAS runs on a UNIX environment, and the PI Server is on a Windows machine. When SAS runs on a UNIX environment, you access the data using the PI Web API.

Note: Because the default memory allocation for SAS in a UNIX environment is 2G, you might need to increase memory by setting the MEMSIZE system option to 4G.

Functional Differences When You Access PI System Data from a UNIX Environment

There are some differences in functionality when you access PI System data from a UNIX environment. To access the PI System from a UNIX environment, you must use the PI Web API:

- You can specify the authentication method to use when connecting to the PI System from UNIX with the SAS_PI_WEB_AUTH= environment variable. By specifying the authentication method, you can improve the time it takes to connect to the PI System via the PI Web API. For more information, see “Security for UNIX: Specify the Authentication Method for Accessing the PI System” on page 119.

- Connection to the PI System on UNIX requires an SSL certificate file. You specify the location of this file with the SSLCALISTLOC= system option. For more information, see your installation and configuration documentation for SAS/ACCESS Interface to the PI System.

- Because you use the PI Web API server when you connect to the PI System from UNIX, you must specify the HOST= and PORT= data set options.

- On UNIX, you can only read data from the PI System. You cannot create or modify tags on the PI System.

- Some data set options are not available or are restricted when you access PI System data from UNIX:
  - ATTRIBUTE=. The ATTRIBUTE= option is case-sensitive. Although other options are not case-sensitive when you use the PI Web API, the ATTRIBUTE= option is. Ensure that you specify values with the correct case to match values in the PI System.
    This option is used on the Asset Framework only.
  - COUNT=. For the Data Archive, the COUNT= data set option is available only when you access the Picomp_Summary virtual table. For the Asset
Framework, the COUNT= data set option is not available when you access
the PI System with the PI Web API.

- MEDIAN=. Calculation of median statistics is not available when you access
PI System data with the PI Web API.
- TESTDATE=. The TESTDATE= data set option is not available. This option is
used on the Data Archive only.

- For UNIX environments, date formats that you use must match the Windows
system locale on the PI Web API server. For more information, see "Formatting
Date Values" on page 122.
- If you are accessing the PI System Data Archive for data that uses multi-byte
characters, set the system locale for the PI Web API host machine to the desired
encoding. For more information, see "Multi-Byte Character Data" on page 123.

Security Considerations

Security for a Microsoft Windows Environment

When running on a Microsoft Windows platform, OSIsoft strongly recommends
using Integrated Windows Authentication. This authentication uses the credentials
of the user’s existing Microsoft Windows session to authenticate with the PI System
server. You must configure the PI System server to allow Read, Write, or Read-
Write access for users and user groups. The PI System allows access using trusts.
OSIsoft does not recommend using an explicit user sign-on using a user ID and
password, although this SAS/ACCESS engine does provide options to specify these
credentials.

Security for UNIX: Specify the Location of SSL Certificates

When running on a UNIX platform, you must specify an SSL certificate file to
connect to the PI System via the PI Web API. You specify the location of this file
with the SSLCALISTLOC= system option. For more information, see your
installation and configuration documentation for SAS/ACCESS Interface to the PI
System.

Security for UNIX: Specify the Authentication Method for
Accessing the PI System

You can specify the authentication method that should be used when you connect to
the PI System from a UNIX platform with the SAS_PI_WEB_AUTH= environment
variable. By specifying the authentication method, you can reduce the time it takes
to connect to the PI System. You specify environment variables in the SAS
invocation command or by using the OPTIONS statement with the SET= option. For
more information, see “Defining Environment Variables in UNIX Environments” in
SAS Companion for UNIX Environments.

Here are the possible values for the SAS_PI_WEB_AUTH= environment variable:

- NONE
  indicates that no user name or password is required.
BASIC
indicates that a user name and password (and possibly a domain) are required for authentication.

KERBEROS
indicates that Kerberos authentication via Generic Security Services (GSS) is used to connect to the PI System.

If you do not set a value for SAS_PI_WEB_AUTH=, then SAS first attempts to connect to the PI System using no authentication (NONE) and then using Kerberos authentication (KERBEROS). SAS attempts all possible methods to authenticate until a successful attempt is made or until all methods have been attempted. When you set SAS_PI_WEB_AUTH=, that means that SAS first attempts to use the method that you specify and then tries any remaining authentication methods.

Data Types for the PI System

Supported Data Types

All data types for the PI System engine are fixed except for the Value column.

A Value column can be one of these data types:

- INT16
- INT32
- FLOAT16
- FLOAT32
- FLOAT64
- DIGITAL
- BLOB
- TIMESTAMP
- STRING

SAS Data Types

SAS has two data types, CHARACTER and NUMERIC. SAS character variables (columns) are of a fixed length with a maximum of 32,767 characters. SAS numeric variables are signed 8-byte, floating-point numbers. When SAS numeric values are used in conjunction with SAS formats, they can represent a number of data types, including DATE, TIME, and DATETIME. For more information about SAS data types, see *SAS Programmer’s Guide: Essentials*.

Data Conversion from the PI System to SAS

SAS/ACCESS Interface to the PI System reads numeric values as a DOUBLE, TIMESTAMP values as a SAS DATETIME value, and all other values as CHARACTER.

This table shows the SAS and PI System data types and the default SAS formats that are assigned to SAS variables when SAS reads PI System data.
### Table 5.2 Data Types When Reading PI System Data into SAS

<table>
<thead>
<tr>
<th>PI System Data Type</th>
<th>SAS Data Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT16</td>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>INT32</td>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT16</td>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT32</td>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT64 2</td>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>NUMERIC</td>
<td>DATETIME22.3</td>
</tr>
<tr>
<td>STRING</td>
<td>CHARACTER</td>
<td>$n. 1</td>
</tr>
<tr>
<td>BLOB</td>
<td>CHARACTER</td>
<td>$n. 1</td>
</tr>
<tr>
<td>DIGITAL</td>
<td>CHARACTER</td>
<td>$n. (Enumeration is resolved to a string.) 1</td>
</tr>
</tbody>
</table>

1 The length of the format is based on the value of the DBMAX_TEXT= LIBNAME option.
2 FLOAT64 is equivalent to a DOUBLE.

### Data Conversion from SAS to the PI System

The following table shows the PI System data types and table properties that are assigned when SAS creates a table for the PI System.

**Note:** You cannot create data in the PI System if you use the PI Web API to access the PI System. UNIX environments are required to use the PI Web API to access the PI System.

### Table 5.3 Data Types When Writing from SAS to the PI System

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>SAS Format</th>
<th>PI System Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>$n.</td>
<td>STRING</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>DATETIMEw.d</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>DATEw.</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>TIMEw.d</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>other numeric formats</td>
<td>FLOAT64</td>
</tr>
</tbody>
</table>
Data Considerations

Text Length

When reading PI System data, the default maximum string length for the Value and Status columns is 32 for the Data Archive and 256 for the Asset Framework. If a longer string is read from the PI System, SAS issues a warning message. You can use the DBMAX_TEXT= option to increase the string length to a value that is greater than the default.

Rounding Numeric Values in Output

When printing output, the last significant digit in fractions, such as milliseconds, might be off by 1 when comparing values in SAS output and in PI System Explorer output. This can be seen because SAS rounds the last significant value in the output, and the PI System Explorer truncates the last significant value. For example, when printing to three significant digits, the value 0.1235 would be printed in SAS output as 0.124. In PI System Explorer output, this value would appear as 0.123.

Note: SAS holds the full precision value in memory, and the full value is used in computations. SAS rounds values only when printing to output.

To ensure agreement to a specific precision in the output, provide a format that displays additional digits of precision, so that the values match to the required precision. A SAS numeric format of 10.5 ensures that SAS and PI System Explorer output match to 3 significant digits.

Formatting Date Values

Dates that you specify using the START= and END= data set options are parsed by the PI System using the Windows system locale setting. To ensure correct parsing of dates, specify these dates using the format for your location. For example, July 1, 2014 might be specified as ‘7/1/2014’ in the United States. In France, you would specify the same date as ‘1.7.2014’.

Note: For UNIX platforms, the date format must match the Windows system locale on the PI Web API server.

Ensure that you specify the month and day correctly for your location. Because the parser can adjust date separators, such as ‘/’ and ‘.’, based on location, dates that are specified incorrectly might not result in an error. For example, if you specify ‘7/1/2014’ in France, the month and the day are interpreted as January 7 and not July 1. If you want the date to be July 1, you would specify ‘1/7/2014’. You can use the TESTDATE= LIBNAME option to test how a given date is interpreted on your system. For a sample program, see “Use the TESTDATE Option” on page 145.
Date Limit for Datetime Values

For the PI System, date values must fall on or after the PI System baseline date and time, which is January 1, 1970 00:00:00.001 GMT. Typically, datetime values in your data are stored in GMT. For this reason, you must ensure that all datetime values specified in SAS (as local time) fall after January 1, 1970 00:00:00.001 GMT. For example, the local time zone in Beijing is eight hours ahead of GMT, or UTC+8. Therefore, all datetime values in Beijing must fall on or after January 1, 1970 08:00:00.001. This ensures that when a local datetime value is converted to GMT, the date and time do not fall before the PI System baseline time.

Missing Date Values

In the PI system, a missing value is represented as the PI System zero value of January 1, 1970 00:00:00.000 GMT. SAS reads missing PI System dates as valid datetime values, so a best practice is to treat these values as missing in your data. Also, remember that times are typically converted to local time, so be sure to adjust January 1, 1970 00:00:00.000 GMT to your local time. In general, any datetime value before January 2, 1970 can be treated as missing.

PI System Null Values and SAS Missing Values

The PI System might store NULL values as a result of a bad record status. SAS reads these as SAS missing values and displays them as ".". SAS reads missing strings as blank character strings.

PI System missing datetime values are stored as the zero date value of January 1, 1970 00:00:00.000 GMT. SAS reads missing date values as valid datetime values, so treat these values as missing.

When SAS missing values are written to the Value column, SAS writes "No Data" as the value of the Status column.

Multi-Byte Character Data

The PI System engine supports National Language characters, such as Asian character sets that are stored as multi-byte sequences. Perform these tasks for this to work properly:

- Set the Windows System Locale to the desired character encoding. The character encoding should match the encoding of the PI System. Setting the Windows System Locale is typically not a requirement for other SAS/ACCESS engines. However, this step is needed for SAS/ACCESS Interface to the PI System because of the internal workings of the PI System.

  When you run SAS/ACCESS Interface to the PI System on a Windows machine, set the Windows System Locale on that machine. When you run SAS/ACCESS Interface to the PI System on a UNIX machine, set the Windows System Locale on the PI Web API host machine.

- If you are using the PI Web API to access the PI System Data Archive, set the System Locale for the PI Web API host machine to the same encoding as that used for the PI System.
Start SAS with the desired ENCODING and LOCALE command line options. The ENCODING and LOCALE values that you specify should match those that were used for the PI System.

Set the options VALIDVARNAME=ANY and VALIDMEMNAME=EXTEND before using any commands that use multi-byte characters.

Tag names can contain multi-byte characters, although this is not fully supported by the PI System. There are some limitations, such as multi-byte characters that do not set the high-order bit in each byte. If you get an error, use a different name. Also remember that tags that are created in SAS for the Data Archive are limited to 32 bytes (not 32 characters). This limitation affects the maximum length of a multi-byte tag name.

The Status column might contain certain pre-defined status strings, such as “OK” or “No Data”. Therefore, specifying multi-byte characters for Status has no effect and are read as “No Data”.

The tag descriptor (‘member label’ in SAS) can store any multi-byte character sequence up to 32767 bytes in length with no other restrictions.

---

**Note:** In output from PROC CONTENTS and PROC DATASETS, the member label is truncated to 256 bytes. The full value is stored in SAS.

Any string-based Value column value can also consist of any multi-byte character sequence up to 976 bytes in length with no other restrictions.

---

**Using PROC DATASETS with the PI System**

PROC DATASETS might have limitations if tag names are greater than 32 bytes or if tag names contain special characters. This is due to the SAS restrictions on data set names, which must be 32 bytes or fewer. Although the Tag column displays the full tag name, the Name column can contain generated tag names that do not work with PROC DATASETS commands.

In the following PROC DATASETS output, rows 1–5 contain tag names that comply with SAS rules for data set names and you see the names in the Name column. In rows 6–9, PROC DATASETS determined that the tag name is not a valid SAS name. In these rows, the tag name is longer than 32 bytes or the tag names contain periods ( . ) and spaces. When PROC DATASETS processes a tag that is not a valid SAS name, it writes a generated tag name in the Name column. Rows 6–9 contain generated tag names in the Name column. It is these generated tag names that are not valid as data set names in PROC DATASETS.
When you issue a PROC DATASETS command, such as DELETE, MODIFY, and APPEND, specify the tag name using the same capitalization that is used for the tag name in the PI System. Otherwise, an error occurs. For example, if you want to delete the CDEP158 tag that was generated by SAS, you must match the case exactly by specifying CDEP158 and not cdep158. If you specified cdep158, PROC DATASETS would not find the data set for this tag. Some of these commands are also available using other procedures, such as PROC DELETE and PROC APPEND, and do not have this limitation.

If your PI Server has more than 10,000 tags, a best practice for performance is to limit the tags of interest. To do this, use a LIBNAME command with the PROC_DATASETS_TAGFILTER= option with a wildcard filter string. This filter is pushed down to the PI Server, resulting in a faster response time. For more information, see “PROC_DATASETS_TAGFILTER= LIBNAME Statement Option” on page 20.

Using the PI System Data Archive

Data Structure for the Data Archive

In SAS, a PI System tag is a 7-column data set unless the HIDEFLAGS=YES data set option is set. When HIDEFLAGS=YES, the last three columns of the data set are omitted.

These are the PI System data set columns:

- **Tag**
  - is the name of the PI Point (table). In SAS, a table is referred to as a data set.
Timestamp
is the date and the time, including milliseconds, when the tag value was first recorded. The Timestamp value cannot be null or missing.

Value
is the value of the sample, usually numeric. It could also be a character string or a timestamp.

Status
is the status of the sample. The value can be OK, a status that indicates an error, or another meaningful status.

Questionable
is a flag that indicates whether the event has been marked as questionable.

Annotated
is a flag that indicates whether the event has an annotation. SAS does not handle the annotation itself.

Substituted
is a flag that indicates whether the value has been modified after its original recording.

The Annotated and Substituted columns are read-only. They can be changed only by the PI System when a tag has been annotated or has been substituted.

---

## Naming Conventions for the Data Archive

### SAS and Data Archive Objects

SAS and Data Archive objects include data sets and columns. They follow these naming conventions:

- **SAS data set names** are limited to 32 characters from A-Z, 0–9, and underscore (_). In the PI System, tag names can be up to 1024 characters long and can contain almost any character. If a tag name falls within the conventions of a valid SAS data set name, that tag name can be used as if it were a data set name. Otherwise, you must read from Picomp using the `TAGLIST=` data set option that contains the set of tags that you want to read.

- **Tag names** are stored and displayed using the case in which they were created. However, when a tag is read, the name can be specified in any case. Therefore, `MyTag` and `MYTAG` refer to the same tag. A tag that was originally created as `MyTag` is displayed as `MyTag`.

- **Tags that are created in SAS** are limited to the conventions of valid SAS data set names.

- **Columns names** are all predefined (for example, Tag, Timestamp, Value, and Status), and fall within the conventions of valid SAS names.

For general information, see “SAS Names and Support for DBMS Names” in *SAS/ACCESS for Relational Databases: Reference.*

### PI Server Name Requirements

When you specify a PI System server name in the `LIBNAME` statement, use the name as it is defined by the user in the PI System SDK Utility. Do not use the
(possibly fully qualified) host name. For example, if the PI Server is defined as <myServer>, you must use <myServer> in the LIBNAME= statement, and not <myServer.myCompany.com>. Otherwise, the connection fails.

Reading and Writing with the Data Archive

Connect to the PI System Server

To connect to the PI System server, you use the LIBNAME statement using the engine PISYSTEM. You can use the default PI System server or you can specify the name of the server using the SERVER= LIBNAME option.

```sas
libname pi pisystem;
libname pi pisystem server=myserver;
```

Although options exist to specify user authentication credentials, the best practice is to default to using Integrated Windows Authentication, which does not require explicit LIBNAME authentication options.

You can use these LIBNAME options to configure the server connection:

- `DBMAX_TEXT=n` specifies the maximum length of a character string for the Value and Status columns.
- `DBPROMPT=[YES | NO]` specifies whether to use dialog boxes for authentication credentials.
- `DEFER=[YES | NO]` specifies whether to wait until a tag is opened to connect to the PI System server.
- `HIDEFLAGS=[YES | NO]` specifies whether to hide in output the columns Questionable, Annotated, and Substituted.
- `SHOWFLAGS=[YES | NO]` specifies whether to display in output the columns Questionable, Annotated, and Substituted.
- `TESTDATE= "PI-System-datetime-string"` is a special test-only option. You use it to specify a PI System DATETIME string so that you can see how the value appears as a SAS DATETIME value.

```
Note: This option is not supported when you use the PI Web API to access PI System data from a UNIX environment.
```

```
TIMEZONE=[LOCAL | GMT | UTC] specifies whether SAS reads and writes PI System timestamps in local or GMT time. The default is LOCAL. Use GMT or UTC only when you want to see the time using GMT.
```

For more information, see "LIBNAME Statement Specifics for the PI System Data Archive" on page 134.

Requirements for Writing Data to the Data Archive

When you create a Data Archive tag (table), it can contain the following columns, and the names that you use must match the following list:

- *(Required)* Timestamp. Timestamp values must be numeric and preferably formatted as DATETIME values.
- (Required) Value. Values in the Value column can be of any type.
- (Optional) Status. Values in the Status column must be CHARACTER values. If no Status column is provided, a default status value of OK is used.
- (Optional) Questionable. Values in the Questionable column must be NUMERIC. If no Questionable column is provided, a default value of 0 (false) is used.

The order of the columns can vary, but the names must match the preceding list.

A tag that you create can contain the Tag, Annotated, and Substituted columns, but these columns are not written to a tag on output to the Data Archive. By definition, these columns contain read-only values.

**Read Tags and Metadata Using the PIpoint Virtual Table**

Using the PIpoint data set, you can use the TAGFILTER= data set option and WHERE clauses to acquire the list of tags that you want to process, along with their metadata. Using the TAGFILTER= option, you specify a wildcard filter string to select the tags. For tag metadata, see your PI System documentation.

**Note:** For systems that use the PI Web API to access PI System data, using the PI Web Server 1.7.0 or higher results in improved performance when reading PIpoint data.

This example creates the SAS data set Sinus that contains the tags and their metadata for tag names that begin with 'sinus':

```
libname pi pisystem;
data sinus;
   set pi.pipoint (tagfilter="sinus");
run;
```

This example creates the SAS data set Zero that contains only the tag names for tags where the zero attribute has a value of 100:

```
libname pi pisystem;
data zero;
   set pi.pipoint (keep=tag zero where=(zero=100));
run;
```

The two examples above can be combined to create a data set that contains tag names that begin with 'sinus' and have a zero-attribute value of 100.

```
libname pi pisystem;
data sinus;
   set pi.pipoint(keep=tag zero tagfilter="sinus" where={(zero=100)});
run;
```

Regarding performance, the TAGFILTER= option is processed faster by the PI Server than the WHERE clause that is processed within SAS. For better performance, avoid using WHERE=tag-name.

**Best Practices for Performance Optimization**

Performance is optimized when you can push down processing to the PI Server. Here are some recommendations:
Do not use the DBMAX_TEXT= option with values larger than necessary for reading the data, especially when reading large data volumes. DBMAX_TEXT= defaults to 32 for reading from Picomp, which usually is sufficient.

Reading from PIpoint:
- Use the TAGFILTER= data set option to reduce the number of records returned from the PI Server.
- Use the KEEP= data set option to reduce the number of columns that are created. Specify only those columns (tag attributes) that you are interested in. Always include the Tag column. This can speed up the response by a factor of 3 compared to reading all 128 columns. For more information, see "KEEP= Data Set Option" in SAS Data Set Options: Reference.
- Increasing memory by setting the MEMSIZE= system option to a larger value, such as MEMSIZE=4G, can prevent a job from ending due to memory constraints.
- For systems that use the PI Web API to access the PI System, use the PI Web Server 1.7.0 or higher to improve performance when reading data.

Reading from Picomp or tag-name:
- Constrain the time interval with the START= and END= data set options to minimize the number of records that are returned from the PI Server. Using a WHERE clause results in poor performance because all tag data records are read and then partially discarded when SAS processes the WHERE clause. The START= and END= options are processed by the PI Server.
- Use SELECT=, MIN=, and MAX= data set options when possible, which are also processed by the PI Server.
- For advanced users of the Data Archive, using the FILTER= data set option is preferable to using a SAS WHERE clause. Refer to the OSIsoft documentation for details about how to formulate complex filter expressions. The SELECT=, MIN=, and MAX= options are used to dynamically build a filter expression and are therefore incompatible with the FILTER= option.
- SAS WHERE clause filtering occurs in SAS. Use the WHERE clause when data set options that push filtering to the PI Server are insufficient. You can combine data set option filtering with SAS WHERE clause filtering.

Write Data Archive Data

Writing Data Archive data is allowed only for new tags or for existing tags that are created in SAS, as determined by the PointSource attribute. Tag names, or data sets, must follow the naming rules for SAS data sets. The TAGLIST= option is ignored. You can write to only one tag at a time.

For examples, see "Create a New Tag from a SAS Data Set" on page 141.
Special Virtual Tables for the Data Archive

Virtual Table Descriptions

A common practice when reading data in the Data Archive is to read multiple tags at one time using a filter. This enables you to process a subset of events that are located in multiple tags. Tag names can be up to 1024 characters long, whereas the length of a table name in SAS can be a maximum of 32 characters. Tag names can also contain characters that do not comply with rules for SAS data set names. To accommodate processing PI System tags and reading multiple tags at once, you create a data set that contains the tags that you want to read.

SAS uses these virtual tables in the Data Archive to read a directory of tags and to read the data:

- **PIpoint** represents a 128-column data set that consists of all Data Archive tags and their attributes. PIpoint can be considered a directory of available tags. You use PIpoint to create a table of tags that you want to read.

- **Picomp** produces a seven-column data set that contains the time series data that is stored in each tag. When you read from Picomp, you must specify the TAGLIST= data set option. This option specifies the data set that contains the tags that you want to read. This is the data set that you created using the PIpoint virtual table. The tag name is in the Tag column. This enables tag names that do not conform to the SAS data set naming rules to be accessed, as well as reading from multiple tags at once.

- **Picomp_Summary** generates a report of summary statistics per time interval over a specified time range. For example, you might obtain hourly summary statistics over the duration of one day, the time interval for the summary statistics is one hour.

When you process data in the Data Archive, you typically filter PIpoint to build a data set that consists of selected tags. You then use this data set as the value of the TAGLIST= option when you process Picomp or Picomp_Summary. For more information, see “Connect to the PI System Server” on page 127.

For examples, see “Read Multiple Tags Simultaneously: Create a TAGLIST= Subset of Tags” on page 142 and the two examples that follow it.

Picomp_Summary: Summary Statistics That Are Available

Here is the list of columns in the Picomp_Summary table. This list includes the summary statistics that are calculated and stored in the Picomp_Summary virtual table. The summary statistics are based on a time interval that you specify. For example, if you want a list of hourly summary statistics over the duration of one day, the time interval for the summary statistics is one hour.
Table 5.4 Columns in Picomp_Summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>String</td>
<td>Tag name</td>
</tr>
<tr>
<td>StartTime</td>
<td>Timestamp</td>
<td>Start time of a summary interval</td>
</tr>
<tr>
<td>EndTime</td>
<td>Timestamp</td>
<td>End time of a summary interval</td>
</tr>
<tr>
<td>Count</td>
<td>Numeric</td>
<td>Sample count for event-weighted statistics 1/(number of seconds) for time-weighted statistics</td>
</tr>
<tr>
<td>Total</td>
<td>Numeric</td>
<td>Total of sample values in an interval for event-weighted statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average of sample values in an interval for time-weighted statistics</td>
</tr>
<tr>
<td>Minimum</td>
<td>Numeric</td>
<td>Smallest sample value in an interval</td>
</tr>
<tr>
<td>MinimumTime</td>
<td>Timestamp</td>
<td>Timestamp of the smallest sample value</td>
</tr>
<tr>
<td>Maximum</td>
<td>Numeric</td>
<td>Largest sample value in an interval</td>
</tr>
<tr>
<td>MaximumTime</td>
<td>Timestamp</td>
<td>Timestamp of the largest sample value</td>
</tr>
<tr>
<td>Range</td>
<td>Numeric</td>
<td>Difference of the Maximum and Minimum values for an interval</td>
</tr>
<tr>
<td>Average</td>
<td>Numeric</td>
<td>Mean of the sample values over an interval</td>
</tr>
<tr>
<td>Median¹</td>
<td>Numeric</td>
<td>Median of sample values over an interval</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Median statistics are not available when you use the PI Web API to access the PI System.</td>
</tr>
<tr>
<td>MedianCount¹</td>
<td>Numeric</td>
<td>Number of sample values used to determine the Median</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: Median statistics are not available when you use the PI Web API to access the PI System.</td>
</tr>
<tr>
<td>StdDev</td>
<td>Numeric</td>
<td>Standard deviation of sample values over an interval</td>
</tr>
<tr>
<td>PopulationStdDev</td>
<td>Numeric</td>
<td>Population standard deviation of sample values over an interval</td>
</tr>
<tr>
<td>PercentGood</td>
<td>Numeric</td>
<td>Percentage of sample values with a status of “OK”</td>
</tr>
</tbody>
</table>
### Picomp_Summary: Specifying the Report Duration and Time Intervals for Summary Statistics

Use the following data set options to specify the duration of a summary report and to determine the time intervals that are used to calculate summary statistics. The resulting records are saved to the Picomp_Summary table.

- **START=** specifies when the report period begins. To generate summary statistics over a 24-hour period that begins at 8:00 A.M. on March 1, 2016, specify `START='3/1/2016 08:00:00.000 AM'`.

- **END=** specifies when the report period ends. Using the START= value above, to get summary statistics over a 24-hour period, specify `END='3/2/2016 07:59:59.999 AM'`.

- **COUNT=** specifies the number of sample intervals to evenly divide the report duration into. You specify the duration of the report with the START= and END= values.

  Interaction: If you specify COUNT=, do not specify INTERVAL=. If you specify INTERVAL= and COUNT= for a tag, then the value of INTERVAL= takes precedence.

- **INTERVAL=** specifies the length of the time interval to use for summary statistics. If the interval that you specify does not divide evenly into the duration specified by START= and END=, then remaining records are discarded. For example, if you specify a 5-hour interval for a report that spans 24 hours, then four summary records are stored in Picomp_Summary, and data from the remaining four hours is not summarized. For this reason, for a report that spans 24 hours, the recommended intervals are 1, 2, 3, 4, 6, 8, or 12 hours.

  Interaction: If you specify INTERVAL=, do not specify COUNT=. If you specify INTERVAL= and COUNT= for a tag, then the value of INTERVAL= takes precedence.

### Picomp_Summary: Calculating Median Values

There is no median statistic calculated in the PI System. Therefore, before you can calculate median values, SAS/ACCESS Interface to the PI System must read the data into SAS. This might require reading a very large number of records and might significantly increase computation times. For this reason, the Median and MedianCount columns in the Picomp_Summary table are disabled by default.

---

**Note:** Median statistics are not supported when you use the PI Web API to access the PI System.

---

Here are the data set options that control calculation of median statistics:

---

1. The median statistics are optional, but if requested (via the MEDIAN= data set option), they are calculated by reading all relevant data into SAS. This can significantly increase processing time.
- MEDIAN= specifies whether to calculate the median statistics. Use this data set option to control the number of data points to use when calculating median statistics. For more information, see “MEDIAN= Data Set Option” on page 63.

- CALCULATION_BASIS= specifies whether to use observed events or to interpolate data points across a summary interval. For more information, see “CALCULATION_BASIS= Data Set Option” on page 43.

Note: Only values of EventWeighted or TimeWeighted are valid when calculating median statistics.

**Plpoint: Performance Consideration**

The Plpoint table contains 128 columns that include the tag name and additional tag attributes. If you need only the list of tag names in the Plpoint table, it is recommended that you specify the KEEP=TAG data set option when you read from Plpoint. Because the list of tags can be very large, sometimes including 10,000 tags or more, using KEEP=TAG substantially improves performance when you read from this table.

**Picomp_Summary: Performance Considerations**

The Picomp_Summary table requires possibly extensive calculations in the PI Server. For better performance, use the KEEP= data set option to retain only those columns that you require.

The Median and MedianCount columns are disabled by default because calculation of the median is not performed by the PI System. Therefore, data must be loaded into SAS before the median is calculated. As a best practice, do not enable the Median and MedianCount columns (using the MEDIAN= data set option) unless they are truly required.

Note: Median statistics are not supported when you use the PI Web API to access the PI System.

**Controlling the Date Ranges When Reading from Multiple Tags**

You can specify dates to limit the records that are read from a tag. For an individual tag, you can use the START= and END= data set options to specify the date range of interest. However, if you are reading from multiple tags, you typically list those tags in a data set that you reference with the TAGLIST= data set option. You can also specify start and end dates for each tag in Start and End columns in the TAGLIST= data set. Values in the Start and End columns in the TAGLIST= data set override any values given in the START= or END= data set options for the referenced tag. For more information, see “Data Set Options for the Data Archive” on page 137.

Suppose that you need to create Start and End columns for some other purpose in the data set that you reference with the TAGLIST= data set option. In this case, you can use Start_Time and End_Time, respectively, to specify the date range to use when you read from a tag. If there is a Start and Start_Time column, then the
Start_Time column takes precedence over the Start column. Similarly, the End_Time column takes precedence over the End column.

You can specify the date value in the Start and End (or Start_Time and End_Time) columns in the following ways:

- as a character string. Use any timestamp string that is acceptable by the PI System. This is the same format that you specify for the START= and END= data set options. The PI System expects timestamps to be in a localized format, based on the settings for your system. For example, a system in the United States might require a value such as "1/20/2014 00:00:00". However, a system in France might require a value such as "20.1.2014 00:00:00".

- as a SAS date, time, or datetime value with related formats. The value that you supply is used as given. For example, you might supply a datetime value as '01JAN2014 12:00:00'dt or a date value as '01JAN2014'd.

- as a numeric value. This is not recommended and generates a warning in the log file. Numeric values are interpreted as the number of seconds since January 1, 1960 00:00:00, which is the SAS baseline date. (This differs from the PI System baseline date of January 1, 1970 00:00:00.)

For examples, see "Read Multiple Tags Simultaneously: Create a TAGLIST= Data Set with SAS Datetime Ranges" on page 144 and "Read Multiple Tags Simultaneously: Create a TAGLIST= Data Set with Text Date Ranges" on page 144.

LIBNAME Statement Specifics for the PI System Data Archive

Syntax

This LIBNAME statement associates a libref with the PI System Data Archive and enables you to read and write Data Archive tags.

LIBNAME libref PISYSTEM <SERVER=server-name> <LIBNAME-options>;

Required Arguments for the Data Archive

**libref**

specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of data sets and views.

**PISYSTEM**

specifies the SAS engine name to connect to the PI System Data Archive.

LIBNAME Options for the Data Archive

Here are the connection options that control how SAS/ACCESS manages the timing and concurrence of connections to the Data Archive.

**SERVER=<">server-name</"**

specifies the PI System server name to connect to. The value server-name is the name of the PI Server as defined in *PI System Management Tools*. Note that server-name is not necessarily the host name of the machine that is running the PI Server.
<table>
<thead>
<tr>
<th>Aliases</th>
<th>Default</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATASRC</td>
<td>the default PI Server specified for the PI System client that is installed on your computer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use quotation marks if server-name contains spaces or nonalphanumeric characters or if the server name is an IP address.</td>
<td></td>
</tr>
<tr>
<td>DOMAIN=&lt;&gt;domain-name&lt;&gt;</td>
<td>specifies the domain for authentication. Use this with USERNAME= and PASSWORD= options.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the current domain.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This option is supported when you use the PI Web API to access PI System data from a UNIX environment as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use quotation marks if domain-name contains spaces.</td>
<td></td>
</tr>
<tr>
<td>USERNAME=&lt;&gt;user-name&lt;&gt;</td>
<td>specifies the user name for authentication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Integrated Windows Authentication</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This option is supported when you use the web API to access PI System data from a UNIX environment as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use quotation marks if user-name contains spaces.</td>
<td></td>
</tr>
<tr>
<td>PASSWORD=pass</td>
<td>specifies the password for authentication.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Integrated Windows Authentication</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This option is supported when you use the web API to access PI System data from a UNIX environment as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enclose the password in quotation marks.</td>
<td></td>
</tr>
</tbody>
</table>
**HOST=host-name**
specifies the network host name of the PI Web API server. This server might
differ from the server that hosts the PI System database.

**UNIX specifics**
This option is required when you use the PI Web API to access
PI System data from a UNIX environment.

**PORT=port-ID**
specifies the port ID for the PI Web API server.

**Default**
443

**UNIX specifics**
This option is applicable only when you use the PI Web API to
access PI System data from a UNIX environment.

**DBPROMPT=YES | NO**
specifies whether to use dialog boxes to enter server, domain, user name, and
password information.

**YES** specifies to prompt the user using dialog boxes for authentication
information. When you specify YES, the DEFER= option is set to
YES.

**NO** specifies not to prompt the user for authentication information.

**Default**
NO

Here are additional LIBNAME options that you can specify in the LIBNAME
statement for the PI System Data Archive.

*Table 5.5 LIBNAME Options for the PI System Data Archive*

<table>
<thead>
<tr>
<th>LIBNAME Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMAX_TEXT=</td>
<td>32</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>HIDEFLAGS=</td>
<td>NO</td>
</tr>
<tr>
<td>PROC_DATASETS_TAGFILTER=</td>
<td>*</td>
</tr>
<tr>
<td>SHOWFLAGS=</td>
<td>YES</td>
</tr>
<tr>
<td>TESTDATE=</td>
<td>none</td>
</tr>
<tr>
<td>TIMEZONE=</td>
<td>LOCAL</td>
</tr>
</tbody>
</table>
Data Set Options for the Data Archive

These data set options are supported by the PI System. Default values are provided where applicable. For other data set options, see *SAS Data Set Options: Reference*.

Table 5.6  Data Set Options for the PI System Data Archive

<table>
<thead>
<tr>
<th>Data Set Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALCULATION_BASIS=</td>
<td>TimeWeighted</td>
</tr>
<tr>
<td>COUNT=</td>
<td>For Picomp, do not interpolate.</td>
</tr>
<tr>
<td></td>
<td>For Picomp_Summary, create only one interval that covers the time spanned by START= and END= values.</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>32</td>
</tr>
<tr>
<td>DELAY=</td>
<td>0</td>
</tr>
<tr>
<td>END=</td>
<td>12/31/9999 11:59:59 PM</td>
</tr>
<tr>
<td>FILTER=</td>
<td>none</td>
</tr>
<tr>
<td>HIDEFLAGS=</td>
<td>NO</td>
</tr>
<tr>
<td>INTERVAL=</td>
<td>For Picomp, do not interpolate.</td>
</tr>
<tr>
<td></td>
<td>For Picomp_Summary, create only one interval that covers the time spanned by START= and END= values.</td>
</tr>
<tr>
<td>LABEL=</td>
<td>none</td>
</tr>
<tr>
<td>MAX=</td>
<td>none</td>
</tr>
<tr>
<td>MEDIAN=</td>
<td>NO</td>
</tr>
<tr>
<td>MIN=</td>
<td>none</td>
</tr>
<tr>
<td>MIXED=</td>
<td>NO</td>
</tr>
<tr>
<td>SELECT=</td>
<td>ALL</td>
</tr>
<tr>
<td>SHOWFLAGS=</td>
<td>YES</td>
</tr>
<tr>
<td>SHOWINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>START=</td>
<td>1/1/1970 12:00:00.001 AM GMT</td>
</tr>
</tbody>
</table>
Sample Code for the Data Archive

Introduction to Sample Code for the Data Archive

References to specific PI tags, such as SINUSOID, SINUSOID1, and SINUSOID2, might apply to data that is not present on your system. In this case, these examples serve as illustrations of the tasks that you can perform with your own data.

In addition to the sample programs that are provided in this documentation, there are sample programs that are delivered with your software. These samples are available in \SASHOME\9.4\accesssample\sample.

Note: Because the PI System is read-only under UNIX, there is not a way to deliver additional samples on UNIX systems.

Assign the LIBNAME Statement

/* Use the default server. */
libname db pisystem;

/* Specify the server name */
libname db pisystem server=myserver;

Use PROC DATASETS and a PROC_DATASETS_TAGFILTER for List Tags

libname db pisystem proc_datasets_tagfilter="sinus*";
proc datasets lib=db;
quit;

Use PROC CONTENTS to List Tag Metadata

libname db pisystem;
proc contents data=db.pipoint varnum;
run;
proc contents data=db.sinusoid varnum;
The SAS System

The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>DB.sinusoid</th>
<th>Observations</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>7</td>
</tr>
<tr>
<td>Engine</td>
<td>PISYSTEM</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>10/14/2013 11:41:30</td>
<td>Observation Length</td>
<td>80</td>
</tr>
<tr>
<td>Last Modified</td>
<td>06/19/2014 17:00:31</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>PI POINT</td>
<td>Sorted</td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td>12 Hour Sine Wave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables in Creation Order

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tag</td>
<td>Char</td>
<td>8</td>
<td>$</td>
<td>$.</td>
</tr>
<tr>
<td>2</td>
<td>Timestamp</td>
<td>Num</td>
<td>8</td>
<td>DATETIME22.3</td>
<td>DATETIME22.3</td>
</tr>
<tr>
<td>3</td>
<td>Value</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Status</td>
<td>Char</td>
<td>32</td>
<td>$</td>
<td>$.</td>
</tr>
<tr>
<td>5</td>
<td>Questionable</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Annotated</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Substituted</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use Various Options to Print Tag Data

libname db pisystem;

proc print data=db.sinusoid;
run;

proc print data=db.sinusoid (start="07/08/2014" end="07/09/2014" interval="1:00:00");
run;

proc print data=db.sinusoid (start="07/08/2014" end="07/09/2014" count=10);
run;

proc print data=db.sinusoid (select=snapshot);
run;
Here is the output when `select=snapshot`:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Tag</th>
<th>Timestamp</th>
<th>Value</th>
<th>Status</th>
<th>Questionable</th>
<th>Annotated</th>
<th>Substituted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SINFUSOID</td>
<td>19JUN2014:17:00:31.000</td>
<td>74.8048</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Read and Write Tags**

When you use the native API, you can use SAS to create tags. You can modify (append or delete) only tags that were originally created by using SAS/ACCESS Interface to the PI System.

When you use the PI Web API, remember that you cannot write to the PI System. Therefore, you cannot create, modify, or delete a tag.

```sas
libname db pisystem;
proc delete data=db.newtag;
run;
data sinusoid;
    set db.sinusoid;
run;

data db.newtag (desc="My new tag");
    set sinusoid;
run;
```
Append Data to a PI System Tag

When you use the native API, you can modify (append or delete) only tags that were originally created by using SAS/ACCESS Interface to the PI System.

When you use the PI Web API, remember that you cannot write to the PI System. Therefore, you cannot create, modify, or delete a tag.

```
libname db pisystem;
proc delete data=db.zz;
run;
data db.zz (desc="My new tag");
   set db.sinusoid;
run;
proc append base=db.zz data=db.sinusoid;
run;
proc print data=db.zz (obs=100);
run;
```

Create a New Tag from a SAS Data Set

When you use the PI Web API, remember that you cannot write to the PI System. Therefore, you cannot create, modify, or delete a tag.

```
/* Use a data type of double. */
/* The PI System does not allow dates that are in the future. */
libname db pisystem showflags=no;
proc delete data=db.zz;
run;
data db.zz (desc="Values are double");
   format Timestamp DATETIME22.3;
   Timestamp='01jan2014:10:11:12.123'dt;
   Value=10;
   output;
   Timestamp='02feb2014:11:12:13'dt;
   Value=20;
   output;
   Timestamp='03mar2014:12:13:14'dt;
   Value=30;
   output;
run;
proc print data=db.zz;run;

/* Use a data type of string. */
libname db pisystem showflags=no;
proc delete data=db.zz;
run;
data db.zz (desc="Values are string");
   format Timestamp DATETIME22.3 Value $32.;
   Timestamp = '01jan2014:10:11:12.123'dt;
   Value="First event";
   output;
   Timestamp = '02feb2014:11:12:13'dt;
   Value="Second event";
   output;
```
Use PROC SQL

libname db pisystem;

/* WHERE is processed by SAS */
proc sql;
  select * from db.sinusoid where (Value >= 99);
quit;

/* Same result, but filter processing is pushed to the PI System */
proc sql;
  select * from db.sinusoid (min = 99);
quit;

Read Multiple Tags Simultaneously: Create a TAGLIST= Subset of Tags

Create a TAGLIST= Subset of Tags

The following code reads all tags that begin with 'sinus', and prints the last entry for each tag (select=snap):

libname db pisystem;
data list; set db.pipoint (tagfilter="sinus*");
run;

proc print data=db.picomp (taglist=list select=snap);
run;
By modifying the LIBNAME statement options, you can hide the three flag columns:

```
libname db pisystem showflags=no;
data list; set db.pipoint (tagfilter="sinus*");
run;
```

```
proc print data=db.picomp (taglist=list select=snap);
run;
```

---

## The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Tag</th>
<th>Timestamp</th>
<th>Value</th>
<th>Status</th>
<th>Questionable</th>
<th>Annotated</th>
<th>Substituted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SINUSOID</td>
<td>19JUN2014:17:00:31.000</td>
<td>74.8048</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>sinusoid1</td>
<td>19JUN2014:16:59:59.000</td>
<td>1.7047</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>sinusoid2</td>
<td>19JUN2014:17:00:31.000</td>
<td>74.8048</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>sinusoid3</td>
<td>19JUN2014:17:00:31.000</td>
<td>85.5932</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>sinusoid4</td>
<td>19JUN2014:17:01:29.000</td>
<td>7.3600</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>sinusoid5</td>
<td>19JUN2014:17:01:59.000</td>
<td>34.9829</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>sinusoid6</td>
<td>19JUN2014:16:53:59.000</td>
<td>99.3808</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>sinusoid7</td>
<td>19JUN2014:17:01:59.000</td>
<td>40.0054</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>sinusoid8</td>
<td>19JUN2014:17:01:29.000</td>
<td>5.4636</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>sinusoid9</td>
<td>19JUN2014:17:01:59.000</td>
<td>82.4985</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>SINUSOIDU</td>
<td>19JUN2014:17:00:59.000</td>
<td>0.0018</td>
<td>OK</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Read Multiple Tags Simultaneously: Create a TAGLIST= Data Set with SAS Datetime Ranges

```sas
data mytags;
  format Tag $20. Start DATETIME22.3 End DATETIME22.3;
  Tag="SINUSOID"; Start='01JAN2014 12:00:00'dt; End='01JAN2014 18:00:00'dt;
  output;
  Tag="SINUSOID1"; Start='02JAN2014 12:00:00'dt; End='02JAN2014 18:00:00'dt;
  output;
  Tag="SINUSOID2"; Start='03JAN2014 12:00:00'dt; End='03JAN2014 18:00:00'dt;
  output;
run;
```

```sas
proc print data=mytags; run;
```

```sas
libname db pisystem;
proc print data=db.PICOMP (TAGLIST=mytags); run;
```

This code results in two listings. The first listing prints the Mytags data set that you define in the first DATA step. The second listing prints the sensor readings from 12 noon until 6 p.m. on January 1, 2014 through January 3, 2014 for tags SINUSOID, SINUSOID1, and SINUSOID2.

Read Multiple Tags Simultaneously: Create a TAGLIST= Data Set with Text Date Ranges

```sas
data mytags;
  format Tag $20.;
  Tag="SINUSOID"; Start="1/1/2014 12:00:00"; End="1/1/2014 18:00:00";
  output;
  Tag="SINUSOID1"; Start="1/2/2014 12:00:00"; End="1/2/2014 18:00:00";
  output;
  Tag="SINUSOID2"; Start="1/3/2014 12:00:00"; End="1/3/2014 18:00:00";
  output;
run;
```

```sas
proc print data=mytags; run;
```

```sas
libname db pisystem;
proc print data=db.PICOMP (taglist=mytags); run;
```

This code results in two listings. The first listing prints the Mytags data set that you define in the first DATA step. The second listing prints the sensor readings from 12 noon to 6 p.m. on January 1, 2014 through January 3, 2014 for tags SINUSOID, SINUSOID1, and SINUSOID2.

Use Complex Nested Picomp and Plpoint Virtual Tables

```sas
libname db pisystem;
proc compare
  base=db.picomp (taglist=db.plpoint (where={lower(tag)=:"sinus"}))
```
compare=db.picomp (taglist=db.pipoint (tagfilter="sinus*"));
run;

This results in an equal comparison because the WHERE clause and the
TAGFILTER= option that is used in the base and compare tables generate the same
results.

Read PI Data from a Microsoft Excel Spreadsheet
This example uses Microsoft Excel to provide the list of tags to read. This example
assumes that you have SAS/ACCESS Interface to PC Files.

libname xls pcfiles path="C:\MyTagList.xlsx";
proc print data=db.picomp (taglist="xls.sheet1$" );
run;

Use the TESTDATE Option
This output from this example shows how PI System dates are displayed in SAS.

libname db pisystem
testdate="*"
testdate="*-1h"

Print Summaries from the Picomp_Summary Table
Assume that the List data set was created as shown in "Read Multiple Tags
Simultaneously: Create a TAGLIST= Subset of Tags" on page 142. Use the
following PRINT statements to print different summary data.

/* For each tag 'list' dataset, print a single 24-hr event-weighted
summary */
proc print data=db.picomp_summary (taglist=list start="8/1/2016"
end="8/2/2016"
calcbasis=ew);
run;

/* For each tag 'list' dataset, print a single 24-hr time-weighted
summary */
/* with
median                                                               */
proc print data=db.picomp_summary (taglist=list start="8/1/2016"
end="8/2/2016"
calcbasis=tw median=yes);
run;

/* For each tag 'list' dataset, print hourly time-weighted summaries
with median*/
proc print data=db.picomp_summary (taglist=list start="8/1/2016"
end="8/2/2016"
count=24 calcbasis=tw median=yes);
run;

/* For each tag 'list' dataset, print hourly time-weighted summaries
with median*/
Using the PI System Asset Framework

LIBNAME Statement Specifics for the PI System Asset Framework

Syntax
Here is the LIBNAME statement syntax for accessing the Asset Framework.

LIBNAME libref PIAF <LIBNAME-options>;

Required Arguments for the Asset Framework

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, or group of tables and views.

PIAF
specifies the SAS/ACCESS engine name for the PI System Asset Framework interface.

Alias PISYSAF

LIBNAME Options for the Asset Framework

Here are the connection options that control how SAS/ACCESS manages the timing and concurrence of connections to the Asset Framework.

DATABASE=<'database-name'>
specifies the name of the PI System Asset Framework database. If the database name contains spaces or nonalphanumeric characters, enclose the name in quotation marks.

Alias SCHEMA=

Windows default PI System database server

UNIX default None. You must supply a value when you use the PI Web API.

DBPROMPT=YES | NO
specifies whether to use dialog boxes to prompt for server, domain, user name, and password information.
DOMAIN=<domain-name>
specifies the domain for authentication. Use the DOMAIN=, USER=, and PASSWORD= options together, if you specify those options. This option is not required if you use Integrated Windows Authentication for your system.

Default: current domain

Restriction: Use of the DOMAIN=, USER=, and PASSWORD= options is discouraged by OSIsoft.

UNIX specifics: This option is supported when you use the web API to access PI System data as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.

HOST=host-name
specifies the network host name of the PI Web API server. This server might differ from the server that hosts the PI System database.

UNIX specifics: This option is required when you access PI System data by using the PI Web API.

PASSWORD=<password>
specifies the password for authentication. Use the DOMAIN, USER, and PASSWORD options together, if you specify those options. This option is not required if you use Integrated Windows Authentication for your system.

Alias: PASS=

Restriction: Use of the DOMAIN=, USER=, and PASSWORD= options is discouraged by OSIsoft.

UNIX specifics: This option is supported when you use the web API to access PI System data as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.

PORT=port-ID
specifies the port ID for the PI Web API server.

Default: 443

UNIX specifics: This option is applicable only when you access PI System data by using the PI Web API.

SERVER=<server-name>
specifies the PI System server name. The value of server-name is the name of the PI server as defined in the PI System Management Tools. If the server name contains spaces or nonalphanumeric characters or if it is an IP address, enclose it in quotation marks.

Windows default: Default PI System server

UNIX default: Value of HOST=
**USER= user-name**

specifies the user name for authentication. Use the DOMAIN, USER, and PASSWORD options together, if you specify those options. This option is not required if you use Integrated Windows Authentication for your system.

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Restriction</strong></td>
<td>Use of the DOMAIN=, USER=, and PASSWORD= options is discouraged by OSIsoft.</td>
</tr>
<tr>
<td><strong>UNIX specifics</strong></td>
<td>This option is supported when you use the web API to access PI System data as long as you use BASIC authentication. Use the SAS_PI_WEB_AUTH= environment variable to specify the authentication method.</td>
</tr>
</tbody>
</table>

Here are the additional LIBNAME options that you can specify in the LIBNAME statement for the PI System Asset Framework.

**Table 5.7 LIBNAME Options for the PI System Asset Framework**

<table>
<thead>
<tr>
<th>LIBNAME Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMAX_TEXT=</td>
<td>256</td>
</tr>
<tr>
<td>DEFER=</td>
<td>YES</td>
</tr>
<tr>
<td>HIDEOPTVARS=</td>
<td>NO</td>
</tr>
<tr>
<td>TIMEZONE=</td>
<td>LOCAL</td>
</tr>
</tbody>
</table>

**Data Set Options for the Asset Framework**

These data set options pertain to the virtual tables in the PI System Asset Framework. For other data set options, see *SAS Data Set Options: Reference*.

**Table 5.8 Data Set Options for the PI System Asset Framework**

<table>
<thead>
<tr>
<th>Data Set Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE=</td>
<td>none</td>
</tr>
<tr>
<td>BOUNDARY=</td>
<td>none</td>
</tr>
<tr>
<td>CATEGORY=</td>
<td>none</td>
</tr>
<tr>
<td>CHILD=</td>
<td>ALL</td>
</tr>
<tr>
<td>COUNT=</td>
<td>none</td>
</tr>
</tbody>
</table>
### Accessing the Asset Framework via Virtual Tables

Due to the hierarchical structure of the Asset Framework's elements and event frames, SAS maps access to these through a set of virtual tables. Specify virtual table names in SAS as if they were regular table names, such as `mydb.element_attribute`. Here is the list of virtual tables that SAS/ACCESS uses:

- Element
- Element_Attribute
- Eventframe
- Eventframe_Attribute
- Eventframe_TSdata
- TSdata_Attribute
- TSdata_PIpoint
- TSdata_Summary

Use the `SEARCHBY=` data set option in conjunction with the `NAME=`, `PATH=`, `ATTRIBUTE=`, `CATEGORY=`, `TEMPLATE=`, or `GUID=` data set options to select the desired Asset Framework elements or event frames. The following example shows how to define a `LIBNAME` connection to the PI System Asset Framework and query
the T_385 element at a specific path. The second SELECT statement creates a
table called Testdata by querying elements that start with “T_38*”.  

libname mydb piaf server=myAFsvr database=PI_AF;

proc sql;  
    * query an element in a location;
    select * from mydb.element(searchby=path
                                path="\\host\pi_af\Region\Equipment\T_385");
    * create a table;
    create table work.testdata as
    select * from mydb.element(searchby=name name="T_38*");
    quit;

---

### About the Asset Framework Virtual Tables

#### Element Virtual Table

The Element table is a virtual table in the Asset Framework. This table contains information about elements in the Asset Framework. You can use the following data set options when you work with the Element table:

- **CHILD=**
- **DBMAX_TEXT=**
- **HIDEOPTVARS=**
- **SEARCHBY=** with related data set options
  - **ATTRIBUTE=**
  - **NAME=**
  - **CATEGORY=**
  - **PATH=**
  - **GUID=**
  - **TEMPLATE=**
- **TIMEZONE=**

#### Table 5.9  Columns in the Element Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElementID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>ElementName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Type</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Path</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>HasChildren</td>
<td>Numeric</td>
<td></td>
</tr>
</tbody>
</table>
IsRoot  Numeric
Description  Character
Categories  Character
Template  Character
Parent  Character

Element.Attribute Virtual Table

The Element.Attribute table is a virtual table that contains information about element attributes. You can use the following data set options when you work with the Element.Attribute table:

- ATTRIBUTE=
- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- SEARCHBY= with related data set options
  - ATTRIBUTE=  NAME=
  - CATEGORY=  PATH=
  - GUID=  TEMPLATE=
- TIMEZONE=

Table 5.10  Columns in the Element.Attribute Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>AttributeName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Type</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Path</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Categories</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>Template</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>
Eventframe Virtual Table

The Eventframe virtual table contains a list of event frames and properties for a set of queried event frames. You can specify the following data set options when you work with the Eventframe table:

- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- SEARCHBY= with related data set options
  - ATTRIBUTE= NAME=
  - CATEGORY= PATH=
  - GUID= TEMPLATE=
- START= and END=
- TIMEZONE=

Table 5.11  Columns in the Eventframe Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>EventName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>StartTime</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>EndTime</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Description</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>Categories</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>PI System Data Type</td>
<td>Displayed When HIDEOPTVARS=YES</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Template</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>PrimaryElementID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>PrimaryElementPath</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

**Eventframe_Attribute Virtual Table**

The Eventframe_Attribute virtual table contains a list of attributes and properties for a set of queried Event Frames. You can use the following data set options when you work with the Eventframe_Attribute table:

- ATTRIBUTE=
- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- SEARCHBY= with related data set options
  - ATTRIBUTE= NAME=
  - CATEGORY= PATH=
  - GUID= TEMPLATE=
- START= and END=
- TIMEZONE=

*Table 5.12  Columns of the Eventframe_Attribute Table*

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>AttributeName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>AttributePath</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Type</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Description</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>Categories</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>
## Eventframe_TSdata Virtual Table

The Eventframe_TSdata virtual table contains the time series data for event frames. You can use the following data set options when you work with the Eventframe_TSdata table:

- `ATTRIBUTE=`
- `CHILD=`
- `DBMAX_TEXT=`
- `HIDEOPTVARS=`
- `MODEL=` with related data set options
  - `BOUNDARY=`
  - `INTERVAL=`
  - `COUNT=`
  - `START=` and `END=`
- `SEARCHBY=` with related data set options
  - `ATTRIBUTE=`
  - `NAME=`
  - `CATEGORY=`
  - `PATH=`
  - `GUID=`
  - `TEMPLATE=`
  - `START=` and `END=`
  - `TIMEZONE=`

### Table 5.13: Columns of the Eventframe_TSdata Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>AttributeName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Column</td>
<td>PI System Data Type</td>
<td>Displayed When HIDEOPTVARS=YES</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>ValueN</td>
<td>Numeric</td>
<td>●</td>
</tr>
<tr>
<td>Status</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Questionable</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Annotated</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Substituted</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>PointClass</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>PointType</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>PIPointID</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>PIPointPath</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>EventID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>EventPath</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

**TSdata_Attribute Virtual Table**

The TSdata_Attribute table is a virtual table that contains time series data for an element attribute. You can use the following data set options when you work with the TSdata_Attribute table:

- ATTRIBUTE=
- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- MODEL= with related data set options
  - BOUNDARY= INTERVAL=
  - COUNT= START= and END=
- SEARCHBY= with related data set options
  - ATTRIBUTE= NAME=
  - CATEGORY= PATH=
  - GUID= TEMPLATE=
- TIMEZONE=
### Table 5.14  Columns in the TSdata_Attribute Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>AttributeName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Type</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Status</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Questionable</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Annotated</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Substituted</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>DataReference</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>PIPointID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>PIPointPath</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>ElementID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>ElementPath</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

### TSdata_PIpoint Virtual Table

The TSdata_PIpoint virtual table contains time series data for an element PI point. You can use the following data set options when you work with the TSdata_PIpoint table:

- ATTRIBUTE=
- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- MODEL= with related data set options
- BOUNDARY=  INTERVAL=
- COUNT=  START= and END=
- SEARCHBY= with related data set options
### Table 5.15  Columns in the TSDATA_PIPOINT Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPointID</td>
<td>Numeric</td>
<td>●</td>
</tr>
<tr>
<td>PIPointPath</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>ValueN</td>
<td>Numeric</td>
<td>●</td>
</tr>
<tr>
<td>Status</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Questionable</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Annotated</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>Substituted</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>PointClass</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>PointType</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>ElementID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>ElementPath</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

### TSdata_Summary Virtual Table

The TSdata_Summary virtual table contains a summary of time series data from PI Points in queried elements. You can use the following data set options when you work with the TSdata_Summary table:

- ATTRIBUTE=
- CHILD=
- DBMAX_TEXT=
- HIDEOPTVARS=
- SEARCHBY= with related data set options
Table 5.16  Columns in the TSDATA_SUMMARY Table

<table>
<thead>
<tr>
<th>Column</th>
<th>PI System Data Type</th>
<th>Displayed When HIDEOPTVARS=YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID</td>
<td>GUID</td>
<td>●</td>
</tr>
<tr>
<td>AttributeName</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Statistics</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Value</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Unit</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Status</td>
<td>Character</td>
<td>●</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Datetime</td>
<td>●</td>
</tr>
<tr>
<td>PIPointID</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>PIPointPath</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>ElementID</td>
<td>GUID</td>
<td></td>
</tr>
<tr>
<td>ElementPath</td>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

Sample Code for the Asset Framework

About the Sample Code for the Asset Framework

Because the Asset Framework is read-only, the examples in this chapter work only if the OSIsoft-supplied sample data is installed on your server. References to specific PI tags, such as K_023, might apply to data that is not present on your system. If the referenced PI tags are not present, these examples serve as illustrations of the tasks that you can perform with your own data.

There are additional code samples that are delivered with your software. These samples are available in %SASHOME%/9.4/accesssample/sample.
These code samples are read-only and read data that might be available in OSIsoft sample data files. If the sample data files are not available on your system, then the code samples serve as illustrations of code usage.

**Note:** Because the PI System is read-only under UNIX, there is not a way to deliver additional samples on UNIX systems.

---

Connect to Asset Framework Server and List Elements

```sql
/* Connect to specified server and database */
libname db piaf server=myserver database=mydatabase;

/* List all member elements in the database */
proc print data=db.element (searchby=name name="*" );
run;
```

Use PROC PRINT to list elements in the PI System Asset Framework. Using PROC DATASETS results in a list of the Asset Framework virtual tables.

---

List the Attributes for an Element

```sql
proc print data=db.element_attribute (searchby=path
    path="\host\PI_AF\region\K_023" );
run;
```

The PRINT procedure displays the attributes of the K-023 element.

**Note:** K_023 is the SAS version of the K-023 element name. K_023 is listed when you run the DATASETS procedure to see the element names in a database.

---

Access Data by Using Virtual Tables

```sql
proc sql;

* select variables from an element;
select Path, AttributeName, Value, Timestamp from
    db.element (searchby=path path="\host\PI_AF\region\K_023"
    hideoptvars=yes);

* find element under specified path ;
select ElementName from db.element (searchby=path path="\host\PI_AF
\region\K_023"
    child=all);

* find named attribute and its value under element path ;
select Path, Value from db.element_attribute
    (searchby=path path="\host\PI_AF\region\cityA\System Process\T_038" 
    child=leaves)
    where AttributeName = "Power Savings" ;
```
* list the recorded time series data of the attributes of elements:
  H-431;
  select ElementID, AttributeName, Timestamp, Value, DataReference
    from db.tsdata_attribute
    (searchby=name name="H-431" start='y' end='t' datamodel=archive)
    where DataReference = "PI Point";

* list the recorded time series data of the pipoints of element:
  H-431;
  select ElementID, PIPointPath, Timestamp, Value from db.tsdata_pipoint
    (searchby=name name="H-431" start='y' end='t' datamodel=archive);

* list the recorded time series data of the pipoints of attribute:
  Process Feedrate of element: H-431;
  select ElementID, PIPointPath, Timestamp, Value from db.tsdata_pipoint
    (searchby=name name="H-431" attribute="Process Feedrate"
     start='y' end='t' datamodel=sampled interval="1d");

* query all templates used in the connect database;
  select unique Template from db.element(searchby=name name="*");

* query all categories used in the connect database;
  select unique Categories from db.element(searchby=name name="*");

* list the recorded time series summary data of element "B-235";
  select * from db.tsdata_summary
    (searchby=name name="B-235" start='y' end='t');

* list the recorded eventframe data for element "B-235";
  select * from db.EventFrame
    (searchby=name name="B-235" start='y' end='t');

* find event attribute and its value for element "B-235";
  select * from db.EventFrame_Attribute
    (searchby=name name="B-235" start='y' end='t');

* list the recorded time series event frame data for element "B-235";
  select * from db.EventFrame_tsdata
    (searchby=name name="B-235" start='y' end='t');

quit;

Use DATASETS and CONTENTS Procedures and Print Element Data
libname db clear;

  * maximum string length default to 256;
libname db piaf server=user database=user_database dbmax_text=976;

  * to list all elements in the connected database;
proc print data=db.element (searchby-name name="*");
run;

  * to list information for element B_235 using the full path;
Using the PI System Asset Framework

```sql
proc contents data=db.element(searchby=path path="\host\PI_AP\region\B_235")
   varnum;
run;

* variables selection with extended string length;
proc contents data=db.element(searchby=path path="\host\PI_AP\region\B_235")
   keep=AttributeName Timestamp Value Path dbmax_text=976) varnum; run;
proc contents data=db.element(searchby=path path="\host\PI_AP\region\B_235")
   drop=AttributeID Type UnitOfMeasure) varnum; run;
proc contents data=db.element(searchby=path path="\host\PI_AP\region\B_235")
   hideoptvars=yes) varnum; run;

* data read;
proc sql;
select * from db.element(searchby=path path="\host\PI_AP\region\B_235");
* select columns;
select AttributeName, Timestamp, Value, Path from 
db.element(searchby=path path="\host\PI_AP\region\B_235");
* select columns with keep option;
select * from db.element(searchby=path path="\host\PI_AP\region\B_235"
   keep=AttributeName Timestamp Value Path);
* set gmt, notice the column name change;
select * from db.element(searchby=path path="\host\PI_AP\region\B_235"
   timezone=utc keep=AttributeName Timestamp_gmt Value);
quit;

proc print data=db.element(searchby=path path="\host\PI_AP\region\B_235");
run;
* hide optional columns;
proc print data=db.element(searchby=path path="\host\PI_AP\region\B_235"
   hideoptvars=yes); run;
* use utc time;
proc print data=db.element(searchby=path path="\host\PI_AP\region\B_235"
   hideoptvars=yes timezone=utc); run;
* use local time - default;
proc print data=db.element(searchby=path path="\host\PI_AP\region\B_235"
   hideoptvars=yes timezone=local); run;

* select columns;
proc print data=db.element(searchby=path path="\host\PI_AP\region\B_235"
   keep=AttributeName Timestamp Value Path); run;
* refer Timestamp_GMT instead of Timestamp when timezone=gmt;
```
proc print data=db.element(searchby=path path="\\host\PI_AF\region \B_235"
    keep=AttributeName Timestamp_GMT Value Path hideoptvars=yes
    timezone=gmt);
run;

/* Use START= and END= to limit the time range for the Time Series data */
/* Omitting START= and END= might result in an excessive number of records */
proc print data=db.tsdata_attributes(searchby=name name='B-235'  
    start="01/01/1970 12:00:00.001 am gmt" end="*"
    start=01/01/1970 12:00:00.001 am gmt end="*"
); run;
proc print data=db.tsdata_pipoint(searchby=name name='B-235'
    start='y' end='t'); run;
proc print data=db.tsdata_pipoint(searchby=name name='B-235'
    start='y' model=count count=5); run;
About SAS/ACCESS Interface to Salesforce

Overview

Salesforce is a customer relationship management (CRM) platform rather than a standard relational database. Here are some ways that you can use it through its customer portal.

- handle case and task management
- automatically route and escalate important events
- track your own cases
join conversations about your company on social networking websites with its social networking plug-in

receive email alerts, perform Google searches, and access your entitlement and contracts by using Salesforce analytical tools and other services

For more information about Salesforce, see What Is Salesforce?

SAS/ACCESS Interface to Salesforce lets you transfer data between your Salesforce document database and the SAS client. For more information about Salesforce, see your Salesforce documentation.

Note: SAS/ACCESS Interface to Salesforce does not support CREATE VIEW.

Support for SAS/ACCESS Interface to Salesforce is added in April 2019.

Beginning in SAS Viya 3.5, SAS/ACCESS Interface to Salesforce includes SAS Data Connector to Salesforce. The data connector enables you to transfer large amounts of data between Salesforce and the CAS server for parallel processing. For more information, see these topics:


Security Considerations

With only a basic connection with no options, the Salesforce interface defaults to a secure Transport Layer Security (TLS) connection with server-side authentication. In this case, your session data is encrypted over the wire, and SAS verifies the identity of Salesforce using a public certificate that a code authority signs.

The Salesforce interface also supports mutual authentication. After you enable this feature, you can upload your own certificate that a code authority to Salesforce signs. This authority then verifies your identity upon connection and also for each subsequent query.

For more information about security, see Salesforce Security Guide.

Supported Features for SAS/ACCESS Interface to Salesforce

Here are the features that SAS/ACCESS Interface to Salesforce supports. To find out which versions of Salesforce are supported, see your system requirements documentation.

There is no support for writing to Salesforce from SAS.
Table 6.1  Features by Host Environment for Salesforce

<table>
<thead>
<tr>
<th>Platform</th>
<th>SAS/ACCESS LIBNAME Statement</th>
<th>SQL Pass-Through Facility</th>
<th>Bulk-Load Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Linux</td>
<td>●</td>
<td>●</td>
<td>no</td>
</tr>
</tbody>
</table>

LIBNAME Statement for the Salesforce Engine

Syntax

This section describes the LIBNAME statement that SAS/ACCESS Interface to Salesforce supports. This LIBNAME statement associates a libref with a Salesforce data source and enables you to read from Salesforce.

Here is the LIBNAME statement syntax for accessing Salesforce.

LIBNAME  libref  SFORCE  <connection-options>  <LIBNAME-options>;

Required Arguments for Salesforce

libref specifies any SAS name that serves as an alias to associate SAS with a data source, schema, server, or group of tables and views.

SFORCE specifies the SAS/ACCESS engine name for the Salesforce interface.

Optional Arguments for Salesforce

connection-options provide connection information and control how SAS manages the timing and concurrence of the connection to the Salesforce data source.

NOPROMPT='Salesforce-connection-options'
specifies connection options for Salesforce. Separate multiple options with a semicolon. If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you with the connection string.

PWD='Salesforce-password'
USING='Salesforce-password'
specifies the password for authentication.

Alias: PASS=, PASSWORD=, PW=
USER="Salesforce-user-name' specifies the user information for authentication.

Alias: UID=

Table 6.2 SAS/ACCESS LIBNAME Options for Salesforce

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Valid in CONNECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>API_TRACE=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>AUTHDOMAIN=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>AUTHENDPOINT=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
<td>●</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
<td>●</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
<td>●</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
<td>●</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>MUTUAL_AUTH=</td>
<td>NO</td>
<td>●</td>
</tr>
<tr>
<td>PRESERVE_TAB_NAMES=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>PROXY_HOST=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>PROXY_PASS=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>PROXY_PORT=</td>
<td>8080</td>
<td>●</td>
</tr>
<tr>
<td>PROXY_USER=</td>
<td>none</td>
<td>●</td>
</tr>
<tr>
<td>QUOTE_CHAR=</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>ROWSET_SIZE=</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SHOW_METADATA=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>SHOW_RECYCLED=</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>SPOOL=</td>
<td>YES</td>
<td>●</td>
</tr>
<tr>
<td>SQL_FUNCTIONS_COPY=</td>
<td>none</td>
<td>●</td>
</tr>
</tbody>
</table>
### Salesforce LIBNAME Statement Examples

This example specifies the JAR file path that the Salesforce interface uses on a Windows system.

`'c:\lib1;c:\lib2'`

In this example, USER= and PWD= are connection options.

```
libname x sforce user='myusr1' pwd='mypwd1';
proc datasets lib=x;quit;
proc contents varnum data=x.Account;run;
```

### Data Set Options for Salesforce

All SAS/ACCESS data set options in this table are supported for Salesforce. Default values are provided where applicable. When you specify both the LIBNAME and data set option, the data set option value takes precedence.

Table 6.3  SAS/ACCESS Data Set Options for Salesforce

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>LIBNAME option value</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBMAX_TEXT=</td>
<td>1024</td>
</tr>
<tr>
<td>DBNULL=</td>
<td>YES</td>
</tr>
<tr>
<td>Option</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSASTYPE=</td>
<td>see Data Types for Salesforce</td>
</tr>
<tr>
<td>LABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>ROWSET_SIZE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>none</td>
</tr>
</tbody>
</table>

Known Limitation When Working with SAS/ACCESS Interface to Salesforce

SAS/ACCESS Interface to Salesforce does not support positional inserts, updates, or deletions. Therefore, the MODIFY statement in a DATA step is not supported for Salesforce.

SQL Pass-Through Facility Specifics for Salesforce

Key Information

The Salesforce interface supports only explicit Salesforce Object Query Language (SOQL) pass-through.

Passing SAS Functions to Salesforce

SAS/ACCESS Interface to Salesforce automatically passes the following SAS functions to Salesforce for processing. Where the Salesforce function name differs from the SAS function name, the Salesforce name appears in parentheses. For a
multiple libref join to pass to Salesforce, all of these components of the LIBNAME statements must match exactly.

<table>
<thead>
<tr>
<th>AVG</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT (*)</td>
<td>MAX</td>
</tr>
<tr>
<td>fieldName</td>
<td>SUM</td>
</tr>
</tbody>
</table>

Data Types for Salesforce

Overview

Every column in a table has a name and a data type. The data type tells Salesforce how much physical storage to set aside for the column and the form in which the data is stored. This section includes information about Salesforce data types, null and default values, and data conversions.

Salesforce data types are classified as primitive or field types. Primitives are comparable to SAS data types and represent such basic types as Booleans, integers, and timestamps. Field types are formatted data that still use underlying primitive types. These distinctions are not relevant for users, who can simply select a data type from a list primitive and field types.

For more information about Salesforce data types and to determine which data types are available for your version of Salesforce, see your Salesforce documentation.

Supported Salesforce Data Types

Here are the data types that SAS/ACCESS Interface to Salesforce supports.

- Character data
  - ADDRESS
  - ANYTYPE
  - BASE64
  - COMBOBOX
  - DATACATEGORYGROUPREFERENCE
  - EMAIL
  - ENCRYPTEDSTRING
  - ID
  - JUNCTIONIDLIST
  - LOCATION
  - MASTERRECORD
  - MULTIPICKLIST
  - PHONE
  - PICKLIST
  - REFERENCE
  - STRING
  - TEXTAREA
  - URL

- Numeric data
  - BOOLEAN
  - DOUBLE
  - BYTE
  - INT
Date and time data

Table 6.4  Supported Salesforce Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>No time-zone conversion.</td>
</tr>
<tr>
<td>TIME</td>
<td>No time-zone conversion. This is displayed as it is stored in the database, as is done in the Salesforce web interface.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>The database stores these in UTC (GMT). When writing data, SAS converts from local time to UTC. When reading data, SAS converts from UTC to local time. An important consideration is how SAS determines the local time-zone offset—namely, that it could change when Daylight Saving Time (DST) is in effect. For example, the offset for the US east coast is -5 hours in winter and -4 hours in summer, when DST in effect. By default, SAS uses the current date to determine whether DST is in effect. Users typically want to use the date of the actual timestamp to be converted as the basis to determine whether DST is in effect. So that this can work as intended, use the SAS TIMEZONE system option with the long form of the time zone. Here are the ways that you can specify the TIMEZONE option.</td>
</tr>
<tr>
<td></td>
<td>- On the command line that starts SAS:</td>
</tr>
<tr>
<td></td>
<td>sas -timezone &quot;America/New_York&quot; [other options...]</td>
</tr>
<tr>
<td></td>
<td>- Within SAS or any AUTOEXEC script:</td>
</tr>
<tr>
<td></td>
<td>option timezone=&quot;America/New_York&quot;;</td>
</tr>
</tbody>
</table>

**IMPORTANT** Do not use the short form TIMEZONE option values (such as ET, EST, or EDT) because they use today’s date to determine the time-zone offset instead of reading the actual date.

Data Type Conversions When Loading Salesforce Data into SAS

The following table shows the data type conversions for data that is loaded from Salesforce into SAS.
Table 6.5  Data Type Conversions from Salesforce into SAS

<table>
<thead>
<tr>
<th>Salesforce Data Type</th>
<th>SAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Data</td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>CHAR</td>
</tr>
<tr>
<td>ANYTYPE</td>
<td>CHAR</td>
</tr>
<tr>
<td>BASE64</td>
<td>CHAR</td>
</tr>
<tr>
<td>COMBOBOX</td>
<td>CHAR</td>
</tr>
<tr>
<td>DATACATEGORYGROUPREFERENCE</td>
<td>CHAR</td>
</tr>
<tr>
<td>EMAIL</td>
<td>CHAR</td>
</tr>
<tr>
<td>ENCRYPTEDSTRING</td>
<td>CHAR</td>
</tr>
<tr>
<td>ID</td>
<td>CHAR</td>
</tr>
<tr>
<td>JUNCTIONIDLIST</td>
<td>CHAR</td>
</tr>
<tr>
<td>LOCATION</td>
<td>CHAR</td>
</tr>
<tr>
<td>MASTERRECORD</td>
<td>CHAR</td>
</tr>
<tr>
<td>MULTIPICKLIST</td>
<td>CHAR</td>
</tr>
<tr>
<td>PHONE</td>
<td>CHAR</td>
</tr>
<tr>
<td>PICKLIST</td>
<td>CHAR</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>CHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>CHAR</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td>CHAR</td>
</tr>
<tr>
<td>URL</td>
<td>CHAR</td>
</tr>
<tr>
<td>Numeric Data</td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BYTE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>INT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>PERCENT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>Date and Time Data</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>