SAS® Life Science Analytics Framework: SAS Macro API 2.4 User's Guide

2.4*

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

SAS® Documentation
December 16, 2020

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® Life Science Analytics Framework: SAS Macro API 2.4 User's Guide*. Cary, NC: SAS Institute Inc.

SAS® Life Science Analytics Framework: SAS Macro API 2.4 User's Guide

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

December 2020

2.4-P1:lsafmapiug

Contents

1

Audience

Audience

This guide is intended for users who want to access functionality within the SAS Life Science Analytics Framework using macros.

You must be familiar with SAS Life Science Analytics Framework functionality, such as type definitions, contexts, files, and access permissions. For reference information about SAS Life Science Analytics Framework functionality, see the SAS Life Science Analytics Framework online Help and User's Guide.

2

Installing the Macros

Overview

This chapter describes how to install the SAS Life Science Analytics Framework Macro API, which is distributed in the lsaf-sas-macro-2.4.zip.

Requirements

The SAS Life Science Analytics Framework Macros requires these items:

- SAS Life Science Analytics Framework Java API client version 2.4
- For PC SAS, at least SAS 9.4M7

Install Macros on Microsoft Windows

1 Follow the instructions in the *Getting Started with the SAS Life Science Analytics Framework Java API* document to install the Java API client.

Be sure to note the location of the lib directory, which is typically:

C:\lsaf-java-api-client-2.4\lib

2 Unzip the contents of lsaf-sas-macro-2.4.zip to C:\.

This step creates these files and folders in C:\:

- **lsaf-sas-macro-2.4\conf**

  This folder contains example configuration files.

- **lsaf-sas-macro-2.4\docs**

  This folder contains the documentation for the SAS Life Science Analytics Framework Macro API, which includes the detailed documentation that describes all of the macros that are delivered with the distribution.

- **lsaf-sas-macro-2.4\lib**

  This folder contains the sas.lsaf.api.macro.jar file.

- **lsaf-sas-macro-2.4\sasmacros**

  This folder contains the SAS Life Science Analytics Framework macros as .sas files.

3 Determine the location of your SAS installation and the configuration file.

In a typical Windows Unicode support installation, !sasroot points to this location:

C:\Program Files\SASHome\SASFoundation\9.4\nls\u8

4 Back up the file **!sasroot\sasv9.cfg**.

You will edit it in the next step.

CAUTION

Use extreme care when you edit this file, and modify only the indicated text. Ensure that you do not insert any carriage returns in the sas.app.class.dirs option. If you have any questions, concerns, or problems, contact SAS Technical Support.

5 Edit the file **!sasroot\sasv9.cfg** to add these lines near the top of the file, immediately above the comment box with the “WARNING:” label in it:

```
/* define the location of the SAS Life Science Analytics Framework Macro API */
-append sasautos "C:\lsaf-sas-macro-2.4\sasmacros"

/* put both the macro and java api client jars on the classpath */
-JREOPTIONS (-Dsas.app.class.dirs=C:\lsaf-sas-macro-2.4\lib;C:\lsaf-java-api-
client-2.4\lib)

/* prevent a classpath not set warning from javaobj */
-SET CLASSPATH !CLASSPATH
```

6 Save the file and start a new SAS session to verify the installation.

Verify the Installation

In the SAS session, run the following SAS code to display the settings for the JREOPTIONS and to verify that the JRE is configured properly.

This code also verifies that the SAS Life Science Analytics Framework macros are installed and functioning as expected.

Note: Replace *lsaf-instance*, *lsaf-user-ID*, *lsaf-password*, and *valid-container-path* with the values for your instance of the SAS Life Science Analytics Framework.

```
/* verify the JRE settings */
options mprint;
proc javainfo;
run;


/* initiate a connection to SAS Life Science Analytics Framework */
%lsaf_login(lsaf_url=%str(https://lsaf-instance),
lsaf_userid=%str(lsaf-user-ID),
lsaf_password=%str(lsaf-password));


/* print version information */
%lsaf_getapiversions();


/* List the contents of a folder in the SAS Life Science Analytics Framework repository
*/
%lsaf_getchildren(lsaf_path=%str(valid-container-path));

proc print;
    title "List of Items in valid valid-container-path";

run;


/* terminate the connection to the SAS Life Science Analytics Framework */

%lsaf_logout();
```

The code generates a list of the contents in the specified container that is in the SAS Life Science Analytics Framework repository.

The SAS log file contains information that might be useful for debugging the installation of the SAS Life Science Analytics Framework macros.

3

SAS Life Science Analytics Framework Macros

Introduction

The SAS Life Science Analytics Framework SAS Macro API enables you to use familiar SAS macro syntax to act on the content that is in the repository and workspace.

SAS Macro Return Codes

After the execution of each macro, the global macro variable _LSAFRC_ contains a return code that indicates the success or failure of the operation. The global macro variable _LSAFMSG_ contains text information that indicates the success or the cause of the failure.

Table 3.1 SAS Macro Return Codes

Value	Explanation
0	The macro executed without error.
-1	The macro executed with an error. See return message for error details.
-100	There is no SAS Life Science Analytics Framework session. This is applicable only when calling a macro from PC SAS.
-200	Invalid records were found in the input data set for a query macro.
-300	One or more notes were reported as a result of a clinical import operation.
-301	One or more warnings were reported as a result of a clinical import operation.
-302	One or more errors were reported as a result of a clinical import operation.
-500	An unexpected error has occurred.
-999	No return code was set.

Using the Macros

Using the Ampersand Character (&) in URLs

For a macro with a parameter that specifies a URL, such as a macro that sets properties, you cannot embed the ampersand character (&) in the URL. The ampersand character is a special character in SAS. If you embed an ampersand character, SAS attempts to resolve the subsequent text as a macro variable.

The Proper Case for Parameter Values

Although SAS is case insensitive, the parameter values passed to the SAS Life Science Analytics Framework might be case sensitive.

Quoting Parameter Values

To ensure consistent results, it is recommended that parameters of type String be specified one of the string functions, such as %str() or %nrbquote(). Using double quotation marks results in a SAS syntax error.

4

Queries

Overview

Query macros extract data that is stored in the SAS Life Science Analytics Framework. The query is built from an input data set. The extracted data is stored in a comma-separated values (.csv) file in your workspace or in the repository.

Input Data Set

The input data set must contain at least these columns. All other columns are ignored.

Table 4.1 *Input Data Set Columns*

Column Name	Description	Valid Values
recordType	The type of query element that the record represents.	SELECT The specified columnName is included in the output file. If no observations with recordType=SELECT are included in the data set, all allowable columns are included in the query. ORDER_ASCENDING The output file is sorted by the specified columnName in ascending order, and the level is indicated by the specified value. ORDER_DESCENDING The output file is sorted by the specified columnName in descending order, and the level is indicated by the specified value. CONSTRAINT The data to use to create a single comparison condition for the query. CONSTRAINT_RANGE The data to use to create a single comparison to determine whether a column value falls between the two specified values. This option is available only for query columns of type numeric and date. LOGICAL_OPERATOR The data to use to join the query.
columnClass	The class of the column to use with the query operation that is associated with recordType.	ColumnClass is not required when recordType is LOGICAL_OPERATOR and, if specified, is ignored. To get the columnClass data and the data types, call the macro %LSAF_GETQUERYCOLUMNS.

Column Name	Description	Valid Values
columnName	The name of the column to use with the query operation that is associated with recordType.	ColumnName is not required when recordType is LOGICAL_OPERATOR and, if specified, is ignored. To get the columnName data and the data types, call the macro %LSAF_GETQUERYCOLUMNS.
value	The value to use with the query operation that is associated with recordType.	The value varies, depending on recordType. ■ recordType is SELECT A value is not required and, if specified, is ignored. ■ recordType is ORDER_ASCENDING or ORDER_DESCENDING The sort level for the specified columnNames. The value is required, must be an integer, and must be unique among the ORDERx records in the data set. They do not need to be sequential or start at 1. The ORDERx records are sorted by value prior to processing. ■ recordType is CONSTRAINT A value is required and must match the expected type of the specified columnName (such as date or numeric). For character values, the wildcard character * is allowed when the comparator is LIKE or NOT_LIKE. ■ recordType is CONSTRAINT_RANGE Two comma-separated values of like types (such as date), where the first value precedes the second and the type is consistent with the type of the specified columnName. For example: 04Jan2018:20:48:41, 19Jan2019:40:00:00 ■ recordType is LOGICAL_OPERATOR How the constraint records are joined: AND or OR. The values are case-insensitive.
comparator	Indicates how the query should handle the constraint.	A value is required when recordType is CONSTRAINT. If specified for any other record type, it is ignored. EQUAL NOT_EQUAL LESS_THAN GREATER_THAN LESS_THAN_OR_EQUAL GREATER_THAN_OR_EQUAL LIKE NOT_LIKE

Column Name	Description	Valid Values
isCaseSensitive	Indicates whether a column value is treated as case-sensitive.	Case sensitivity is applicable when recordType is ORDER_ASCENDING, ORDER_DESCENDING, or CONSTRAINT and the associated column is of type STRING. Otherwise, the value, if specified, is ignored. If a value is not specified for applicable record/column types, the default value is used. 0 1 The default is 1.

Validating the Input Data Set

The input data set that contains the query is validated prior to running the query. The results of the validation are stored in an output data set.

If the input data set validation succeeds, the query is run and the resulting query is printed to the SAS log file.

If the input data set validation fails, the query is not run nor printed to the SAS log file. You can review the validation results in the data set that you specify with the LSAF_VALIDATEDDATASET parameter. The validationNote column lists the errors.

Running the Query

To run the query, you must have the proper privilege. The reference information for each macro lists the required privilege.

If the query results in more than one million records, the macro fails.

Global Macro Variables

Global macro variables that are specific to each query contain the full path of the query results that are specified in a comma-separated values file. The macro variable includes the file extension. If the query is not executed or if the macro processing results in a failure, the macro variable is blank.

Query Macros

General Syntax of the Query Macros

%LSAF_QUERY*query-name*(
LSAF_QUERYDATASET=*query-data-set-name*,
LSAF_VALIDATEDDATASET=*validated-data-set-name*,
LSAF_EXPORTLOCATION=REPOSITORY | WORKSPACE,
LSAF_EXPORTPATH=*export-path*
<, LSAF_OVERWRITE=0 | 1>
<, LSAF_ENABLEVERSIONING=0 | 1>
<, LSAF_VERSIONTYPE=MAJOR | MINOR | CUSTOM>
<, LSAF_CUSTOMVERSION=*custom-version*>
<, LSAF_COMMENT=*comment*>);

Required Parameters

LSAF_QUERYDATASET=*query-data-set-name*
The name of the input SAS data set that contains the query metadata, specified as LIBREF.DATASET. When querying against the columnName MODE with the columnClass of AuditEntry, the valid values are USER, SYSTEM, and ADMIN.

LSAF_VALIDATEDDATASET=*validated-data-set-name*
The name of the output SAS data set to contain the results of the validation of the records from the input data set lsaf_querydataset, specified as LIBREF.DATASET. The default depends on the query macro.

LSAF_EXPORTLOCATION=REPOSITORY | WORKSPACE
The case-insensitive output location for the exported CSV file.

LSAF_EXPORTPATH=*export-path*
The case-sensitive output path for the exported CSV file.

Optional Parameters

LSAF_OVERWRITE=0 | 1
Indicates whether an existing nonversioned file is overwritten. The default is 0.

LSAF_ENABLEVERSIONING=0 | 1
Indicates whether versioning is enabled for a file that is added to the repository. When LSAF_EXPORTLOCATION=WORKSPACE, this value is ignored. The default is 0.

LSAF_VERSIONTYPE=MAJOR | MINOR | CUSTOM
The version type to use to create a file in the repository. When LSAF_EXPORTLOCATION is WORKSPACE or LSAF_ENABLEVERSIONING is 0, this value is ignored. The default is MAJOR.

LSAF_CUSTOMVERSION=*custom-version*
The version number to use to create a customized versioned file in the repository when LSAF_VERSIONTYPE is CUSTOM. When LSAF_EXPORTLOCATION is WORKSPACE, LSAF_ENABLEVERSIONING is 0, or LSAF_VERSIONTYPE is not CUSTOM, this value is ignored.

LSAF_COMMENT=*comment*
The check-in comment to associate with the action of adding a file to the repository. When LSAF_EXPORTLOCATION=WORKSPACE, this value is ignored.

Query Audit Records

Overview

In the SAS Life Science Analytics Framework user interface, you can query the audit history using simple queries. For more complex queries, you can use SAS macros in a SAS session.

There are several important pieces of information that you generally need to create a query:

- The identifier of the object type to query.
- The user actions for the object type.
- The classes and names of the metadata columns.

Identify the Values to Use in the Query

1. Run %lsaf_getAllTypes.

   This macro creates a data set that contains all of the object type identifier values for which audit actions are recorded.

2. Open the data set that was created in the previous step, and record the **id** of the object type that you need for the query.

3. Run %lsaf_getAuditActions with the **id** from the previous step.

   This macro creates a data set that contains all of the user actions that are applicable to the object type identifier.

4 Open the data set that was created in the previous step, and record the **action** that you need for the query.

   You can specify multiple **action**s in a query.

5 Run %lsaf_getQueryColumns.

   This macro creates a data set of the columns that can be queried.

   To query the audit entry columns, specify AuditEntryQuery. To query the audit detail columns, specify AuditEntryDetailQuery.

6 Open the data set that was created in the previous step, and record the **columnName** that you need for the query.

   You can specify multiple **columnName**s in a query.

Create the Input Data Set for the Query

1 Run %lsaf_getQueryTemplateDataset.

   This macro creates a zero-observation data set that contains the variable metadata that is required to create the input data set for the query.

2 Create and run a DATA step that uses the values that you recorded in "Identify the Values to Use in the Query" on page 16 to create a data set that is the query.

See Also

- "Input Data Set" on page 12
- "Validating the Input Data Set" on page 14

Run the Query

Run %lsaf_queryAuditEntries.

This macro validates and runs the query to produce a CSV file that contains the results.

See Also

- "Running the Query" on page 14
- "Query Macros" on page 15

Example 1: Find All Records of Successful Log On or Log Off for a Single User

Identify the Values to Use in the Query

1 Get all of the object type values:

```
%lsaf_getAllTypes;
```

In the lsafgetalltypes data set, the relevant value is **sas:user**.

2 Get all of the actions for sas:user:

```
%lsaf_getAuditActions(lsaf_typeID=%str(sas:user),
sas_dsName=user_actions);
```

In the user_actions data set, the relevant values are **logonSuccessful** and **logoffSuccessful**.

3 Get a list of the columns that can be queried:

```
%lsaf_getQueryColumns(lsaf_queryType=AuditEntryQuery);
```

In the lsafgetquerycolumns data set, the relevant values are **userId** and **action**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data logonLogoff;
  if  0 then set lsafGetQueryTemplateDataset;

  * NOTE: Below, replace user_identifier with a valid user identifier;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; value="user_identifier";
output;

  * Logical operator;
  recordType="LOGICAL_OPERATOR"; columnClass=""; isCaseSensitive=0;
  columnName=""; comparator=""; value="AND"; output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="logonSuccessful";
output;
```

```
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="logoffSuccessful";
output;
run;
```

Note: Replace *user_identifier* with a valid user identifier.

Run the Query

1 Run the query:

```
%lsaf_queryAuditEntries(lsaf_queryDataSet=logonLogoff,
lsaf_validatedDataSet=validate_logonLogoff,
lsaf_exportLocation=workspace,
lsaf_exportPath=%str(/Users/user_identifier/logonLogoff.csv),
lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open logonLogoff.csv and review the results.

Example 2: Find All Records of the Contexts That Were Created or Permanently Deleted by Several Users

Identify the Values to Use in the Query

1 Get all of the object type values:

```
%lsaf_getAllTypes;
```

In the lsafgetalltypes data set, the relevant value is **sas:context**.

2 Get all of the actions for sas:context:

```
%lsaf_getAuditActions(lsaf_typeID=%str(sas:context),
sas_dsName=file_actions);
```

In the file_actions data set, the relevant values are **created** and **permanentlyDeleted**.

3 Get a list of the columns that can be queried:

```
%lsaf_getQueryColumns(lsaf_queryType=AuditEntryQuery);
```

In the lsafgetquerycolumns data set, the relevant values are **userId**, **action**, and **sourceTypeId**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data contexts;
  if  0 then set lsafGetQueryTemplateDataset;

  * Contexts created or permanently deleted by 3 users;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="sourceTypeId"; comparator="EQUAL"; value="sas:context";
output;

  * Logical Operator;
  recordType="LOGICAL_OPERATOR"; columnClass=""; isCaseSensitive=0;
  columnName=""; comparator=""; value="AND"; output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="created"; output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="permanentlyDeleted";
output;

  * NOTE: Below, replace the user_identifiers with valid user
identifiers.
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; Value="user_identifier1";
output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; value="user_identifier2";
output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; value="user_identifier3";
output;
run;
```

Note: Replace the *user_identifier*s with valid user identifiers.

Run the Query

1 Run the query:

```
%lsaf_queryAuditEntries(lsaf_queryDataSet=contexts,
lsaf_validatedDataSet=validate_contexts,
lsaf_exportLocation=workspace,
lsaf_exportPath=%str(/Users/user_identifier/contexts.csv),
lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open contexts.csv and review the results.

Example 3: Find All of the Files That Were Checked Out by a User from a Folder and Its Subfolders, within a Time Frame

Identify the Values to Use in the Query

1 Get all of the object type values:

```
%lsaf_getAllTypes;
```

In the lsafgetalltypes data set, the relevant value is **sas:file**.

2 Get all of the actions for sas:file:

```
%lsaf_getAuditActions(lsaf_typeID=%str(sas:file),
sas_dsName=file_actions);
```

In the file_actions data set, the relevant value is **checkedOut**.

3 Get a list of the columns that can be queried:

```
%lsaf_getQueryColumns(lsaf_queryType=AuditEntryQuery);
```

In the lsafgetquerycolumns data set, the relevant values are **userId**, **action**, and **sourceTypeId**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data checkedoutFiles;
  if  0 then set lsafGetQueryTemplateDataset;

  * NOTE: Below, replace location with a ;
  * valid repository location. The wildcard (*) selects all
subfolders.;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="sourceLocation"; comparator="LIKE"; value="location*";
output;

  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="checkedOut"; output;


  * NOTE: Below, replace user_identifier with a ;
  * valid user identifier.;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; value="user_identifier";
output;

  RecordType="LOGICAL_OPERATOR"; ColumnClass=""; isCaseSensitive=0;
  ColumnName=""; Comparator=""; Value="AND"; output;

  * Constraints - DATE_RANGE;
  recordType="CONSTRAINT_RANGE"; columnClass="AuditEntry";
isCaseSensitive=0;
  columnName="timestamp"; comparator="";
  value="01MAR2020:00:00:00,31MAR2020:21:55:59"; output;
run;
```

Note: Replace *location* with a valid repository location. Replace *user_identifier* with a valid user identifier.

Run the Query

1 Run the query:

```
%lsaf_queryAuditEntries(lsaf_queryDataSet=checkedoutFiles,
lsaf_validatedDataSet=validate_checkedoutFiles,
lsaf_exportLocation=workspace,
lsaf_exportPath=%str(/Users/user_identifier/checkedoutFiles.csv),
lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open checkedoutFiles.csv and review the results.

Example 4: Find Details about the User Roles That Were Created by a User

Identify the Values to Use in the Query

Note: %lsaf_getAllTypes is not needed in this example.

1 Get all of the actions:

```
%lsaf_getAuditActions(sas_dsName=file_actions);
```

In the file_actions data set, the relevant value is **roleAdded**.

Note: **roleAdded** is available for custom contexts and several standard SAS Life Science Analytics Framework objects. For this reason, lsaf_typeID was not specified in the call to %lsaf_getAuditActions.

2 Get a list of the columns that can be queried:

```
%lsaf_getQueryColumns(lsaf_queryType=AuditEntryQuery);
```

In the lsafgetquerycolumns data set, the relevant values are **action** and **userId**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data userRoles;
  if  0 then set lsafGetQueryTemplateDataset;

  * NOTE: Below, replace user_identifier with a valid user identifier.;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="userId"; comparator="EQUAL"; value="user_identifier";
output;

  recordType="LOGICAL_OPERATOR"; columnClass=""; isCaseSensitive=0;
  columnName=""; comparator=""; value="AND"; output;

  * Test to see roles have been added by user_identifier;
  recordType="CONSTRAINT"; columnClass="AuditEntry"; isCaseSensitive=0;
  columnName="action"; comparator="EQUAL"; value="roleAdded";
output;
```

```
run;
```

Note: Replace *user_identifier* with a valid user identifier.

Run the Query

Note: Unlike the other examples, this one calls %LSAF_QUERYAUDITDETAILS. This macro provides the details about attribute changes.

1 Run the query:

```
%lsaf_queryAuditDetails(lsaf_queryDataSet=userRoles,
lsaf_validatedDataSet=validate_userRoles,
lsaf_exportLocation=workspace,
lsaf_exportPath=%str(/Users/user_identifier/userRoles.csv),
lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open userRoles.csv and review the results.

Query the Recycle Bin

Overview

In the SAS Life Science Analytics Framework user interface, you can query the recycle bin using simple queries. For more complex queries, you can use SAS macros in a SAS session.

Identify the Values to Use in the Query

1 Run %lsaf_getquerycolumns.

This macro creates a data set that contains all of the object properties that can be queried. The parameter lsaf_querytype specifies the type of recycle bin object to query.

2 Open the data set that was created in the previous step, and record the **columnName** that you need for the query.

You can specify multiple **columnName**s in a query.

Create the Input Data Set for the Query

1. Run %lsaf_getquerytemplatedataset.

   This macro creates a zero-observation data set that contains the variable metadata that is required to create the input data set for the query.

2. Create and run a DATA step that uses the values that you recorded in “Identify the Values to Use in the Query” on page 24 to create a data set that is the query.

See Also

- “Input Data Set” on page 12
- “Validating the Input Data Set” on page 14

Run the Query

Run %lsaf_queryrecyclebincontainer, %lsaf_queryrecyclebinfile, %lsaf_queryrecyclebinfileversion, %lsaf_queryrecyclebinfileversion, or %lsaf_queryrecyclebinitem.

Each macro queries a specific type of recycle bin item. The macros validate and run the query to produce a CSV file that contains the results.

See Also

- “Running the Query” on page 14
- “Query Macros” on page 15

Example 1: Find All Items That Were Deleted after a Specific Date and Time within a Context

Identify the Values to Use in the Query

Get a list of all of the object properties that can be queried:

```
%lsaf_getQueryColumns(lsaf_querytype=%str(rbitem));
```

In the lsafgetquerycolumns data set, the relevant values of **columnName** are **path** and **deletedDate**.

Create the Input Data Set for the Query

1. Create a zero-observation data set with the variable metadata needed to create the input data set:

   ```
   %lsaf_getQueryTemplateDataset;
   ```

2. Create the input data set for the query:

```
data deleted_after_datetime;
  if 0 then set lsafGetQueryTemplateDataset;

  * Records deleted by any user under the /SAS context after;
  * Oct. 1, 2020;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "path"; Comparator="LIKE";
  Value="/SAS*";
  output;

  * Logical Operator;
  RecordType = "LOGICAL_OPERATOR"; ColumnClass = "";
  isCaseSensitive=0; ColumnName = ""; Comparator=""; Value="AND";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "deletedDate";
  Comparator="GREATER_THAN_OR_EQUAL"; Value="01Oct2020:00:00:00";
  output;
run;
```

Run the Query

1 Run the query:

```
%lsaf_queryrecyclebinitem(
lsaf_querydataset=deleted_after_datetime,
lsaf_validateddataset=validated_deleted_after_datetime,
lsaf_exportlocation=workspace,
lsaf_exportpath=%str(/Users/user_identifier/
deleted_after_datetime.csv),
   lsaf_overwrite=1
);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open deleted_after_datetime.csv and review the results.

Example 2: Find All Unversioned SAS Data Sets That Are Greater Than 100,000KB within a Context

Identify the Values to Use in the Query

Get a list of all of the object properties that can be queried:

```
%lsaf_getQueryColumns(lsaf_querytype=%str(rbitem));
```

In the lsafgetquerycolumns data set, the relevant values of **columnName** are **path**, **typeID**, **size**, and **version**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data unversioned_ds_gt100k;
   if 0 then set lsafGetQueryTemplateDataset;

  * Unversioned SAS data sets with size > 100000 deleted by any user;
  under the /SAS context;
```

```
  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "path"; Comparator="LIKE";
  Value="/SAS*";
  output;

  * Logical Operator;
  RecordType = "LOGICAL_OPERATOR"; ColumnClass = ""; isCaseSensitive=0;
  ColumnName = ""; Comparator=""; Value="AND";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "typeId"; Comparator="EQUAL";
  Value="sas:sasdataset";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinFile";
  isCaseSensitive=0; ColumnName = "size"; Comparator="GREATER_THAN";
  Value="100000";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinFile";
  isCaseSensitive=0; ColumnName = "version"; Comparator="EQUAL";
  Value="-";
  output;
run;
```

Run the Query

1 Run the query:

```
%lsaf_queryrecyclebinfile(
lsaf_querydataset=unversioned_ds_gt100k,
lsaf_validateddataset=val_unversioned_ds_gt100k,
lsaf_exportlocation=workspace,
lsaf_exportpath=%str(/Users/user_identifier/unversioned_ds_gt100k.csv),
   lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open unversioned_ds_gt100k.csv and review the results.

Example 3: Find All Files That Have More Than Two Versions within a Context

Identify the Values to Use in the Query

Get a list of all object properties that can be queried:

```
%lsaf_getquerycolumns(lsaf_querytype=rbfileversion);
```

In the lsafgetquerycolumns data set, the relevant values of **columnName** are **path** and **totalVersions**.

Create the Input Data Set for the Query

1. Create a zero-observation data set with the variable metadata needed to create the input data set:

   ```
   %lsaf_getQueryTemplateDataset;
   ```

2. Create the input data set for the query:

   ```
   data rbfiles_gt2vers;
     if 0 then set lsafGetQueryTemplateDataset;

     * Versioned records with total versions > 2 deleted by any user;
     * under the /SAS context;

     RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
     isCaseSensitive=0; ColumnName = "path"; Comparator="LIKE";
     Value="/SAS*";
     output;

     * Logical Operator;
     RecordType = "LOGICAL_OPERATOR"; ColumnClass = ""; isCaseSensitive=0;
     ColumnName = ""; Comparator=""; Value="AND";
     output;

     RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinFileVersion";
     isCaseSensitive=0; ColumnName = "totalVersions";
     Comparator="GREATER_THAN"; Value="2";
     output;
   run;
   ```

Run the Query

1 Run the query:

```
%lsaf_queryrecyclebinfileversion(
lsaf_querydataset=rbfiles_gt2vers,
lsaf_validateddataset=validated_rbfiles_gt2vers,
lsaf_exportlocation=workspace,
lsaf_exportpath=%str(/Users/user_identifier/rbfiles_gt2vers.csv),
lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open rbfiles_gt2vers.csv and review the results.

Note: %lsaf_queryrecyclebinfile provides information about both unversioned and versioned files in the recycle bin that meet the criteria for a query. If a file is versioned, only the information from the most recent file version is displayed. %lsaf_queryrecyclebinfileversion provides information for all versions of a file in the recycle bin that meet the criteria for a query.

Example 4: Find All Folders That Are Greater Than 200,000KB within a Context

Identify the Values to Use in the Query

Get a list of all object properties that can be queried:

```
%lsaf_getquerycolumns(lsaf_querytype=rbcontainer);
```

In the lsafgetquerycolumns data set, the relevant values of **columnName** are **path**, **typeID**, **size**.

Create the Input Data Set for the Query

1 Create a zero-observation data set with the variable metadata needed to create the input data set:

```
%lsaf_getQueryTemplateDataset;
```

2 Create the input data set for the query:

```
data folders_gt200k;
```

```
  if 0 then set lsafGetQueryTemplateDataset;

  * Folders deleted by any user with size > 200000 under the ;
  * /SAS context;
  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "path"; Comparator="LIKE";
  Value="/SAS*";
  output;

  * Logical Operator;
  RecordType = "LOGICAL_OPERATOR"; ColumnClass = ""; isCaseSensitive=0;
  ColumnName = ""; Comparator=""; Value="AND";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinItem";
  isCaseSensitive=0; ColumnName = "typeId"; Comparator="EQUAL";
  Value="sas:folder";
  output;

  RecordType = "CONSTRAINT"; ColumnClass = "RecycleBinContainer";
  isCaseSensitive=0; ColumnName = "size"; Comparator="GREATER_THAN";
  Value="200000";
  output;
run;
```

Run the Query

1 Run the query:

```
%lsaf_queryrecyclebincontainer(
lsaf_querydataset=folders_gt200k,
lsaf_validateddataset=validated_folders_gt200k,
lsaf_exportlocation=workspace,
lsaf_exportpath=%str(/Users/user_identifier/folders_gt200k.csv),
   lsaf_overwrite=1);
```

Note: Replace *user_identifier* with your user identifier.

2 In your workspace, open folders_gt200k.csv and review the results.