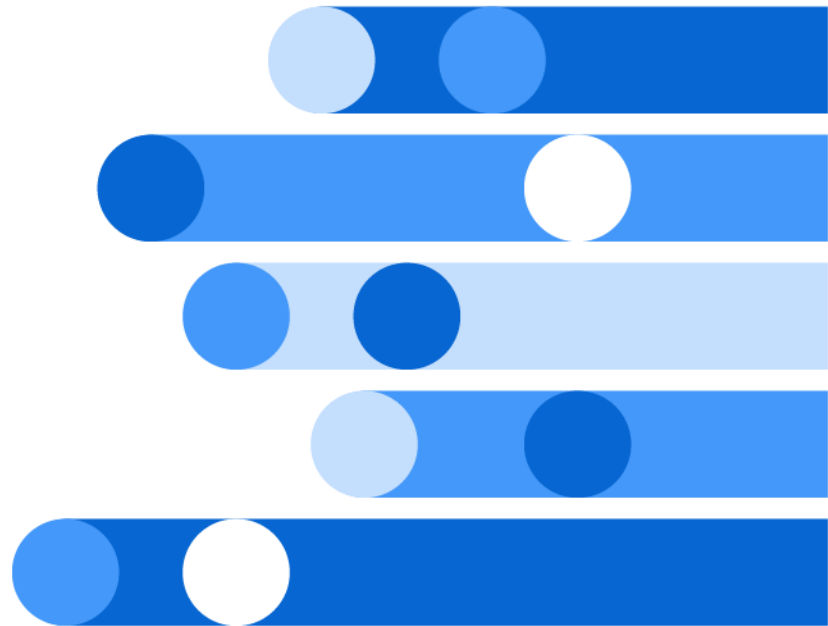




SAS[®] Job Execution Web Application: User's Guide

2020.1 - 2026.07*



* 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.



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® Job Execution Web Application: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® Job Execution Web Application: User's Guide

Copyright © 2020, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

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

July 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_002-P1:jobexecug

Contents

Chapter 1 / Using the Web Application	1
Overview	1
Working with the SAS Job Execution Web Application	4
Executing Jobs	14
Modifying Your Settings	28
Chapter 2 / Administration	31
Administrative Tasks	31
Security for SAS Viya Platform Jobs	38
Testing the Installation	44
Chapter 3 / Developing Jobs	45
Development Concepts	46
Working with Dynamic Data in Prompts and HTML Forms	74
Creating HTML Forms with Dynamic Data	75
Advanced Programming	85
Samples	95
Reference	177
Chapter 4 / Using Job Chaining Macros	197
Overview of %JOBEXEC_* Autocall Macros	197
Job Definitions for %JOBEXEC_* Autocall Macro Examples	198
Dictionary	200

Using the Web Application

Overview	1
Working with the SAS Job Execution Web Application	4
Accessing the Application	4
Accessing Content	5
Managing Jobs	9
Scheduling a Job	10
Installing the Samples	12
Accessing Application Pages through a URL	12
Executing Jobs	14
Executing a Job Using the SAS Job Execution Web Application	14
Specifying a Job to Run	14
Executing a Job Using Direct URL Access	15
Using Job Input Parameters	15
Prompting for Input Parameters with Job Forms	16
Specifying Output Files	18
Using Output Job Parameters	19
Modifying the Job Execution	19
Using the Command-Line Interface (CLI)	26
Executing a Job as a Compute Task	27
Modifying Your Settings	28
General	28
Region and Language	29
Accessibility	30

Overview

A SAS Viya *platform job* consists of a program and its definition, which includes information such as the job name, author, and creation date and time. You can use jobs for web reporting, performing analytics, building web applications, and delivering content to clients. One such client is the SAS Job Execution Web Application. This client works with SAS compute server jobs, which are jobs that contain SAS code and execute on the compute server. These jobs can access any

SAS data source or external file and create new tables, files, or other data targets that are supported by SAS.

By default, jobs that are submitted through the SAS Job Execution Web Application begin with the %JESBEGIN macro and end with the %JESEND macro. These macros produce HTML output by default.

An input form can be created using HTML or the SAS prompting interface to provide a user interface to the job. The job definitions and user interface definitions are assigned unique identifiers, stored in the SAS Infrastructure Data Server, and executed in real time by client applications.

The SAS Job Execution Web Application is a web-based client used to create, manage, and execute jobs. This application, written in Go, provides access to data in combination with a powerful array of analysis and presentation procedures running on a server. No SAS software is required on the client machine.

To access and analyze data, a web user typically completes an input form displayed by the SAS Job Execution Web Application. When the user selects the option to submit the information, data specified in the form is passed to a waiting SAS session as global macro variables. The SAS program runs and the results are returned to the web browser.

You do not need Java or script programming experience to use the SAS Job Execution Web Application. You can create the web user interface and retrieve SAS data for display on the web using only HTML or the SAS prompting interface and SAS code.

Note: If you are interesting in creating jobs from SAS Stored Processes by promoting them to the SAS Viya platform, see [SAS Viya Platform: Content Migration from SAS Viya 4](#) for more information.

Use the SAS Job Execution Web Application if you

- want to analyze and display information dynamically on the web and let your web users immediately retrieve the information that they need.
- have SAS programming experience but little or no web programming experience. You can create the web user interface and retrieve the SAS data for display on the web.
- want to create applications that provide web output without investing a lot of programming time.
- want to create applications that run on a variety of web browsers.

The SAS Job Execution Web Application has several types of users, as follows:

- End users enter information in an input form, select a link, or view an inline image that is displayed in a web browser.
- Web page authors create the input forms or pages that collect and submit input to the SAS job. These individuals could be SAS application developers.
- SAS job program component developers create the SAS programs that receive information entered in the input form.

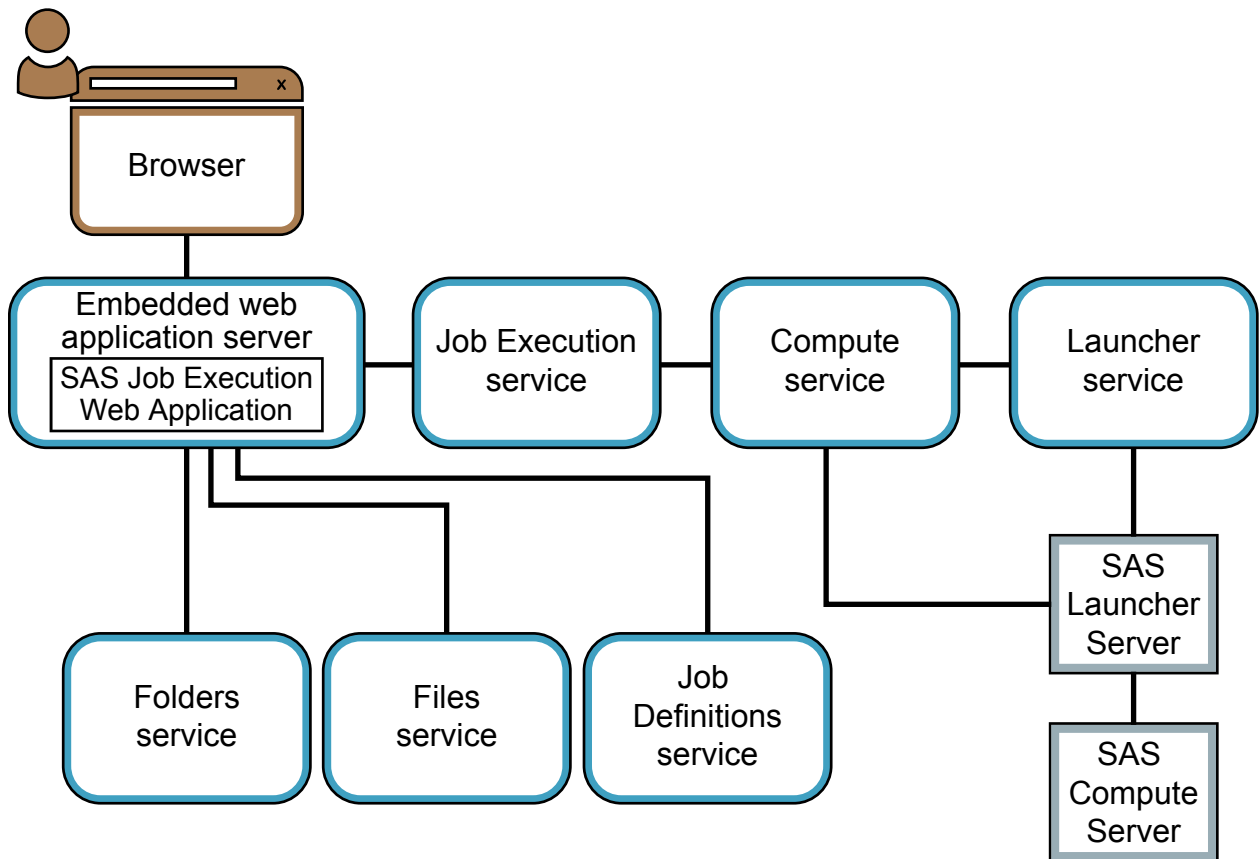
The SAS Job Execution Web Application executes a job when it receives a request with a parameter indicating the location of the job. The request is initiated by accessing a URL with the following general format:

```
http://host:port/SASJobExecution/?_program=/SomeFolder/Hello World
```

In this example, the Hello World job stored in the `/SomeFolder` location is executed.

Various SAS Viya platform services are used to retrieve the job, submit it for processing, and display the job output. The following figure shows the components that are used to run a job:

Figure 1.1 Components of Running a Job



The execution flow consists of the following steps:

- 1 A request is submitted to the SAS Job Execution Web Application, usually initiated by a web browser client.
- 2 If the job is referenced using a folder path, then the job location is obtained from the Folders service. This step is skipped if the job location was specified in the `_JOB` parameter.
- 3 The job definition is retrieved from the Job Definitions service by using the job location obtained in step 2.
- 4 Input parameters, execution parameters, and the job definition are submitted to the Job Execution service.

- 5 The request is passed to the Compute service, which uses the Launcher service to start a compute server.
- 6 The SAS program runs in the compute server and creates output in the SAS Infrastructure Data Server using the Files service. The files are associated with the job execution object to facilitate retrieval.

Note: The compute server can access the SAS Cloud Analytic Services (CAS) server through the CAS procedure or through the CAS LIBNAME engine. For more information about the CAS procedure, see [“Overview: CAS Procedure” in SAS Cloud Analytic Services: CASL Reference](#). For more information about the CAS LIBNAME engine, see [“CAS LIBNAME Engine Overview” in SAS Cloud Analytic Services: User’s Guide](#).

- 7 The Job Execution service monitors the job execution and alerts the SAS Job Execution Web Application when the job has completed.
- 8 The SAS Job Execution Web Application instructs the web browser to use the Files service to retrieve the desired output files associated with the job execution object. If the `_SAVEFOLDER` option is used, then the output is saved to the specified folder.
- 9 The job execution object has an expiration time, which is set when the job has finished executing. When it expires, the job execution object and all files associated with it are deleted.

Working with the SAS Job Execution Web Application

Accessing the Application

Access the SAS Job Execution Web Application using this URL:

`http://host:port/SASJobExecution`

If you do not specify a value for *port*, the default value of 80 is used.

Accessing Content

Overview

The SAS Content page enables you to locate, manage, and execute job-related files. If the output of a job is saved to SAS content folders, it can also be displayed from this page. The SAS Content page appears by default when you enter the SAS Job Execution Web Application, but if you are on another page in the web application, you can click  in the left navigation menu to access it.

The left pane of the SAS Content page displays folders and their contents, and the right pane is used as a presentation and working area for the various job functions. Separate tabs are created in the right pane when you perform different functions (editing, viewing, displaying forms, or executing jobs). Tabs are reused based on function, member name, and type (source code, HTML form, or prompts). A new tab is created every time you edit any folder member, regardless of whether you already have an editing tab open.

The full version of the SAS Content page is available to users who have authorization to develop jobs. A more limited SAS Content page is available to users who are allowed only to execute existing jobs and view output. The display and capabilities are automatically adjusted according to the authorization of each user. For example, HTML input forms are displayed only for developers. By default, all users have authorization to develop jobs. See [“Changing Access to Application Functions” on page 38](#) for more information about how to assign users access to developer or user views.

The following actions are common to both views when you select a job:

- Click  to submit a job for execution.
- Click  to retrieve the output from the most recent execution of the job, if it is still available.
- Click  to open the following window with execution options:

You can add parameters using the format *name1=value1&name2=value2*.

The **Show log** option adds the `_debug=log` parameter setting to the request. You can use this option to open a new tab that contains the log when the job has completed (if **New window** is not selected). The **Show log** option does not work if `_debug=trace` is active.

The **New window** option displays the job output (and input form, if one exists) in a new browser tab or window, depending on how your browser is configured. Otherwise, the job output is displayed in a new tab within the SAS Job Execution Web Application. These options remain in effect for job submissions during the current browser session.

- Click  to access the [Scheduling](#) page, where you can display jobs that have been prepared for scheduling or are already scheduled.

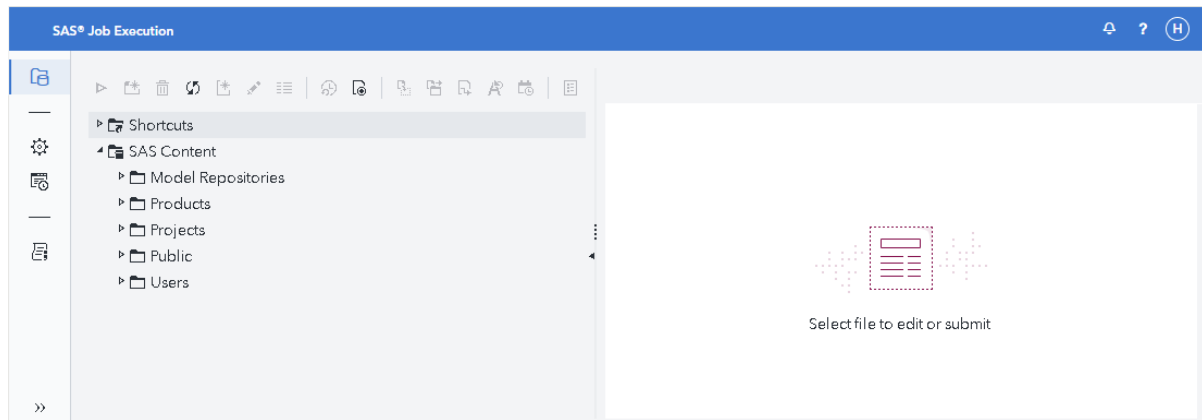
The `_FOLDER` URL parameter can be used to alter the folder list for the Content page. Specifying `_folder=/SomeFolder` opens the display at the indicated folder path. Here is an example:

```
http://host:port/SASJobExecution/?_folder=/SomeFolder
```

See [“Accessing Application Pages through a URL”](#) on page 12 for more information about accessing pages through a URL.

Content for Developers

Users who have authorization to develop jobs can access the full version of the SAS Content page using the `/developer` path.



Developers can add, delete, and rename folders and jobs. You cannot restore jobs that have been deleted. Both users and developers can also modify job execution options, submit jobs for execution, schedule jobs, and retrieve output from recent job executions. The following actions are available only for developers when an object in a folder is selected:

Note: Depending on the width of the left pane, you might need to click  to access some of these actions.

- Click  to add a new object to a folder. The object can be a job definition, HTML input form, XML prompts, or a text file. If you are adding a job definition, you can select the type of server and the compute context for the job.

Note: The compute context contains information such as the user identity and any SAS options or autoexec file parameters to be used when starting the server. For example, it is important to select the correct compute context so that your program is able to access the correct data.

The default compute context is determined by the contextname configuration parameter. When you select a compute context, the `_contextName` parameter is added to the job definition. Any change that you make to either the **Compute context** selection or the value for the `_contextName` job definition parameter will be reflected in both places.

If you are creating a text file, you can select from various content types, as shown:

New file [X]

[Reset](#)

Name: *

Description:

File type:

Content:

- CSS
- CSV
- HTML
- Javascript
- Plain text
- SAS program

For more information about including a file, see [“Including a File in an HTML Form” on page 94](#).

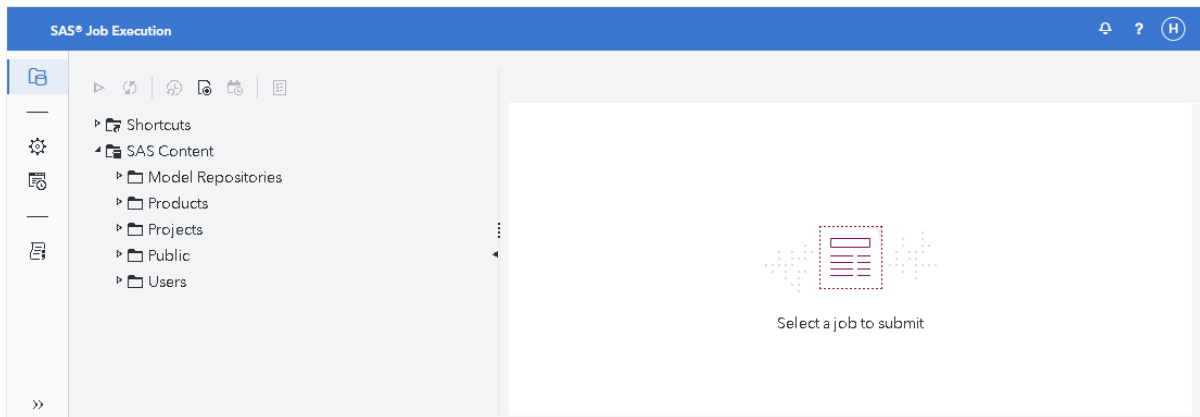
- Click to add or edit the contents of an object, such as the source code, HTML input form, or prompts for a job.
- Click to view an object, such as the source code, HTML input form, or prompts for a job.
- Click to display the contents of an object, such as an input form or prompts for a job.
- Click to make a copy of the object. In the Save As window, specify the folder location and file name for the new object.
- Click to move the object to another folder. In the Choose a Location window, specify the folder location for the object.
- Click to add a copy of the object to your **My Favorites** folder. You can modify or delete this copy without affecting the original object.
- Click to rename the object.
- Click to access properties and parameters for the object. Developers can edit or add parameters to jobs. See [“Passing User Input to a Job Using a Job](#)

[Definition Parameter” on page 50](#) for more information about setting parameters.

See the [Creating a Simple Job that Uses DATA Step Code and Passing User Input to Your Job Using an HTML Input Form](#) sections in [“Development Concepts” on page 46](#) for more information about creating and editing files.

Content for Users

Users who are allowed only to execute existing jobs and view output can access a more limited SAS Content page using the `/user` path.



Users can select an object and click  to view the properties and parameters for the object. Both users and developers can also modify job execution options, submit jobs for execution, schedule jobs, and retrieve output from recent job executions.

Managing Jobs

Click  in the left navigation menu to access the Jobs page, where you can display and maintain jobs that have run previously and have not yet expired. Each row of the table contains a previously submitted job, with information about the location of the job, its status, run time, who submitted the job and when, and when the job expires.

SAS® Job Execution						
Jobs						
Program	Status	Run Time	Submitted	Submitted By	Expires	
Display Macro Variables	✓	00:00:37.7	Mar 4, 2021 04:51 PM	omitest	Mar 4, 2021 05:22 PM	
Hello World	✓	00:00:29.8	Mar 4, 2021 04:51 PM	omitest	Mar 4, 2021 05:22 PM	
Simple ODS HTML	✓	00:00:26.1	Mar 4, 2021 04:51 PM	omitest	Mar 4, 2021 05:22 PM	

The following actions are available when a job is selected:

- Click  to specify the date and time at which to delete a job.
- Click  to display the job output in a new browser tab or window.
- Click  to display the SAS log for the job execution.
- Click  to save all output from the job. In the Save As window, specify the folder location and file name for the output. If multiple output files were created, then you are prompted for a location and file name for each file. Any existing files with the same name are overwritten.
- Click  to open the Schedule Job window on the [Scheduling](#) page. You can rename the scheduled job, edit the description, and add a new trigger to schedule how often and when to run the job again. If you are an administrator, you can select which user profile is used to run the scheduled job. See [“Scheduling a Job” on page 23](#) for more information about using the `_ACTION` parameter to schedule a job.
- Click  to cancel a job that is currently running.

You can specify the `_PROGRAM` input parameter to display results for a single job. For example, the following URL displays results only for the Hello World job:

```
http://host:port/SASJobExecution/jobs?_program=/SomeFolder/Hello World
```

The value of the `_PROGRAM` parameter is case-sensitive and must exactly match the location and name of the job.

Scheduling a Job

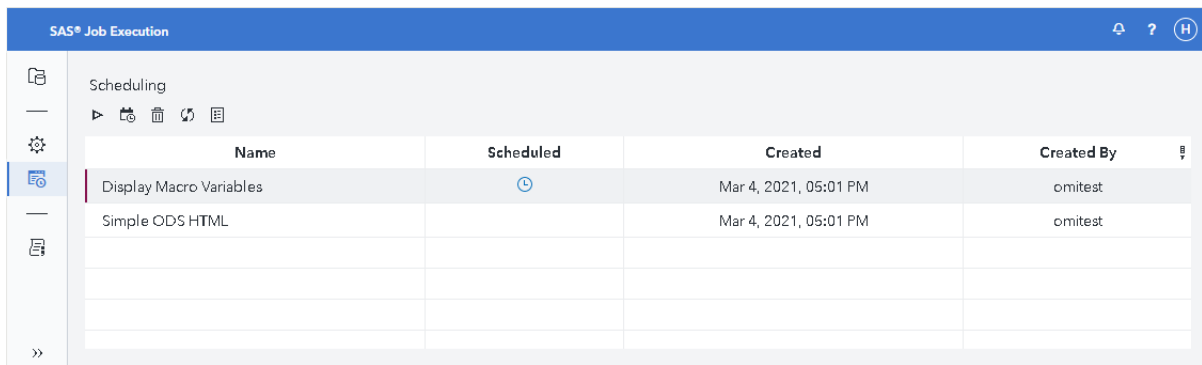
You can schedule a job before or after it has been run. Scheduling a job after execution enables you to set additional parameters such as those within job forms or prompts. To schedule a job after it has been run, select the job on the Jobs page. If you schedule a job before running it, it is limited to using only job definition parameters. To schedule a job before it has been run, select the job on the SAS Content page. For either scenario, click  to open the Schedule Job window on the Scheduling page.

Note: You can also use `_action=schedule` parameter when submitting a job if you want to prepare the job for scheduling. After the job has run, it is available for scheduling on the Scheduling page.

The source code and parameters for the scheduled job are a copy of the job definition. Any changes to the original job do not affect the scheduled job.

All scheduled jobs are submitted directly to the Job Execution service rather than to the web application.

You can click  in the left navigation menu to access the Scheduling page, where you can display jobs that have been prepared for scheduling or are already scheduled. Each row of the table contains a job, whether it has been scheduled to run, and who prepared the job for scheduling and when.



Name	Scheduled	Created	Created By
Display Macro Variables		Mar 4, 2021, 05:01 PM	omitest
Simple ODS HTML		Mar 4, 2021, 05:01 PM	omitest

The following actions are available when an object in a folder is selected:

- Click  to open the Schedule Job window. You can rename the scheduled job and edit the description, which is useful if you want to schedule different variations of the job. An administrator can select which user profile is used to run the scheduled job — the current user's profile is used by default. You can add a new trigger to schedule how often and when to run the job. See [“Schedule Jobs” in SAS Environment Manager: User's Guide](#) for more information about scheduling jobs and creating a time trigger.

Note: If you are an administrator and you want to change the value for **Run as** after the job has been scheduled, you must delete the scheduled job and schedule it again.

- Click  to submit a job for execution. Click  to access the Jobs page, where you can view the job output.
- Click  to view the properties, arguments (which include parameters), and scheduling information for a job. You can edit the job's arguments or source code, which affects only this instance of the scheduled job, not the original job definition.

Note: Arguments that are used by the Job Execution service are case-sensitive in this window. However, when using these arguments as job definition parameters, they are case-insensitive.

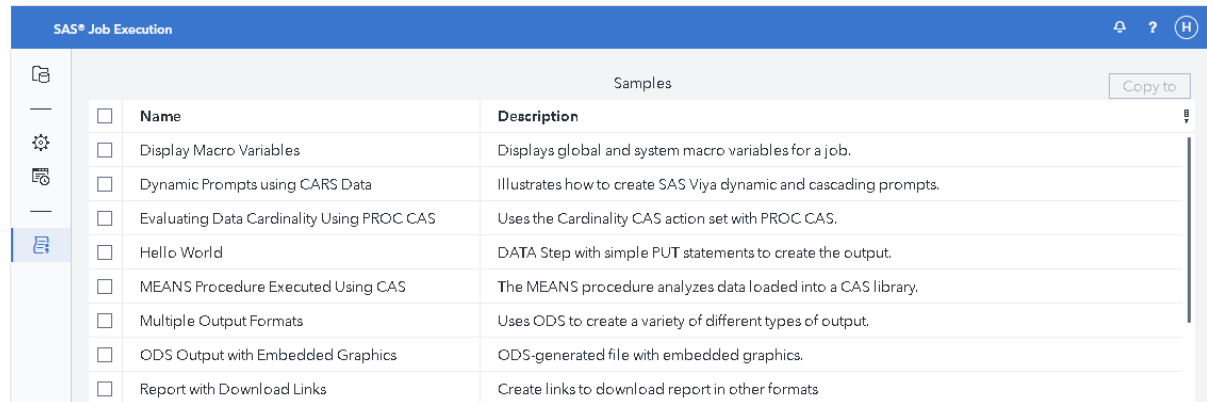
Some arguments (such as `_EXPIRATION`, `_DEBUG`, `_ACTION`, and `_TIMEOUT`) must be set in the original job definition rather than in the instance of the scheduled job.

Source code could also be included from a Files service file. See [“Assigning a Fileref to Reference Source Code Stored in the SAS Infrastructure Data Server” on page 64](#) for more information.

Installing the Samples

A set of sample jobs is available for installation. These samples illustrate various SAS program coding techniques and how jobs are executed with the SAS Job Execution Web Application.

Click  in the left navigation menu to access the Samples administration page, where you can view a table of available samples. This is a list of items available for installation, not those that have already been installed.



☐	Name	Description
☐	Display Macro Variables	Displays global and system macro variables for a job.
☐	Dynamic Prompts using CARS Data	Illustrates how to create SAS Viya dynamic and cascading prompts.
☐	Evaluating Data Cardinality Using PROC CAS	Uses the Cardinality CAS action set with PROC CAS.
☐	Hello World	DATA Step with simple PUT statements to create the output.
☐	MEANS Procedure Executed Using CAS	The MEANS procedure analyzes data loaded into a CAS library.
☐	Multiple Output Formats	Uses ODS to create a variety of different types of output.
☐	ODS Output with Embedded Graphics	ODS-generated file with embedded graphics.
☐	Report with Download Links	Create links to download report in other formats

To install samples and use them, copy the sample job definitions supplied by SAS to a folder. Select the desired items and then click **Copy To**. In the folder selector window, select a destination folder and then click **OK** to copy the items. If an item already exists in that location, an option to skip, replace the item, or quit copying is presented.

Copying sample job definitions to a folder is typically a one-time operation performed as a post-installation step.

Accessing Application Pages through a URL

The page that appears when you access the base SAS Job Execution Web Application URL depends on your authorization level. The default page lists job files and provides access to editing and administration tools. You can use SAS Environment Manager to create authorization rules that determine which users

have access to different pages. See “[Changing Access to Application Functions](#)” on [page 38](#) for more information.

Several pages are available with the SAS Job Execution Web Application. The different pages are displayed by adding paths to the base SAS Job Execution Web Application URL:

Table 1.1 Paths to SAS Job Execution Web Application Pages

Path	Page Description
<code>/apiMeta</code>	Returns the application version and build information.
<code>/csrf</code>	Returns the current CSRF token for the SAS Job Execution Web Application.
<code>/developer</code>	Displays all files and administration tools. Depending on a user's authorization, this might be the default Content page, which is used to create and maintain job-related files.
<code>/env</code>	Displays system environment information such as properties and parameters (restricted to SAS Administrators).
<code>/jobs</code>	Enables you to administer previous job runs.
<code>/logout</code>	Logs out of the application.
<code>/samples</code>	Enables you to administer sample jobs.
<code>/scheduling</code>	Enables you to schedule jobs.
<code>/search</code>	Returns a table of all available job definitions, including their folder locations and descriptions.
<code>/theme</code>	Returns the CSS for the current theme.
<code>/user</code>	Displays and executes job files. Depending on a user's authorization, this might be the default Content page.

For example, use the following URL to display and execute job files:

`http://host:port/SASJobExecution/user`

You can use the `/search` path to access a table of all available job definitions, including their folder locations and descriptions. You can use the following additional modifiers to limit the search results:

`_MATCH=string`

returns job definitions that contain the specified case-insensitive string.

`_FIELD=field-name`

specifies which field to search. Valid values are **name** and **description**. By default, the name field is searched.

`_ACTION=JSON`

returns the URLs for job definitions that are matched in the search results.

For example, use the following URL to search the description field for the word binary:

`http://host:port/SASJobExecution/search/?_match=binary&_field=description`

Executing Jobs

Executing a Job Using the SAS Job Execution Web Application

The Content page of the SAS Job Execution Web Application provides a basic user interface to create, list, and execute jobs.

Note: Users who do not have access to developer functionality see fewer icons and files. Developers see the job forms as well as the job definitions. Other users see only the job definitions.

In the left pane, navigate to the job that you want to execute, select it, and then click ► to execute the job.

The job output is displayed in the right pane.

Specifying a Job to Run

Jobs are typically referenced by a multi-level path. Jobs can be created in any folder, and they are referenced using the `_PROGRAM` parameter:

`_program=/SomeFolder/jobname`

When the SAS Job Execution Web Application receives this parameter, the job location is obtained from the program path and the job definition is retrieved.

Alternatively, an exact location can be entered using the `_JOB` parameter. The Job Definitions URI, which contains the unique identifier for the job, is specified in the following form:

`_job=/jobDefinitions/definitions/3b9f3a5e-deb1-4873-a90a-be6280e35deb`

To obtain the value for `_JOB`, open the Properties window for the job and click **Details**. The **Job details** field contains the location of the job definition details.

Executing a Job Using Direct URL Access

You can execute a job by entering a URL into the address bar of your web browser. The URL for the example in “Creating a Simple Job That Uses DATA Step Code” on page 47 is as follows:

```
http://host:port/SASJobExecution/?_program=/SomeFolder/Simple HTML
```

Specify the complete path and name of the job in the `_PROGRAM` URL parameter. Perform the following steps to get a copy of the URL without input parameters:

- 1 Click a job to select it, and then click  and select **Properties** to display the Properties window.
- 2 Click **Details** to see the location of the job definition details and the URL for submitting the job. For the Simple HTML example, you might see values such as the following:

Job details /jobDefinitions/definitions/493ba851-303b-4fb1-ac12-22199e085320

Job submit http://host:port/SASJobExecution/?_program=/SomeFolder/Simple HTML

- 3 Copy the value for Job submit.

SAS global macro variables are created from all query string parameters to the right of the question mark (?) in the URL. These macro variables are available for use in your SAS program. In this case, a macro variable named `_PROGRAM` is created with a value of `/SomeFolder/Simple HTML`.

The program creates the following output:

Hello World!

You can also execute a job by accessing a link in a web page. Specify the previous URL in the `HREF` attribute of an anchor tag:

```
<a href="http://host:port/SASJobExecution/?_program=/SomeFolder/
Simple HTML">Click here to execute job</a>
```

Using Job Input Parameters

Job input parameters pass data to your SAS program as a list of name/value pairs. The name/value pairs can be specified in the URL, in fields in an HTML input form, in prompt fields, in job definition parameters, or in the SAS Job Execution Web

Application configuration pane. SAS global macro variables, which are created from these name/value pairs, are available for use in your program.

Note:

- When prompts are displayed, they are populated with any parameter values that are specified in the URL or job definition. URL parameters take precedence over job definition parameters. Values for both URL parameters and job definition parameters take precedence over parameter values that are set in prompt XML.
 - When you use an HTML input form or XML prompts, parameters must be included in the job definition, HTML input form, or XML prompts. You cannot specify a new parameter in the URL.
-

For example, when a job is executed using the following URL:

```
http://host:port/SASJobExecution/form/?_program=/Folder1/myJob&p=123
```

The following macro variables are created:

Macro Variable Name	Macro Variable Value
_program	/Folder1/myJob
p	123

To avoid security vulnerabilities, the compute server processes incoming variable values to handle "special" characters. Unsafe characters that are masked for input parameters are the ampersand (&), apostrophe ('), percent sign (%), and quotation marks ("). Semicolons (;) are replaced with blank characters. See ["Specify Input Parameters" in SAS Viya Platform: Programming Run-Time Servers](#) for more information about special characters that are masked. See [SAS Functions and CALL Routines: Reference](#) for more information about using the COMPSRV_OVAL and COMPSRV_UNQUOTE2 functions to unmask some or all of the unsafe characters in an input parameter.

Prompting for Input Parameters with Job Forms

You can create HTML input forms or prompt forms that are stored separately from the job definition. Storing the form within the job definition makes it simpler to keep track of the form and to move or copy job definitions and forms. However, creating forms that are stored separately enables you to share forms between multiple job definitions.

To create a separate form, navigate to the folder where you want to store the form and click . Specify a name for the form, and then select **HTML form** or **Prompts** from the **File type** drop-down list. If you do this, you can either use the _FORM job definition parameter to specify the location of the form, or the form and job

definition can be in the same location and have the same name. The `_FORM` parameter can be specified in the URL as an input parameter or preset as a job definition parameter using the Properties function of the Content page.

Note: You must have access to developer functionality to modify properties for a job.

Here is an example of specifying the HTML input form using an input parameter:

```
http://host:port/SASJobExecution/form/?_program=/Folder1/myJob&_form=/Folder2/myJobForm
```

The HTML input form named `myJobForm` stored in `/Folder2` is displayed. Note that the HTML input form does not need to be in the same folder as the job (`/Folder1`) nor does it need to have the same name as the job (`myJob`).

If the `_FORM` parameter is not specified, then an HTML input form with the same name and location as the corresponding job is displayed, if it exists:

```
http://host:port/SASJobExecution/form/?_program=/Folder1/myJob
```

The HTML input form named `myJob` stored in `/Folder1` is displayed. The sample jobs provided by SAS take advantage of this feature.

Alternatively, you can use the `_ACTION` input parameter in the URL instead of the `/form` path. The previous examples become:

```
http://host:port/SASJobExecution/?_action=form&_program=/Folder1/myJob
&_form=/Folder2/myJobForm
```

and

```
http://host:port/SASJobExecution/?_action=form&_program=/Folder1/myJob
```

To display a prompt form with the same name and location as the corresponding job, you can use the `_ACTION=prompts` input parameter in the URL, or the `/prompts` path as follows:

```
http://host:port/SASJobExecution/prompts/?_program=/Folder1/myJob
```

Note: If you specify a value for the `_ACTION` input parameter in the URL, that value overrides any value for `_ACTION` that is specified as a job definition parameter.

Use the Properties function of the Content page to specify `_FORM` as a preset job parameter. This technique is useful when you have multiple jobs that need to use the same HTML input form.

Note: You must have access to developer functionality to modify properties for a job.

Suppose that you have two jobs that use the HTML input form name `Shared Input Form` stored in the `/Folder2` folder. Specify `/Folder2/Shared Input Form` for the `_FORM` parameter of each job. The input form is then displayed when both jobs are run using `/form` or `_action=form`, without the need for specifying `_FORM` in the URL:

```
http://host:port/SASJobExecution/form/?_program=/Folder1/myJob1
```

and

```
http://host:port/SASJobExecution/?_action=form&_program=/Folder1/myJob2
```

See “Passing User Input to a Job Using a Job Form” on page 51 for an example of how to use an HTML input form to pass input to a job.

Specifying Output Files

When a job is run by the compute server, files stored in the SAS Infrastructure Data Server can be returned to the web browser for display. After the job has completed, the SAS Job Execution Web Application directs the web browser to retrieve and display the desired files.

A file matching the pattern `_webout.*` is returned by default. If more than one file matches this pattern, then they are all returned in alphabetical order with the case and file extension considered in the ordering. The first file is displayed in an IFRAME in the HTML page, and links are provided to display the remaining files. Click a link to display the content of the file in the IFRAME. If the IFRAME is unable to render the content (for example, an RTF file or a SAS data set), then the file is downloaded.

Use the `_RESULTFILE` parameter to indicate which files should be returned. The parameter accepts a comma-separated list and supports the `""` (multi-character) and `"?"` (single-character) wildcards. The following table provides some examples:

_RESULTFILE Value	Returns
<code>*</code>	All files
<code>*.csv</code>	Only CSV files
<code>*.html,*.htm</code>	HTML and HTM files, in that order
<code>*.htm*,*.pdf,*.csv</code>	HTM, HTML, PDF, and CSV files, in that order
<code>state??.csv</code>	CSV files beginning with <code>state</code> and ending with any two characters (for example, <code>state01.csv</code>)
<code>retail.htm,retail.pdf,class.htm,class.pdf</code>	Only the files that are listed in the order in which they are listed

Use `_action=json` if you have an application that requires a list in JSON format of output files that are created by the job. Here is an example:

```
http://host:port/SASJobExecution/?_program=/Folder1/myJob&_action=json
```

This returns the following JSON instead of the _webout.htm file:

```
{ "items": [
  {
    "name": "_webout.htm",
    "href": "/files/files/a680029d-da79-4d1a-83be-2440c6e6b89e/content"
  }
]}
```

Use the URI in the HREF property to retrieve the _webout.htm file.

The following example uses the _RESULTFILE and _OMITTEXTLOG parameters to return all files that are created as a result of running the job, including the SAS log:

```
http://host:port/SASJobExecution/?_program=/Folder1/myJob&_action=json
&_resultfile=*&_omittextlog=false
```

The following JSON is returned:

```
{ "items": [
  {
    "name": "7BC34A5D-42BD-FD4C-AAF1-87D3A55AB8D9.log.txt",
    "href": "/files/files/bff66afe-eaf3-4af1-8261-b64ec9b72198/content"
  },
  {
    "name": "Class.pdf",
    "href": "/files/files/7ffad6f2-f6f4-4093-af4e-0548c4dfe8b0/content"
  },
  {
    "name": "Class.xml",
    "href": "/files/files/1987b95d-196c-4fad-9590-7afc2f6e8be3/content"
  },
  {
    "name": "_webout.htm",
    "href": "/files/files/6ded3921-0ae7-4e97-bf97-a654df996d27/content"
  }
]}
```

Using Output Job Parameters

A status message can be displayed to the user at the end of the job execution. If the SAS macro variable _STATUS_MESSAGE contains a value, then the value is displayed in a window. See [“Sending ODS Output to an Email Recipient” on page 62](#) for an example of how to use _STATUS_MESSAGE.

Modifying the Job Execution

Job parameters can be used in the URL to change how jobs are executed. Default values for many of these job parameters can be set as configuration properties. See [“Setting Configuration Properties” on page 31](#) for more information about setting

configuration properties. Job parameters specified in the URL become macro variables, and they are available for use within your SAS program.

The `_ACTION` job parameter performs many different functions, as described in some of the following sections. See [“Prompting for Input Parameters with Job Forms” on page 16](#) for more information about how to use it to display HTML input forms and prompts. The `nobanner` value is reserved for future use.

Specifying SAS System Options

You can use special input parameters to set SAS system options for a job. These options are specified upon SAS invocation. The general format is as follows:

`%opt-unique-name option-specification`

Always begin with `%opt-` (case-insensitive) followed by a name not used by another input parameter. These parameters can be specified in an HTML input form, in a prompt form, as job definition parameters, or in the query string. No SAS macro variables are created for this type of input parameter.

The following table shows sample values for setting a single option using an HTML input form, prompt form, or job definition parameter:

Parameter Name	Value
<code>%opt-myopt</code>	<code>linesize=max</code>

The following table shows sample values for setting more than one option using an HTML input form, prompt form, or job definition parameter:

Parameter Name	Value
<code>%opt-myopt1</code>	<code>linesize=max</code>
<code>%opt-myopt2</code>	<code>nobyline</code>

When specifying the parameter as part of the query string, be sure to URL encode the percent sign (“%”) in `%OPT-` and also the equal sign (“=”) if it is present in your option value. For example, use this URL parameter to specify 200 for the `LINESIZE` option:

`http://host:port/SASJobExecution/?_program=/Folder1/myJob&%25opt-myopt=linesize%3d200`

Job Execution Time-out

By default, the SAS Job Execution Web Application waits 300 seconds for a job to complete. You can change the time-out value for a specific request by specifying

the `_TIMEOUT` job parameter in the URL. For example, to specify a value of 480 seconds, use the following value:

```
_timeout=480
```

The default time-out value for all jobs can be set using the `EXECUTETIMEOUT` configuration property. See [“Setting Configuration Properties” on page 31](#) for more information about setting configuration properties.

If you execute a job that is expected to take more than 300 seconds to complete and you want the browser to wait, you can use `_action=wait`, which has no default time-out. Also, jobs that run using the `_action=background` parameter do not have a time-out value.

By default, the proxy server time-out is 300 seconds for NGINX clusters. If you execute a job that requires more than 300 seconds to complete, then you might not get any output or you might receive a 504 Gateway Time-out error message.

You can edit the YAML file to change this time-out value. For example, to specify a value of 480 seconds, perform the following steps:

- 1 Enter the following command to edit the `sas-job-execution-app` container annotation:

```
kubectl edit ingress/sas-job-execution-app
```

- 2 Add the following lines to the YAML file:

```
metadata:
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "480"
```

Note: When you use the `kubectl edit` command, any changes will be overwritten when you update the environment. You will need to reapply the changes after each update.

Job Output Expiration

The output files created by a job are available for 30 minutes by default. Jobs that are scheduled or that run in the background expire after 24 hours by default. You can use the Jobs page to set the expiration after a job has executed. You can also use the `_EXPIRATION` job parameter to specify a different duration. The value is formatted according to the W3C XML duration data type. For example, the following value specifies that the job expires within 60 minutes after execution has completed:

```
_expiration=PT60M
```

Specifying zero (0) indicates no expiration, and the output files are not deleted. The default value can be set using the `EXPIRATION` configuration property.

Background Processing

Use the `_ACTION` job parameter to perform several operations. Specify the following value to execute a job in the background without waiting for completion:

```
_action=background
```

The following message is displayed after the job is submitted:

```
Job /Folder/job-name submitted for background processing.
```

You can use the Jobs page or  in the Content page to view the results once the job has finished running.

For all jobs that are submitted, the job ID is returned in the HTTP header `X-SAS-JOBEXEC-ID`. For jobs that run in the background, the job ID can be used as follows to poll the `jobExecution` service to identify the state of the job:

```
http://host:port/SASJobExecution/?_jobexec=/jobExecution/jobs/<job-ID>/state
```

When the job has finished running, you can use the job ID as follows to view the results:

```
http://host:port/SASJobExecution/?_jobexec=/jobExecution/jobs/<job-ID>
```

Jobs run in the background expire after 24 hours, unless a different duration is specified using the `_EXPIRATION` parameter. You can also use the Jobs page to set the expiration after a job has executed.

Perform the following steps to use the `BACKGROUND` configuration property to specify a custom HTML page to display instead of the default message:

- 1 Create an HTML file with your customized message and then save it to a folder. See [“Passing User Input to a Job Using a Job Form” on page 51](#) for more information about creating an HTML file.
- 2 Follow the instructions in [“Setting Configuration Properties” on page 31](#) to create the `BACKGROUND` property with the full path and name of the HTML file. For example, specify `/Folder/My Custom Background Message` as the value.

Wait Screen

You can use the `_ACTION` job parameter to display a wait screen with informational text while the job is executing:

```
_action=wait
```

The default value for the text is **Please wait**. Use the `WAITTEXT` configuration property to specify the text of the wait message.

Retrieving Previous Results

Output from a previous job execution can be displayed if it has not yet expired:

```
_action=lastjob
```

This avoids the time needed to execute the job in the compute server and can be used to provide semi-static reports. This feature can improve performance and reduce server load if multiple users request a job that produces exactly the same results.

If there is no previous output, then you receive an error message indicating that previous job output does not exist.

The `_ACTION` job parameter also accepts a comma-separated list of values. Add `execute` to force the job to execute if the previous job output does not exist:

```
_action=lastjob,execute
```

Scheduling a Job

When you execute a job, you can use the `_ACTION` parameter as follows to save a copy of the job definition and run-time parameters (arguments) in the Scheduling table in SAS Environment Manager:

```
_action=schedule
```

See [“Scheduling a Job” on page 10](#) for more information about scheduling jobs.

By default, scheduled jobs expire after 24 hours. See [“Job Output Expiration” on page 21](#) for more information about how to change the expiration value for a job.

Saving Job Output

Job output files can be saved for later viewing. Use the `_SAVEFOLDER` parameter as follows to specify the folder in which to save the output:

```
_savefolder=/Folder1/Folder2
```

By default, the file names that are specified in the `NAME` option of the Files service access method `FILENAME` statement are used. Existing files are replaced. See the sections about assigning a fileref in [“Development Concepts” on page 46](#) for more information about how to use the `FILENAME` statement.

The `_SAVEFILE` parameter specifies the file to save:

```
_savefile=_webout.html
```

Only the `_webout.html` file, specified in the `NAME` option of the Files service access method `FILENAME` statement, is saved. Omit this parameter to save all job output files or if the job creates only one file. Note that by default, only files named with

the pattern `_webout.*` are returned. Use the `_RESULTFILE` parameter to specify return files with names that do not match this pattern. See [“Specifying Output Files” on page 18](#) for more information.

Specify the parameter multiple times to save only some of the output files created by a job. Only the `_webout.html` and `_webout.pdf` files are saved from a job that creates multiple files:

```
_savefile=_webout.html&_savefile=_webout.pdf
```

and

```
_resultfile=*&_savefile=MyOutput.html&_savefile=MyOutput.pdf
```

The `_SAVEFILE` parameter can be used to save a file with a new name:

```
_savefile=original-name,new-name
```

The `_webout.html` file is saved as `MyOutput.html`:

```
_savefile=_webout.html,MyOutput.html
```

See [“Managing Jobs” on page 9](#) for more information about manually saving job output.

Debugging

Debugging information can be returned to assist in program development and execution. Like the `_ACTION` job parameter, `_DEBUG` accepts a single value or a comma-separated list of values:

```
_debug=fields,log,time,trace
```

If you are using the numeric values, you can also set more than one debug option by adding the flag values together. For example, to set both the options `2048` and `2`, use the value $(2048 + 2) = 2050$.

The following values are available:

Value	Information Returned
<code>fields</code> (or 1)	List of all input parameters. Those listed in the Arguments section become SAS global macro variables.
<code>time</code> (or 2)	Total time in seconds to execute the job; incremental and total time are reported when used in conjunction with <code>trace</code> .
<code>log</code> (or 128)	SAS job log.
<code>trace</code> (or 2048)	Information for each step of the job.

Use `trace` to display detailed information about the job execution, including preset and input parameters, the job definition and SAS code, and results created by the job.

By default, the `_DEBUG` parameter can be used by any authenticated user. To restrict its usage on a group or user basis, change the authorization rules for the `/SASJobExecution/debug/` key using SAS Environment Manager. See [“Security for SAS Viya Platform Jobs” on page 38](#) for more information.

Use the `DEBUGDISALLOW` configuration property to disallow the use of specific debug values for all users.

Logging in the web application is controlled by the `SAS_LOG_LEVEL_JOBEXECAPP` environment variable. Perform the following steps to change the amount of information that is generated in the log:

- 1 Click  in SAS Environment Manager, and then select **Definitions** from the **View** drop-down list.
- 2 Click **logging.level** to display the properties that are used to configure logging levels.
- 3 Specify `JOBEXECAPP` in the **Filter** field at the top of the right panel.
- 4 If no properties are found, then you must add one. Click **New Configuration** and then supply the required value in the New logging.level Configuration window. Otherwise, select the desired value from the **level** drop-down list. For example, specify the following values to set the level to report debug messages:
 - For **Services**, specify `SAS Job Execution`.
 - For **level**, specify `DEBUG`.
 - For **name**, specify `JOBEXECAPP`.

New logging.level Configuration

Services:

SAS Job Execution

Services to which this configuration instance applies. 'Global' indicates the configuration instance applies to all services.

level: *

DEBUG

Logger level.

name: *

JOBEXECAPP

Logger name.

Save Cancel

- 5 You can use Kubernetes `kubectl` commands to view the web application log. Use the following command to find the `sas-job-execution-app-pod-number` pod:

```
kubectl get pods
```

Use the following command to view the log:

```
kubectl logs -f sas-job-execution-app-pod-number
```

Using the Command-Line Interface (CLI)

The SAS Job Execution Web Application can be called from a command-line interface. A request with URL parameters is sent to the web application and results are returned and displayed as text (including any HTML tags).

To use the command-line interface, you must first download the job plug-in using the instructions in [Get the CLI and Its Plug-Ins](#). Create a profile using the instructions in [Create at Least One Profile](#), with the following differences:

- To initialize a new profile, enter the following command:

```
sas-viya profile init
```

- When prompted for the output type, specify **text**.
- When prompted about whether you want to enable ANSI colored output, specify **Y**.

You can use a command such as the following to execute a job, where *query-string* is any valid combination of URL parameters:

```
sas-viya job webapp execute --query-string "_program=ping"
```

The command returns the same output that the web application generates, including raw HTML. For example, you can use `_action=json` to get JSON results.

You can enter `sas-viya job --help` to show other supported commands.

For more information about using command-line interfaces, see [SAS Viya Platform: Using the Command-Line Interface](#).

Executing a Job as a Compute Task

Starting in the 2025.09 release, you can execute a short job as a compute task. Executing the job in this way uses the Compute service rather than the Job Execution service. This method is useful for executing SAS Stored Processes that have been migrated to SAS Viya from SAS 9.4. You can submit a SAS program, macro variables, and input files. The Compute service creates a session, executes the program, and returns the results to the client.

Note:

- Performance has been improved while running short jobs as compute tasks. However, it is best practice to execute long-running jobs or large files using the Job Execution service rather than the Compute service. The default maximum time-out value for compute tasks is 30 seconds, which does not include SAS Job Execution Web Application or network overhead. If you encounter a time-out error while executing a job as a compute task, consider submitting the job using the Job Execution service instead.
- If you execute a job as a compute task:
 - ☐ The job is not listed on the Jobs page.
 - ☐ Output files cannot be downloaded.
 - ☐ The job cannot be scheduled.

Before you can use this feature, you must set up a [compute context for a reusable server](#). Make sure the jobs that you want to run as compute tasks are using that compute context.

Use the `_EXECUTIONTASKS` job definition parameter to specify whether you want to execute the job as a compute task. Set `_executionTasks=True` to run the job as a compute task. In this case, `_WEBOUT` is assigned a temporary location. If you set `_executionTasks=False`, the job runs using the Job Execution service. In this case,

_WEBOUT is assigned a URI such as /files/files/. If you do not set this job definition parameter, the job is executed using the Job Execution service.

You can use the _WEBIN_* macro variables in your program to provide input to the job if you are running it as a compute task. See [“Reserved Macro Variables” on page 181](#) for more information about these macro variables. For example, you can use the _WEBIN_FILEREF variable in your program to reference an uploaded CSV file, as follows:

```
options validvarname=any validmemname=extend;

proc import datafile=&_WEBIN_FILEREF
    dbms=csv
    out=work.mydata
    replace;

    getnames=yes;
run;
```

You can use the _WEBIN_SASNAME variable in your program to reference an uploaded CSV file, as follows:

```
proc print data=work.&_WEBIN_SASNAME;
run;
```

To ensure that output displays correctly, edit the CONTENTSECURITYPOLICY configuration property. For more information, see [“Setting the Content Security Policy” on page 43](#).

For more information about compute tasks, see [“Compute Tasks” in SAS Viya Platform: Programming Run-Time Servers](#). For more information about how to work with compute tasks, see [“Work with Compute Tasks” in SAS Viya Platform: Programming Run-Time Servers](#).

Modifying Your Settings

You can use the Settings window to edit user preferences or customize accessibility settings. Changing these settings does not impact other users. To access these settings, click the user button in the application bar and select **Settings**.

General

The **General** section includes settings that enable users to change the appearance of the web application, enable warning and information messages to be displayed, and choose a profile picture. Here are the settings:

- You can change the appearance of the web application by using the **Theme** setting. The default theme is set by an administrator. The theme specifies the collection of colors, graphics, and fonts that appear in the application. You can

also apply other themes, including both SAS themes and custom themes, if available.

Select **Choose a theme**, and then select another theme from the drop-down list to change the look of the application. The theme change takes effect after you close the **Settings** window.

Note: SAS themes and custom themes are separated by a horizontal separator line. SAS themes appear at the top.

SAS themes:

High Contrast

Presents a dark background with high-contrast foreground elements to meet the needs of users with low vision.

Dark

Presents an all-dark color palette for a better low-light experience.

Light

Presents a light and clean color palette. The Light theme is the default application theme.

- If you want messages to display that you previously asked not to display, click **Reset Messages**. By default, all warning and information messages are displayed.
- You can select a profile picture to display in the application bar, as well as in other places within the application that use profile pictures. The default is the first initial or character of your first name.

Click **Choose Picture** and then select an image file to upload. The image file's size can be up to 1 MB. The valid file types are BMP, GIF, JPEG, JPG, and PNG.

Region and Language

The **Region and Language** section includes the following settings that enable users to specify the locale for regional formats and sorting, as well as for offline processes:

- The **Locale for regional formats and sorting** setting specifies the locale that is used for sorting data and formatting values such as dates, times, numbers, and currency. By default, the browser locale is used. Changes take effect after you sign out and sign back in.
- The **Locale for offline processes** setting specifies the locale that is used for offline jobs or background processes such as report distributions or notifications. By default, the locale of the Java Runtime Environment is used.

Accessibility

Several settings in the **Accessibility** section can assist people who rely on assistive technologies:

- Select **Enable sounds** to hear audio indicators for events that occur within the user interface. Changes take effect after you sign out and sign back in.
- Select **Enable visual effects** to show visual effects that indicate state changes. For example, when this setting is selected, you will see a subtle movement in the user interface if you delete an item.
- Select **Adjust the display duration for pop-up notifications** to specify how long temporary pop-up notification messages are displayed. Increasing the amount of time that notifications are displayed can help users discover and read messages that disappear automatically.
- Select **Invert application colors** to make the user interface easier to see for users with sensitivity to certain bright colors. You can also use the Ctrl+` (Ctrl +back quote) keyboard shortcut to invert the application colors.
- Select **Display tooltips when using the keyboard to navigate** to enable keyboard users to access tooltips. When this option is selected, putting keyboard focus on a control also displays the tooltip on the screen. You can also select the location in the browser window to display the tooltip. By default, the tooltip is displayed in the bottom right corner of the browser window.
- The focus indicator is an outline that indicates which user interface component is active. You can make the focus indicator easier to see by selecting **Customize the focus indicator settings** and adjusting the color, thickness, and opacity.

Administration

Administrative Tasks	31
Setting Configuration Properties	31
Setting Preset Parameters	33
Configuring the Compute Server	36
Modifying Server Files	37
Security for SAS Viya Platform Jobs	38
Setting Authorization for a Folder	38
Changing Access to Application Functions	38
Enabling Guest Access	40
Sharing Jobs and Files with Group Members	42
Preventing Cross-Site Request Forgery (CSRF)	42
Setting the Content Security Policy	43
Testing the Installation	44
Executing the Ping Program	44
Running a Sample Job	44

Administrative Tasks

Administration of the SAS Job Execution Web Application consists of setting configuration properties and preset job parameters, enabling logon options, installing sample programs, and configuring the compute server. An account belonging to the SASAdministrators group is required to perform these actions. When performing administrative tasks, make sure to log in to SAS Environment Manager as an administrator and accept the administrator rights.

Setting Configuration Properties

Configuration properties control the default behavior of the SAS Job Execution Web Application. Use SAS Environment Manager to view and modify these parameters.

In SAS Environment Manager, select  in the left pane. Select **All services** from the drop-down menu in the left pane and then select **SAS Job Execution**.

The configuration properties are displayed in the right pane and can be changed by selecting  in the upper right corner of the sas.jobexecapp section. To add a new property, scroll to the bottom of the list, click **Add property**, and then specify the property name and value in the Add Property window.

The following properties can be changed or added:

Table 2.1 Configuration Properties

Property	Default Value	Description
actiondisallow		Comma-separated list of _ACTION values to disallow
background		SAS folder path and name of HTML file to display for background processing
contentsecuritypolicy		Content-Security-Policy setting for jobs. SCRIPT tags are disabled in order to prevent Cross-Site Scripting (XSS) attacks. For more information, see “Setting the Content Security Policy” on page 43 .
contextname	SAS Job Execution compute context	Compute service context name
createapplist	false	Specifies whether to return the _APSLIST macro variable when a job is executed
debugdisallow		Comma-separated list of debug values to disallow
executetimeout	300	Job run time-out in seconds
expiration	PT30M	Job expiration duration (30 minutes) in the format defined in the W3C XML duration data type
getrunpaths		Comma-separated list of locations for programs that are allowed to run using a GET request. If you enable this property and leave the value blank, then all jobs must be executed using a POST request. You can use wildcards when specifying locations. An asterisk (*) wildcard can be used to match 0 or more characters. A question mark (?)

Property	Default Value	Description
		wildcard can be used to match any single character. For example, a value of <code>/Public/myjob,/Test/*</code> allows the myjob job and any jobs located in the Test folder to execute using a GET request.
indextitle	SAS Job Execution	Banner title for the application
maxfilecount	5	File upload maximum file count
maxfilesize	100000000	File upload maximum file size in bytes. Note: If you increase the maxfilesize value for the SAS Job Execution service, you should also increase the value of maxFileSize for Files service .
waittext	Please wait	Message to display on job wait display. You can include HTML tags to control the formatting of the message.
welcome		SAS folder path of HTML file to display if no parameters are specified

Setting Preset Parameters

Like configuration properties, preset parameters are defined using SAS Environment Manager. The parameter name/value pairs are set when a job executes and SAS global macro variables are created from the name/value pairs. The preset parameter definitions are in the same pane as the configuration properties. Follow the instructions in [“Setting Configuration Properties” on page 31](#) to access this pane, and then scroll to the Preset Parameters section.

These parameters can be set to fixed strings or values that are substituted when the job is run. The properties that are available for substitution are listed in the **Request Properties**, **System Properties**, and **HTTP Request Headers** sections displayed by the System Environment page. The following table contains some common request properties:

Table 2.2 Common Request Properties

Recommended SAS Variable Name	Request Property Name	Description
_AUTHTYP	jobexec.auth.type	Name of the authentication scheme that is used (for example, BASIC, SSL, or blank if no protection).
	jobexec.character.encoding	Name of the character encoding that is used in the body of the request.
	jobexec.content.length	Length (in bytes) of the request body, which is made available by the data source. If the length is not known, the value is -1.
	jobexec.content.type	MIME type of the body of the request. If the type is not known, the value is blank.
	jobexec.context.path	Portion of the request URL that indicates the context of the request.
	jobexec.cookies	Cookie strings that the client sent with this request.
	jobexec.header	All HTTP request headers.
	jobexec.header. <i>name</i>	A particular HTTP request header line as it was received, where <i>name</i> is the header name.
_HTUA	jobexec.header. <i>user-agent</i>	A particular HTTP request header line as it was received, where <i>user-agent</i> is the name of the user agent.
	jobexec.jsessionid	Web application session ID.
_USERLOCALE	jobexec.locale	Preferred locale in which the client accepts content, based on the Accept-Language header.

Recommended SAS Variable Name	Request Property Name	Description
	jobexec.method	HTTP method used when this request was made (for example, GET, POST, or PUT).
	jobexec.path	URL pathname.
	jobexec.protocol	Name and version of the protocol that the request uses in the form <i>protocol/majorVersion.minorVersion</i> (for example, HTTP/1.1).
	jobexec.query.string	Query string that is contained in the request URL after the path.
_RMTADDR	jobexec.remote.addr	Internet Protocol (IP) address of the client that sent the request.
_RMTHOST	jobexec.remote.host	Fully qualified name of the client that sent the request or the IP address of the client if the name cannot be determined.
	jobexec.request.inputencoding	Request input encoding. The default encoding is UTF-8.
	jobexec.scheme	Name of the scheme that was used to make this request (for example, HTTP, HTTPS, or FTP).
	jobexec.secure	A value of true or false, indicating whether this request was made using a secure channel, such as HTTPS.
	jobexec.user	Login ID of the user that made this request if the user has been authenticated. If the user has not been authenticated, the value is blank.
_URL	jobexec.uri	Part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.

Recommended SAS Variable Name	Request Property Name	Description
<code>_VERSION</code>	<code>jobexec.version</code>	SAS Job Execution Web Application version.
<code>_XFORWARD</code>	<code>jobexec.header.x-forwarded-host</code>	Request HTTP header value for <i>x-forwarded-host</i> .
<code>_CLIENTNAME</code>	<code>SASJobExecution</code>	Name of the client.

Use a dollar sign character (\$) followed by the property name to perform run-time substitution. Any unresolved values result in the corresponding parameter being set to a zero-length string. The following table provides some examples:

Table 2.3 Example Macro Variables

Macro Variable Name	Value	Description
<code>_SOME_TEXT</code>	<i>This is some text</i>	Hardcoded text
<code>_SCHEME</code>	<code>\$jobexec.scheme</code>	Name of the scheme for the request
<code>_HTTP_HOST</code>	<code>\$jobexec.header.host</code>	Host and port number for the request

The last example illustrates how to assign the value of an HTTP request header to a macro variable using the general syntax `$jobexec.header.name`. Be sure to specify `header` when referencing an HTTP header.

It is good practice to use an underscore character (`_`) at the beginning of system macro variable names. Refrain from creating macro variables in your SAS program that begin with an underscore character to avoid overwriting system macro variable values.

Configuring the Compute Server

By default, each client user must have a user account on the SAS compute server machine to run a request. However, you can use service accounts with the compute server, which enables the administrator to configure a compute server context to be associated with a service account. Then all compute server sessions for that context are run as the specified service account (a shared account). This removes the need for end-users to have operating system accounts on the hosts where the compute server is running. These alternate compute contexts can be used for

specific jobs by specifying the context name in the `_contextName` parameter for the job, or by selecting a value for **Compute context** in the Properties window for the job.

After you configure the compute server to run under a shared account, you can configure the server to be reusable. This enables the server to be reused by a later session instead of terminating when a session ends, which in turn helps performance because you do not have to wait for a new server to start.

See [“SAS Compute: Server, Service, and Contexts” in SAS Viya Platform: Programming Run-Time Servers](#) for more information about configuring the compute server.

Modifying Server Files

Although the server files `autoexec.sas`, `sasv9.cfg`, and `startscript.sh` are not available for editing, you can use configuration instances to make any necessary changes.

Start SAS Environment Manager and then select  in the left pane. Select **All services** from the drop-down menu in the left pane. Enter `Compute` in the **Filter** field and then select **Compute service**.

In the right pane, there is a list of configuration instances. The following instances are included in this list:

- `sas.compute.server: autoexec_code`
- `sas.compute.server: configuration_options`
- `sas.compute.server: startup_commands`

See [“Configuration Instances: Compute Service” in SAS Viya Platform: Programming Run-Time Servers](#) for more information about these configuration instances.

You can click  next to an instance to make changes to that instance. In the Edit Configuration window, specify any changes to the configuration options for the server in the **contents** area. If you want an `autoexec` statement added to each server invocation, add it to the `autoexec.sas` file, which is the replacement for `autoexec_usermods.sas`.

Note: The options that you specify are not validated for correctness.

The **name** field contains the file name for the usermods file that is unique to the server. Do not modify this value unless SAS Technical Support instructs you to change it.

After you save your changes, they are applied by each compute server at launch.

Security for SAS Viya Platform Jobs

Setting Authorization for a Folder

By default, the authorization for a top-level folder allows the owner full access to the folder. For more information about viewing and editing authorization for folders and folder objects, see [SAS Viya Platform: Folders](#).

Changing Access to Application Functions

Access to the SAS Content page for developers and other functions is determined by authorization rules applied to object URIs. Use SAS Environment Manager to locate and then edit or add rules that control which users and groups can access the object URIs.

Using an administrative account, click  in the left navigation bar to access the SAS Environment Manager Rules page. Specify a URI, such as `/SASJobExecution/developer/**`, in the **Object URI** field of the **Rules Filter**, and then click **Apply** to filter the rules of this URI:

Rules Filter [Expand] [Collapse] [Refresh]

- ▼ Object URI [Reset](#)
- > Container URI [Reset](#)
- > Principal (no filter) [Reset](#)
- > Setting (no filter) [Reset](#)
- > Media Type (no filter) [Reset](#)
- > Rule Status (no filter) [Reset](#)
- > Description [Reset](#)
- > Reason [Reset](#)
- > Modified By [Reset](#)
- > Date Modified [Reset](#)
- > Rule ID [Reset](#)

[Apply](#)

If you want to restrict who can create job definitions and job forms, copy the rule for `/SASJobExecution/developer/**`. Change the copy to specify the users or groups that should have developer capabilities. Disable the original rule for `/SASJobExecution/developer/**`.

Note: Be sure to make authorization changes in the new copy of the rule. Any changes you make to the original rule (other than disabling the rule) will reset to the original values when the SAS Job Execution Web Application is restarted.

For more information about general authorization and working with authorization rules or about permissions that you can grant or prohibit, see [SAS Viya Platform: General Authorization](#). For most functions, the SAS Job Execution Web Application checks permissions based on the HTTP request method used to access the URI.

Use the Read permission to control access to the `_DEBUG` job parameter. For all other URIs, use the Read permission to control access to the URI using the HTTP GET method and CREATE for the POST method.

The following table contains the URIs and default principals for all of the application functions:

Table 2.4 Application Function URIs and Principals

Function	Object URI	Default Principal
Web application root	/SASJobExecution	Authenticated Users
Administration interface	/SASJobExecution/ developer/**	Authenticated Users
_DEBUG job parameter	/SASJobExecution/debug/**	Authenticated Users
Display system environment	/SASJobExecution/env/**	SASAdministrators
Display an HTML input form before the job executes	/SASJobExecution/form/**	Authenticated Users
Display prompts before the job executes	/SASJobExecution/prompts/**	Authenticated Users
Standard interface to display and execute jobs	/SASJobExecution/user/**	Authenticated Users
Administer previous job runs	/SASJobExecution/jobs/**	Authenticated Users
Schedule jobs	/SASJobExecution/ scheduling/**	Authenticated Users
Administer sample jobs	/SASJobExecution/samples/**	Authenticated Users
Return the CSS for the current theme	/SASJobExecution/theme	Authenticated Users

Enabling Guest Access

Authorization rules can be used to run jobs using the Guest account. Perform the following steps to enable guest access:

- 1 Set up a [compute context for a reusable server](#) with a shared service account. Make sure the jobs that you want the Guest account to run are using that compute context.
- 2 Add the COMPUTE_SERVER_ALLOW_GUEST environment variable for the compute server at the launcher context level.
 - a In SAS Environment Manager, select  in the left pane. Select **Launcher contexts** from the drop-down menu in the left pane and then select **SAS Job Execution launcher context**.

- b The basic properties are displayed in the right pane and can be changed by selecting  in the upper right corner.

To add a new environment variable, click **+** above the **Environment Variables** table. In the New Environment Variable window, specify the `COMPUTE_SERVER_ALLOW_GUEST` as the variable and `true` as the value. Click **OK**.
 - c Click **Save**.
 - 3 In SAS Environment Manager, enable the `sas.logon.provider.guest` configuration.
 - a In SAS Environment Manager, select  in the left pane. Select **Definitions** from the drop-down menu in the left pane and then select **`sas.logon.provider.guest`**.
 - b Click **New Configuration** in the right pane.
 - c In the New `sas.logon.provider.guest` Configuration window, click the slider to enable the property. Click **Save**.

.....

Note: Enabling this configuration also enables all other guest rules in the system.

.....
 - 4 Grant Read permission to Guest for any folders that contain jobs (or you can save the jobs to the `Public` folder).
 - a In SAS Environment Manager, select  in the left pane. Select the folder to which you want to grant Read access for guest users.
 - b Right-click and select **Edit authorization**.
 - c In the Edit Authorization window, click  and in the Add Identities window, add Guest. Grant Read and Read (convey) access.
 - d Click **Save**.
 - 5 Apply authorization rules for Guest access to SAS Job Execution Web Application.
 - a Download the [rules.json](#) file.
 - b You can add authorization rules using the following CLI command:


```
sas-viya --profile cl --insecure auth login
sas-viya plugins install --repo SAS authorization
sas-viya --profile cl authorization create-rules --file rules.json
```
 - 6 Use the `SSO_GUEST` URL parameter to run a job with the Guest account. Here is an example:


```
http://host:port/SASJobExecution/?_program=jobname&sso_guest=true
```

For more information about enabling guest access for SAS Viya platform applications, see [“Guest Access” in SAS Viya Platform: Authentication](#).

Sharing Jobs and Files with Group Members

You can use SAS Environment Manager to add the following new rules if you want to enable members of a particular group to view jobs and files (such as output and logs) on the Jobs page.

Table 2.5 Rules and Values for Sharing Jobs and Files

Rule Field	Value for Enabling Members to View Files	Value for Enabling Members to View Jobs
Object URI	/files/files/**	/jobExecution/jobs/**
Principal type	Group	Group
Principal	<i>group-name</i>	<i>group-name</i>
Rule type	Grant	Grant
Condition	<code>groupsForUser(#file?.createdBy).contains("group-name")</code>	<code>groupsForUser(#job?.createdBy).contains("group-name")</code>
Permissions	Read	Read

For more information about how to add rules, see [“Rules Page” in SAS Environment Manager: User’s Guide](#).

You can use a group-managed service account to enable members of a scheduling group to execute or schedule a job or job flow that was created by another member of that group. When executed or scheduled, the job or job flow is initiated by the service account. For more information, see [“Using a Group-Managed Service Account” in SAS Viya Platform: Jobs and Flows](#).

Preventing Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack that gets users to perform unintentional, malicious actions on a website that they trust, using their own authentication. There are some ways that you can protect against CSRF.

By default, the SAS Job Execution Web Application allows users to execute jobs using GET requests, without any restrictions. One way to protect against CSRF attacks is to prevent jobs from being run using a GET request. You can use the

GETRUNPATHS configuration property to create an allowlist, which specifies locations for programs that are allowed to run using a GET request. If you enable the GETRUNPATHS configuration property but do not specify any locations, then all jobs must be executed using a POST request. You can use wildcards when specifying locations. You can set permissions for the allowed locations if you want to limit job creation in those directories. See [“Setting Configuration Properties” on page 31](#) for more information about how to use the GETRUNPATHS configuration property.

When you make a POST request using an input form, you can use the `_CSRF` input parameter to ensure that the request is considered non-malicious by sending a CSRF token to the server. Use `$CSRF$` as the value for the input parameter, and when the input form is displayed, the CSRF token for the request is substituted for `$CSRF$`. See [“HTML Input Form” on page 173](#) for an example of how to use the `_CSRF` input parameter in an input form.

Setting the Content Security Policy

SCRIPT tags are disabled in order to prevent Cross-Site Scripting (XSS) attacks. If you have existing code that uses SCRIPT tags, you can edit the CONTENTSECURITYPOLICY configuration property to enable SCRIPT tags. See [“Setting Configuration Properties” on page 31](#) for more information about how to access the CONTENTSECURITYPOLICY configuration property.

Starting in the 2024.02 release, the default value of this configuration property is as follows:

```
default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self';
style-src 'self'; frame-ancestors 'self'; form-action 'self';
```

To allow scripts in jobs, edit the configuration property to add `'unsafe-inline'` to the script-src section of the configuration property value, as follows:

```
default-src 'none'; script-src 'self' 'unsafe-inline'; connect-src 'self';
img-src 'self'; style-src 'self'; frame-ancestors 'self'; form-action 'self';
```

To use the default SAS style in jobs, edit the configuration property to add `'unsafe-inline'` to the style-src section of the configuration property value, as follows:

```
default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self';
style-src 'self' 'unsafe-inline'; frame-ancestors 'self'; form-action 'self';
```

If you are executing a job using the Job Execution Service and you are producing SVG output, edit the configuration property as follows to ensure that output displays correctly. Starting in the 2025.09 release, you must also make this change if you are executing a job as a compute task, regardless of output type.

```
default-src 'none'; script-src 'self' 'unsafe-inline'; connect-src 'self';
img-src 'self' data:; style-src 'self' 'unsafe-inline'; frame-ancestors 'self';
form-action 'self';
```

Testing the Installation

Executing the Ping Program

A special internal program is available to determine whether the system is functioning correctly:

```
http://host:port/SASJobExecution/?_program=ping
```

An HTML page is returned when this job runs to completion with the message:

```
Job completed successfully
```

Running a Sample Job

You can further test the installation by running one or more sample jobs, if they are installed on your system. See [“Samples” on page 95](#) for a list of samples and their expected output.

Developing Jobs

Development Concepts	46
Overview of %JESBEGIN and %JESSEND Macros	46
Creating a Simple Job That Uses DATA Step Code	47
Passing User Input to a Job Using the Query String	49
Passing User Input to a Job Using a Job Definition Parameter	50
Passing User Input to a Job Using a Job Form	51
Substituting Parameters Into a Job Form	53
Creating Simple HTML Output Using ODS	54
Creating Simple PDF, RTF, or DOCX Output Using ODS	55
Using Input Parameters with Multiple Values	55
Linking One Job to Another (Drill Down)	58
Sending ODS Output to an Email Recipient	62
Assigning a Fileref for HTML Output	62
Assigning a Fileref for Other Types of Output	63
Assigning a Fileref to Reference Source Code Stored in the SAS Infrastructure Data Server	64
Using Sessions	65
Working with Dynamic Data in Prompts and HTML Forms	74
Creating HTML Forms with Dynamic Data	75
Overview	75
Initializing the Dynamic Library	75
Selecting Libraries and Tables	76
Retrieving Table Column Data	76
Using Cascading Fields	77
Using Multiple Selections	78
Using a Combo Box	79
Using a Dual Selector	79
Using Paging	80
Filtering	81
Entering Values	81
Visibility	82
Enablement	82
Attributes	83
Advanced Programming	85
Sending JSON Data to a Job Using an Input Parameter	85
Sending JSON Data to a Job by Uploading a File	86
Saving Uploaded Files to a Folder	87
Executing a Job Using JavaScript - Sending Small Data to the Job	87

Executing a Job Using JavaScript - Sending Large Data to the Job	89
Executing a Job Using JavaScript - Sending Non-form Data to the Job	90
Returning JSON Data from a Job	91
Returning a List of Output Files in JSON Format	92
Errors Returned from JSON Requests	93
Including a File in an HTML Form	94
Working with SAS Viya Platform Services	94
Samples	95
Accessing the Samples	95
Items Common to Most Samples	95
Display Macro Variables	97
Dynamic Prompts Using CARS Data	109
Evaluating Data Cardinality Using PROC CAS	115
Hello World	122
MEANS Procedure Executed Using CAS	126
Multiple Output Formats	132
ODS Output with Embedded Graphics	138
Report with Download Links	145
Return GCHART Image Output	151
Return SGPLOT Image Output	156
Simple JSON	160
Simple ODS HTML	163
Upload a CSV File	167
Upload a File	172
Reference	177
%JESBEGIN and %JESEND Autocall Macros	177
Reserved Macro Variables	181
_ACTION Input Parameter Values	187
%PARAM_LIST Macro	187
Generated Macro Variables by Prompt Type	191

Development Concepts

Overview of %JESBEGIN and %JESEND Macros

The %JESBEGIN and %JESEND utility macros are automatically added to your code stream. There is no need to include them in your program code.

The %JESBEGIN utility macro sets up the job execution environment and executes before your SAS code. You use job input parameter values to control the macro.

For example, specify `_OUTPUT_TYPE=ods_html5` if your job uses ODS to create HTML output. The macro assigns a fileref named `_WEBOUT` to return output to the web browser or client application and issues an ODS HTML5 statement.

Note: The %JESBEGIN macro creates HTML5 output by default.

The %JESEND autocall macro cleans up the job execution environment and executes after the SAS code.

In some cases, you might need to use very specific FILENAME and ODS statements. In these cases, it is best to prevent the %JESBEGIN utility macro from generating these statements for you.

Specify the input parameter `_OUTPUT_TYPE=none` to suppress the generation of these statements. Alternatively, you can specify the `_ADDJESBEGINENDMACROS=false` input parameter, which prevents the automatic inclusion of the %JESBEGIN and %JESEND macro invocations in the code stream. An advantage of specifying `_OUTPUT_TYPE=none` is that the %JESBEGIN macro displays the macro variables created from input parameters.

See [“Samples” on page 95](#) and [“%JESBEGIN and %JESEND Autocall Macros” on page 177](#) for more information about using these macros.

Creating a Simple Job That Uses DATA Step Code

The following example creates a simple job that uses DATA step code to return HTML to the client.

You can use the SAS Job Execution Web Application to create jobs by using the Content page:

`http://host:port/SASJobExecution`

Note: You must have access to developer functionality to create jobs.

Perform the following steps to create the sample job:

- 1 Navigate to a folder location where you want to store the job, and then click .
- 2 Specify the name for the file (this example uses **Simple HTML**) and accept the default values for the **File type**, **Server**, and **Compute context** fields. Click **OK**.
- 3 Click the **Simple HTML** job, and then click  and select **Properties** to display the Properties window.

Properties [X]

▼ General [Reset](#)

Name:
/Public/Simple HTML

File type:
Job definition

Server:
Compute ▼

Compute context:
SAS Job Execution compute context ▼

Description:

> Details

> Parameters* [Reset](#)

- Click **Parameters**, and then click **Add a new parameter**. Specify the following properties for the parameter:

Property	Value
Name	<code>_OUTPUT_TYPE</code>
Default Value	html
Field Type	Character

Click **Save**.

This job creates HTML output but not HTML output generated by ODS. See [“Creating Simple HTML Output Using ODS” on page 54](#) for more information about creating ODS output from a job.

- 5 Click  and select **Source code**.
- 6 Enter the following code into the editor window:

```
* Write the custom HTML to _webout;

data _null_;
file _webout;
put '<html>';
put '<head><title>Hello World!</title></head>';
put '<body>';
put '<h1>Hello World!</h1>';
put '</body>';
put '</html>';
run;
```

- 7 Click , and then click **Close** to save the job and close the editor window.

This code writes HTML to the _WEBOUT fileref that is assigned by the %JESBEGIN macro. This fileref is assigned based on the value specified for the _OUTPUT_TYPE job parameter. The SAS Job Execution Web Application displays the HTML written to this fileref. See the sections about assigning a fileref in [“Development Concepts” on page 46](#) for more information about this type of FILENAME statement.

For information about how to execute a job using the web application, see [“Executing a Job Using the SAS Job Execution Web Application” on page 14](#). For information about how to execute a job using a URL, see [“Executing a Job Using Direct URL Access” on page 15](#).

Passing User Input to a Job Using the Query String

Most jobs require information from the client to perform their intended function. This information can be in the form of presentation options for a report, selection criteria for data to be analyzed, names of data tables to be used or created, or an unlimited number of other possibilities.

Input parameters are the most common way to deliver information from a client to a job. They are defined as name/value pairs and appear in your SAS program as global macro variables. The _PROGRAM parameter used in [“Executing a Job Using Direct URL Access” on page 15](#) is an example of a URL parameter.

The simplest way to pass user input to a job is by specifying name/value pairs in the SAS Job Execution Web Application query string. Consider this modification to the web address:

```
http://host:port/SASJobExecution/?_program=/SomeFolder/Simple HTML&myname=John
```

A global macro variable named MYNAME with a value of John is created before the SAS code is executed, and it is available for use in your SAS program. Make the

following change to the code from “Creating a Simple Job That Uses DATA Step Code” on page 47:

```
* Declare input parameter;

%global MYNAME;

* Write the custom HTML to _webout;

data _null_;
file _webout;
put '<html>';
put '<head><title>Hello World! </title></head>';
put '<body>';
put "<h1>Hello %sysfunc(htmlencode(&MYNAME))!</h1>";
put '</body>';
put '</html>';
run;
```

It is good practice to declare input parameters at the beginning of the program so that macro variables resolve. Also, use double quotation marks in the PUT statement to resolve the macro variable.

Executing the job using the previous web address creates the following output:

Hello John!

Passing User Input to a Job Using a Job Definition Parameter

Run the Simple HTML job that you created previously without specifying a value for the MYNAME input parameter in the URL:

```
http://host:port/SASJobExecution/?_program=/SomeFolder/Simple HTML
```

The following output is displayed:

Hello !

Perform the following steps to specify a default value for the MYNAME input parameter by adding a job definition parameter:

- 1 Use the Content page to navigate to the Simple HTML job, select it, and then click  and select **Properties**.

Note: You must have access to developer functionality to modify properties for a job.

- Click **Parameters** and then click **Add a new parameter**. Specify the following properties for the parameter:

Property	Value
Name	<code>myname</code>
Default Value	<code>John</code>
Field Type	<code>Character</code>

- Select **Required** only if you want to ensure that a non-blank value be specified for the parameter at run time.
- Click **Save** to save the parameter.
- Run the job again using the previous URL. The following output is now displayed:

Hello John!

Passing User Input to a Job Using a Job Form

You can use the Content page to create HTML input forms or prompts that accept input from a user and then pass that input to the SAS code. This example uses the Simple HTML job that was created in [“Creating a Simple Job That Uses DATA Step Code” on page 47](#). The example creates an HTML input form to replace the technique of passing input to the SAS program using the query string.

Note: You must have access to developer functionality to create job forms.

When you create a job form with the same name as the job that is located in the same folder as the job, or if the job form is embedded in the job, it can be automatically displayed when the job executes. Create a job parameter named `_ACTION` with the following properties to take advantage of this behavior for an HTML input form:

Property	Value
Name	<code>_ACTION</code>
Default Value	<code>form,execute</code>
Field Type	<code>Character</code>

If an HTML input form exists, then it is displayed. Otherwise, the job executes.

Perform the following steps to create an HTML input form that is embedded in the job:

- 1 Click the Simple HTML job definition that you just created, and then click  and select **New HTML form**.
- 2 Enter the following HTML code into the editor window:

```
<!DOCTYPE html>
<head>
<title>Simple HTML Example</title>
</head>
<body>

<h1>Simple HTML Example</h1>

<form action="/SASJobExecution/" target="_tab">
  <input type="hidden" name="_program" value="$PROGRAM$">
  <input type="hidden" name="_action" value="execute">

  <label>Specify a name for the greeting: </label>
  <input type="text" name="myname" value="World" required>

  <br/>
  <br/>

  <input type="submit" value="Run code">
</form>

</body>
</html>
```

Alternatively, you can use a text editor or web development tool to create the HTML, and then copy and paste it into the editor window.

Always specify `/SASJobExecution/` in the ACTION attribute of the FORM tag to indicate that the form data is submitted to the SAS Job Execution Web Application for processing. You can specify `_tab` in the TARGET attribute to force the output to always appear in a new tab within the application. Omit this attribute if you do not want this behavior.

The first INPUT tag specifies that a non-visual object named `_PROGRAM` has a value of `$PROGRAM$`. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

The second INPUT tag specifies a value of `execute` for the `_ACTION` parameter, which overrides the default value specified in the job parameter and executes the job.

The third INPUT tag prompts the user for the name to use in the greeting. This object is named `MYNAME` and its default value is `World`.

The last INPUT tag displays a button with the label **Run code**.

Click , and then click **Close** to save the HTML input form and close the editor window.

- Click the Simple HTML job to select it, and then click ▷. The following HTML input form appears in the right pane:

Simple HTML Example

Specify a name for the greeting:

- Specify **Jane** as the name for the greeting, and then click **Run code**.

The web browser uses data from all form elements except the submit button to automatically construct a URL similar to the URL in [“Passing User Input to a Job Using the Query String” on page 49](#). The form data is submitted when the button is clicked, resulting in the following URL:

```
http://host:port/SASJobExecution/?_program=/SomeFolder/Simple HTML&myname=Jane
```

The results are displayed in a new browser window or tab, using the updated value of the MYNAME macro variable:

Hello Jane!

Substituting Parameters Into a Job Form

You can use `$param$` in the HTML input form if you want to substitute a value for *param* when the HTML input form is displayed. The value for *param* can be specified in a URL or job definition parameter. The string `$param$` is replaced with the value for the parameter named *param* or removed if the parameter is not found.

Note: Use lowercase letters in the `$param$` string.

In the previous Simple HTML example, the INPUT tag can be changed to

```
<input type="text" name="myname" value="$myname$" required>
```

The value for a job definition or URL parameter called myname is substituted into the form.

You can also use this functionality anywhere in a form to substitute data from a file. For example, using `$/folder1/folder2/file$` includes the data from the specified file. This enables you to save HTML text and incorporate it into multiple job forms.

Creating Simple HTML Output Using ODS

The Output Delivery System (ODS) enables you to generate different types of output from your procedure code. An ODS destination controls the type of output that is generated (HTML, RTF, PDF, and so on). An ODS style controls the appearance of the output.

Many jobs create ODS HTML as their primary type of output. The %JESBEGIN macro can issue an ODS statement in addition to the _WEBOUT fileref. See [“Overview of %JESBEGIN and %JESEND Macros” on page 46](#) for more information. Add the following parameters to the job to use the ODS HTML5 destination and the HTMLBlue style:

Property	Value
Name	<code>_ODSSTYLE</code>
Default Value	<code>HTMLBlue</code>
Field Type	<code>Character</code>

This job uses PROC PRINT to display all of the data in the SASHELP.CLASS data set:

```
* Display the SASHELP.CLASS data set;

title 'Student Data';

proc print data=sashelp.class noobs;
  var name sex age height weight;
run; quit;
```

Here is the output:

Student Data				
Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0

Creating Simple PDF, RTF, or DOCX Output Using ODS

You can make small changes to the sample in [“Creating Simple HTML Output Using ODS” on page 54](#) to create PDF, RTF, or DOCX output. Specify `ods_pdf` for the `_OUTPUT_TYPE` input parameter to create PDF output, `ods_rtf` to create RTF output, and `ods_word` to create DOCX output. The PDF output is rendered by the web browser, and the RTF output is downloaded as a file named `SASResults.rtf` that can be opened using an application such as Microsoft Word. The DOCX output can also be opened using Microsoft Word.

Using Input Parameters with Multiple Values

Parameters with multiple values (or, alternatively, multiple input parameters with the same name) can be useful in some jobs. For example, an HTML input form might contain a multiple selection list box named `COLS` that allows the user to choose which columns of a data set to display. This example shows a parameter with multiple values.

Create a job definition named `Multiple Input Values` with the following parameters:

Name	Default Value	Field Type
_ACTION	form,execute	Character
_OUTPUT_TYPE	ods_html5	Character
_ODSSTYLE	HTMLBlue	Character

Add the following code:

```
<param_list macro definition here>

* Convert the selections to a space-separated list;

%param_list(mvar=cols, outvar=column_list)

title 'Student Data';

proc print data=sashelp.class;
  var &COLUMN_LIST;
run; quit;
```

This example uses a multiple selection list box to choose the columns to display from the SASHELP.CLASS data set. Click the Multiple Input Values job definition that you just created, and then click  and select **New HTML form** to create an HTML input form with the following HTML:

```
<!DOCTYPE html>
<head>
<title>Multiple Input Values Example</title>
</head>
<body>

<h1>Multiple Input Values Example</h1>

<form action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$">
<input type="hidden" name="_action" value="execute">

<div>Use Ctrl+Click to choose columns to display: </div>

<br/>

<select name="cols" multiple required size="5">
  <option value='name'>First Name</option>
  <option value='sex'>Gender</option>
  <option value='age'>Age (y) </option>
  <option value='height'>Height (in) </option>
  <option value='weight'>Weight (lb) </option>
</select>

<br/>
<br/>

<input type="submit" value="Run code">
```

```
</form>

</body>
</html>
```

The MULTIPLE attribute of the SELECT tag indicates that multiple selections are allowed, and the optional SIZE attribute specifies the number of rows to display when the HTML page is rendered. The name of the SAS macro variable COLS is specified in the NAME attribute.

The OPTION tags specify the values sent to the SAS program as well as the values displayed when the HTML page is rendered. This example uses display values that differ from the macro variable values:

Display Value	Macro Variable Value
First Name	name
Gender	sex
Age (y)	age
Height (in)	height
Weight (lb)	weight

Here is a partial view of the rendered HTML file:

Use Ctrl+Click to choose columns to display:

First Name ▲

Gender

Age (y)

Height (in)

Weight (lb) ▼

Run code

If you select only **First Name** and then submit the form, a global macro variable named COLS is created with a value of **name**. If you select **First Name**, **Age (y)**, and **Height (in)** and then submit the form, the following macro variables are created:

Macro Variable Name	Macro Variable Value	Description
COLS	name	Specifies the first value
COLS0	3	Specifies the number of values
COLS1	name	Specifies the first value

Macro Variable Name	Macro Variable Value	Description
COLS2	age	Specifies the second value
COLS3	weight	Specifies the third value
COLS_COUNT	3	Specifies the number of values

Because macro variables cannot hold more than one value, a numeric suffix is added to the parameter name to distinguish between values. The number of values is set in the *param-name0* and *param-name_COUNT* variables. The first value is set in the *param-name1* variable, and so on, as shown in the previous table. Note that the original parameter macro variable (*param-name*) is always set to the first parameter value.

This format is seldom useful in SAS code. For example, the pseudo-array of user selections must be transformed before they can be used in a VAR or SELECT statement. You can use the %PARAM_LIST macro available in [“%PARAM_LIST Macro” on page 187](#) to convert the user selections into a usable format.

The COLS selection list in the HTML input form is used to choose one or more columns in the SASHELP.CLASS data set. The %PARAM_LIST macro takes the individual selections and converts them to a single list that is used in the VAR statement.

For example, if the user selects **First Name**, **Age (y)**, and **Height (in)**, then the macro variable COLUMN_LIST, which was created by the %PARAM_LIST macro, resolves to the following:

```
name age height
```

See [“%PARAM_LIST Macro” on page 187](#) for more information and examples.

Linking One Job to Another (Drill Down)

You might want to display summarized information with the option to click a link to display more detailed data. This is an example of performing drill down. Developing this type of application usually involves at least two different jobs: one job to create the summarized information with the links, and a second job that displays the detailed data related to that link.

This example summarizes sales data in the SASHELP.SHOES data set by sales region, and then uses PROC PRINT to display it with links to the detailed data:

Sales Totals by Region	
Region	Sales
Africa	\$2,342,588
Asia	\$460,231
Canada	\$4,255,712
Central America/Caribbean	\$3,657,753
Eastern Europe	\$2,394,940
Middle East	\$5,631,779
Pacific	\$2,296,794
South America	\$2,434,783
United States	\$5,503,988
Western Europe	\$4,873,000
	\$33,851,566

Detailed sales information is displayed when you click a link. Here is the detailed data for Asia:

Detailed Sales Information for Asia				
Subsidiary	Shoe Style	Sales	Inventory	Returns
Bangkok	Boot	\$1,996	\$9,576	\$80
	Men's Dress	\$3,033	\$20,831	\$52
	Sandal	\$3,230	\$15,087	\$120
	Slipper	\$3,019	\$16,075	\$127
	Women's Casual	\$5,389	\$16,251	\$185
		\$16,667	\$77,820	\$564
Seoul	Boot	\$60,712	\$160,589	\$1,296
	Men's Casual	\$11,754	\$2,176	\$833
	Men's Dress	\$116,333	\$251,803	\$2,443
	Sandal	\$4,978	\$21,483	\$105
	Slipper	\$149,013	\$469,007	\$2,941
	Sport Shoe	\$937	\$455	\$10
	Women's Casual	\$20,448	\$36,576	\$790
	Women's Dress	\$78,234	\$140,628	\$1,891
		\$442,409	\$1,082,717	\$10,309
Tokyo	Sport Shoe	\$1,155	\$15,602	\$22
		\$1,155	\$15,602	\$22
		\$460,231	\$1,176,139	\$10,895

Create a job definition named Drilldown with the following parameters:

Name	Default Value	Field Type
_OUTPUT_TYPE	ods_html5	Character
_ODSSTYLE	HTMLBlue	Character

Add the following code to produce the summarized report with the drill-down links:

```
* Summarize the data;

proc means data=sashelp.shoes sum noprint;
```

```

var sales;
class region;
output out=work.shoes_summary(where=(_type_ eq 1)) sum=sales;
run; quit;

* Define the base URL for the drill-down link;

%let BASE_URL=&_URL.?_program=
    /SomeFolder/Drilldown2&nrstr(&_action)=wait&nrstr(&region=);

* Set the ODS escape character;

ods escapechar='^';

* Add the drill-down links to the summarized data;

data work.shoes_summary;
set work.shoes_summary;

length region_link varchar($1024);

region_link = "^{style [url='&BASE_URL" ||
               urlencode(strip(region)) ||
               "]" ||
               strip(region) ||
               '}';

run;

* Display the summarized data with drill-down links;

title 'Sales Totals by Region';

proc print data=work.shoes_summary noobs label;
var region_link sales;
sum sales;
label region_link = 'Region'
      sales       = 'Sales';
format sales dollar11.;
run; quit;

```

The `BASE_URL` macro variable is used to create a portion of the drill-down link. The `_URL` reserved macro variable ensures that the URL is valid. The job that displays the detail data, `Drilldown2`, is referenced here. This job is created in the next step, and you must specify the full path to that job.

The complete drill-down link text is stored in the `REGION_LINK` variable. ODS inline formatting is used to create the link using the `URL STYLE` attribute. This technique creates links for HTML as well as other ODS output formats. The general syntax of the inline style is as follows:

```
^{style [url='URL-of-second-job']link-text}
```

Here is an example of the value of the `REGION_LINK` variable created in the code:

```
^{style [url='/SASJobExecution/?_program=
    /SomeFolder/Drilldown2&region=Asia']Asia}
```

The value of the sales region of interest is passed to the Drilldown2 job using the REGION input parameter.

The STRIP function removes any trailing blanks in the data value, and the URLENCODE function handles parameter values that need to be encoded:

```
^{\style [url='/SASJobExecution/?_program=
/SomeFolder/Drilldown2&region=Central%20America%2FCaribbean']
Central America/Caribbean}
```

The PRINT procedure displays the summarized data with links to execution of the detail data program.

Next, create a job definition named Drilldown2 that displays the detail data using the following parameters:

Name	Default Value	Field Type
_OUTPUT_TYPE	ods_html5	Character
_ODSSTYLE	HTMLBlue	Character

Add the following code to produce the detailed report for a specified region:

```
* Declare input parameter;

%global REGION;

* Include the value of the REGION input parameter in the report title;

title "Detailed Sales Information for &REGION";

proc report data=sashelp.shoes nowd;
  where (region eq "&REGION");

  column region subsidiary product sales inventory returns;

  define region      / order noprint 'Region';
  define subsidiary  / order        'Subsidiary';
  define product     / order        'Shoe Style';
  define sales       / sum          'Sales';
  define inventory   / sum          'Inventory';
  define returns     / sum          'Returns';

  break after subsidiary / summarize suppress style=header;
  rbreak after          / summarize;
run; quit;
```

Though not required, it is good practice to declare all input parameters using a %GLOBAL statement.

The value of the REGION input parameter, which is passed to the program as a URL parameter from the link in the Drilldown job output, is used in the report title and also to subset the data so that only the detailed information is displayed for the specified region.

Run the Drilldown job, and then click a link for a sales region. The detailed data for the region appears.

Sending ODS Output to an Email Recipient

The following example sends ODS output as the body of an email message.

Create a job definition named Email Report where the `_OUTPUT_TYPE` parameter has a value of `none`. Add the following code:

```
* Close all open destinations;

ods _all_ close;

* ODS output is sent directly to the email recipient;

filename mail email 'email-recipient@email-recipient-domain'
  subject='Your SAS Report' type='text/html';

* The HTML3 destination provides better rendering in some email clients;

ods html3 file=mail style=HTMLBlue;

title 'Student Data';

proc print data=sashelp.class; run; quit;

ods html3 close;

%let _STATUS_MESSAGE=Email sent.;
```

Specify appropriate values for *email-recipient* and *email-recipient-domain*. Use the HTML3 destination because some email clients do not support the HTML created by the HTML4 and HTML5 destinations.

The job does not create visual output displayed by the web browser client. You can use `_STATUS_MESSAGE` to display a message in the web browser.

You might have to specify one or more system options to successfully send email. See [SAS System Options: Reference](#) for more information about email communications system options.

Assigning a Fileref for HTML Output

A `FILENAME` statement is required to define the location of your output. You can choose any valid name for the fileref, but the access method (`FILESRV`) and the `PARENTURI` option should be specified exactly as follows:

```
filename _webout filesrv parenturi="&SYS_JES_JOB_URI"
  name='_webout.htm';
```

Unlike the DISK device type, the FILESRVC access method writes files to the SAS Infrastructure Data Server using the Files service, not to the external file system. The FILESRVC access method creates global macro variables of the form `_FILESRVC_fileref_URI`, where *fileref* is the fileref used in the FILENAME statement. This macro variable provides a relative URL that can be used to reference and retrieve the file using the Files service (for example, `/files/files/74d8179e-e922-4b58-a8fe-0863b2aa3bfc`). See [“Report with Download Links” on page 145](#) for an example that uses this macro variable.

The SYS_JES_JOB_URI macro variable provides a reference to the job execution object. When used with the PARENTURI option, this ensures that the file is associated with the job execution object. All files associated with a job execution object can be displayed by the SAS Job Execution Web Application until the job expires. By default, job output is deleted 30 minutes after it is created. See [“Job Output Expiration” on page 21](#) for more information about how to change the expiration time. See [“Saving Job Output” on page 23](#) for more information about how to save a permanent copy of your job output files.

In most cases, you should use the name `_webout.htm` because the SAS Job Execution Web Application searches the job results object for an entry named `_webout.*` and displays the first result that it finds. This behavior can be altered using the `_RESULTFILE` parameter. See [“Specifying Output Files” on page 18](#) for more information.

Assigning a Fileref for Other Types of Output

The FILENAME statement that supports other types of ODS output is similar to the format used for HTML. The following examples show some common ODS output formats. See [“FILENAME Statement” in SAS Global Statements: Reference](#) for more information about the FILENAME statement.

Use the following format if your web browser supports rendering PDF files:

```
filename _webout filesrv parenturi=&SYS_JES_JOB_URI"
  name='_webout.pdf';
```

Use the following format if your web browser does not support rendering PDF files or if you want the content to be downloaded as a file. Specify the desired file name in the FILENAME attribute of the CONTENTDISP option:

```
filename _webout filesrv parenturi=&SYS_JES_JOB_URI"
  name='_webout.pdf'
  contentdisp=attachment; filename="MyFile.pdf";
```

See [“Report with Download Links” on page 145](#) for an example that uses this format.

Use the following format to download RTF content:

```
filename _webout filesrv parenturi=&SYS_JES_JOB_URI"
  name='_webout.rtf'
  contentdisp=attachment; filename="MyFile.rtf";
```

Most web browsers support rendering XML content:

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.xml';
```

Use the following format to download XML content:

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.xml'
contentdisp='attachment; filename="MyFile.xml"';
```

Use one of the following formats to handle JSON content:

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.json';
```

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.json'
contentdisp='attachment; filename="MyFile.json"';
```

Use the following format if your program uses the tagsets.ExcelXP ODS destination:

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.xml'
contenttype='application/vnd.ms-excel'
contentdisp='attachment; filename="MyFile.xml"';
```

See [“Report with Download Links” on page 145](#) for an example that uses this format.

Use the following format if your program uses the Excel ODS destination:

```
filename _webout filesrvc parenturi="&SYS_JES_JOB_URI"
name='_webout.xlsx'
contenttype='application/vnd.ms-excel'
contentdisp='attachment; filename="MyFile.xlsx"';
```

Assigning a Fileref to Reference Source Code Stored in the SAS Infrastructure Data Server

The following examples show how to reference code that is already stored. If you have a large program or want to share code with other users, you can store source code in the SAS Infrastructure Data Server. An additional benefit is that if you use this code in a scheduled job, the source code can be changed without rescheduling the job.

The following example shows how to format the source code if you want to access the file using a URI:

```
filename mycode filesrvc '/files/files/ID';
%include mycode;
```

The following format shows how to format the source code if you want to access the file by name and folder:

```
filename mycode filesrvc folderpath='/SomeFolder/programs' name='MyProgram.sas';
%include mycode;
```

In both of these examples, mycode is the fileref that you assign. See [“FILESRV Options” in SAS Global Statements: Reference](#) for more information about how to use the FILESRVC access method.

Using Sessions

Overview of Sessions

The web is a stateless environment. A client request to a server does not know about preceding requests. Often, programmers want to carry information from one request to the next. This is known as maintaining state. Starting in the 2025.11 release, sessions provide a convenient way to maintain state across multiple job requests.

A session is the data that is saved from one job execution to the next. The session data consists of macro variables and library members (data sets and catalogs) that the job has explicitly saved. The session data is scoped so that all users have independent sessions.

See [“COMPSRV_SESSION Function” in SAS Functions and CALL Routines: Reference](#) for more information about the syntax for the COMPSRV_SESSION function.

Creating a Session

The job must explicitly create a session in a compute server with the COMPSRV_SESSION function, as follows:

In macro:

```
%let rc=%sysfunc(compsrv_session(create));
```

In the DATA step:

```
rc=compsrv_session('create');
```

Creating a session sets the global macro variables _SESSIONID and _THISSESSION and creates the SAVE session library.

Using the Session

Between calls to the Compute service, the session remains as it was at the end of the last job. This includes all macro variables, data sets, library references, and so on. These items are available to all future jobs that execute in the same session.

Creating a session causes the automatic variables _THISSESSION and _SESSIONID to be set. The value for the _URL macro variable is used by the COMPSRV_SESSION function to create the _THISSESSION macro variable value,

which resolves to `&_URL?_SESSIONID=&_SESSIONID`. Sample values for these variables are as follows:

```
%let rc=%sysfunc(compsrv_session(create));

%put _SESSIONID=&_SESSIONID;
_SESSIONID=7CF645EB-6E23-4853-8042-BBEA7F866B55

%put _THISSESSION=&_THISSESSION;
_THISSESSION=/SASJobExecution?_sessionid=7CF645EB-6E23-4853-8042-BBEA7F866B55
```

These variables can be used to construct URLs or HTML forms that execute another job in the same session. For example:

```
%let rc=%sysfunc(compsrv_session(create));
data _null;
  file _webout;
  put '<HTML>';
  put '<BODY>';
  put '<H1>Session Test Page</H1>';
  /* Link to another job in the same session */
  put '<A HREF="" "&_THISSESSION" "&_PROGRAM=/Test/Test2">Test</A>';
  put '</BODY>';
  put '</HTML>';
run;
```

Deleting the Session

Sessions that are created using the `COMPSRV_SESSION` function must also be explicitly destroyed using the `COMPSRV_SESSION` function, as follows:

In macro:

```
%let rc=%sysfunc(compsrv_session(delete));
```

In the DATA step:

```
rc=compsrv_session('delete');
```

Submitting this code does not immediately delete the session. The session is marked for deletion only at the completion of the job. For this reason, a job cannot delete a session and create a new session. When the session is deleted, so are the associated `SAVE` library and all associated macro values. Any further attempts to access the session result in an invalid or expired session error.

For more information about sessions on the compute server and when they are terminated, see [“Configure How a Task Is Executed” in SAS Viya Platform: Programming Run-Time Servers](#).

Using Sessions in a Sample Web Application

Overview of the Sample Web Application

The following sample web application demonstrates some of the features of sessions. The sample application is an online library. Users can log on, select one or more items to check out of the library, and request by email that the selected items be delivered. The sample code shows how to create a session and then create, modify, and view macro variables and data sets in that session.

Note: The sample web application includes several jobs. Submit the Main Aisle job to access all features; the other jobs are supporting components.

Sample Data

This sample requires a LIB_INVENTORY data set in the SAMPDAT library. You can create the data set using the following code:

```
libname SAMPDAT 'library-path';
data SAMPDAT.LIB_INVENTORY;
length type $10 desc $80;
input refno 1-5 type 7-14 desc 16-80;
datalines;
17834 BOOK      SAS Viya Platform Fundamentals
32345 BOOK      Getting Started with Machine Learning
52323 BOOK      SAS Enterprise Miner: High-Performance Procedures
54337 BOOK      SAS Visual Analytics: Designing Reports
35424 BOOK      SAS Visual Analytics: Viewing Reports
93313 AUDIO     The Zen of SAS
34222 VIDEO     Getting Started with SAS Studio
34223 VIDEO     SAS Model Manager Overview Demo
34224 VIDEO     Demo: Develop a Web Application to Extract SAS Data
70001 HARDWARE  Microphone - Model 5153
70002 HARDWARE  Video Projector - Model 79F15
;;;
;
```

Main Aisle Job

The main aisle page is generated by the Main Aisle job. This page acts as a welcome page to new users. A session is created the first time a user executes this job.

```
/* Main Aisle of the Online Library */
data _null_;
file _webout;
if libref('SAVE') ne 0 then rc = compsrv_session('create');
put '<HTML>';
put '<HEAD><TITLE>Online Library Main Aisle</TITLE></HEAD>';
put;
put '<BODY>';
```

```

put '<H1>Online Library Main Aisle</H1>';
put;
put 'Select one of the following areas of the library:';
put '<UL>';
length hrefroot $400;
hrefroot = symget('_THISSESSION') || '&_PROGRAM=/SomeFolder/';
put '<LI><A HREF="' hrefroot + (-1) 'Aisles&type=Book">Book
    aisle</A></LI>';
put '<LI><A HREF="' hrefroot + (-1) 'Aisles&type=Video">Video
    aisle</A></LI>';
put '<LI><A HREF="' hrefroot + (-1) 'Aisles&type=Audio">Audio
    aisle</A></LI>';
put '<LI><A HREF="' hrefroot + (-1) 'Aisles&type=Hardware">Hardware
    aisle</A></LI>';
put '<LI><A HREF="' hrefroot + (-1) 'Shopping Cart">View my shopping
    cart</A></LI>';
put '<LI><A HREF="' hrefroot + (-1) 'Logout">Logout</A></LI>';
put '</UL>';
put '</BODY>';
put '</HTML>';
run;

```

The main aisle page consists of a list of links to specific sections of the Online Library.



Each link in this page is built using the `_THISSESSION` macro variable. This variable includes both the `_URL` value pointing back to the SAS Job Execution Web Application and the `_SESSIONID` value that identifies the session.

Aisles Job

The library is divided into aisles for different categories of library items. The pages for each aisle are generated by one shared Aisles job. The job accepts a `TYPE` input variable that determines which items to display.

```

/* Aisles - List items in a specified aisle.
The aisle is specified by the TYPE variable. */
libname SAMPDAT 'library-path';

/* Build a temporary data set that contains the selected type,
and add links for selecting and adding items to the shopping cart. */
data tempolist;
if libref('SAVE') ne 0 then rc = compsrv_session('create');
set SAMPDAT.LIB_INVENTORY;

```

```

where type="%UPCASE(&type)";
length select $200;
select = '<A HREF="' || symget("_THISSESSION") ||
        '&_program=/SomeFolder/Add+Item&REFNO=' ||
        trim(left(refno)) || '&TYPE=' || "&TYPE" || '">Add to cart</A>';
run;
ods html body=_webout(nobot) rs=none;
title Welcome to the &type Aisle;
proc print data=templist noobs label;
var refno desc select;
label refno='RefNo' desc='Description' select='Select';
run;
ods html close;
data _null_;
file _webout;
put '<P>';
put 'Return to <A HREF="' "&_THISSESSION"
    '&_PROGRAM=/SomeFolder/Main+Aisle'">main aisle</A><BR>';
put 'View my <A HREF="' "&_THISSESSION"
    '&_PROGRAM=/SomeFolder/Shopping+Cart'">shopping cart</A><BR>';
put '</BODY>';
put '</HTML>';
run;

```

The job selects a subset of the LIB_INVENTORY data set by using a WHERE clause, and then uses PROC PRINT to create an HTML table. A temporary data set is created. This data set contains the selected items that users can use to add items. An additional column is generated from the LIB_INVENTORY data set that has an HTML link that users can use to add the item to their shopping cart.

In this job, both ODS and a DATA step are used to generate HTML. The ODS HTML statement includes the NOBOT option that indicates that more HTML is appended after the ODS HTML CLOSE statement. The navigation links are then added using a DATA step. The following display shows the contents of the Book aisle.

Welcome to the Book Aisle		
RefNo	Description	Select
17834	SAS Viya Platform Fundamentals	Add to cart
32345	Getting Started with Machine Learning	Add to cart
52323	SAS Enterprise Miner: High-Performance Procedures	Add to cart
54337	SAS Visual Analytics: Designing Reports	Add to cart
35424	SAS Visual Analytics: Viewing Reports	Add to cart

Return to [main aisle](#)
View my [shopping cart](#)

Add Item Job

The Add Item job is run when the user clicks the **Add to cart** link in the aisle item table. The specified item is copied from the LIB_INVENTORY data set to a shopping cart data set in the session library (SAVE.CART). The session and the data set

remain accessible to all programs in the same session until the session is deleted or it times out.

```
/* Add Item - Add a selected item to the shopping cart.
This job uses REFNO and TYPE input variables to identify the item. */
libname SAMPDAT 'library-path';
/* Perform REFNO and TYPE verification here. */
/* Append the selected item. */
proc append base=SAVE.CART data=SAMPDAT.LIB_INVENTORY;
where refno=&refno;
run;
/* Print the page. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>Selected Item Added to Shopping Cart</TITLE>
    </HEAD>';
put '<BODY>';
put "&<H1>Item &refno Added</H1>";
put 'Return to <A HREF="' & _THISSESSION"
    '&_PROGRAM=/SomeFolder/Aisles'&TYPE=' '&TYPE" '>' &TYPE aisle
    </A><BR>';
put 'Return to <A HREF="' & _THISSESSION"
    '&_PROGRAM=/SomeFolder/Main+Aisle'&TYPE=' '>main aisle</A><BR>';
put 'View my <A HREF="' & _THISSESSION"
    '&_PROGRAM=/SomeFolder/Shopping+Cart'&TYPE=' '>shopping cart</A><BR>';
put '</BODY>';
put '</HTML>';
run;
```

The program prints an information page that has navigation links.

Item 17834 Added

Return to [Book aisle](#)
 Return to [main aisle](#)
 View my [shopping cart](#)

Shopping Cart Job

The Shopping Cart job displays the contents of the shopping cart.

```
/* Shopping Cart - Display contents of the shopping cart
* (SAVE.CART data set). */
%macro lib_cart;
%let CART=%sysfunc(exist(SAVE.CART));
%if &CART %then %do;
/* This program could use the same technique as the LIB_AISLE program
in order to add a link to each line of the table that removes items
from the shopping cart. */
/* Print the CART contents. */
ods html body=_webout(nobot) rs=none;
title Your Selected Items;
proc print data=SAVE.CART noobs label;
```

```

var refno desc;
label refno='RefNo' desc='Description';
run;
ods html close;
%end;
%else %do;
/* No items in the cart. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>No items selected</TITLE></HEAD>';
put '<BODY>';
put '<H1>No Items Selected</H1>';
put;
run;
%end;
/* Print navigation links. */
data _null_;
file _webout;
put '<P>';
if &CART then do;
put '<FORM ACTION="' &_url" '>';
put '<INPUT TYPE="HIDDEN" NAME="_program"
    VALUE="/SomeFolder/Logout">';
put '<INPUT TYPE="HIDDEN" NAME="_sessionid"
    VALUE="' &_sessionid" '>';
put '<INPUT TYPE="HIDDEN" NAME="CHECKOUT" VALUE="YES">';
put '<INPUT TYPE="SUBMIT" VALUE="Request these items">';
put '</FORM><P>';
end;
put 'Return to <A HREF="' &_THISSESSION"
    '&_PROGRAM=/SomeFolder/Main+Aisle'">main aisle</A><BR>';
put '<A HREF="' &_THISSESSION"
    '&_PROGRAM=/SomeFolder/Logout' '&CHECKOUT=NO">Logout</A><BR>';
put '</BODY>';
put '</HTML>';
run;
%mend;
%lib_cart;

```

The contents of the shopping cart are displayed using a PROC PRINT statement. The page also includes a request button and navigation links. The request button is part of an HTML form. In order to connect to the same session, include the `_SESSIONID` value in addition to the `_PROGRAM` value. These values are usually specified as hidden fields. This program also has a hidden `CHECKOUT` field that is initialized to `YES` in order to indicate that the user is requesting the items in the cart.

The program prints a page that contains the contents of the shopping cart.

Your Selected Items

RefNo	Description
17834	SAS Viya Platform Fundamentals

[Return to main aisle](#)
[Logout](#)

Logout Job

The Logout job checks the user out of the Online Library. If the CHECKOUT input variable is YES, then all of the items in the user's shopping cart are requested through email.

```

/* Logout - logout of Online Library application.
Send email to the library@abc.com account with requested item if
CHECKOUT=YES is specified. */
%macro lib_logout;
%global CHECKOUT;
    /* Define CHECKOUT in case it was not input. */
    %if %UPCASE(&CHECKOUT) eq YES %then %do;
    /* Checkout - send an email request to the library. See the
documentation for the email access method on your platform
for more information on the required options. */
    /* ***** disabled for demo ***** */
    filename RQST EMAIL 'library@abc.com'
    SUBJECT='Online Library Request for &_username';
    ods listing body=RQST;
    title Request for &_username;
    proc print data=SAVE.CART label;
    var refno type desc;
    label refno='RefNo' type='Type'
    desc='Description';
    run;
    ods listing close;
    * ***** */
    data _null_;
    file _webout;
    put '<HTML>';
    put '<HEAD><TITLE>Library Checkout</TITLE></HEAD>';
    put '<BODY>';
    put '<H1>Library Checkout</H1>';
    put;
    put 'The items in your shopping cart have been requested.';
    put '<P>Requested items will normally arrive via interoffice';
    put 'mail by the following day. Thank you for using the Online
Library.';
    put '<P><A HREF="' '&_URL"
        '?_PROGRAM=/SomeFolder/Main+Aisle">Click here</A>';
    put 'to re-enter the application.';
    put '</BODY>';
    put '</HTML>';

```

```

run;
%end;
%else %do;
/* Logout without requesting anything. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>Logout</TITLE></HEAD>';
put '<BODY>';
put '<H1>Library Logout</H1>';
put;
put '<P>Thank you for using the Online Library.';
put '<P><A HREF="' '&_URL"
    '?_PROGRAM=/SomeFolder/Main+Aisle">Click here</A>';
put 'to re-enter the application.';
put '</BODY>';
put '</HTML>';
run;
%end;
%mend;
%lib_logout;
/* User is finished - delete the session. */
%let rc=%sysfunc(compsrv_session(delete));

```

An information page is displayed if the user chooses to request the shopping cart items.

Library Checkout

The items in your shopping cart have been requested.

Requested items will normally arrive via interoffice mail by the following day. Thank you for using the Online Library.

[Click here](#) to re-enter the application.

A logoff screen is displayed if the user selects the Logout link.

Library Logout

Thank you for using the Online Library.

[Click here](#) to re-enter the application.

Working with Dynamic Data in Prompts and HTML Forms

You can use dynamic data in prompt pages and HTML forms. The following XML is an example of how you can use dynamic data in prompt pages:

```
<DataSource active="true" name="DataSource1" defaultValue="prmtplib.mytable">
```

The following HTML is an example of how you can use dynamic data in HTML forms:

```
<select name="myparam" id="myparam" data-colname="mycol" data-library="prmtplib" data-table="mytable">
```

When you use this feature, the library must be assigned to the compute context that is associated with the job. You can determine the compute context for the job by accessing the **General** tab in the Properties window for the job definition.

In SAS Environment Manager, click  to open the Contexts page. For **View**, select **Compute contexts**. Select the compute context you want to edit and click . In the Edit Compute Context window, click the **Advanced** tab. You can enter each autoexec statement on a new line.

The following LIBNAME statement is an example of how to specify a SAS data set:

```
libname prmtplib 'path-to-disk-location';
```

The following LIBNAME statement is an example of how to specify a CAS table:

```
libname prmtplib cas caslib=CASLibrary;
```

See [“Dynamic Prompts Using CARS Data” on page 109](#) for a larger example of how to use dynamic prompts in a job. For information about creating HTML forms with dynamic data, see [“Creating HTML Forms with Dynamic Data” on page 75](#).

Creating HTML Forms with Dynamic Data

Overview

A JavaScript library has been developed that enables you to obtain data from SAS libraries and tables — referred to as *dynamic data* — when constructing HTML input forms. The library and table data are retrieved using a compute context name to connect to a compute server.

The following JavaScript library must be included in the form along with the installed JQuery library:

```
<script src="/SASJobExecution/resources/sap/ui/thirdparty/jquery.js"></script>
```

```
<script src="/SASJobExecution/resources/dynamic.min.js"></script>
```

The standard HTML input and select components have been augmented with attributes that define which data is to be included and the way it is to be presented. The custom DATA-XXX attributes are described in [“Attributes” on page 83](#). Any standard HTML attributes including styling can also be used with the components.

Note: Before you create forms that use dynamic data, see [“Working with Dynamic Data in Prompts and HTML Forms” on page 74](#) for information about how to assign a library to the compute context that is associated with the job.

Initializing the Dynamic Library

The dynamic library must be initialized when the form is loaded. You can do this using an HTML BODY tag:

```
<body onload="SASDynamic.init()">
```

This tag calls the library initialization function. An optional parameter can be passed to the function indicating which compute context is to be used for retrieval. If not specified, the default compute context for the session is used. Because this is a SAS Job Execution Web Application job form, you can use a parameter substitution value for the `_contextName` run-time parameter:

```
<body onload="SASDynamic.init('$_contextname$') ">
```

For long forms, you can keep the form hidden until all of the components are ready:

```
<body onload="SASDynamic.init()" style="visibility:hidden;">
```

If there are any problems with the initialization or data retrieval, scan the browser Developer Tools (F12) console for any errors.

Selecting Libraries and Tables

Lists can be constructed for all available libraries and tables within a library. You can use the following code to construct a library selector:

```
<label for="libs">Choose a library:</label>

<select name="libs" id="libs" data-library="*" data-default="SASHELP">

</select>
```

This code generates the following output:

Choose a library:

The asterisk (*) is used as a selection value to retrieve all available libraries. The DATA-DEFAULT attribute indicates the default selection.

To choose a table for the given selection, you can construct another selection list to obtain the available table values:

```
<label for="tables">Choose a table:</label>

<select name="tables" id="tables" data-table="*" data-uses="libs" >

</select>
```

This code generates the following output:

Choose a table:

The DATA-USES attribute indicates that the table retrieval depends on the value of the component with an ID equal to `libs`. As the library selection is changed, the list of tables is updated.

Retrieving Table Column Data

The main function of the JavaScript library is to construct lists with data retrieved from library tables. A standard HTML SELECT component can be used to provide a single-selection list.

For example, to retrieve the Make column from the CARS table in the SASHELP library, you can include the following entry:

```
<label for="cars">Choose a car:</label>
```

```
<select name="cars" id="cars" data-colname="make" data-library="SASHELP" data-table="CARS">
</select>
```

This code generates the following output:

Choose a car:

This code produces an HTML selection list that contains the table data. A user-selected value is sent on form submission and is provided to the SAS program in a CARS macro.

Using Cascading Fields

The use of one dynamic data field to provide values to be used in the data retrieval for another field is called cascading fields. The DATA-USES attribute specifies which other fields are needed to query the specified table for values.

For example, you can add a new selection list to the form to indicate that an additional field requires the car data that was defined previously:

```
<label for="type">Choose a type:</label>

<select name="type" id="type" data-colname="type" data-library="SASHELP"
      data-table="CARS" data-uses="cars">

</select>
```

This code generates the following output:

Choose a car: Choose a type:

The **Choose a type** field uses whatever value is selected in the **Choose a car** field to retrieve data for the selection list. The values in the **Choose a type** list change whenever a different value is selected for the **Choose a car** field.

You can use multiple fields to construct a query to retrieve data for a field. Another field could be added to the above fields that uses both the **Choose a car** and **Choose a type** fields. The DATA-USES attribute contains the field IDs separated by commas.

```
<label for="model">Choose a model:</label>

<select name="model" id="model" data-colname="model" data-library="SASHELP"
      data-table="CARS" data-uses="cars, type" >

</select>
```

This code generates the following output:

Choose a car: Choose a type: Choose a model:

The **Choose a model** field contains data based on the selections for the **Choose a car** and **Choose a type** fields. The entire form might look like the following code:

```
<html>
<head>
<script src="/SASJobExecution/resources/sap/ui/thirdparty/jquery.js"></script>
<script src="/SASJobExecution/resources/dynamic.min.js"></script>

</head>
<body onload="SASDynamic.init()" >
  <form action="/SASJobExecution/">
    <input type="hidden" name="_program" value="$PROGRAM$"/>
    <input type="hidden" name="_action" value="execute"/>

    <label for="cars">Choose a car make:</label>
    <select name="cars" id="cars" data-colname="make"
      data-library="SASHELP" data-table="CARS" >
    </select>

    <label for="type">Choose a type:</label>
    <select name="type" id="type" data-colname="type"
      data-library="SASHELP" data-table="CARS" data-uses="cars">
    </select>

    <label for="model">Choose a model:</label>
    <select name="model" id="model" data-colname="model"
      data-library="SASHELP" data-table="CARS" data-uses="cars, type" >
    </select>
    <br/>
    <input type="submit" value="Run program"/>
  </form>
</body>
</html>
```

Using Multiple Selections

The selection list can be configured to allow multiple selections by adding the standard HTML attribute `multiple` to the component, as follows:

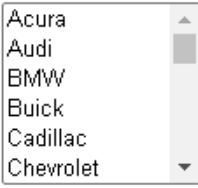
```
<label for="cars">Choose a car:</label>

<select name="cars" id="cars" data-colname="make" multiple size="6"
  data-library="SASHELP" data-table="CARS" >

</select>
```

This code generates the following output:

Choose a car:



This code also enables you to use multiple values in a column query if another field is dependent on this field. The standard attribute size indicates the number of rows to display.

Using a Combo Box

A combo box selection list can be created for data retrieved from a table. An HTML INPUT tag is used to define the component with an empty DATALIST component added to hold the data. As characters are entered in the input field, the matching list items are displayed for selection. Values that are not included in the list can also be entered in the field.

```
<label for="cars">Choose a car:</label>

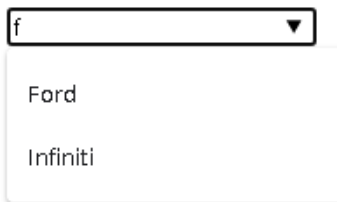
<input type="text" id="cars" name="cars"
       value="" list="makelist" data-colname="make"
       data-library="SASHELP" data-table="CARS">

<datalist id="makelist">

</datalist>
```

This code generates the following output:

Choose a car:



Using a Dual Selector

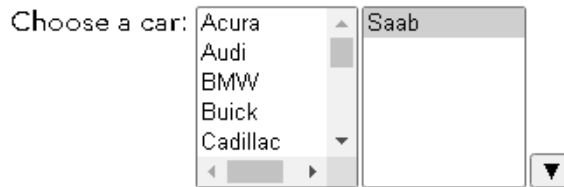
You can define a dual selector list to contain dynamic data. This type of list consists of two panels. The left panel contains the retrieved data and the right panel contains the user selections. The DATA-DUAL attribute indicates that the component is a dual selector.

```
<label for="cars">Choose a car:</label>
```

```
<select name="cars" id="cars" data-colname="make"
      data-library="SASHELP" data-dual="true" data-table="CARS"
      data-default="Saab" size="5" style="width:7em;overflow:auto;">

</select>
```

This code generates the following output:



This example illustrates a dual selector with a default value specified. As items are selected from the left panel, they are disabled and added to the right panel. Selecting values in the right panel removes them and re-enables them in left panel. A standard HTML style attribute is used to set the panel widths and to indicate that a horizontal scroll bar can be added if needed. If the dual selector is specified in another component's DATA-USES attribute, then the down-arrow apply button is shown to allow the selections to be used in the next data retrieval.

Using Paging

If the number of items in a selection list exceeds the value for the DATA-THRESHOLD attribute, then paging buttons can be added to a component. The buttons retrieve the next or previous threshold number of items. Set the DATA-PAGING attribute to `true` to display forward and backward paging buttons. If this attribute is set to `auto`, then the paging buttons are displayed only if the amount of data available exceeds the threshold value.

```
<label for="cars">Choose a car:</label>

<select name="cars" id="cars" data-colname="make" data-threshold="30"
      data-library="SASHELP" data-table="CARS" data-paging="true"
      style="width:7em">

</select>
```

This code generates the following output:



For dual selectors, the paging buttons appear between the two panels:



Filtering

Items in the selection list can be limited by adding a filter input area. The DATA-FILTER attribute adds a filter area over the selection. Characters that you enter in the filter change the list to display only values that contain the case-insensitive filter characters.

```
<label for="cars">Choose a car:</label>

<select name="cars" id="cars" data-colname="make"
        data-library="SASHELP" data-table="CARS"
        data-dual=true required style="width:8em;" data-filter="true">

</select>
```

This code generates the following output:

Choose a car: 

Acura	
Audi	
BMW	
Buick	

Entering "an" in the filter displays only the values that contain those characters:

Choose a car:

Land Rover	
Nissan	

Entering Values

A list can be created from user-entered values using an HTML INPUT tag and the DATA-DUAL attribute. Values can be entered in the input area and added to the list in the right panel using the plus (+) button. Selecting a value in the right panel removes it from the list.

```
<label for="cars">Enter a car:</label>

<input type="text" name="cars" id="cars"
        data-dual="true" style="width:8em" placeholder="Enter values"/>
```

This code generates the following output:

Enter a car:

The standard HTML placeholder attribute can be used to add text to the input area.

Visibility

HTML areas can be hidden and made visible based on the selected value of a component using the DATA-VISIBLE attribute. The DATA-VISIBLE attribute has three parts:

"id, comparison, value"

id specifies the component ID to compare

comparison must be **eq** or **ne**

value specifies the string to compare

Typically, the DATA-VISIBLE attribute is added to an HTML DIV tag:

```
<div data-visible="cars,eq,Ford">
  <label for="type">Choose a type:</label>
  <select name="type" id="type" data-colname="type"
    data-library="SASHELP" data-table="CARS" data-uses="cars">
  </select>
</div>
```

This code displays all of the HTML inside the DIV section when the component with ID equal to **cars** contains the selected value **Ford**.

You can use an empty string to match when any value is selected:

```
<div data-visible='type,ne,""'>
  <label for="model">Choose a model:</label>
  <select name="model" id="model" data-colname="model"
    data-library="SASHELP" data-table="CARS" data-uses="cars, type">
  </select>
</div>
```

Enablement

HTML areas can be enabled and disabled based on the selected value of a component using the DATA-ENABLE attribute. The DATA-ENABLE attribute has three parts:

"id, comparison, value"

id specifies the component ID to compare

comparison must be **eq** or **ne**
value specifies the string to compare

Typically, the DATA-ENABLE attribute is added to an HTML DIV tag:

```
<div data-visible="cars,eq,Ford">
  <label for="type">Choose a type:</label>
  <select name="type" id="type" data-colname="type"
    data-library="SASHELP" data-table="CARS" data-uses="cars">
  </select>
</div>
```

This code enables all of the HTML inside the DIV section when the component with ID equal to **cars** contains the selected value **Ford**.

You can use an empty string to match when any value is selected:

```
<div data-enable='type,ne,"">
  <label for="model">Choose a model:</label>
  <select name="model" id="model" data-colname="model"
    data-library="SASHELP" data-table="CARS" data-uses="cars, type">
  </select>
</div>
```

Attributes

Various attributes are available to control the retrieval and display of dynamic data. The format for the attributes is **DATA-XXX = "value"**.

Table 3.1 Dynamic Data Attributes

Attribute	Value	Description
DATA-CALLBACK	count, ID	Function to call when data is retrieved, where count returns the number of values retrieved ID returns the prompt element ID
DATA-COLNAME	Column name	Table column name to use for data retrieval
DATA-DEFAULT	def1, def2	Comma-separated list of selection list default values. The syntax for a default value is <i>value:display</i> , where <i>value</i> matches a table data value and <i>display</i> is an optional string to use as a display value when one is not available from the table.

Attribute	Value	Description
DATA-DISTINCT	true or false	Return only distinct values for data retrieval
DATA-DUAL	true or false	Display list as a dual selector
DATA-ENABLE	ID, comparison, string	Enable or disable HTML based on comparison, where ID specifies the component ID to compare comparison must be eq or ne value specifies the string to compare
DATA-FILTER	true or false	Display data filter input area
DATA-FORMAT	Format	Format value used to display data (an empty string ("") means that the data is unformatted)
DATA-LABELCOLNAME	Column name	Column to use for displayed values
DATA-LIBRARY	Library name	Library name to use for retrieval (use an asterisk (*) to get the library list)
DATA-PAGING	true , false , or auto	Display paging buttons. Specify auto to display only buttons when the amount of data available is greater than the threshold value. For single-selection components with paging buttons, the default value is not invoked until the corresponding page is displayed. For dual selectors with paging buttons, the display value might need to be specified if the default value is not in the first page.
DATA-SORTBY	value or label	Data retrieval sort indicator (the default is value)
DATA-SORTDIRECTION	none , ascending , or descending	Data retrieval sort direction (the default is none)
DATA-TABLE	Table name	Table name to use for retrieval (use an asterisk (*) to get the table list)

Attribute	Value	Description
DATA-THRESHOLD	Value	Max number to retrieve (the default is 1000)
DATA-USES	id1, id2	HTML component IDs used for data retrieval
DATA-VISIBLE	ID, comparison, string	Change visibility for HTML based on comparison, where ID specifies the component ID to compare comparison must be eq or ne value specifies the string to compare
DATA-WHERECOLNAME	Column name	Alternate column to use for data retrieval

Advanced Programming

Sending JSON Data to a Job Using an Input Parameter

Data in JSON format can be sent to a job as an input parameter if the data contains fewer than 32,767 bytes. Data that is sent in this way can be accessed as a macro variable in the SAS job. In the following example, the MYJSON parameter contains JSON data that was sent to the JSON1 job. Note that %7B and %7D represent the URL-encoded values for the left and right brace characters, { and }, respectively:

```
http://server:port/SASJobExecution/?_program=/Test/json1&myjson=%7B"aaa": "AAA",
"bbb":222, "ccc":false%7D
```

The SAS program for the JSON1 job uses the JSON LIBNAME engine to read the JSON data from the input parameter and then to convert it to SAS data sets:

```
* Declare input parameter;

%global MYJSON;

* Copy the JSON data from input parameter to a file;
```

```

filename indata temp;

data _null_;
file indata;
length str $32767;
str = resolve(symget('myjson'));
put str;
run;

* Use the JSON engine to provide read-only sequential access to JSON data;

libname indata json;

title 'ALLDATA Table from JSON Input';
proc print data=indata.alldata; run; quit;

title 'ROOT Table from JSON Input';
proc print data=indata.root; run; quit;

```

The PROC PRINT output is shown in the following figure:

ALLDATA Table from JSON Input				
Obs	P	P1	V	Value
1	1	aaa	1	AAA
2	1	bbb	1	222
3	1	ccc	1	false

ROOT Table from JSON Input				
Obs	ordinal_root	aaa	bbb	ccc
1	1	AAA	222	0

Sending JSON Data to a Job by Uploading a File

JSON data that exceeds 32,767 bytes can be sent to the job as a file. See [“Upload a File” on page 172](#) for information about how to upload a file. Use the following SAS program to access the uploaded data:

```

* Reference the uploaded JSON data;

filename indata filesrv "&_WEBIN_FILEURI";

* Use the JSON engine to provide read-only sequential access to JSON data;

libname indata json;

title 'ALLDATA Table from JSON Input';
proc print data=indata.alldata; run; quit;

```

```
title 'ROOT Table from JSON Input';
proc print data=indata.root; run; quit;
```

The output is shown in [“Sending JSON Data to a Job Using an Input Parameter” on page 85](#).

Saving Uploaded Files to a Folder

After you have uploaded files, you can use the following steps to save the uploaded files to a folder:

- 1 Create a fileref for the uploaded file using the following code:

```
filename upload filesrvc parenturi="%SYS_JES_JOB_URI"
name="_WEBIN_FILENAME"
contenttype="_WEBIN_CONTENT_TYPE"
recfm=n;
```

- 2 Create a fileref for the saved file using the following code:

```
filename save filesrvc folderpath='/Public'
name="Saved_&_WEBIN_FILENAME"
contenttype="_WEBIN_CONTENT_TYPE"
recfm=n;
```

- 3 Make a copy of the uploaded file using the following code:

```
data _null_;
length rc sysrc 8 sysmsg $1024;
rc = fcopy('upload', 'save');
sysrc=sysrc(); sysmsg=sysmsg();
***TODO: Add error checking;
run;
```

See [“Upload a File” on page 172](#) for information about how to upload a file.

Executing a Job Using JavaScript - Sending Small Data to the Job

You can use this technique to control how job output is handled. The following example uses the FormData JavaScript object and strings to send input parameters to the previous JSON1 job. This is another way to send JSON data containing less than 32,767 bytes to a job.

The JavaScript code in the following HTML input form dynamically creates a form with parameters, submits the form using the POST method, and then displays the output in a DIV element:

```
<!DOCTYPE html>
<html>
```

```

<head>
<title>JavaScript Job Execution</title>

<script>
function submitForm() {
    var formData = new FormData();
    // Your small JSON object here
    var json = {aaa:"AAA", bbb:222, ccc:false};
    // Create the input parameter for the JSON data
    formData.append("myjson", JSON.stringify(json));
    // Create other input parameters
    formData.append("_program", "/Folder/json1");
    formData.append("_action", "execute");
    formData.append("_csrf" , "$CSRF$");
    // Create the request object
    var request = new XMLHttpRequest();
    request.addEventListener("error", function(event) {
        alert("Something went wrong.");
    });
    request.onreadystatechange = function() {
        if (this.readyState == 4) {
            if (this.status == 200) {
                // Display the results in the DIV
                document.getElementById("JobResults").innerHTML = this.responseText;
            }
            else {
                document.getElementById("JobResults").innerHTML = "Status: " + this.status;
            }
        }
    };
    request.open("post", "/SASJobExecution/");
    // Submit the form
    request.send(formData);
    // Display a temporary message in the DIV
    document.getElementById("JobResults").innerHTML = "Please wait ...";
}
</script>

</head>
<body>

<!-- Other content of your web application here -->

<div id="JobResults"></div>

<script>submitForm();</script>

<!-- Other content of your web application here -->

</body>
</html>

```

When you make a POST request, you must specify the `_csrf` input parameter exactly as shown. This tag ensures that the request is considered non-malicious by sending a Cross-Site Request Forgery token to the server.

See [“Sending JSON Data to a Job Using an Input Parameter” on page 85](#) for the results and the SAS code used to process the data.

Executing a Job Using JavaScript - Sending Large Data to the Job

The following example is like the previous one, except that more than 32,767 bytes of data can be sent to the job. This is accomplished using the JavaScript Blob object in the HTML input form:

```
<!DOCTYPE html>
<html>

<head>
<title>JavaScript Job Execution</title>

<script>
function submitForm() {
    var formData = new FormData();
    // Your large JSON object here
    var json = {aaa:"AAA", bbb:222, ccc:false};
    var blob = new Blob([JSON.stringify(json)], {type : 'application/json'});
    // Create the input parameter for the JSON data
    formData.append("myjsonfile", blob);
    // Create other input parameters
    formData.append("_program", "/Folder/json2");
    formData.append("_action", "execute");
    formData.append("_csrf" , "$CSRF$");
    // Create the request object
    var request = new XMLHttpRequest();
    request.addEventListener("error", function(event) {
        alert("Something went wrong.");
    });
    request.onreadystatechange = function() {
        if (this.readyState == 4) {
            if (this.status == 200) {
                // Display the results in the DIV
                document.getElementById("JobResults").innerHTML = this.responseText;
            }
            else {
                document.getElementById("JobResults").innerHTML = "Status: " + this.status;
            }
        }
    };
    request.open("post", "/SASJobExecution/");
    // Submit the form
    request.send(formData);
    // Display a temporary message in the DIV
    document.getElementById("JobResults").innerHTML = "Please wait ...";
}
</script>
```

```

</head>
<body>

<!-- Other content of your web application here -->

<div id="JobResults"></div>

<script>submitForm();</script>

<!-- Other content of your web application here -->

</body>
</html>

```

The JSON data is uploaded as a file using the technique in [“Sending JSON Data to a Job by Uploading a File” on page 86](#). See that section for the results and the SAS code used to process the data.

Executing a Job Using JavaScript - Sending Non-form Data to the Job

The following example shows how to send non-form data to the job in the body of a request. The Content-type header must be set to a non-form type.

```

<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Job Execution</title>
    <script>
      function submitForm() {

        // Create the request object
        var request = new XMLHttpRequest();
        request.addEventListener("error", function(event) {
          alert("Something went wrong.");
        });
        request.onreadystatechange = function() {
          if (this.readyState === 4) {
            if (this.status === 200 || this.status === 202) {
              // Display the results in the DIV
              document.getElementById("JobResults").innerHTML = this.responseText;
            }
            else {
              document.getElementById("JobResults").innerHTML = "Status: " + this.status;
            }
          }
        };
        request.open("post", "/SASJobExecution/?_action=execute&_program=$PROGRAM$");
        request.setRequestHeader('Content-Type', 'application/json');
        request.setRequestHeader("X-CSRF-TOKEN" , "$CSRF$");

        // Submit the data

```

```

var body = JSON.stringify({abc: "abc",def: "def",xyz:"xyz"});
request.send(body);

// Display a temporary message in the DIV
document.getElementById("JobResults").innerHTML = "Please wait ...";
}
</script>
</head>
<body>
  <!-- Other content of your web application here -->
  <div id="JobResults"></div>
  <script>submitForm();</script>
  <!-- Other content of your web application here -->
</body>
</html>

```

When you make a POST request, you must specify the `_CSRF` input parameter exactly as shown. This tag ensures that the request is considered non-malicious by sending a Cross-Site Request Forgery token to the server.

The data is uploaded as a file using the technique in [“Sending JSON Data to a Job by Uploading a File” on page 86](#). See that section for the results and the SAS code used to process the data.

Returning JSON Data from a Job

The Simple JSON sample (see [“Simple JSON” on page 160](#)) displays the SASHELP.CLASS data set in JSON format. You can use JavaScript to execute the job and then post-process the JSON data instead of displaying it. For example, you might want to use the returned JSON data with a JavaScript object, such as a grid or a chart object.

The following HTML input form executes the Simple JSON sample and then stores the returned JSON in the `JSONString` variable:

```

<!DOCTYPE html>
<html>

<head>
<title>JavaScript Job Execution</title>

<script>
function submitForm() {
  var formData = new FormData();
  // Create input parameters
  formData.append("_program", "/Folder/Simple JSON");
  formData.append("_action", "execute");
  formData.append("_csrf" , "$CSRF$");
  // Create the request object
  var request = new XMLHttpRequest();
  request.addEventListener("error", function(event) {
    alert("Something went wrong.");
  });
  request.onreadystatechange = function() {

```

```

        if (this.readyState == 4) {
            if (this.status == 200) {
                // Store the returned JSON data in a variable for later use
                var jsonString = this.responseText;
                // Your code to post process the JSON data here
            }
            else {
                alert("Status: " + this.status); }
        }
    };
    request.open("post", "/SASJobExecution/");
    // Submit the form
    request.send(formData);
}
submitForm();
</script>

</head>
<body>

<!-- Content of your web application here -->

</body>
</html>

```

Returning a List of Output Files in JSON Format

The Report with Download Links sample (see [“Report with Download Links” on page 145](#)) creates XML for use with Microsoft Excel, PDF, and HTML output files. Your application might retrieve this output and provide special handling of the files. You can use the `_ACTION` and `_RESULTFILE` input parameters, which is discussed in [“Specifying Output Files” on page 18](#), to return a list of output files in JSON format. For example, the following JSON output is returned to the client when you specify `_ACTION=json` and `_RESULTFILE=Class.pdf,Class.xml,_webout.htm`:

```

{"items": [
  {
    "name": "Class.pdf",
    "href": "/files/files/3dba9d66-106b-4403-aea9-65a6ea3cb514/content"
  },
  {
    "name": "Class.xml",
    "href": "/files/files/dcef367b-4c37-48c8-b53d-29e6189c2dc2/content"
  },
  {
    "name": "_webout.htm",
    "href": "/files/files/3eelceef-8ff3-46ae-8193-6e408dfa0a4f/content"
  }
]}

```

Your application can use the URIs in the HREF keys to retrieve the file content and handle it appropriately.

The following HTML input form executes the Report with Download Links sample and then stores the returned list of files in the JSONString variable:

```
<!DOCTYPE html>
<html>

<head>
<title>JavaScript Job Execution</title>

<script>
function submitForm() {
    var formData = new FormData();
    // Create input parameters
    formData.append("_program", "/Folder/Report with Download Links");
    formData.append("_action", "json,execute");
    formData.append("_resultfile", "*");
    formData.append("_csrf" , "$CSRF$");
    // Create the request object
    var request = new XMLHttpRequest();
    request.addEventListener("error", function(event) {
        alert("Something went wrong.");
    });
    request.onreadystatechange = function() {
        if (this.readyState == 4) {
            if (this.status == 200) {
                // Store the returned JSON data in a variable for later use
                var JSONString = this.responseText;
                // Your code to post process the JSON data here
            }
            else {
                alert("Status: " + this.status);
            }
        }
    };
    request.open("post", "/SASJobExecution/");
    // Submit the form
    request.send(formData);
}
submitForm();
</script>

</head>
<body>

<!-- Content of your web application here -->

</body>
</html>
```

Errors Returned from JSON Requests

Any errors from requests that use `_ACTION=json` return the error information in a JSON structure that contains the following fields:

```
{
  "errorCode": "-80",
  "message": "A procedure in the SAS program provided for the job completed execution with
    at least one error.",
  "details": "180-322: Statement is not valid or it is used out of proper order.\n",
  "log": "/files/files/2baa843c-0198-4635-9aab-c321ca66b2ed"
}
```

In addition, the return header X-SAS-JOBEXEC-ERROR is set.

Including a File in an HTML Form

If you want to import files such as JavaScript or CSS files into an HTML form, you can set up another web server and retrieve the files from there. However, this can often lead to issues with Content Security Policy (CSP) that need to be overridden.

Another way to serve files is to upload them to the Files service and use SAS Job Execution Web Application to retrieve them. The MIME type for the file is determined by the operating system file type. For example, importing a file named `myjslib.js` to the `Public` folder would upload it to the Files service with a MIME type of `text/javascript`.

Using the SAS Job Execution Web Application, the file can be included in an HTML form with a `SCRIPT` tag:

```
<script src="/SASJobExecution/?_file=/Public/myjslib.js"></script>
```

The `_FILE` parameter specifies the folder location and name of the file that you want to include.

Working with SAS Viya Platform Services

You can use the HTTP procedure to access any resource that supports HTTP requests. This includes external web services and SAS Viya platform services. See [“HTTP Procedure” in Base SAS Procedures Guide](#) for more information about PROC HTTP. See [SAS Viya REST APIs](#) for more information about SAS Viya platform services.

You can use the following code to call the Job Definitions service to retrieve the first 50 job definitions in JSON format stored on your system and then display some of the fields:

```
* Base URI for the service call;

%let BASE_URI=%sysfunc(getoption(servicesbaseurl));
```

```

* FILEREFS for the response and the response headers;

filename resp      temp;
filename resp_hdr temp;

proc http url="&BASE_URI/jobDefinitions/definitions/?limit=50"
  method='get'
  oauth_bearer=sas_services
  out=resp
  headerout=resp_hdr
  headerout_overwrite;
run; quit;

* Use the JSON engine to provide read-only sequential access to JSON data;

libname resp json;

title 'Job Definitions';

proc print data=resp.items;
  var name creationTimeStamp createdBy modifiedTimeStamp modifiedBy description;
run; quit;

```

The SAS_SERVICES keyword specified in the OAUTH_BEARER option ensures that an access token is obtained using the identity of the user executing the job.

Samples

Accessing the Samples

You can use the Content page to navigate to the location of the sample jobs that are supplied by SAS, click a job to select it, and then click ► to execute it. The HTML input form or prompts are displayed in the right pane, and the output is displayed in a new tab within the application.

Items Common to Most Samples

Many samples use the same or similar job definition parameters and HTML markup in their input forms. These common items are explained in this section instead of repeating the explanation for each sample.

Job Definition Parameters

All sample jobs include the `_ACTION` input parameter to display their respective HTML input forms or prompts. The sample jobs also include the `_contextName` parameter, which is set to the value of the `contextname` configuration parameter at the time the sample is copied. By default, the value for the `contextname` configuration parameter is **SAS Job Execution compute context**. Additional parameters are specified to ensure that the jobs run successfully if no parameters are specified in the URL, as discussed in [“Executing a Job Using Direct URL Access” on page 15](#).

%JESBEGIN and %JESEND Macros

The `%JESBEGIN` macro is automatically executed before the first line of the program code. This macro uses the values of the `_OUTPUT_TYPE` and `_ODS*` parameters to configure job output. In most cases, a `FILENAME` statement is issued to return output to the web browser. An `ODS` statement is issued in addition to the `FILENAME` statement for samples that use `ODS` to create output. If the `_OUTPUT_TYPE` and `_ODS*` parameters are not defined, the `%JESBEGIN` macro configures the job to create HTML output using `ODS`.

The `%JESEND` macro executes after the last line of code and closes all open `ODS` destinations.

Note: The `%JESBEGIN` and `%JESEND` macros are included by default and do not usually need to be added. If they do need to be manually added or omitted, then `_ADDJESBEGINENDMACROS` must be specified with a value of `false`.

Cascading Style Sheet Code in HTML Input Forms

The following code that appears at the beginning of the HTML input form retrieves the Cascading Style Sheet (CSS) for the current theme and controls the appearance of the input form:

```
<link rel="stylesheet" href="/SASJobExecution/theme">
```

You can omit this `LINK` element if you do not want to use the theme in your HTML input form.

FORM Tags

The following lines appear near the beginning of the HTML input form:

```
<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
```

Always specify `/SASJobExecution/` for the value of the ACTION attribute of the FORM tag. This ensures that the SAS Job Execution Web Application processes the form data when the form is submitted. You can specify `_tab` for the value of the TARGET attribute of the FORM tag if you want output to display in a new tab within the application.

As the name implies, HIDDEN elements are not displayed on screen, but they pass data to the application specified in the ACTION attribute. Specify `$PROGRAM$` as the value for the `_PROGRAM` INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

HTML Attributes to Support Accessibility

The HTML in the job forms that are included with the samples includes the following attributes, which support US government Section 508 accessibility standards:

```
<html lang="en">

<body role="main">

< ... for="element-ID" ... />

< ... aria-label="label-text" ... />

< ... aria-labelledby="checkboxfields element-ID" ... />
```

Prompts

Most of the sample jobs include prompts. You can use `_ACTION=prompts` to display the prompts for a sample. The [“Display Macro Variables”](#) and [“Dynamic Prompts Using CARS Data”](#) sample sections contain the XML code for the prompts. See [SAS Data and AI Studio Developer's Guide: Working with Jobs](#) for more information about the SAS prompting interface and how to create prompts using XML.

Note: There are no prompts for the Upload a CSV File or Upload a File samples.

Display Macro Variables

This sample illustrates how fields from an HTML input form are converted to global macro variables. The global and system macro variables are displayed in HTML format.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	ods_html5	Specifies that ODS HTML5 output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Display Macro Variables

The program invoked by this HTML page is used to display the macro variables created for a job. The SAS Job Execution Web Application creates multiple macro variables when the same name is used for multiple fields in the HTML form.

The following three text fields are all named TEXT.

- 1

The next three checkboxes are all named CBOX. The value shown is the value specified in the INPUT tag.

- 2 ☒ First ☐ Second ☒ Third

Now we have a selection box, LBOX, that allows multiple selections.

- 3

- 4 ODS style:

- 5 ☐ Exclude blank macro variables from the report.

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Display Macro Variables</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<input type="hidden" name="_output_type" value="ods_html5"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>
```

```

<h1 class="jobexec_sample_name">Display Macro Variables</h1>

<p>
The program invoked by this HTML page is used to display the macro
variables created for a job.
The SAS Job Execution Web Application creates multiple macro variables
when the same name is
used for multiple fields in the HTML input form.
</p>

<hr/>
<p>
The following three text fields are all named TEXT.
</p>

1 <input type="text" name="text" value="First"
class="jobexec_sample_input_text"
  aria-label="First text field"/>
<input type="text" name="text" value="Second"
class="jobexec_sample_input_text"
  aria-label="Second text field"/>
<input type="text" name="text" value="Third"
class="jobexec_sample_input_text"
  aria-label="Third text field"/>

<p id="checkboxfields">
The next three checkboxes are all named CBOX. The value shown is the
value
specified in the INPUT tag.
</p>

2 <input type="checkbox" name="cbox" id="cbox1" value="First" checked
class="jobexec_sample_input_checkbox" aria-
labelledby="checkboxfields cbox1"/>
<label for="cbox1">First</label>

<input type="checkbox" name="cbox" id="cbox2" value="Second"
class="jobexec_sample_input_checkbox" aria-
labelledby="checkboxfields cbox2"/>
<label for="cbox2">Second</label>

<input type="checkbox" name="cbox" id="cbox3" value="Third" checked
class="jobexec_sample_input_checkbox" aria-
labelledby="checkboxfields cbox3"/>
<label for="cbox3">Third</label>

<p>
Now we have a selection box, LBOX, that allows multiple selections.
</p>

3 <select name="lbox" multiple class="jobexec_sample_multiselect"
  aria-label="Selection box named LBOX">
  <option value="First">First</option>
  <option value="Second" selected>Second</option>
  <option value="Third">Third</option>
  <option value="Fourth" selected>Fourth</option>

```

```

</select>

<br/>
<br/>

4 <label for="_odsstyle">ODS style:</label>
<select name="_odsstyle" id="_odsstyle" class="jobexec_sample_select">
  [More values here]
  <option value="HTMLBlue" selected>HTMLBlue</option>
  [More values here]
</select>

<br/>
<br/>

5 <input type="checkbox" name="blanks" id="blanks"
value="ExcludeBlanks"
  class="jobexec_sample_input_checkbox"/>
<label for="blanks">Exclude blank macro variables from the report.</
label>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
  class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 This INPUT element, along with elements 2 and 3, enables you to specify multiple values. The creation of multiple value global macro variables and their usage is discussed in [“Using Input Parameters with Multiple Values” on page 55](#). The output shows the values for the macro variables created from these elements when the default values are selected and the form is submitted.
- 2 This INPUT element enables you to specify multiple values.
- 3 This INPUT element enables you to specify multiple values.
- 4 The final SELECT tag creates a drop-down list that enables you to select the ODS style to apply to your output. The HTMLBlue style is selected by default. The selected value is stored in the _ODSTYLE global macro variable.
- 5 All macro variables are displayed in the output by default. Select the **BLANKS** check box to exclude macro variables with blank values.

Specify `$PROGRAM$` as the value for the _PROGRAM INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

View the output to see the macro variables that are created when the form is submitted.

The HTML in this form has some of the same elements that are used in [“ODS Output with Embedded Graphics” on page 138](#).

Prompts

▶Submit

SAS® Job Execution

Display Macro Variables

The program invoked by this prompt page is used to display the macro variables created for a job. The SAS Job Execution Web Application creates multiple macro variables when the same name is used for multiple fields in the prompt page.

The following three text fields are all named TEXT.

First Second Third

This is dual selector named DUAL. [Edit](#)
Select Edit to select values.

First
Third

Now we have a selection box, LBOX, that allows multiple selections.

☐ First
☒ Second
☐ Third
☒ Fourth

ODS style: HTMLBlue ▼

Exclude blank macro variables from the report: No ▼

Show SAS Log: No ▼

The following XML is the source for the prompts:

```
<?xml version="1.0" encoding="utf-8"?>
<Task schemaVersion="7.2">
  <Registration>
    <Name>Display Macro Variablest Prompts</Name>
```

```

    <Description></Description>
    <Version>5.2</Version>
  </Registration>
  <Metadata>
    <DataSources/>
    <Options>
      1 <Option name="_program" inputType="inputtext" active="true"
defaultValue=""/>
        <Option name="_action" inputType="inputtext" active="true"
defaultValue="execute"/>

      2 <Option name="blankline" inputType="string"/>

      <Option name="header"
inputType="markdown">&lt;h1&gt;SAS&amp;#174;
      Job Execution&lt;/h1&gt;</Option>

      <Option name="name" inputType="markdown">&lt;b&gt;Display Macro
      Variables&lt;/b&gt;</Option>
      <Option name="desc1" inputType="string" readOnly="true">
        The program invoked by this prompt page is used to display
the macro variables created
        for a job. The SAS Job Execution Web Application creates
multiple macro variables when
        the same name is used for multiple fields in the prompt page.
      </Option>

      2 <Option name="hr" inputType="markdown">&lt;hr&gt;</Option>

      <Option name="desc2" inputType="string" readOnly="true">
        The following three text fields are all named TEXT.
      </Option>

      3 <Option name="text_1" inputType="inputtext"
defaultValue="First" returnName="text">
        </Option>
        <Option name="text_2" inputType="inputtext"
defaultValue="Second" returnName="text">
        </Option>
        <Option name="text_3" inputType="inputtext" defaultValue="Third"
returnName="text">
        </Option>

      3 <Option name="dual" inputType="dualselector"
defaultValue="dual_1,dual_3"
        returnName="dual">
        This is dual selector named DUAL.
        Select Edit to select values.
      </Option>
      <Option inputType="string" name="dual_1"
returnValue="First">First</Option>
      <Option inputType="string" name="dual_2"
returnValue="Second">Second</Option>
      <Option inputType="string" name="dual_3"
returnValue="Third">Third</Option>

```

```

        <Option name="desc4" inputType="string" readOnly="true">
            Now we have a selection box, LBOX, that allows multiple
            selections.
        </Option>

3 <Option name="lbox" inputType="select"
  defaultValue="lbox_2,lbox_4" returnName="lbox">
    </Option>
    <Option inputType="string" name="lbox_1"
  returnValue="First">First</Option>
    <Option inputType="string" name="lbox_2"
  returnValue="Second">Second</Option>
    <Option inputType="string" name="lbox_3"
  returnValue="Third">Third</Option>
    <Option inputType="string" name="lbox_4"
  returnValue="Fourth">Fourth</Option>

    <Option name="_odsstyle_label" inputType="string"
  readOnly="true">ODS style:</Option>
4 <Option name="_odsstyle" inputType="combobox"
  defaultValue="_odsstyle_15"></Option>
    [More values here]
    <Option inputType="string" name="_odsstyle_15"
  returnValue="HTMLBlue">HTMLBlue</Option>
    [More values here]

    <Option name="blanks_label" inputType="string">Exclude blank
  macro variables from the
  report:</Option>
5 <Option name="blanks" inputType="combobox"
  defaultValue="blanks_1"></Option>
    <Option inputType="string" name="blanks_1" returnValue="">No</
  Option>
    <Option inputType="string" name="blanks_2"
  returnValue="ExcludeBlanks">Yes</Option>

    <Option name="_debug_label" inputType="string">Show SAS Log:</
  Option>
6 <Option name="_debug" inputType="combobox"
  defaultValue="_debug_1"></Option>
    <Option inputType="string" name="_debug_1" returnValue="">No</
  Option>
    <Option inputType="string" name="_debug_2"
  returnValue="log">Yes</Option>
  </Options>
</Metadata>
7 <UI>
  <OptionItem option="header"/>
  <OptionItem option="name"/>
  <OptionItem option="desc1"/>

  <OptionItem option="hr"/>

  <OptionItem option="desc2"/>

  <HorizontalLayout>

```

```

        <OptionItem option="text_1"/>
        <OptionItem option="text_2"/>
        <OptionItem option="text_3"/>
    </HorizontalLayout>
    <OptionItem option="blankline"/>

    <OptionChoice option="dual">
        <OptionItem option="dual_1"/>
        <OptionItem option="dual_2"/>
        <OptionItem option="dual_3"/>
    </OptionChoice>
    <OptionItem option="blankline"/>

    <OptionItem option="desc4"/>

    <OptionChoice option="lbox">
        <OptionItem option="lbox_1"/>
        <OptionItem option="lbox_2"/>
        <OptionItem option="lbox_3"/>
        <OptionItem option="lbox_4"/>
    </OptionChoice>
    <OptionItem option="blankline"/>

    <HorizontalLayout>
        <OptionItem option="_odsstyle_label"/>
        <OptionChoice option="_odsstyle">
            [More values here]
        </OptionChoice>
    </HorizontalLayout>
    <OptionItem option="blankline"/>

    <HorizontalLayout>
        <OptionItem option="blanks_label"/>
        <OptionChoice option="blanks">
            <OptionItem option="blanks_1"/>
            <OptionItem option="blanks_2"/>
        </HorizontalLayout>
    </OptionChoice>

    <OptionItem option="hr"/>

    <HorizontalLayout>
        <OptionItem option="_debug_label"/>
        <OptionChoice option="_debug">
            <OptionItem option="_debug_1"/>
            <OptionItem option="_debug_2"/>
        </OptionChoice>
    </HorizontalLayout>
</UI>
<CodeTemplate/>
</Task>

```

- 1 The path and name of the program to execute are substituted at run time by the SAS Job Execution Web Application. The value for _PROGRAM is not needed in the XML because the value for the _PROGRAM URL parameter is used.

- 2 The `blankline` and `hr` objects are used to add vertical white space and a horizontal line to the prompt form, respectively.
- 3 These elements enable you to specify multiple values.
- 4 This element creates a drop-down list that enables you to select the ODS style to apply to your output. The HTMLBlue style is selected by default. The selected value is stored in the `_ODSTYLE` global macro variable.
- 5 All macro variables are displayed in the output by default. This element enables you to specify whether to exclude macro variables with blank values.
- 6 The `_debug` object creates a drop-down list with two choices. Selecting **Yes** returns a value of `log` in the `_DEBUG` macro variable.
- 7 The UI section displays the objects that were previously defined.

See *SAS Data and AI Studio Developer's Guide: Working with Jobs* for more information about the SAS prompting interface and how to create prompts using XML. If you are using a prompting interface that uses JSON, see the JSON file [here](#).

Program

```
* Declare input parameter;

%global BLANKS;

%macro blanks;
  %if %length(&BLANKS) ne 0 %then and compress(value) ne ' ';
%mend;

*;
* Get the macro variables in the current SAS session.
* The RESOLVE function insures that the unmasked macro
* variable values are obtained.
*;

proc sort data=sashelp.vmacro out=work.vmacro sortseq=ebcdic;
  by scope name offset;
run; quit;

data work.globalvars
  work.systemvars;
set work.vmacro;
where (name ne 'TCPLISTN' %BLANKS) and
      (name not like '_RR_%') and
      (name not like 'SQL%') and
      (name not like 'SYS_SQL%');
by scope name;
length full_value $32767;
retain full_value;
keep name full_value;

if (first.name)
  then full_value=value;
  else full_value=cats(full_value, value);
```

```

if last.name then do;
    full_value = htmlencode(resolve(full_value));

    if (scope eq 'GLOBAL') then output work.globalvars;
    else if (scope eq 'AUTOMATIC') then output work.systemvars;
end;

label name          = 'Macro Variable Name'
      full_value = 'Macro Variable Value';

run;

* Specify PROTECTSPECIALCHARS=off to prevent "double encoding" the
values;

title 'Global Macro Variables';
footnote;

proc print data=work.globalvars label noobs
    style(header)=[just = center]
    style(column)=[protectspecialchars=off];
run;
quit;

title 'Automatic (System) Macro Variables';
footnote;

proc print data=work.systemvars label noobs
    style(header)=[just = center]
    style(column)=[protectspecialchars=off];
run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return HTML output to the web browser because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- issuing an ODS statement for the HTML5 destination because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- writing ODS output to the `_WEBOUT` fileref that is assigned by the %JESBEGIN macro. The style specified in the `_ODSSYLE` input parameter controls the appearance of the output.

The BLANKS macro generates code to omit blank macro variables from the output, if this option was selected in the input form.

Macro variable information spans multiple records in the SASHELP.VMACRO view when the value exceeds 200 characters. The DATA step code collects information from multiple records and then stores the first 32,767 characters of the value.

The HTMLENCODE function is used when resolving the full value to prevent execution of malicious code in the web browser.

The PRINT procedure displays the global and system macro variable names and values.

The %JESSEND macro executes after the last line of code and closes all open ODS destinations.

Output

The following sample output contains selected values, assuming that default selections were made in the HTML input form or prompts.

Global Macro Variables	
Macro Variable Name	Macro Variable Value
_ACTION	execute
_ADDJESBEGINENDMACROS	true
_CLIENTNAME	SASJobExecution
_CONTDISP_FILEEXT	html
_CONTDISP_FILENAME	SASResults
_CONTEXTNAME	SAS Job Execution compute context
_COOKIE	jobexec_1568914894340
_CSRF	6484-bd25-45b1-1029-4884-4884-4884-4884

	First
CBOX_COUNT	2
CBOX0	2
CBOX1	First
CBOX2	Third
LBOX	Second
LBOX_COUNT	2
LBOX0	2
LBOX1	Second
LBOX2	Fourth
SYS_COMPUTE_DATA	/opt/ses/viya/config/data/compsrv/default
SYS_COMPUTE_JOB_ID	3C7A173D-D336-214A-B7F5-E1FD9AF74905
SYS_COMPUTE_SESSION_ID	fd5d523d-dc62-454f-b38d-f310b7b8d0db-ses0000
SYS_COMPUTE_SESSION_OWNER	omitest
SYS_JES_JOB_URI	/jobExecution/jobs/772653b5-9e1e-4866-b557-afac4886a987
TEXT	First
TEXT_COUNT	3
TEXT0	3
TEXT1	First
TEXT2	Second
TEXT3	Third

Dynamic Prompts Using CARS Data

This sample illustrates how to create SAS Viya platform dynamic and cascading prompts. *Dynamic prompts* allow the lookup of possible prompt values from a data source such as a SAS data set. *Cascading prompts* populate prompt values based on selections in other prompts. In this sample, prompts are sourced dynamically from the SASHELP.CARS data set and provide a user interface to the program. After you select a value for the vehicle make, you can select a value for the vehicle type. The values that are available for vehicle type depend on the value that was selected for vehicle make. An HTML input form is not available for this sample.

Note: Before you create prompts that use dynamic data, see [“Working with Dynamic Data in Prompts and HTML Forms” on page 74](#) for information about how to assign a library to the compute context that is associated with the job.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	<code>prompts,execute</code>	Displays the prompts if available; otherwise, executes the job
_OUTPUT_TYPE	<code>ods_html5</code>	Specifies that ODS HTML5 output is created by the job
MAKE	<code>Saab</code>	Specifies the make of the car
TYPE	<code>Sedan</code>	Specifies the type of car
_contextName	<code>SAS Job Execution compute context</code>	Specifies the compute context for executing the job

Prompts

▶ Submit

SAS® Job Execution

Dynamic Prompts Using CARS Data

This sample uses data in the SASHELP.CARS data set to source dynamic prompts.

Data set:

SASHELP.CARS

Select a vehicle make: *

Saab ▼

Select a vehicle type: *

Sedan ▼

Show SAS Log: No ▼

The following XML is the source for the prompts:

```
<?xml version="1.0" encoding="utf-8"?>
<Task schemaVersion="7.2">
  <Registration>
    <Name>Dynamic Prompts Using CARS Data Prompts</Name>
    <Description></Description>
    <Version>5.2</Version>
  </Registration>
  <Metadata>
    <DataSources>
      1 <DataSource name="dataset" defaultValue="sashelp.cars"
where="false" required="true"
      readOnly="true">
        <Filters>
          2 <Filter name="filterMake">
            <Column column="make" sortBy="label"
sortDirection="ascending"/>
          </Filter>
          3 <Filter name="filterType">
            <Column column="type" sortBy="label"
sortDirection="ascending"/>
          </Filter>
        </Filters>
      </DataSource>
    </DataSources>
  </Metadata>
  <Where>
    <Expressions>
      <Expression operation="eq">
```

```

        <Column column="make" sortBy="label"/>
        <Value option="make"/>
    </Expression>
</Expressions>
</Where>
</Filter>
</Filters>
</DataSource>
</DataSources>

<Options>
    4 <Option name="_program" inputType="inputtext" active="true"
defaultValue="/">
        <Option name="_action" inputType="inputtext" active="true"
defaultValue="wait"/>

    5 <Option name="blankline" inputType="string"/>

    <Option name="header"
inputType="markdown">&lt;h1&gt;SAS&amp;#174; Job Execution&lt;/h1&gt;
    </Option>

    <Option name="name" inputType="markdown">&lt;b&gt;Dynamic
Prompts Using CARS Data&lt;/b&gt;
    </Option>
    <Option name="desc" inputType="string" readOnly="true">
        This sample uses data in the SASHELP.CARS data set to source
dynamic prompts.
    </Option>

    5 <Option name="hr" inputType="markdown">&lt;hr&gt;</Option>

    <Option name="dataset_label" inputType="string">Data set:</
Option>

    6 <Option name="make" inputType="combobox" filter="filterMake"
defaultValue="Saab"
        required="true">Select a vehicle make:</Option>
    7 <Option name="type" inputType="combobox" filter="filterType"
defaultValue="Sedan"
        required="true">Select a vehicle type:</Option>

    <Option name="_debug_label" inputType="string">Show SAS Log:</
Option>
    8 <Option name="_debug" inputType="combobox"
defaultValue="_debug_1"></Option>
    <Option inputType="string" name="_debug_1" returnValue="">No</
Option>
    <Option inputType="string" name="_debug_2"
returnValue="log">Yes</Option>
</Options>
</Metadata>
    9 <UI>
    <OptionItem option="header"/>
    <OptionItem option="name"/>
    <OptionItem option="desc"/>

```

```

<OptionItem option="hr"/>

<OptionItem option="dataset_label"/>
<DataItem data="dataset"/>
<OptionItem option="blankline"/>

<OptionChoice option="make"/>
<OptionItem option="blankline"/>

<OptionChoice option="type"/>

<OptionItem option="hr"/>

<HorizontalLayout>
  <OptionItem option="_debug_label"/>
  <OptionChoice option="_debug">
    <OptionItem option="_debug_1"/>
    <OptionItem option="_debug_2"/>
  </OptionChoice>
</HorizontalLayout>
</UI>
<CodeTemplate/>
</Task>

```

- 1 The `DataSource` element specifies the SAS data set to use as a data source for the prompts. You cannot select a value at run time because the `readOnly` attribute is set to `true`.
- 2 The first `Filter` element specifies that the values of the `MAKE` variable in `SASHELP.CARS` are displayed.
- 3 The second `Filter` element specifies that the values of the `TYPE` variable in `SASHELP.CARS` are displayed. Only values matching the selected value of `MAKE` are displayed because a `WHERE` expression is specified.
- 4 The path and name of the program to execute are substituted at run time by the SAS Job Execution Web Application. The value for `_PROGRAM` is not needed in the XML because the value for the `_PROGRAM` URL parameter is used.
- 5 The `blankline` and `hr` objects are used to add vertical white space and a horizontal line to the prompt form, respectively.
- 6 The `make` object creates a drop-down list populated with the values of the `MAKE` variable. The data displayed is determined by the `FILTER` attribute.
- 7 The `type` object creates a drop-down list populated with the values of the `TYPE` variable, but only for observations that match the selected value of `MAKE`.
- 8 The `_debug` object creates a drop-down list with two choices. Selecting **Yes** returns a value of `log` in the `_DEBUG` macro variable.
- 9 The UI section displays the objects that were previously defined.

See [SAS Data and AI Studio Developer's Guide: Working with Jobs](#) for more information about the SAS prompting interface and how to create prompts using XML. If you are using a prompting interface that uses JSON, see the JSON code [here](#).

When the prompts are submitted, the following global macro variables are defined just before SAS code execution, but only the MAKE and TYPE variables are used in the program code:

Macro Variable Name	Macro Variable Value
_ACTION	wait
_OUTPUT_TYPE	ods_html5
_PROGRAM	/Folder/Dynamic Prompts Using CARS Data
MAKE	Saab
TYPE	Sedan
_DEBUG	log (if Yes is selected in the drop-down list)

Program

```

* Declare input parameters;

%global MAKE
      TYPE;

* Subset the data based on prompt selections;

proc sql;
  create table work.cars as
  select make, type, model, enginesize, horsepower, msrp
  from sashelp.cars
  where (make eq "&MAKE") and
        (type eq "&TYPE")
  order by enginesize, model, msrp;
quit;

title 'Data Based on Prompt Selections';

proc report data=work.cars;
  column ('Prompt Selections' make type) model enginesize horsepower
  msrp;
  define make--type / order;
  define enginesize / format=4.1;
run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return HTML output to the web browser because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- issuing an ODS statement for the HTML5 destination because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- writing ODS output to the `_WEBOUT` fileref that is assigned by the %JESBEGIN macro. The default style for the HTML5 destination is used because the `_ODSSYLE` input parameter was not specified.

The SQL procedure selects data from the SASHELP.CARS data set based on the user selections in the prompt form, and the REPORT procedure displays this data.

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

Output

Data Based on Prompt Selections					
Prompt Selections					
Make	Type	Model	Engine Size (L)	Horsepower	MSRP
Saab	Sedan	9-3 Aero 4dr	2.0	210	\$33,360
		9-3 Aero convertible 2dr	2.0	210	\$43,175
		9-3 Arc Sport 4dr	2.0	210	\$30,860
		9-3 Arc convertible 2dr	2.0	210	\$40,670
		9-5 Aero 4dr	2.3	250	\$39,465
		9-5 Arc 4dr	2.3	220	\$35,105

Evaluating Data Cardinality Using PROC CAS

This sample determines the limited cardinality (up to 254 levels) for each variable in the specified SAS data set, and assigns each variable one of three roles: CLASS, ID, or INTERVAL. The cardinality action set, included with SAS Visual Statistics, is used with the CAS procedure to perform the analysis.

Note: You must have a license for SAS Visual Statistics to run this sample.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
DATASET	sashelp.baseball	Specifies the SAS data set to analyze
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Evaluating Data Cardinality Using PROC CAS

This sample determines the limited cardinality (up to 254 levels) for each variable in the specified SAS data set, and assigns each variable one of three roles: CLASS, ID, or INTERVAL. The Cardinality action set is used with the CAS procedure to perform the analysis.

Note: You must have a license for SAS® Visual Statistics to run this sample.

Data set: SASHELP.BASEBALL ▼

Output format: (default) ▼

ODS style: (default) ▼

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
<title>Evaluating Data Cardinality Using PROC CAS</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Evaluating Data Cardinality Using PROC
CAS</h1>

<p>
This sample determines the limited cardinality (up to 254 levels) for
each variable in
the specified SAS data set, and assigns each variable one of three
roles: CLASS,
ID, or INTERVAL. The Cardinality action set is used with the CAS
procedure
to perform the analysis.
</p>

<p>
Note: You must have a license for SAS<sup>&#174;</sup> Visual
Statistics to run this sample.
</p>

<hr/>

<label for="dataset">Data set:</label>
<select name="dataset" id="dataset" class="jobexec_sample_select">
  <option value="sashelp.baseball" selected>SASHELP.BASEBALL</option>
  <option value="sashelp.prdsal2">SASHELP.PRDSAL2</option>
  <option value="sashelp.shoes">SASHELP.SHOES</option>
  <option value="sashelp.zipcode">SASHELP.ZIPCODE</option>
</select>

<br/>
<br/>

<label for="_output_type">Output format:</label>
<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="" selected>(default)</option>
  <option value="ods_html">HTML</option>
  <option value="ods_html5">HTML5</option>
  <option value="ods_pdf">Portable Document Format (PDF)</option>
  <option value="ods_excel">Microsoft Excel (XLSX)</option>
  <option value="ods_word">Microsoft Word (DOCX)</option>
  <option value="ods_rtf">Rich Text Format (RTF)</option>

```

```

    <option value="ods_csv">Comma-separated Value (CSV)</option>
    <option value="ods_xml">Extensible Markup Language (XML)</option>
    <option value="ods_tagsets.rtf">RTF Tagset</option>
    <option value="ods_tagsets.excelxp">Microsoft Excel (XML)</option>
    <option value="ods_ps">Postscript (PS)</option>
    <option value="ods_latex">LaTeX</option>
  </select>

  <br/>
  <br/>

  <label for="_odsstyle">ODS style:</label>
  <select name="_odsstyle" id="_odsstyle" class="jobexec_sample_select">
    <option value="" selected>(default)</option>
    <option value="Analysis">Analysis</option>

    [More values here]

  </select>

  <br/>
  <br/>

  <hr/>

  <input type="submit" value="Run code"
  class="jobexec_sample_input_submit"/>
  <input type="checkbox" name="_debug" id="_debug" value="log"
  class="jobexec_sample_input_checkbox"/>
    <label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

The following table contains the display values for the first drop-down list and the corresponding values for the DATASET global macro variable:

Display Value	Macro Variable Value
SASHELP.BASEBALL	<code>sashelp.baseball</code>
SASHELP.PRDSAL2	<code>sashelp.prdsal2</code>
SASHELP.SHOES	<code>sashelp.shoes</code>
SASHELP.ZIPCODE	<code>sashelp.zipcode</code>

The following table contains the display values for the second drop-down list and the corresponding values for the _OUTPUT_TYPE global macro variable:

Display Value	Macro Variable Value
(default)	
HTML	<code>ods_html</code>
HTML5	<code>ods_html5</code>
Portable Document Format (PDF)	<code>ods_pdf</code>
Microsoft Excel (XLSX)	<code>ods_excel</code>
Microsoft Word (DOCX)	<code>ods_word</code>
Rich Text Format (RTF)	<code>ods_rtf</code>
Comma-Separated Value (CSV)	<code>ods_csv</code>
Extensible Markup Language (XML)	<code>ods_xml</code>
RTF_Tagset	<code>ods_tagsets.rtf</code>
Microsoft Excel (XML)	<code>ods_tagsets.excelxp</code>
PostScript (PS)	<code>ods_ps</code>
LaTeX	<code>ods_latex</code>

When the form is submitted, the following global macro variables are defined just before SAS code execution, but only the DATASET variable is used in the program code:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	<code>execute</code>
<code>_ODSDEST</code>	Depends on selection (for example, <code>tagsets.rtf</code>)
<code>_ODSSTYLE</code>	Depends on selection (for example, <code>HTMLBlue</code>)
<code>_OUTPUT_TYPE</code>	Depends on selection (see previous table)
<code>_PROGRAM</code>	<code>/Folder/Evaluating Data Cardinality Using PROC CAS</code>
<code>_DEBUG</code>	<code>log</code> (if the check box is selected)
<code>DATASET</code>	Depends on selection (see previous table)

The `_ODSDEST` macro variable is derived from the value of `_OUTPUT_TYPE` and indicates the ODS destination that is used.

The HTML in this form uses some of the same fields as [“Multiple Output Formats” on page 132](#). See that sample for more information.

Program

```
* Declare input parameter;

%global DATASET;

* Declare program variable;

%global TABLE;

%let TABLE=%scan(&DATASET, 2);

* Start the CAS session;

cas mysession;

* Load the data into the active CAS library;

proc casutil;
  load data=&DATASET replace;
run; quit;

* Suppress display of the ODS procedure title text;

ods noproctitle;

proc cas;

* Analyze the data and determine cardinality;

cardinality.summarize /
  table={name="&TABLE"},
  cardinality={name='Cardinality'},
  options={nLevels=254},
  varList={};

* Display the results to the currently open ODS destination;

title "Analysis of the %upcase(&DATASET) Data Set";

table.fetch /
  table='Cardinality'
  fetchVars={ {name='_varname_'},
               {name='_type_'},
               {name='_cardinality_'},
               {name='_rlevel_'},
               {name='_nmiss_'}}
             }
```

```

index=false
sortBy={ {name='_rlevel_' } }
to=65535;

run; quit;

* End the CAS session;

cas mysession terminate;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return the type of output that is specified in the _OUTPUT_TYPE input parameter to the web browser. If a blank value is specified for the _OUTPUT_TYPE input parameter, then a FILENAME statement for HTML output is issued.
- issuing an ODS statement for the appropriate destination. If a blank value is specified for the _OUTPUT_TYPE input parameter, then the ODS HTML5 destination is used.
- writing ODS output to the _WEBOUT fileref that is assigned by the %JESBEGIN macro. The style specified in the _ODSSYLE input parameter controls the appearance of the output.

The data set name is determined from the two-level name specified in the DATASET input parameter and is stored in the TABLE macro variable. A CAS session is started and the CASUTIL procedure loads the specified data set into the active CAS library. For more information about the CASUTIL procedure, see [“CASUTIL Procedure” in SAS Cloud Analytic Services: User’s Guide](#).

The CAS procedure provides a programming environment that enables you to run CAS actions. For more information about the CAS procedure, see [“Overview: CAS Procedure” in SAS Cloud Analytic Services: CASL Reference](#).

The summarize action of the cardinality action set, available with SAS Visual Statistics, determines the limited cardinality for each variable in the specified CAS table. Each variable is assigned one of three roles: CLASS, ID, or INTERVAL. Results are stored in the Cardinality CAS table and are displayed using the fetch method of the table action set. For more information, see the [cardinality](#) action set in *SAS Visual Statistics: Programming Guide*. For more information about the table action set, see [“Table Action Set” in SAS Viya Platform: System Programming Guide](#).

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

HTML and PDF output are displayed by the web browser, and all other output is downloaded so that it can be saved and opened with an appropriate application.

Output

The following sample output is the result of selecting default values in the HTML input form or prompts.

Analysis of the SASHELP.BASEBALL Data Set				
Selected Rows from Table CARDINALITY				
Variable name	Type of the raw values	Number of levels	Recommended level for analytics	Number of missing values
nError	N	29	CLASS	0
Salary	N	150	CLASS	59
nHits	N	136	CLASS	0
logSalary	N	150	CLASS	59
nAtBat	N	246	CLASS	0
Div	C	4	CLASS	0
nBB	N	85	CLASS	0
YrMajor	N	22	CLASS	0
nHome	N	37	CLASS	0
Team	C	24	CLASS	0
CrHome	N	151	CLASS	0
nRuns	N	91	CLASS	0
nRBI	N	100	CLASS	0
CrBB	N	248	CLASS	0
League	C	2	CLASS	0
Division	C	2	CLASS	0
Position	C	25	CLASS	0
nOuts	N	234	CLASS	0
nAssts	N	161	CLASS	0
Name	C	254	ID	0
CrRbi	N	254	INTERVAL	0
CrRuns	N	254	INTERVAL	0
CrHits	N	254	INTERVAL	0
CrAtBat	N	254	INTERVAL	0

Hello World

This sample creates HTML output using DATA step code and returns it to the web browser. An HTML input form, which accepts a name for the greeting, provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	html	Specifies that non-ODS HTML output is created by the job
myname	World	Specifies the default value used in the greeting
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

1 SAS® Job Execution

2 Hello World

This sample uses a DATA Step with simple PUT statements to create the output.

3 Specify a name for the greeting:

4 ☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Hello World</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">
```

```

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="wait,execute"/>
<input type="hidden" name="_output_type" value="html"/>
1 <div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job
Execution</div>

2 <h1 class="jobexec_sample_name">Hello World</h1>

<p>
This sample uses a DATA Step with simple PUT statements to create the
output.
</p>

<hr/>

<br/>

3 <label for="myname">Specify a name for the greeting:</label>
<input type="text" name="myname" id="myname" value="World" required
class="jobexec_sample_input_text"/>

<br/>
<br/>

<hr/>

4 <input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>
</html>

```

- 1 The DIV tag displays a page heading.
- 2 The name of the sample is displayed in the H1 tag.
- 3 The code prompts the user to specify a name used in the greeting. This field is required, and the default value for the name is **World**.
- 4 The **Run Code** button submits the job for execution. Select the **Show SAS Log** check box if you want to view the SAS log with the output.

A `Please wait` message is displayed while the program is running when the value for `_ACTION` is set to `wait,execute`. Specify only `execute` if you do not want to see this message. The value of `_OUTPUT_TYPE` indicates that non-ODS HTML output is created by the job. The `%JESBEGIN` macro issues a `FILENAME` statement that supports HTML output, but no ODS statement is issued.

When the form is submitted, the following global macro variables are defined just before SAS code execution:

Macro Variable Name	Macro Variable Value
_ODSDEST	blank
_OUTPUT_TYPE	html
_PROGRAM	/Folder/Hello World
myname	World
_DEBUG	log (if the check box is selected)

The _ODSDEST macro variable is derived from the value of _OUTPUT_TYPE. It is blank because ODS is not being used. The MYNAME macro variable is used in the program code.

Program

```
* Declare input parameter;

%global MYNAME;

* Write the custom HTML to _webout;

data _null_;
file _webout;
put '<!DOCTYPE html>';
put '<html lang="en">';
put '<head><title>Hello World!</title></head>';
put '<body role="main">';
put "<h1>Hello %sysfunc(htmlencode(&MYNAME))!</h1>";
put '</body>';
put '</html>';
run;
```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including issuing a FILENAME statement to return HTML output to the web browser because **html** is specified as the value for the _OUTPUT_TYPE input parameter.

The DATA step code writes simple HTML to the _WEBOUT fileref that is assigned by the %JESBEGIN macro, and that HTML is rendered and displayed by the web browser. The value of the MYNAME macro variable used in the PUT statement is obtained from the value that was specified in the HTML input form. The

HTMLENCODE function is used here to prevent execution of malicious code in the web browser.

The %JESEND macro executes after the last line of code, but it does not close all open ODS destinations because this sample does not use ODS.

Output

Hello World!

MEANS Procedure Executed Using CAS

This sample computes a two-sided 90% confidence limit for the mean values of two variables. Data is loaded into a CAS library, and the MEANS procedure performs the computations using the CAS data. This code for this sample is based on the code from [“Computing a Confidence Limit for the Mean” in Base SAS Procedures Guide](#).

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

MEANS Procedure Executed Using CAS

This sample computes a two-sided 90% confidence limit for the mean values of two variables. Data is loaded into a CAS library, and the MEANS procedure performs the computations using the CAS data.

Output format:

ODS style:

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>

<html lang="en">
<head>
<title>MEANS Procedure Executed Using CAS</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">MEANS Procedure Executed Using CAS</h1>

<p>
This sample computes a two-sided 90% confidence limit for the mean
values
of two variables. Data is loaded into a CAS library, and the MEANS
procedure
performs the computations using the CAS data.
</p>

<hr/>

<label for="_output_type">Output format:</label>
<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="" selected>(default)</option>
```

```

    <option value="ods_html">HTML</option>
    <option value="ods_html5">HTML5</option>
    <option value="ods_pdf">Portable Document Format (PDF)</option>
    <option value="ods_excel">Microsoft Excel (XLSX)</option>
    <option value="ods_word">Microsoft Word (DOCX)</option>
    <option value="ods_rtf">Rich Text Format (RTF)</option>
    <option value="ods_csv">Comma-separated Value (CSV)</option>
    <option value="ods_xml">Extensible Markup Language (XML)</option>
    <option value="ods_tagsets.rtf">RTF Tagset</option>
    <option value="ods_tagsets.excelxp">Microsoft Excel (XML)</option>
    <option value="ods_ps">Postscript (PS)</option>
    <option value="ods_latex">LaTeX</option>
  </select>

  <br/>
  <br/>

  <label for="_odsstyle">ODS style:</label>
  <select name="_odsstyle" id="_odsstyle" class="jobexec_sample_select">
    <option value="" selected>(default)</option>
    <option value="Analysis">Analysis</option>

    [More values here]

  </select>

  <br/>
  <br/>

  <hr/>

  <input type="submit" value="Run code"
  class="jobexec_sample_input_submit"/>
  <input type="checkbox" name="_debug" id="_debug" value="log"
  class="jobexec_sample_input_checkbox"/>
    <label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

The following table contains the display values for the first drop-down list and the corresponding values for the `_OUTPUT_TYPE` global macro variable:

Display Value	Macro Variable Value
(default)	
HTML	<code>ods_html</code>
HTML5	<code>ods_html5</code>
Portable Document Format (PDF)	<code>ods_pdf</code>

Display Value	Macro Variable Value
Microsoft Excel (XLSX)	<code>ods_excel</code>
Microsoft Word (DOCX)	<code>ods_word</code>
Rich Text Format (RTF)	<code>ods_rtf</code>
Comma-Separated Value (CSV)	<code>ods_csv</code>
Extensible Markup Language (XML)	<code>ods_xml</code>
RTF_Tagset	<code>ods_tagsets.rtf</code>
Microsoft Excel (XML)	<code>ods_tagsets.excelxp</code>
PostScript (PS)	<code>ods_ps</code>
LaTeX	<code>ods_latex</code>

When the form is submitted, the following global macro variables are defined just before SAS code execution:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	<code>execute</code>
<code>_ODSDEST</code>	Depends on selection (for example, <code>tagsets.rtf</code>)
<code>_ODSSTYLE</code>	Depends on selection (for example, <code>HTMLBlue</code>)
<code>_OUTPUT_TYPE</code>	Depends on selection (see previous table)
<code>_PROGRAM</code>	<code>/Folder/MEANS Procedure Executed Using CAS</code>
<code>_DEBUG</code>	<code>log</code> (if the check box is selected)

The `_ODSDEST` macro variable is derived from the value of `_OUTPUT_TYPE` and indicates the ODS destination that is used.

The HTML in this form uses some of the same fields as [“Multiple Output Formats” on page 132](#). See that sample for more information.

Program

```
* Start the CAS session;
```

```

cas mysession;

* Sample data;

data work.charity;
input School $ 1-7 Year 9-12 Name $ 14-20
      MoneyRaised 22-26 HoursVolunteered 28-29;
datalines;
Monroe 2016 Allison 31.65 19
Monroe 2016 Barry 23.76 16
Monroe 2016 Candace 21.11 5

      [More data lines here]

Kennedy 2018 Sid 27.45 25
Kennedy 2018 Will 28.88 21
Kennedy 2018 Morty 34.44 25
;
run;

* Load the data into the active CAS library;

proc casutil;
  load data=work.charity;
run; quit;

* Assign a library for the data using the CAS engine;

libname casdata cas;

* Suppress display of the ODS procedure title text;

ods noproctitle;

* Perform the analysis using the CAS data;

title1 'Confidence Limits for Fund Raising Statistics';
title2 '2016 - 2018';

proc means data=casdata.charity fw=8 maxdec=2 alpha=0.1 clm mean std;
  class Year;
  var MoneyRaised HoursVolunteered;
run; quit;

* End the CAS session;

cas mysession terminate;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return the type of output that is specified in the _OUTPUT_TYPE input parameter to the web browser. If a blank value is specified for the _OUTPUT_TYPE input parameter, then a FILENAME statement for HTML output is issued.
- issuing an ODS statement for the appropriate destination. If a blank value is specified for the _OUTPUT_TYPE input parameter, then the ODS HTML5 destination is used.
- writing ODS output to the _WEBOUT fileref that is assigned by the %JESBEGIN macro. The style specified in the _ODSSYLE input parameter controls the appearance of the output.

A CAS session is started and the CASUTIL procedure loads the specified data set into the active CAS library. For more information about the CASUTIL procedure, see [“CASUTIL Procedure” in SAS Cloud Analytic Services: User’s Guide](#).

The CAS LIBNAME engine enables your SAS code to access the in-memory data. The LIBNAME statement is associated with the active CAS library, making the Charity table available to PROC MEANS. The following note in the log indicates that the MEANS procedure uses the aggregate CAS action to perform the analysis:

NOTE: The CAS aggregation.aggregate action will be used to perform the initial summarization.

For more information about the CAS LIBNAME engine, see [“CAS LIBNAME Engine Overview” in SAS Cloud Analytic Services: User’s Guide](#). For more information about the aggregate CAS action, see [“Aggregation Action Set” in SAS Visual Analytics: Programming Guide](#).

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

HTML and PDF output are displayed by the web browser, and all other output is downloaded so that it can be saved and opened with an appropriate application.

Output

The following sample output is the result of selecting default values in the HTML input form or prompts.

Confidence Limits for Fund Raising Statistics 2016 - 2018						
Year	N Obs	Variable	Lower 90% CL for Mean	Upper 90% CL for Mean	Mean	Std Dev
2016	34	MoneyRaised	25.69	32.29	28.99	11.37
		HoursVolunteered	17.68	22.73	20.21	8.71
2017	29	MoneyRaised	24.61	31.61	28.11	11.09
		HoursVolunteered	15.67	20.19	17.93	7.17
2018	46	MoneyRaised	26.73	33.78	30.26	14.23
		HoursVolunteered	19.68	22.63	21.15	5.96

Multiple Output Formats

This sample uses ODS to create different output formats. An HTML input form, which accepts the data set to display and the ODS destination and style, provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
DATASET	SASHELP.CLASS	Specifies the SAS data set to display
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Multiple Output Formats

This sample shows different output formats supported by the Output Delivery System (ODS). Sample data sets can be printed to any of the output formats listed below.

1 Data set:

2 Output format:

3 ODS style:

Run code

☐ Show SAS Log

The following HTML is the source for the input form:

```

<!DOCTYPE html>
<html lang="en">

<head>
<title>Multiple Output Formats</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Multiple Output Formats</h1>

<p>
This sample shows different output formats supported by the
Output Delivery System (ODS). Sample data sets can be printed
to any of the output formats listed below.
</p>

<hr/>

1<label for="dataset">Data set:</label>
<select name="dataset" id="dataset" class="jobexec_sample_select">
  <option value="" selected>&#160;</option>
  <option value="sashelp.retail">SASHELP.RETAIL</option>
  <option value="sashelp.class">SASHELP.CLASS</option>
  <option value="sashelp.revhub2">SASHELP.REVHUB2</option>
  <option value="does_not_exist">does_not_exist</option>
</select>

<br/>
<br/>

2<label for="_output_type">Output format:</label>
<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="" selected>(default)</option>
  <option value="ods_html">HTML</option>
  <option value="ods_html5">HTML5</option>
  <option value="ods_pdf">Portable Document Format (PDF)</option>
  <option value="ods_excel">Microsoft Excel (XLSX)</option>
  <option value="ods_word">Microsoft Word (DOCX)</option>
  <option value="ods_rtf">Rich Text Format (RTF)</option>
  <option value="ods_csv">Comma-separated Value (CSV)</option>
  <option value="ods_xml">Extensible Markup Language (XML)</option>
  <option value="ods_tagsets.rtf">RTF Tagset</option>
  <option value="ods_tagsets.excelxp">Excel (XML)</option>
  <option value="ods_ps">Postscript (PS)</option>
  <option value="ods_latex">LaTeX</option>
</select>

```

```

<br/>
<br/>

3 <label for="_odsstyle">ODS style:</label>
<select name="_odsstyle" id="_odsstyle" class="jobexec_sample_select">
  <option value="" selected>(default)</option>
  [More values here]
</select>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 The first SELECT tag creates a drop-down list with the names of the data set to be displayed, and the first item is selected by default. A macro variable named DATASET is created with the corresponding value in the VALUE attribute of the OPTION tag when you select an item. The first and last items are used to test the cases when no data set is specified and when a non-existent data set is selected, respectively. The SAS program handles these two error conditions.
- 2 The second SELECT tag creates a drop-down list that enables you to select the ODS output format. The first item is selected by default and results in a blank value for the _OUTPUT_TYPE macro variable. The JESBEGIN macro handles this case, and the ODS HTML5 destination creates the output.
- 3 The final SELECT tag creates a drop-down list that enables you to choose the ODS style to apply to your output. The first item is selected by default and results in a blank value for the _ODSSTYLE macro variable. The JESBEGIN macro handles this case, and the output is created using the HTMLBlue ODS style.

The following table contains the display values for the first drop-down list and the values for the DATASET global macro variable:

Display Value	Macro Variable Value
SASHELP.RETAIL	sashelp.retail
SASHELP.CLASS	sashelp.class
SASHELP.REVHUB2	sashelp.revhub2

Display Value	Macro Variable Value
does_not_exist	does_not_exist

The following table contains the display values for the first drop-down list and the values for the `_OUTPUT_TYPE` global macro variable:

Display Value	Macro Variable Value
(default)	
HTML	ods_html
HTML5	ods_html5
Portable Document Format (PDF)	ods_pdf
Microsoft Excel (XLSX)	ods_excel
Microsoft Word (DOCX)	ods_word
Rich Text Format (RTF)	ods_rtf
Comma-Separated Value (CSV)	ods_csv
Extensible Markup Language (XML)	ods_xml
RTF_Tagset	ods_tagsets.rtf
Excel (XML)	ods_tagsets.excelxp
PostScript (PS)	ods_ps
LaTeX	ods_latex

When the form is submitted, the following global macro variables are defined just before SAS code execution, but only the `_ODSDEST` and `DATASET` variables are used in the program code:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	<code>execute</code>
<code>_ODSDEST</code>	Depends on selection (for example, <code>tagsets.rtf</code>)
<code>_ODSSTYLE</code>	Depends on selection (for example, <code>HTMLBlue</code>)

Macro Variable Name	Macro Variable Value
_OUTPUT_TYPE	Depends on selection (see previous table)
_PROGRAM	<i>/Folder/Multiple Output Formats</i>
_DEBUG	log (if the check box is selected)
DATASET	Depends on selection (see previous table)

The _ODSDEST macro variable is derived from the value of _OUTPUT_TYPE and indicates the ODS destination that is used.

The HTML in this form uses some of the same fields as [“Simple ODS HTML” on page 163](#). See that sample for more information.

Program

```

*   Declare input parameter;

%global DATASET;

%macro setup;

%local ERRORTTEXT RC;

%*   Verify that a valid data set was specified;

%if (%qcmpres(&DATASET) eq )
    %then %let ERRORTTEXT=ERROR: You must specify a data set.;
    %else %if not %sysfunc(exist(&DATASET))
        %then %let ERRORTTEXT=ERROR: Data set
            ""%sysfunc(htmlencode(&DATASET))""
            not found.;

%if (%bquote(&ERRORTTEXT) ne ) %then %do;

    %*   Close the currently open destination and write message to the
    browser;

    ods _all_ close;

    %let RC=%sysfunc(fdelete(_webout));

    filename _webout filesrv parenturi="%SYS_JES_JOB_URI"
        name='_webout.htm';

    title;

    ods html5 file=_webout
        text="&ERRORTTEXT";

```

```

ods html5 close;

data _null_;
  abort cancel;
run;
%end;

%mend setup;

%SETUP

title "%sysfunc(htmlencode(%qupcase(&DATASET))) Table in
      %sysfunc(htmlencode(%qupcase(&_ODSDEST))) Format";

proc print data=&DATASET noobs label n; run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return the type of output that is specified in the _OUTPUT_TYPE input parameter to the web browser. If a blank value is specified for the _OUTPUT_TYPE input parameter, then a FILENAME statement for HTML output is issued.
- issuing an ODS statement for the appropriate destination. If a blank value is specified for the _OUTPUT_TYPE input parameter, then the ODS HTML5 destination is used.
- writing ODS output to the _WEBOUT fileref that is assigned by the %JESBEGIN macro. The style specified in the _ODSSYLE input parameter controls the appearance of the output.

The SETUP macro checks the validity of the value specified for the DATASET input parameter. If no value is specified or if the specified data set does not exist:

- An error message is created and returned to the web browser using the HTML5 ODS destination.
- The HTMLENCODE function is used here, and later in the TITLE statement, to prevent execution of malicious code in the web browser.
- All ODS destinations are closed, and the current content of the _WEBOUT fileref is deleted to ensure that only the error message is returned to the web browser. A FILENAME statement is issued to return HTML output to the web browser, and an ODS statement for the HTML5 destination writes the error message to the _WEBOUT fileref. See [“Assigning a Fileref for HTML Output” on page 62](#) for more information about this type of FILENAME statement.
- No additional program statements are executed after the ABORT statement executes.

The PRINT procedure displays the data in the specified data set.

The %JESSEND macro executes after the last line of code and closes all open ODS destinations.

HTML and PDF output are displayed by the web browser, and all other output is downloaded so that it can be saved and opened with an appropriate application.

Output

The following HTML output is one representation of the variety of formats that ODS can produce.

SASHELP.CLASS Table in HTML5 Format

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0
N = 19				

ODS Output with Embedded Graphics

This sample creates HTML, PDF, XLSX, DOCX, or RTF output using ODS and returns it to the web browser. An HTML input form, which accepts the ODS destination, style, and graphic output type, provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	ods_html5	Specifies that ODS HTML5 output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

ODS Output with Embedded Graphics

The SGPLOT procedure creates a bar chart using the SASHELP.CLASS table, followed by a display of the data using the PRINT procedure.

The SVG image format is not supported for RTF output.

Output format:

Graphic image format:

ODS style:

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
<title>ODS Output with Embedded Graphics</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">ODS Output with Embedded Graphics</h1>

<p>
The SGPLOT procedure creates a bar chart using the SASHELP.CLASS
table, followed by a display of the data using the PRINT procedure.
</p>

<p>
The SVG image format is not supported for the RTF output.
</p>

<label for="_output_type">Output format:</label>
1<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="ods_html5">HTML5</option>
  <option value="ods_pdf">Portable Document Format (PDF)</option>
  <option value="ods_excel">Microsoft Excel (XLSX)</option>
  <option value="ods_word">Microsoft Word (DOCX)</option>
  <option value="ods_rtf">Rich Text Format (RTF)</option>
</select>

<br/>
<br/>

<label for="_ods_device">Graphic image format:</label>
2<select name="_ods_device" id="_ods_device"
class="jobexec_sample_select">
  <option value="" selected>(default based on output format)</option>
  <option value="svg">SVG</option>
  <option value="png">PNG</option>
</select>

<br/>
<br/>

<label for="_odsstyle">ODS style:</label>
3<select name="_odsstyle" id="_odsstyle"
class="jobexec_sample_select">
  [More values here]
  <option value="HTMLBlue" selected>HTMLBlue</option>
  [More values here]
</select>

```

```

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 The first SELECT tag creates a drop-down list with output format values. A macro variable named `_OUTPUT_TYPE` is created with the corresponding value in the `VALUE` attribute of the `OPTION` tag.
- 2 The second SELECT tag creates a drop-down list that enables you to select the ODS graphic image format. The first item is initially selected and results in a blank value for the `_ODS_DEVICE` macro variable. The blank value causes a default graphic image format to be used. That format is based on the value specified for `_OUTPUT_TYPE`.
- 3 The final SELECT tag creates a drop-down list that enables you to select the ODS style to apply to your output. The HTMLBlue style is selected by default. The selected value is stored in the `_ODSSTYLE` global macro variable.

Specify `$PROGRAM$` as the value for the `_PROGRAM` INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

The following table contains the display values for the first drop-down list and the corresponding values for the `_OUTPUT_TYPE` global macro variable:

Display Value	Macro Variable Value
HTML5	<code>ods_html5</code>
Portable Document Format (PDF)	<code>ods_pdf</code>
Microsoft Excel (XLSX)	<code>ods_excel</code>
Microsoft Word (DOCX)	<code>ods_word</code>
Rich Text Format (RTF)	<code>ods_rtf</code>

The following table contains the display values for the second drop-down list and the values for the `_ODS_DEVICE` global macro variable:

Display Value	Macro Variable Value
SVG	<code>svg</code>
PNG	<code>png</code>

When the form is submitted, the following global macro variables are defined just before SAS code execution, but they are not used in the program code:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	<code>execute</code>
<code>_ODS_DEVICE</code>	Depends on selection (blank, <code>svg</code> , or <code>png</code>)
<code>_ODSDEST</code>	Depends on selection (<code>html5</code> , <code>pdf</code> , <code>excel</code> , <code>word</code> , or <code>rtf</code>)
<code>_ODSSTYLE</code>	Depends on selection (for example, <code>HTMLBlue</code>)
<code>_OUTPUT_TYPE</code>	Depends on selection (<code>ods_html5</code> , <code>ods_pdf</code> , <code>ods_excel</code> , <code>ods_word</code> , or <code>ods_rtf</code>)
<code>_PROGRAM</code>	<code>/Folder/ODS Output with Embedded Graphics</code>
<code>_DEBUG</code>	<code>log</code> (if the check box is selected)

The `_ODSDEST` macro variable is derived from the value of `_OUTPUT_TYPE` and indicates the ODS destination that is used.

The HTML in this form uses some of the same fields as “Simple ODS HTML” on [page 163](#). See that sample for more information.

Program

```

title 'Student Data - SGPLOT';
proc sgplot data=sashelp.class; hbar age; run; quit;

title 'Student Data';
proc print data=sashelp.class; run; quit;

```

Program Description

The `%JESBEGIN` and `%JESEND` macros are included by default and do not usually need to be added. The `%JESBEGIN` macro performs several tasks before executing the code, including:

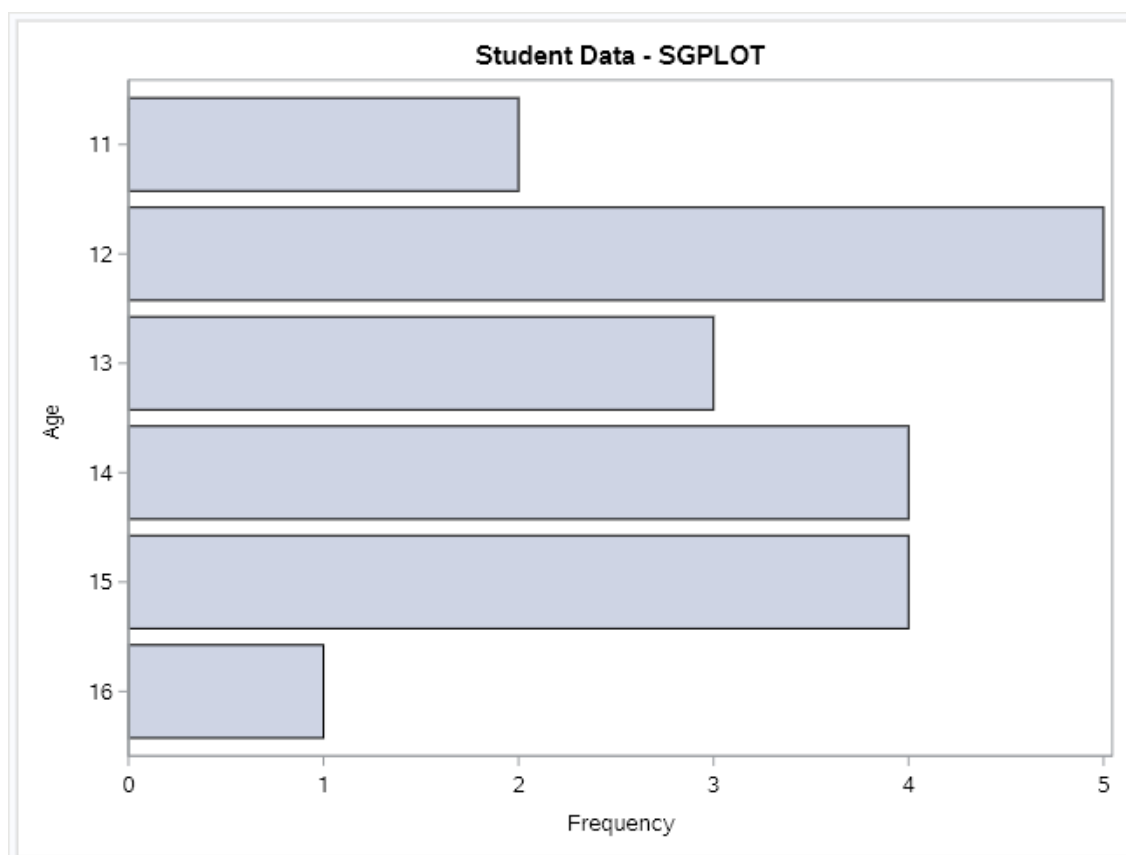
- issuing a FILENAME statement to return the type of output that is specified in the _OUTPUT_TYPE input parameter to the web browser. If a blank value is specified for the _OUTPUT_TYPE input parameter, then a FILENAME statement for HTML output is issued.
- issuing an ODS statement for the HTML5, PDF, EXCEL, WORD, or RTF destination, based on the value specified for the _OUTPUT_TYPE input parameter.
- writing ODS output to the _WEBOUT fileref that is assigned by the %JESBEGIN macro. The style specified in the _ODSSYLE input parameter controls the appearance of the output.
- issuing an ODS GRAPHICS statement that specifies the graphic image format, based on the value specified for the _ODS_DEVICE input parameter. If no value is specified for _ODS_DEVICE, then the best format is used, based on the ODS destination.

The SGPLOT procedure creates the graphic image using data from the SASHELP.CLASS data set, and the PRINT procedure displays the data.

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

The HTML output and the PDF output are displayed by the web browser, and the other output is downloaded so that it can be saved and opened with an appropriate application.

Output



Student Data

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	58.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

Report with Download Links

This sample creates a report in HTML format and provides links to download the report in Excel Spreadsheet XML and PDF formats. An HTML input form provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	none	Suppresses automatic issuing of FILENAME and ODS statements
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS[®] Job Execution

Report with Download Links

The PRINT procedure displays data in the SASHELP.CLASS table in HTML format. Links are provided to download the output in the Excel and PDF formats.

ODS style:

☐ Show SAS Log

The following HTML is the source for the input form:

```

<!DOCTYPE html>
<html lang="en">

<head>
<title>Report with Download Links</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<input type="hidden" name="_output_type" value="none"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Report with Download Links</h1>

<p>
The PRINT procedure displays data in the SASHELP.CLASS table in HTML
format. Links are provided
to download the output in the Excel and PDF formats.
</p>

<label for="_odsstyle">ODS style:</label>
<select name="_odsstyle" id="_odsstyle" class="jobexec_sample_select"
required>
  [More values here]
  <option value="HTMLBlue" selected>HTMLBlue</option>
  [More values here]
</select>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

The HTML in this form uses some of the same fields as [“Simple ODS HTML” on page 163](#).

Specify `$PROGRAM$` as the value for the `_PROGRAM` INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

The FILENAME and ODS statements issued by the %JESBEGIN macro do not meet the needs of this program. Specifying **none** for _OUTPUT_TYPE suppresses the issuing of these statements. The appropriate statements are issued within the program.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but only _ODSSTYLE is used in the program code:

Macro Variable Name	Macro Variable Value
_ACTION	execute
_ODSDEST	blank
_ODSSTYLE	Depends on selection (for example, HTMLBlue)
_OUTPUT_TYPE	none
_PROGRAM	<i>/Folder/Report with Download Links</i>
_DEBUG	log (if the check box is selected)

Program

```

* Close all open destinations;

ods _all_ close;

options nodate nonumber;

* Declare input parameter;

%global _ODSSTYLE;

* Define the escape character for ODS inline formatting;

ods escapechar='^';

* Create a format for the student gender;

proc format;
  value $gender 'F' = 'Female'
               'M' = 'Male';
run; quit;

* Prepare the data;

proc sql;
  create view work.class as
  select name label = 'Name',

```

```

        sex label    = 'Gender' format=$gender.,
        age label    = 'Age',
        height label = 'Height',
        weight label = 'Weight'
    from sashelp.class
    order by sex;
quit;

* Create an ODS document with the report results;

title1 'The CLASS Table';
footnote;

1 ods document name=work.mydoc(write);

proc print data=work.class noobs n label;
    by sex;
    var name age height weight;
run; quit;

ods document close;

* Create the Excel XML and PDF output and associate with the job;

2 filename f_xlzp filesrv parenturi="%SYS_JES_JOB_URI"
  name='Class.xml'
  contenttype='application/vnd.ms-excel'
  contentdisp='attachment; filename="Class.xml"';

3 filename f_pdf filesrv parenturi="%SYS_JES_JOB_URI"
  name='Class.pdf'
  contenttype='application/pdf';

ods pdf file=f_pdf style=&_ODSSTYLE;

ods tagsets.ExcelXP file=f_xlzp style=&_ODSSTYLE
  options(embedded_titles='yes'
    suppress_bylines='yes'
    sheet_name='#byval(sex) Students'
    print_header='&C&A');

proc document name=work.mydoc;
    replay;
run; quit;

ods pdf close;
ods tagsets.ExcelXP close;

* Create download links;

4 %let EXCEL_LINK=%bquote(<a href="%&_FILESRV_F_XLXP_URI/content"
  target="_SASDLResults">Excel</a>);

4 %let PDF_LINK=%bquote(<a href="%&_FILESRV_F_PDF_URI/content"
  target="_SASDLResults">PDF</a>);

```

* Create the HTML output for display in the Web browser;

```
5 filename f_htm filesrvc parenturi "&SYS_JES_JOB_URI"
   name='_webout.htm';

6 ods html5 file=f_htm style=&_ODSSTYLE
   text="<span>^{style systemtitle &EXCEL_LINK^{nbspace 3}
   &PDF_LINK}</span>";

proc document name=work.mydoc;
   replay;
run; quit;

ods html5 close;
```

- 1 The ODS DOCUMENT statement stores the output components from the PRINT procedure in a document named MYDOC. You can later use PROC DOCUMENT to display the results using any ODS destination. This technique is useful when you need to display procedure output several times, but you do not want to rerun the procedure.

- 2 FILENAME and ODS statements are not issued by the %JESBEGIN macro because _OUTPUT_TYPE=none is specified. A FILENAME statement is issued to store the Microsoft XML results, following the general format discussed in [“Assigning a Fileref for Other Types of Output” on page 63](#).

The CONTENTTYPE option creates a MIME header that informs the web browser that the content is intended for a client capable of rendering Excel output. The CONTENTDISP option causes the Excel XML output to be downloaded as a file named Class.xml.

- 3 The next FILENAME statement stores PDF output generated by the ODS PDF destination. The CONTENTTYPE option creates a MIME header that informs the web browser that the content is intended for a client capable of rendering PDF output. Most web browsers can display PDF output, so the CONTENTDISP option is not needed.
- 4 As discussed in [“Assigning a Fileref for HTML Output” on page 62](#), the FILESRVC access method creates global macro variables of the form _FILESRVC_fileref_URI, where fileref is the fileref used in the FILENAME statement. This macro variable provides a relative URL that can be used to reference and retrieve the file using the Files service. These macro variables are used to create the EXCEL_LINK and PDF_LINK macro variables. Later, EXCEL_LINK and PDF_LINK are used to create download links.
- 5 A FILENAME statement is issued to return HTML content to the web browser. See [“Assigning a Fileref for HTML Output” on page 62](#) for more information about this statement.
- 6 The ODS HTML5 destination creates the HTML output using the ODS style specified in the _ODSSTYLE input parameter. The macro variables created earlier provide download links to the Excel XML and the PDF versions of the report. The NBSPACE inline formatting function provides extra blank space between the link text in the output. See [“NBSPACE Function” in SAS Output Delivery System: User’s Guide](#) for more information about the NBSPACE inline formatting function.

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro normally performs several tasks before executing your code. In this case, only a list of global macro variables is displayed because `_OUTPUT_TYPE=none` is specified.

See [“ODS ESCAPECHAR Statement” in SAS Output Delivery System: User’s Guide](#) for more information about ODS ESCAPECHAR.

The SQL procedure selects the data of interest, applies a user-defined format, and then sorts the data.

PROC DOCUMENT executes and replays the PROC PRINT output created earlier and stores it in the XML and PDF files. The ODS destinations are closed after PROC DOCUMENT creates the output files. See [“DOCUMENT Procedure” in SAS Output Delivery System: Procedures Guide](#) for more information about PROC DOCUMENT.

PROC DOCUMENT then executes and replays the PROC PRINT output created earlier, and the HTML output is displayed in the web browser. The %JESEND macro executes after the last line of code, but it does not close all open ODS destinations because `_OUTPUT_TYPE=none` is specified. The HTML5 destination must be explicitly closed.

Output

The CLASS Table

[Excel](#) [PDF](#)

Gender=Female

Name	Age	Height	Weight
Judy	14	64.3	90.0
Jane	12	59.8	84.5
Joyce	11	51.3	50.5
Barbara	13	65.3	98.0
Carol	14	62.8	102.5
Mary	15	66.5	112.0
Louise	12	56.3	77.0
Alice	13	58.5	84.0
Janet	15	62.5	112.5
N = 9			

Gender=Male

Name	Age	Height	Weight
Philip	16	72.0	150.0
James	12	57.3	83.0
Henry	14	63.5	102.5
John	12	59.0	99.5
William	15	66.5	112.0
Alfred	14	69.0	112.5
Jeffrey	13	62.5	84.0
Thomas	11	57.5	85.0
Ronald	15	67.0	133.0
Robert	12	64.8	128.0
N = 10			

Return GCHART Image Output

This sample uses the GCHART procedure to create a bar chart using the SASHELP.CLASS data set, and returns the image data to the client. An HTML input form provides a basic user interface to the program, but a more appropriate use is including SAS generated graphic output in another web application, such as a dashboard. In this case, you can specify the URL in an IMG tag: ``

Note: A SAS/GRAPH license is required to run this sample.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	png	Specifies that binary PNG output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS[®] Job Execution

Return GCHART Image Output

The GCHART procedure creates a bar chart using the SASHELP.CLASS table, and returns the binary image data to the client.

Output format: PNG ▼

ODS style: HTMLBlue ▼

Run code

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Return GCHART Image Output</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
```

```

</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
1<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Return GCHART Image Output</h1>

<p>
The GCHART procedure creates a bar chart using the SASHELP.CLASS
table, and returns the binary image data to the client.
</p>

<label for="_output_type">Output format:</label>
2<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="gif">GIF</option>
  <option value="jpeg">JPEG</option>
  <option value="svg">SVG</option>
  <option value="png" selected>PNG</option>
</select>

<br/>
<br/>

<label for="_odsstyle">ODS style:</label>
3<select name="_odsstyle" id="_odsstyle"
class="jobexec_sample_select">
  [More values here]
  <option value="HTMLBlue" selected>HTMLBlue</option>
  [More values here]
</select>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 Specify **\$PROGRAM\$** as the value for the **_PROGRAM** INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for **\$PROGRAM\$**.

- 2 The first SELECT tag creates a drop-down list that enables you to select the SAS/GRAPH graphic image format. **PNG** is selected by default. The selected value is stored in the `_OUTPUT_TYPE` global macro variable.
- 3 The second SELECT tag creates a drop-down list that enables you to select the ODS style to apply to the graphic output. The **HTMLBlue** style is selected by default. The selected value is stored in the `_ODSTYLE` global macro variable.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but only `_ODSTYLE` and `_OUTPUT_TYPE` are used in the program code:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	execute
<code>_ODSTYLE</code>	Depends on selection (for example, HTMLBlue)
<code>_OUTPUT_TYPE</code>	Depends on selection (gif , jpeg , svg , or png)
<code>_PROGRAM</code>	<i>/Folder/Return GCHART Image Output</i>
<code>_DEBUG</code>	log (if the check box is selected)

The HTML in this form uses some of the same fields as [“Simple ODS HTML” on page 163](#).

Program

```

* Declare input parameters;

%global _ODSTYLE
        _OUTPUT_TYPE;

* Specify the ODS style and graphic output format;

ods listing style=&_ODSTYLE;

options gsfname=_webout gsfmode=replace device=&_OUTPUT_TYPE;

* Create the graphic output;

proc gchart data=sashelp.class; vbar age / discrete; run; quit;
```

Program Description

The `%JESBEGIN` and `%JESEND` macros are included by default and do not usually need to be added. The `%JESBEGIN` macro performs several tasks before executing

the code, including issuing a `FILENAME` statement to return the type of output that is specified in the `_OUTPUT_TYPE` input parameter to the web browser.

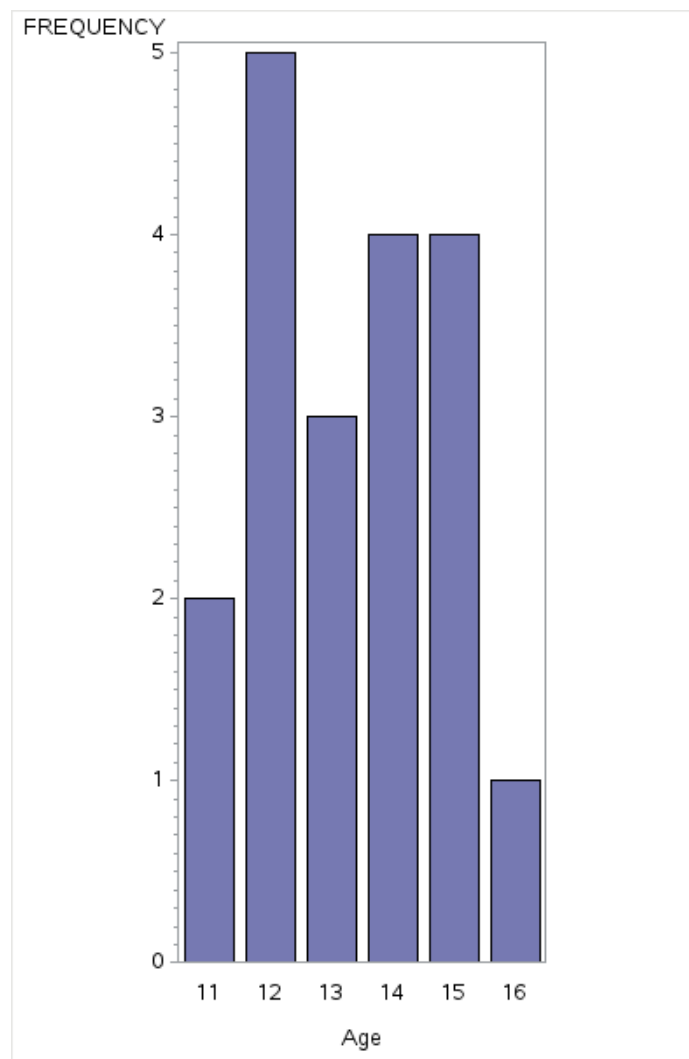
The ODS style specified in the `_ODSSTYLE` parameter controls the graphic output appearance.

The ODS Graphics option to specify the output format is specified.

The `GCHART` procedure creates the graphic image using data from the `SASHELP.CLASS` data set.

The `%JESEND` macro executes after the last line of code and closes all open ODS destinations. The image output is displayed by the web browser, or within an `IMG` tag.

Output



Return SGPLOT Image Output

This sample uses the SGPLOT procedure to create a bar chart using the SASHELP.CLASS data set, and returns the image data to the client. An HTML input form provides a basic user interface to the program, but a more appropriate use is including SAS generated graphic output in another web application, such as a dashboard. In this case, you can specify the URL in an IMG tag: ``

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	png	Specifies that binary PNG output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Return SGPLOT Image Output

The SGPLOT procedure creates a bar chart using the SASHELP.CLASS table, and returns the binary image data to the client.

Output format:

ODS style:

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Return SGPLOT Image Output</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
1<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Return SGPLOT Image Output</h1>

<p>
The SGPLOT procedure creates a bar chart using the SASHELP.CLASS
table, and returns the binary image data to the client.
</p>

<label for="_output_type">Output format:</label>
2<select name="_output_type" id="_output_type"
class="jobexec_sample_select">
  <option value="gif">GIF</option>
  <option value="jpeg">JPEG</option>
  <option value="svg">SVG</option>
  <option value="png" selected>PNG</option>
</select>
```

```

<br/>
<br/>

<label for="_odsstyle">ODS style:</label>
3 <select name="_odsstyle" id="_odsstyle"
class="jobexec_sample_select">
  [More values here]
  <option value="HTMLBlue" selected>HTMLBlue</option>
  [More values here]
</select>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 Specify `$PROGRAM$` as the value for the `_PROGRAM` INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.
- 2 The first SELECT tag creates a drop-down list that enables you to select the ODS graphics output format. `PNG` is selected by default. The selected value is stored in the `_OUTPUT_TYPE` global macro variable.
- 3 The second SELECT tag creates a drop-down list that enables you to select the ODS style to apply to the graphic output. The `HTMLBlue` style is selected by default. The selected value is stored in the `_ODSTYLE` global macro variable.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but only `_ODSSTYLE` and `_OUTPUT_TYPE` are used in the program code:

Macro Variable Name	Macro Variable Value
<code>_ACTION</code>	<code>execute</code>
<code>_ODSSTYLE</code>	Depends on selection (for example, <code>HTMLBlue</code>)
<code>_OUTPUT_TYPE</code>	Depends on selection (<code>gif</code> , <code>jpeg</code> , <code>svg</code> , or <code>png</code>)
<code>_PROGRAM</code>	<code>/Folder/Return SGPlot Image Output</code>
<code>_DEBUG</code>	<code>log</code> (if the check box is selected)

The HTML in this form uses some of the same fields as [“Simple ODS HTML” on page 163](#). See that sample for more information.

Program

```
* Declare input parameters;

%global _ODSSTYLE
        _OUTPUT_TYPE;

* Specify the ODS style and graphic output format;

ods listing style=&_ODSSTYLE;

ods graphics / outputfmt=&_OUTPUT_TYPE;

* Create the graphic output;

proc sgplot data=sashelp.class; hbar age; run; quit;
```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including issuing a FILENAME statement to return the type of output that is specified in the _OUTPUT_TYPE input parameter to the web browser.

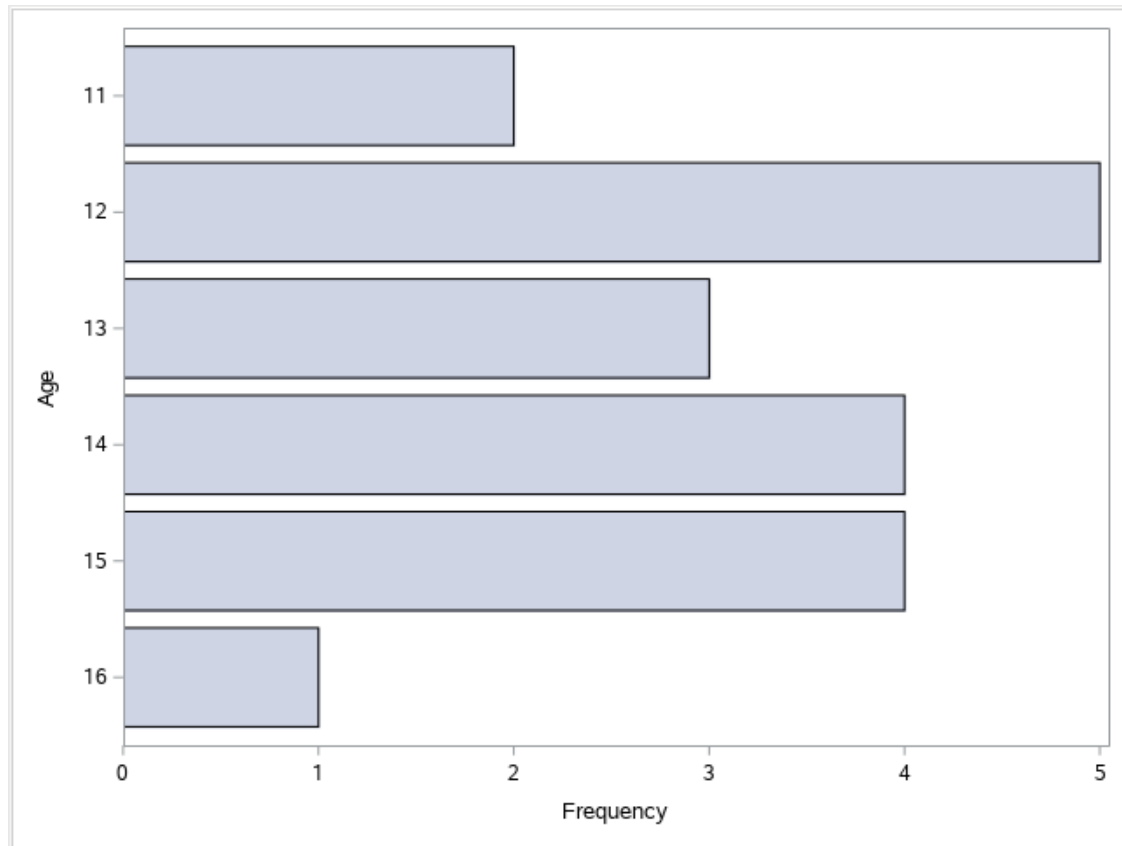
The ODS style specified in the _ODSSTYLE parameter controls the graphic output appearance.

The ODS Graphics option to specify the output format is specified.

The SGPLOT procedure creates the graphic image using data from the SASHELP.CLASS data set.

The %JESEND macro executes after the last line of code and closes all open ODS destinations. The image output is displayed by the web browser, or within an IMG tag.

Output



Simple JSON

This sample uses PROC JSON to display the SASHELP.CLASS data set in JSON format. Use this technique and execute the job using direct URL access if you have an application that requires data in JSON format, as discussed in [“Returning JSON Data from a Job” on page 91](#). An HTML input form provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	json	Specifies that JSON (non-ODS) output is created by the job
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Simple JSON

The JSON procedure creates simple JSON content that returns the data in the SASHELP.CLASS table.

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Simple JSON</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
<input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<input type="hidden" name="_output_type" value="json"/>
<div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

<h1 class="jobexec_sample_name">Simple JSON</h1>

<p>
The JSON procedure creates simple JSON content that returns the data
in the SASHELP.CLASS table.
</p>

<hr/>
```

```

☐

```

The HTML in this form uses some of the same fields as “Simple ODS HTML” on [page 163](#). See that sample for more information.

Specify `$PROGRAM$` as the value for the `_PROGRAM` INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for `$PROGRAM$`.

The value of `_OUTPUT_TYPE` indicates that non-ODS JSON output is created by the job. The `%JESBEGIN` macro issues a `FILENAME` statement that supports JSON output.

Program

```

proc json out=_webout nosastags pretty;
  export sashelp.class;
run; quit;

```

Program Description

The `%JESBEGIN` and `%JESEND` macros are included by default and do not usually need to be added. The `%JESBEGIN` macro performs several tasks before executing the code, including issuing a `FILENAME` statement to return JSON output to the web browser because `json` is specified as the value for the `_OUTPUT_TYPE` input parameter.

The JSON procedure converts the `SASHELP.CLASS` data set to JSON format and then writes JSON data to the `_WEBOUT` fileref that is assigned by the `%JESBEGIN` macro. The JSON is displayed by the web browser.

The `%JESEND` macro executes after the last line of code, but it does not close all open ODS destinations because this sample does not use ODS.

This technique is useful if you have an application such as a JavaScript grid or chart object that requires data in JSON format. In your application, specify the following URL to retrieve the data in JSON format:

```
http://host:port/SASJobExecution/?_program=/Folder/Simple JSON&_action=execute
```

Use the concepts in this sample if your application requires data in another format, such as XML or CSV.

Output

```
[
  {
    "Name": "Alfred",
    "Sex": "M",
    "Age": 14,
    "Height": 69,
    "Weight": 112.5
  },
  {
    "Name": "Alice",
    "Sex": "F",
    "Age": 13,
    "Height": 56.5,
    "Weight": 84
  },
  {
    "Name": "Barbara",
    "Sex": "F",
    "Age": 13,
    "Height": 65.3,
    "Weight": 98
  }, ...
]
```

Simple ODS HTML

This sample creates HTML output using ODS and returns it to the web browser. An HTML input form provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,prompts,execute	Displays the HTML input form or prompts if available; otherwise, executes the job
_OUTPUT_TYPE	ods_html5	Specifies that ODS HTML5 output is created by the job

Name	Value	Description
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

1 SAS® Job Execution

2 Simple ODS HTML

The PRINT procedure creates a simple HTML page that displays the data in the SASHELP.CLASS table.

3 ☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Simple ODS HTML</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" target="_tab">
3 <input type="hidden" name="_program" value="$PROGRAM$"/>
<input type="hidden" name="_action" value="execute"/>
<input type="hidden" name="_output_type" value="ods_html5"/>
1 <div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job
Execution</div>

2 <h1 class="jobexec_sample_name">Simple ODS HTML</h1>

<p>
The PRINT procedure creates a simple HTML page that displays the data
in the SASHELP.CLASS table.
</p>

<hr/>
<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
```

```

        class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 This HTML is similar to the HTML in the Hello World input form.
- 2 This HTML is similar to the HTML in the Hello World input form.
- 3 Specify **\$PROGRAM\$** as the value for the **_PROGRAM** INPUT tag. When the HTML input form is displayed, the path and name of the program to execute are substituted for **\$PROGRAM\$**.

_ACTION is set to **execute** to ensure that the program executes when you click **Run Code**. The value of **_OUTPUT_TYPE** indicates that ODS HTML5 output is created by the job. The **%JESBEGIN** macro issues a **FILENAME** statement that supports HTML output, and it issues an ODS statement using the HTML5 destination.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but they are not used in the program code:

Macro Variable Name	Macro Variable Value
_ACTION	execute
_ODSDEST	html5
_ODSSTYLE	HTMLBlue
_OUTPUT_TYPE	ods_html5
_PROGRAM	/Folder/Simple ODS HTML
_DEBUG	log (if the check box is selected)

The **_ODSDEST** macro variable is derived from the value of **_OUTPUT_TYPE**, and it indicates the ODS destination that is used.

Program

```

title 'Student Data';
proc print data=sashelp.class noobs;
  var name sex age height weight;
run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return HTML output to the web browser because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- issuing an ODS statement for the HTML5 destination because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- writing ODS output to the `_WEBOUT` fileref that is assigned by the %JESBEGIN macro. The HTMLBlue style, specified in the `_ODSSYLE` input parameter, controls the appearance of the output.

The PRINT procedure displays the SASHELP.CLASS data set.

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

Output

Student Data				
Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0

Upload a CSV File

This sample uploads a Comma-Separated Value (CSV) file to the server, imports it into a SAS data set, and then displays the first 10 records of the SAS data set. An HTML input form, which enables you to select a file to upload, provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,execute	Displays the HTML input form before the job is executed
_OUTPUT_TYPE	ods_html5	Specifies that ODS HTML5 output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Upload a CSV File

Use this page to upload a CSV file from your local machine to the SAS server machine. The program imports the file into a SAS table and then uses PROC PRINT to display the first 10 records.

Optionally, you can choose to download the file from the SAS server machine by clicking a link in the output.

Choose a CSV file to upload: No file chosen

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Upload a CSV File</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

<form class="jobexec_form" action="/SASJobExecution/" method="post"
target="_tab"
  enctype="multipart/form-data">
  <input type="hidden" name="_program" value="$PROGRAM$"/>
  <input type="hidden" name="_action" value="execute"/>
  <input type="hidden" name="_output_type" value="ods_html5"/>
  <input type="hidden" name="_csrf" value="$CSRF$"/>
  <div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

  <h1 class="jobexec_sample_name">Upload a CSV File</h1>

  <p>
  Use this page to upload a CSV file from your local
  machine to the SAS server machine. The program imports the
  file into a SAS table and then uses PROC PRINT to display the
  first 10 records.
  </p>

  <p>
```

Optionally, you can choose to download the file from the SAS server machine by clicking a link in the output.

```
</p>

<hr/>
<br/>

<label for=" myfile">Choose a CSV file to upload:</label>
<input type="file" name="myfile" id=" myfile" required
class="jobexec_sample_input_file"/>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="pointerjobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>
```

This input form is functionally equivalent to the form used in [“Upload a File” on page 172](#).

The file that you choose to upload should be a CSV file with the .csv file extension.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but they are not used in the program code:

Macro Variable Name	Macro Variable Value
_CSRF	Cross-Site Request Forgery token for this request
_ACTION	execute
_ODSDEST	html5
_ODSSTYLE	HTMLBlue
_OUTPUT_TYPE	ods_html5
_PROGRAM	/Folder/Upload a CSV File
_DEBUG	log (if the check box is selected)

Additional macro variables with information about the uploaded file are created. See [“Upload a File” on page 172](#) for more information.

Program

```

* Check the file extension to verify that it is a CSV file;

data _null_;
length filename $1024;
filename = htmlencode(strip("&_WEBIN_FILENAME"));
call symputx('_WEBIN_FILENAME', filename);
if (upcase("&_WEBIN_FILEEXT") ne 'CSV') then do;
    rc = dosubl('ods all close;');
    file _webout;
    put '<!DOCTYPE html>';
    put '<html lang="en">';
    put '<head><title>Program Error</title></head>';
    put '<body role="main">';
    put '<h1>ERROR: Uploaded file "' filename +(-1) '" is not a CSV
file.</h1>';
    put '</body>';
    put '</html>';
    abort cancel;
end;
run;

* Create a FILEREF for the uploaded file;

filename upload filesrvc parenturi="&SYS_JES_JOB_URI"
name="&_WEBIN_FILENAME"
contenttype="&_WEBIN_CONTENT_TYPE";

* Set options to support non-SAS name;

options validvarname=any validmemname=extend;

* Import the uploaded CSV file;

proc import datafile=upload
    out=work.mydata
    dbms=csv
    replace;
    getnames=yes;
run; quit;

title 'First 10 Records of Uploaded File ' "&_WEBIN_FILENAME";
footnote link="&_WEBIN_FILEURI/content" 'Click here to download file';

proc print data=work.mydata(obs=10)
    style(header)=[just=center]
    style(column)=[verticalalign=middle];
run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return HTML output to the web browser because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- issuing an ODS statement for the HTML5 destination because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- writing ODS output to the `_WEBOUT` fileref that is assigned by the %JESBEGIN macro. The HTMLBlue style, specified in the `_ODSSYLE` input parameter, controls the appearance of the output.

Additional macro variables with information about the uploaded file are created during the upload process. See [“Upload a File” on page 172](#) for more information.

The HTMLENCODE function encodes the values of input parameters to prevent execution of malicious code in the web browser.

If the value of the `_WEBIN_FILEEXT` macro variable indicates that a CSV file was not uploaded, then action is taken to prevent further code execution. The DOSUBL function closes all open ODS destinations, and an error message is returned to the web browser. No additional program statements are executed after the ABORT statement executes.

The FILENAME statement creates a reference to the temporary location of the uploaded file. The file can be used with PROC IMPORT or with any other code that accepts a fileref. For example, you can use the FCOPY function to make a copy of the file.

PROC IMPORT creates the WORK.MYDATA data set from the uploaded CSV file.

The PRINT procedure displays the first 10 records of the SAS data set that is created by the IMPORT procedure, and a link to download the CSV file is created by the FOOTNOTE statement.

The %JESEND macro executes after the last line of code and closes all open ODS destinations.

Output

The following output is an example of a CSV file that could be uploaded.

First 10 Records of Uploaded File "class.csv"

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5

[Click here to download file](#)

Upload a File

This sample uploads an arbitrary file to the server and then displays information about the file. An HTML input form, which enables you to select a file to upload, provides a basic user interface to the program.

Job Definition Parameters

The following parameters are defined as part of the job definition and can be viewed on the **Parameters** tab in the Properties window:

Name	Value	Description
_ACTION	form,execute	Displays the HTML input form before the job is executed
_OUTPUT_TYPE	ods_html5	Specifies that ODS HTML5 output is created by the job
_ODSSTYLE	HTMLBlue	Specifies the name of the ODS style
_contextName	SAS Job Execution compute context	Specifies the compute context for executing the job

HTML Input Form

SAS® Job Execution

Upload a File

Use this page to upload a file from your local machine to the SAS server machine. The program displays the macro variables with information about the file.

Optionally, you can choose to download the file from the SAS server machine by clicking a link in the output.

Choose a file to upload: No file chosen

☐ Show SAS Log

The following HTML is the source for the input form:

```
<!DOCTYPE html>
<html lang="en">

<head>
<title>Upload a File</title>
<link rel="stylesheet" href="/SASJobExecution/theme">
</head>

<body role="main" class="jobexec_body">

1 <form class="jobexec_form" action="/SASJobExecution/" method="post"
target="_tab"
  enctype="multipart/form-data">
  <input type="hidden" name="_program" value="$PROGRAM$"/>
  <input type="hidden" name="_action" value="execute"/>
  <input type="hidden" name="_output_type" value="ods_html5"/>
2 <input type="hidden" name="_csrf" value="$CSRF$"/>
  <div class="jobexec_sample_header">SAS<sup>&#174;</sup> Job Execution</div>

  <h1 class="jobexec_sample_name">Upload a File</h1>

  <p>
    Use this page to upload a file from your local
    machine to the SAS server machine. The program displays
    the macro variables with information about the file.
  </p>

  <p>
    Optionally, you can choose to download the file from the
    SAS server machine by clicking a link in the output.
```

```

</p>

<hr/>

<br/>
<label for="myfile">Choose a file to upload:</label>
3 <input type="file" name="myfile" id="myfile" required
class="jobexec_sample_input_file"/>

<br/>
<br/>

<hr/>

<input type="submit" value="Run code"
class="jobexec_sample_input_submit"/>
<input type="checkbox" name="_debug" id="_debug" value="log"
class="jobexec_sample_input_checkbox"/>
<label for="_debug">Show SAS Log</label>

</form>

</body>

</html>

```

- 1 When you upload a file, be sure to specify the METHOD and ENCTYPE attributes exactly as shown.
- 2 When you make a POST request, you must specify the _CSRF input parameter exactly as shown. This tag ensures that the request is considered non-malicious by sending a Cross-Site Request Forgery token to the server. When the HTML input form is displayed, the Cross-Site Request Forgery token for this request is substituted for \$CSRF\$.
- 3 A special form of the INPUT tag displays a file selector window when you click a button. Navigate to the file that you want to upload, select it, and then run the code. The file is transferred to the SAS server and stored in the SAS Infrastructure Data Server using the Files service.

Specify \$PROGRAM\$ as the value for the _PROGRAM input parameter. When the HTML input form is displayed, the path and name of the program to execute are substituted for \$PROGRAM\$.

The HTML in this form has some of the same elements and features of the form used in “Simple ODS HTML” on page 163. See that sample for more information.

When the form is submitted, the following global macro variables are defined just before SAS code execution, but they are not used in the program code:

Macro Variable Name	Macro Variable Value
_CSRF	Cross-Site Request Forgery token for this request
_ACTION	execute

Macro Variable Name	Macro Variable Value
<code>_ODSDEST</code>	<code>html5</code>
<code>_ODSSTYLE</code>	<code>HTMLBlue</code>
<code>_OUTPUT_TYPE</code>	<code>ods_html5</code>
<code>_PROGRAM</code>	<code>/Folder/Upload a File</code>
<code>_DEBUG</code>	<code>log</code> (if the check box is selected)

The `_ODSDEST` macro variable is derived from the value of `_OUTPUT_TYPE` and indicates the ODS destination that is used.

Additional macro variables with information about the uploaded file are created. See the Output and Program Description sections for more information.

Program

```
* Create a data set with information about the upload;

data work.upload_info;

length varname $25 value $1024 description $256;

varname      = '_WEBIN_CONTENT_LENGTH';
value        = symget('_WEBIN_CONTENT_LENGTH');
description = 'Specifies the size of the file that was uploaded in
bytes
(supplied automatically by the Web browser).';
output;

varname      = '_WEBIN_CONTENT_TYPE';
value        = resolve(symget('_WEBIN_CONTENT_TYPE'));
description = 'Specifies the content type that corresponds to the file
that was uploaded
(supplied automatically by the Web browser).';
output;

varname      = '_WEBIN_FILE_COUNT';
value        = symget('_WEBIN_FILE_COUNT');
description = 'Specifies the number of files that were uploaded.';
output;

varname      = '_WEBIN_FILEEXT';
value        = resolve(symget('_WEBIN_FILEEXT'));
description = 'Specifies the extension of the file that was uploaded.';
output;

varname      = '_WEBIN_FILENAME';
```

```

value          = resolve(symget('_WEBIN_FILENAME'));
description = 'Specifies the name and original location of the file
that was uploaded.';
output;

varname        = '_WEBIN_FILEURI';
value          = resolve(symget('_WEBIN_FILEURI'));
description = 'Specifies the URI of the location of the uploaded file';
output;

varname        = '_WEBIN_NAME';
value          = resolve(symget('_WEBIN_NAME'));
description = 'Specifies the value that corresponds to the NAME
attribute of the INPUT tag.';
output;

label varname    = 'Variable Name'
      value      = 'Value'
      description = 'Description';

run;

title 'SAS Macro Variables Generated for Uploaded File '
      ""&_WEBIN_FILENAME"";
footnote link=""&_WEBIN_FILEURI/content" "Click here to download file";

proc print data=work.upload_info
  label
  style(header)=[just=center]
  style(column)=[verticalalign=middle];
run; quit;

```

Program Description

The %JESBEGIN and %JESEND macros are included by default and do not usually need to be added. The %JESBEGIN macro performs several tasks before executing the code, including:

- issuing a FILENAME statement to return HTML output to the web browser because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- issuing an ODS statement for the HTML5 destination because `ods_html5` is specified as the value for the `_OUTPUT_TYPE` input parameter.
- writing ODS output to the `_WEBOUT` fileref that is assigned by the %JESBEGIN macro. The HTMLBlue style, specified in the `_ODSSYLE` input parameter, controls the appearance of the output.

Additional global macro variables with information about the file are created as part of the upload process. This information is retrieved and stored in the `WORK.UPLOAD_INFO` data set.

The _WEBIN_FILEURI macro variable is of special interest because it provides a reference to the temporary location of the uploaded file. This location is associated with the job execution object and deleted when the job expires.

The FOOTNOTE statement shows an example of downloading the file by adding / content to the end of the _WEBIN_FILE_URI macro variable. The [“Upload a CSV File” on page 167](#) provides an example of referencing the uploaded file using a FILENAME statement.

The PRINT procedure displays information about the uploaded file.

The %JESSEND macro executes after the last line of code and closes all open ODS destinations.

Output

The following output is an example of a file that could be uploaded.

SAS Macro Variables Generated for Uploaded File "Capture.PNG"			
Obs	Variable Name	Value	Description
1	_WEBIN_CONTENT_LENGTH	13241	Specifies the size of the file that was uploaded in bytes (supplied automatically by the Web browser).
2	_WEBIN_CONTENT_TYPE	image/png	Specifies the content type that corresponds to the file that was uploaded (supplied automatically by the Web browser).
3	_WEBIN_FILE_COUNT	1	Specifies the number of files that were uploaded.
4	_WEBIN_FILEEXT	PNG	Specifies the extension of the file that was uploaded.
5	_WEBIN_FILENAME	Capture.PNG	Specifies the name and original location of the file that was uploaded.
6	_WEBIN_FILEURI	/files/files/f2f37e05-6d97-4616-992b-35815099e480	Specifies the URI of the location of the uploaded file
7	_WEBIN_NAME	myfile	Specifies the value that corresponds to the NAME attribute of the INPUT tag.

[Click here to download file](#)

Reference

%JESBEGIN and %JESEND Autocall Macros

Operation of the %JESBEGIN and %JESEND macros is controlled by global macro variables that are created from job input parameters of the same name. All input parameters are optional.

Table 3.2 Global Macro Variables

Name	Value	Action
_CONTDISP	Any valid value for the FILESRVC access method for the FILENAME	Specifies a value for the CONTENTDISP option in the FILENAME statement.
_CONTDISP_FILEEXT	statement. See “FILENAME Statement: FILESRVC Access Method” in SAS Global Statements: Reference for more information.	Specifies the file extension to use in the CONTENTDISP option in the FILENAME statement if _CONTDISP is not specified. The default value is derived based on the type of output being generated (the value of the _OUTPUT_TYPE parameter).
_CONTDISP_FILENAME		Specifies the file name to use in the CONTENTDISP option in the FILENAME statement if _CONTDISP is not specified. The default value is SASResults .
_CONTTYPE		Specifies a value for the CONTENTTYPE option in the FILENAME statement.
_DEBUG	Any valid value for _DEBUG. See “Debugging” on page 24 for more information.	If one of the values for _DEBUG is trace , then MPRINT is turned on, LS is set to max , and additional debug messages are printed in the log.
_ENCODING	Any valid SAS encoding	Specifies a value for the ENCODING option in the ODS statement.
_FILEREFS_OPTIONS	Any valid value for the FILESRVC access method for the FILENAME statement. See “FILENAME Statement: FILESRVC Access Method” in SAS Global Statements: Reference for more information.	Specifies a value to add to the end of the FILENAME statement.

Name	Value	Action
_GOPT_DEVICE	Any valid value for the DEVICE graphics option	Specifies a value to add to a GOPTIONS statement. See “Graphics Options and Device Parameters Dictionary” in SAS/GRAPH: Reference for more information about graphics options.
_GOPT_HSIZE	Any valid value for the HSIZE graphics option	
_GOPT_VSIZE	Any valid value for the VSIZE graphics option	
_GOPT_XPIXELS	Any valid value for the XPIXELS graphics option	
_GOPT_YPIXELS	Any valid value for the YPIXELS graphics option	
_GOPTIONS	Any value that is valid in a GOPTIONS statement	
_ODS_DEVICE	Any valid graphics device	Specifies a value for the OUTPUTFMT option of the ODS GRAPHICS statement. See “OUTPUTFMT=file-type STATIC” in SAS Output Delivery System: Procedures Guide for more information.
_ODS_EMBED_GRAPHICS	N , NO (case-insensitive)	Specifies ODS options for embedding graphics. This macro variable is valid only for the HTML5 destination. By default, these options are turned on.
_ODSOPTIONS	Any valid ODS options	Specifies a value to add to the ODS statement. See “Dictionary of ODS Language Statements” in SAS Output Delivery System: User’s Guide for more information about statements and their options.
_ODSSTYLE	Any valid ODS style	Specifies a value for the STYLE option in the ODS statement.

Name	Value	Action
_ODSSTYLESHEET_URL	Any valid value	Specifies a value for the URL suboption for the STYLESHEET option in the ODS statement. This macro variable is valid only for HTML-based destinations. See “(URL= ‘Uniform-Resource-Locator’)” in <i>SAS Output Delivery System: User’s Guide</i> for more information. Note: If you are accessing a style sheet outside of the domain, you must use SAS Environment Manager to add a new content security policy. See <i>SAS Viya Platform: Configuration Properties</i> for more information.
_OUTPUT_TYPE	NONE , ods_ods-destination , html , pdf , json , and so on (case-insensitive)	Specifies the type of output to create. Specifying NONE is almost the same as omitting the macros from the user code. Global macro variable values are displayed by the %JESBEGIN macro and then the macro exits. The %JESEND macro exits without performing any tasks. The default value is ods_html5.
_SUPPRESS_MVARS	Y , YES (case-insensitive)	Suppresses the display of macro variables (by the %JESBEGIN macro) before executing user code.

The %JESBEGIN macro creates the following global macro variables:

Table 3.3 Global Macro Variables Created by %JESBEGIN

Name	Value	Action
_JOBERROR	0 for success, nonzero for failure	Indicates whether the %JESBEGIN or %JESEND macro executed successfully.
ODSDEST	Value to the right of ods in the value of the _OUTPUT_TYPE parameter	Specifies which ODS destination is used, if any.

Name	Value	Action
_STATUS_MESSAGE	Any plain text	Passes an error message to the SAS Job Execution Web Application.

Reserved Macro Variables

Many macro variables are reserved for special purposes in jobs. Reserved names generally are prefixed with an underscore character. To avoid conflicts, do not use the underscore prefix for any application variables. The following table contains a complete list of reserved macro variables, including the %JESBEGIN and %JESEND autocall macros.

Note: Do not use CODE as the name of a macro variable, as this macro variable name might generate errors.

Table 3.4 *Reserved Macro Variables*

SAS Variable Name	Description
_ACTION	Specifies the _ACTION job input parameter, if any.
_ADDJESBEGINENDMACROS	Specifies whether to add the %JESBEGIN and %JESEND macros to the code. Valid values are true or false . If no value is present, then the macros are included.
_APSLIST	Specifies a list of job input parameters.
_CLIENTNAME	Specifies the name of the client.
_CONTEXTNAME	Specifies the Compute service context name.
_CONTDISP	Acts as an input parameter for the %JESBEGIN macro.
_CONTDISP_FILEEXT	Acts as an input parameter for the %JESBEGIN macro.
_CONTDISP_FILENAME	Acts as an input parameter for the %JESBEGIN macro.

SAS Variable Name	Description
_CONTTYPE	Acts as an input parameter for the %JESBEGIN macro.
_COOKIE	Returns a value as an HTTP header to the user interface, which indicates that a job is complete.
_CSRF	Specifies the Cross-Site Request Forgery token for this request.
_DEBUG	Specifies the _DEBUG job input parameter, if any.
_ENCODING	Acts as an input parameter for the %JESBEGIN macro.
_EXECUTIONTASKS	Specifies whether the job executes as a compute task. Valid values are true or false (default).
_FILE	Specifies the folder location and name of a file.
_FILEREFS_OPTIONS	Acts as an input parameter for the %JESBEGIN macro. This parameter is valid only when _executionTasks=False (when the job is not being executed as a compute task).
_FILESRVCS_fileref_URI	Specifies the URI of a file created by a fileref using the FILESRVC engine.
_FORM	Specifies the location of the input form file. This value is used only if _ACTION=FORM is also present.
_EXPIRATION	Specifies the expiration time for job output files after a job has executed. By default, this value is set to 30 minutes. The value is formatted according to the W3C XML duration data type. For example, the following value specifies that the job expires within 60 minutes after execution has completed: _expiration=PT60M

SAS Variable Name	Description
_GOPT_DEVICE	Acts as an input parameter for the %JESBEGIN macro.
_GOPT_HSIZE	Acts as an input parameter for the %JESBEGIN macro.
_GOPT_VSIZE	Acts as an input parameter for the %JESBEGIN macro.
_GOPT_XPIXELS	Acts as an input parameter for the %JESBEGIN macro.
_GOPT_YPIXELS	Acts as an input parameter for the %JESBEGIN macro.
_GOPTIONS	Acts as an input parameter for the %JESBEGIN macro.
_HTUA	Specifies the name of the user agent.
_JOB	Specifies a globally unique identifier.
_JOBERROR	Acts as an output parameter for the %JESBEGIN macro.
_JOBRUN_ERRMSG	Acts as an output parameter for the %JOBEXEC_WAITFOR macro.
_JOBRUN_RC	Acts as an output parameter for the %JOBEXEC_* macros.
_JOBRUN_STATE	Acts as an output parameter for the %JOBEXEC_WAITFOR macro.
_MIGRATED_STP	Flags jobs that are imported from SAS Stored Processes.
_ODS_DEVICE	Acts as an input parameter for the %JESBEGIN macro.
_ODS_EMBED_GRAPHICS	Acts as an input parameter for the %JESBEGIN macro.
_ODSDEST	Acts as an output parameter for the %JESBEGIN macro.

SAS Variable Name	Description
_ODSOPTIONS	Acts as an input parameter for the %JESBEGIN macro.
_ODSSTYLE	Acts as an input parameter for the %JESBEGIN macro.
_ODSSTYLESHEET_URL	Acts as an input parameter for the %JESBEGIN macro.
_OMITJSONLISTING	Specifies whether an internal JSON listing file is returned. Valid values are true (default) or false .
_OMITJSONLOG	Specifies whether an internal JSON log file is returned. Valid values are true or false (default).
_OMITSESSIONRESULTS	Specifies whether any results are returned. Valid values are true (default) or false .
_OMITTEXTLISTING	Specifies whether an internal text listing file is returned. Valid values are true (default) or false .
_OMITTEXTLOG	Specifies whether an internal text log is returned. Valid values are true (default) or false .
_OUTPUT_TYPE	Acts as an input parameter for the %JESBEGIN macro.
_PROGRAM	Specifies the path and name of the job.
_REDIRECT	Specifies whether to retrieve job output using a browser redirect to the Files service. Valid values are true or false (default).
_REPLAY	Reserved for future use.
_RESULTFILE	Specifies the output result files to be returned.
_RMTADDR	Specifies the Internet Protocol (IP) address of the client that sent the request.

SAS Variable Name	Description
_RMTHOST	Specifies the fully qualified name of the client that sent the request or the IP address of the client if the name cannot be determined.
_SAVEFILE	Specifies the name of the file for saved output.
_SAVEFOLDER	Specifies the name of the folder for saved output.
_SESSIONID	Specifies a unique identifier for the session. This variable is created when the COMPSRV_SESSION function is used to create a persistent session.
_STATUS_MESSAGE	Specifies the message text that is displayed by the client after a job executes.
_SUPPRESS_MVARS	Acts as an input parameter for the %JESBEGIN macro.
_TARGET	Specifies where to display the job output for prompt pages. For example, you can use _TARGET=_SELF to display the output in the same frame.
_THISSESSION	Specifies a URL that is composed from the values of _URL and _SESSIONID that can be used by clients of the Compute service to reference the current session. This variable is created when the COMPSRV_SESSION function is used to create a persistent session.
_TIMEOUT	Specifies the time-out value in seconds for a specific request.
_URL	Specifies the URL of the web server middle tier that is used to access the job.
_USERLOCALE	Specifies the locale for the user that was set in the user preferences. If this value was not set, it contains the locale sent in the HTTP request Accept-Language header.

SAS Variable Name	Description
_VERSION	Specifies the SAS Job Execution Web Application version number.
_WEBIN_CONTENT_LENGTH	Specifies the size of the file that was uploaded in bytes (this value is returned automatically by the web browser).
_WEBIN_CONTENT_TYPE	Specifies the content type that corresponds to the file that was uploaded (this value is returned automatically by the web browser).
_WEBIN_FILE_COUNT	Specifies the number of files that were uploaded.
_WEBIN_FILEEXT	Specifies the extension of the file that was uploaded.
_WEBIN_FILENAME	Specifies the name and original location of the file that was uploaded.
_WEBIN_FILEREF	Specifies the unique fileref that is automatically assigned to the uploaded file.
_WEBIN_FILEURI	Specifies the URI of the location of the uploaded file.
_WEBIN_NAME	Specifies the value that corresponds to the NAME attribute of the INPUT tag.
_WEBIN_SASNAME	Specifies the name for the SAS table, view, or catalog that is uploaded.
_XFORWARD	Specifies the host and port of the original HTTP request.
SYS_COMPUTE_JOB_ID	Specifies the Compute service job ID.
SYS_COMPUTE_SESSION_ID	Specifies the Compute service session ID.
SYS_COMPUTE_SESSION_OWNER	Specifies the user name that submitted the job.
SYS_JES_JOB_URI	Specifies the Job Execution service object URI.

_ACTION Input Parameter Values

The following values are supported by the _ACTION parameter:

Table 3.5 _ACTION Input Parameter Values

Value	Description
background	Executes the job in the background.
execute	Executes the job.
form	Displays an HTML input form file stored in the folder structure before job execution.
json	Returns a list of unexpired jobs or sample jobs in JSON format.
lastjob	Displays output from a previous job execution if it has not yet expired.
prompts	Displays a prompt file stored in the folder structure before job execution.
schedule	Indicates that a job is to be scheduled.
wait	Displays a wait screen with informational text while the job is executing.

%PARAM_LIST Macro

The %PARAM_LIST macro is used to convert the parameter list generated by a multiple-value prompt into a form that is useful in your SAS code.

Arguments

Table 3.6 %PARAM_LIST Macro Arguments

Name	Description
mvar	Required. Specifies the name of the macro variable that corresponds to the prompt name.

Name	Description
outvar	Optional. Specifies the name of the macro variable that contains the converted parameter list. If the name is not specified, an underscore (_) is added to the beginning of the value specified in mvar.
dlim	Optional. Specifies a character that is used to delimit values in the converted parameter list. A blank space is used by default. If a character is specified, the delimiter followed by a blank space is used.
quote	Optional. Specify y (case-insensitive) to quote the individual values in the converted parameter list. By default, the value is n .

The value of the macro variable specified in the OUTVAR argument is valid when one or more values are selected in the prompt. If no values are selected, then the macro variable is assigned a blank value.

Example: Using Prompt Values in a VAR Statement

This example assumes that a prompt is used to specify one or more column names in the SASHELP.CLASS data set using the PRINT procedure:

```
%param_list(mvar=prompt_vals, outvar=column_list)
```

```
proc print data=sashelp.class;
  var &COLUMN_LIST;
run; quit;
```

The code fails if the PROMPT_VALS macro variable is blank because the COLUMN_LIST macro variable does not have a value. This might happen if no values were selected in the prompt. One way to avoid this problem is to use the IFC function:

```
proc print data=sashelp.class;
  %sysfunc(ifc(%sysfunc(length(&COLUMN_LIST)) gt 0,
    "var &COLUMN_LIST",
  ));
run; quit;
```

Example: Using Prompt Values in a SELECT Statement

This example assumes that a prompt is used to specify one or more column names in the SASHELP.CLASS data set using the SQL procedure:

```
%param_list(mvar=prompt_vals, outvar=column_list, dlm=%str(,))

proc sql;
  create table work.class as
  select &COLUMN_LIST
  from sashelp.class;
run; quit;
```

See the previous example for information about handling missing prompt values.

Example: Using Prompt Values in a WHERE Statement

This example assumes that a prompt is used to select one or more age values (for example, 12, 14, and 16):

```
%param_list(mvar=prompt_vals, outvar=value_list, dlm=%str(,))

proc print data=sashelp.class;
  where age in (&VALUE_LIST);
run; quit;
```

This example assumes that a prompt is used to select **M**, **F**, or both values (case-sensitive). Specify **y** for the quote argument because the values of character variables must be quoted:

```
%param_list(mvar=prompt_vals, outvar=value_list, dlm=%str(,), quote=y)

proc print data=sashelp.class;
  where sex in (&VALUE_LIST);
run; quit;
```

Source Code

```
%macro param_list(mvar=, outvar=, dlm=, quote=n);

%local I PARAMLIST;

%if (%bquote(&MVAR) eq ) %then %do;
  %put ERROR: You must specify a value for the MVAR argument.;
  data _null_;
  abort cancel nolist;
  run;
  %goto exit;
%end;

%if (%symexist(&MVAR) ne 1) %then %do;
  %put ERROR: Macro variable "&MVAR" does not exist.;
  data _null_;
  abort cancel nolist;
  run;
```

```

    %goto exit;
%end;

%if (%bquote(%upcase(&QUOTE)) ne Y) and
    (%bquote(%upcase(&QUOTE)) ne N) %then %do;
    %put ERROR: You must specify either Y or N for the QUOTE argument.;
    data _null_;
    abort cancel nolist;
    run;
    %goto exit;
%end;

%let QUOTE=%upcase(&QUOTE);

%if (%bquote(&OUTVAR) eq )
    %then %let OUTVAR = _&MVAR;

%if (%sysfunc(nvalid(%bquote(&OUTVAR), v7)) ne 1) %then %do;
    %put ERROR: Please specify a valid macro variable name for the OUTVAR argument.;
    data _null_;
    abort cancel nolist;
    run;
    %goto exit;
%end;

%global &OUTVAR;

%global &MVAR.0;

%if (%bquote(&&&MVAR.0) eq ) %then %do;
    %if (&QUOTE eq Y)
        %then %let PARAMLIST=%sysfunc(quote(%bquote(&&&MVAR)));
        %else %let PARAMLIST=%bquote(&&&MVAR);
    %end;
%else %do I = 1 %to &&&MVAR.0;
    %if (&I eq 1) %then %do;
        %if (&QUOTE eq Y)
            %then %let PARAMLIST=%sysfunc(quote(%bquote(&&&MVAR&I)));
            %else %let PARAMLIST=%bquote(&&&MVAR&I);
        %end;
    %else %do;
        %if (&QUOTE eq Y)
            %then %let PARAMLIST=&PARAMLIST.&DLM %sysfunc(quote(%bquote(&&&MVAR&I)));
            %else %let PARAMLIST=&PARAMLIST.&DLM %bquote(&&&MVAR&I);
        %end;
    %end;
%end;

%let &OUTVAR=&PARAMLIST;

%exit:
%mend param_list;

```

Generated Macro Variables by Prompt Type

One or more global macro variables is automatically generated for each prompt when the job is executed. The values that are specified for the prompts at run time are assigned to the generated macro variables. The following table lists, by prompt type, the macro variables that are generated and how their values are determined. Examples of the generated macro variables are also provided.

Note: All dates are based on the Gregorian calendar. Each year begins with the month of January, and each week begins on Monday and ends on Sunday regardless of locale.

For more information about generated macro variables for multiple-value prompts, see [“Using Input Parameters with Multiple Values” on page 55](#).

Table 3.7 Generated Macro Variables by Prompt Type for Prompt MYPROMPT

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
Text	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt. No additional macro variables are generated for single-value prompts.	Hello World!	MYPROMPT	Hello World!
Numeric	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt. No additional macro variables are generated for single-value prompts.	12	MYPROMPT	12
Numeric range	A base macro variable is not generated for range prompts.			
	The <i>PROMPT-NAME_MIN</i> macro variable contains the lower boundary of the specified prompt range.	From: 23 To: 45	MYPROMPT_MIN	23
	The <i>PROMPT-NAME_MAX</i> macro variable contains the upper boundary of the specified prompt range.	From: 23 To: 45	MYPROMPT_MAX	45
Date (Day)	The <i>PROMPT-NAME</i> macro variable contains the value of	April 04, 2008	MYPROMPT	04Apr2008

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
	the prompt in the format <i>ddmmmyyyy</i>. Here is an explanation of the syntax: <i>dd</i> specifies a two-digit integer that represents the day of the month. <i>mmm</i> specifies the first three letters of the month. <i>yyyy</i> specifies a four-digit integer that represents the year.			
Date (Week, Month, Quarter, Year)	The <i>PROMPT-NAME</i> macro variable contains the first day of the week, month, quarter, or year that is specified for the prompt. The format of the macro variable value is <i>ddmmmyyyy</i>. Here is an explanation of the syntax: <i>dd</i> specifies a two-digit integer that represents the day of the month. <i>mmm</i> specifies the first three letters of the month. <i>yyyy</i> specifies a four-digit integer that represents the year.	Week 36 2008	MYPROMPT	01Sep2008
	The <i>PROMPT-NAME_END</i> macro variable contains the last day of the week, month, quarter, or year that is specified for the prompt. See the above base macro variable entry for the format that is used.	Week 36 2008	MYPROMPT_END	07Sep2008

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
Date range (Day)	A base macro variable is not generated for range prompts.			
	The <i>PROMPT-NAME_MIN</i> macro variable contains the lower boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: August 31, 2008 To: September 06, 2008	MYPROMPT_MIN	31Aug2008
	The <i>PROMPT-NAME_MAX</i> macro variable contains the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: August 31, 2008 To: September 06, 2008	MYPROMPT_MAX	06Sep2008
Date range (Week, Month, Quarter, Year)	A base macro variable is not generated for range prompts.			
	The <i>PROMPT-NAME_MIN</i> macro variable contains the first day of the lower boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: September 2008 To: June 2009	MYPROMPT_MIN	01Sep2008
	The <i>PROMPT-NAME_MAX</i> macro variable contains the first day of the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: September 2008 To: June 2009	MYPROMPT_MAX	01Jun2009
	The <i>PROMPT-NAME_MIN_END</i> macro variable contains the last day of the lower boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: September 2008 To: June 2009	MYPROMPT_MIN_END	30Sep2008

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
	The <i>PROMPT-NAME_MAX_END</i> macro variable contains the last day of the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.	From: September 2008 To: June 2009	MYPROMPT_MAX_END	30Jun2009
Time	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt in the format <i>hh:mm:ss</i>. Here is an explanation of the syntax: <i>hh</i> specifies a one-digit or two-digit integer that represents the hour of a 24-hour day. <i>mm</i> specifies a two-digit integer that represents the minute of the hour. <i>ss</i> specifies a two-digit integer that represents the second of the minute.	02:05:28 PM	MYPROMPT	14:05:28
Timestamp	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt in the format <i>ddmmyyyy hh:mm:ss</i>. Here is an explanation of the syntax: <i>dd</i> specifies a two-digit integer that represents the day of the month. <i>mmm</i> specifies the first three letters of the month. <i>yyyy</i> specifies a four-digit integer that represents the year.	September 02, 2008 02:07:03 PM	MYPROMPT	02Sep2008 14:07:03

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
	<i>hh</i> specifies a one-digit or two-digit integer that represents the hour of a 24-hour day. <i>mm</i> specifies a two-digit integer that represents the minute of the hour. <i>ss</i> specifies a two-digit integer that represents the second of the minute.			
Time range, Timestamp range	A base macro variable is not generated for range prompts.			
	The <i>PROMPT-NAME_MIN</i> macro variable contains the lower boundary of the specified prompt range. See the base macro variable entries for time and timestamp prompts for the formats that are used.	From: 01:13:04 AM To: 02:13:14 PM	MYPROMPT_MIN	1:13:04
	The <i>PROMPT-NAME_MAX</i> macro variable contains the upper boundary of the specified prompt range. See the base macro variable entries for time and timestamp prompts for the formats that are used.	From: 01:13:04 AM To: 02:13:14 PM	MYPROMPT_MAX	14:13:14
Color	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt in the format <i>cxrrggb</i>. Here is an explanation of the syntax: <i>rr</i> specifies the red component. <i>gg</i> specifies the green component. <i>bb</i> specifies the blue component.	CXFF0000 (bright red)	MYPROMPT	cxff0000

Prompt Type	Macro Variable	Prompt Value	Macro Variable Name	Macro Variable Value
Data source	The <i>PROMPT-NAME</i> macro variable contains the value of the prompt.	SASHELP.CARS	MYPROMPT	SASHELP.CARS
Data library	The <i>PROMPT-NAME</i> macro variable contains the libref of the library that is specified for the prompt.	SASHELP	MYPROMPT	SASHELP

Using Job Chaining Macros

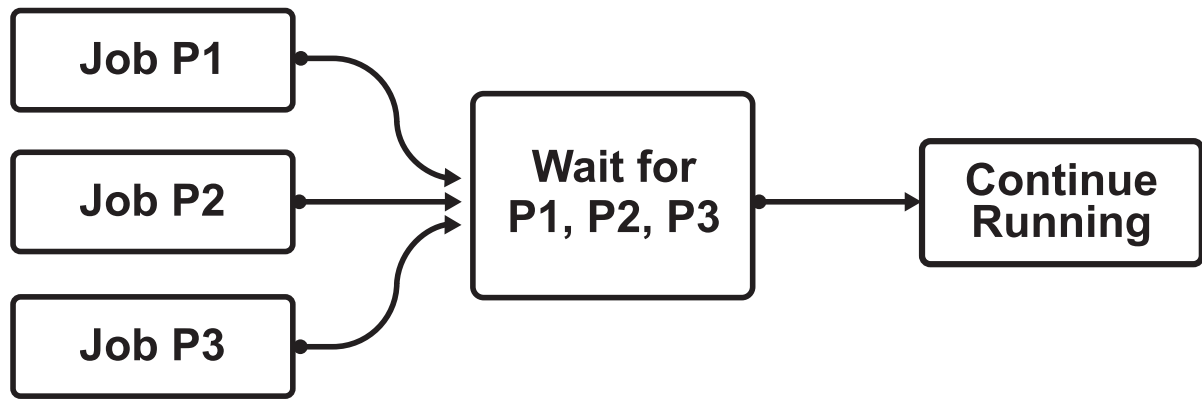
<i>Overview of %JOBEXEC_* Autocall Macros</i>	197
<i>Job Definitions for %JOBEXEC_* Autocall Macro Examples</i>	198
J1 Job Definition	198
J2 Job Definition	198
<i>Dictionary</i>	200
%JOBEXEC_RUN Autocall Macro	200
%JOBEXEC_WAITFOR Autocall Macro	209
%JOBEXEC_CANCEL Autocall Macro	215

Overview of %JOBEXEC_* Autocall Macros

You can use the %JOBEXEC_* autocall macros to execute and manage jobs, which is useful for performing the following tasks:

- executing job definitions from SAS code using SAS Studio or from other job definitions
- running multiple jobs in the background and waiting for them all to finish before executing more statements

The following figure illustrates an example of a job chaining flow using these utility macros. Specifically, it shows using the %JOBEXEC_WAITFOR macro to wait for multiple jobs to finish running before executing additional code. See [“Example 2: Waiting for Multiple Jobs to Finish Running” on page 213](#) for an example of this technique.



Submit the following code one time per active SAS session to make the utility macros available:

```
%jesutil()
```

Job Definitions for %JOBEXEC_* Autocall Macro Examples

To run the %JOBEXEC_* autocall macro examples, you must first create the J1 and J2 job definitions in the /Public/Job Chaining folder. If you select a different folder location, then you must specify that folder location in the _PROGRAM parameter in the example code.

J1 Job Definition

The J1 job loads the SASHELP data set that is specified in the DSNAME parameter to the CASUSER library for use by the J2 job.

For job definition parameters, specify _OUTPUT_TYPE=none. The DSNAME parameter does not need to be specified in the job definition, but is required for execution. Use it to specify the name of the SASHELP data set that was loaded when job J1 was run. You can specify the optional SLEEP parameter to cause the program to wait for the specified number of seconds before the execution completes. This technique works well with the %JOBEXEC_WAITFOR and %JOBEXEC_CANCEL macros.

Use the following SAS code for this job:

```
* Declare the input parameters;
```

```
%global DSNAME
        SLEEP;
```

```
* Start a CAS session and store the data in the CASUSER CAS library;
```

```

cas mysession;

data work.&DSNAME;
set sashelp.&DSNAME;
run;

proc casutil;
  droptable incaslib=casuser casdata="&DSNAME" quiet;
  run;
  load data=work.&DSNAME outcaslib=casuser promote;
  run;
quit;

cas mysession terminate;

%if (&SLEEP ne ) %then %do;
  data _null_;
  seconds = sleep(&SLEEP, 1);
  run;
%end;

```

J2 Job Definition

The J2 job displays the CAS table that was loaded by the J1 job.

You do not need to specify any job definition parameters, but the DSNAME parameter is required for execution. Use it to specify the name of the SASHELP data set to load to the CASUSER library. You must specify the same value for the DSNAME parameter in both jobs.

You can specify the optional SLEEP parameter to cause the program to wait for the specified number of seconds before the execution completes. This technique works well with the %JOBEXEC_WAITFOR and %JOBEXEC_CANCEL macros.

Use the following SAS code for this job:

```

* Declare the input parameters;

%global DSNAME
        SLEEP;

* Start a CAS session and use the data in the CASUSER CAS library;

cas mysession;

libname mydata cas caslib=casuser;

title ""&DSNAME"" Data Set";
proc print data=mydata.&DSNAME; run; quit;

cas mysession terminate;

%if (&SLEEP ne ) %then %do;

```

```
data _null_;
seconds = sleep(&SLEEP, 1);
run;
%end;
```

Dictionary

%JOBEXEC_RUN Autocall Macro

Runs a job definition.

Syntax

```
%JOBEXEC_RUN(BASEURL="URL-to-target",<IN=fileref | FORM (arguments) |
"string",><QUERY=(<NOENCODE> "parm1"="value1"
"parm2"="value2">,>JOBID_VARNAME=ID-variable<,<optional-arguments>)
```

Required Arguments

BASEURL="*URL-to-target*"

specifies the fully qualified base URL path for the SAS Viya platform server, used in the PROC HTTP URL argument. See "[URL="*URL-to-target*"](#)" in *Base SAS Procedures Guide* for more information.

IN=fileref | FORM (*arguments*) | "*string*"

specifies the value used with the PROC HTTP IN argument, which specifies the input data. This argument cannot be used if the QUERY argument is used. There are multiple ways to submit input data. See "[IN=fileref | "*string*" | FORM \(*arguments*\) | MULTI <FORM | "*type*" > \(*inputs*\)](#)" in *Base SAS Procedures Guide* for more information.

QUERY=(<NOENCODE> "*parm1*"="*value1*" "*parm2*"="*value2*")

specifies the value used with the PROC HTTP QUERY argument, which provides an alternate method for submitting query parameters. This argument cannot be used if the IN argument is used. You must specify the _ACTION=BACKGROUND job parameter when using this argument. See "["QUERY=\(<NOENCODE> "*parm1*"="*value1*" "*parm2*"="*value2*"\)](#)" in *Base SAS Procedures Guide* for more information.

JOBID_VARNAME=*ID-variable*

specifies the name of the macro variable that is used to store the job ID.

Optional Arguments

CLEAR_FILEREFs=true | false | blank

Default blank (filerrefs are not cleared)

specifies whether filerefs used by the macro should be cleared after execution. Filerefs are cleared only if **true** is specified.

DEBUG_LEVEL=0 | 1 | 2 | 3

Default blank (the DEBUG statement is not included)

specifies the value used with the PROC HTTP DEBUG statement, which writes debugging information to the SAS log. See [“DEBUG Statement” in Base SAS Procedures Guide](#) for more information about valid values.

DEBUG=blank | 1 | 2

Default blank (no debugging messages are displayed)

specifies the level of debugging messages that are displayed by the macro.

FOLLOWLOC=true | false | blank

Default blank (neither argument is included)

specifies whether the PROC HTTP FOLLOWLOC or NOFOLLOWLOC arguments are included. The FOLLOWLOC argument enables write methods to automatically follow URL redirections. The NOFOLLOWLOC argument prevents the GET method from following URL redirections. See [“FOLLOWLOC” in Base SAS Procedures Guide](#) for more information.

HEADERIN=fileref-to-request-header-file

Default blank (the argument is not included)

specifies the value to use with the PROC HTTP HEADERIN argument, which is a fileref to a text file that contains one line per request header in the format *key:value*. See [“HEADERIN=fileref-to-request-header-file” in Base SAS Procedures Guide](#) for more information.

HEADEROUT_OVERWRITE=true | false | blank

Default true

specifies whether the PROC HTTP HEADEROUT_OVERWRITE argument should be included. The argument is included only if **true** is specified. See [“HEADEROUT_OVERWRITE” in Base SAS Procedures Guide](#) for more information.

HEADERS="HeaderName"="HeaderValue"

Default blank (the statement is not included)

specifies whether the PROC HTTP HEADERS statement is included. See [“HEADERS Statement” in Base SAS Procedures Guide](#) for more information and valid values.

HTTP_OPTIONS=<options>

Default blank (the options are not included)

You can use this argument to specify additional PROC HTTP options that are not already handled by other arguments. See [“PROC HTTP Statement” in Base SAS Procedures Guide](#) for more information about options.

OAUTH_BEARER=token

Default sas_services

specifies the value to use with the PROC HTTP OAUTH_BEARER argument. Specify blank to omit this argument. See [“OAUTH_BEARER=token” in Base SAS Procedures Guide](#) for more information.

RESP_FILEREF=fileref-to-response-data-file

Default resp

specifies the name of the fileref to use in the PROC HTTP OUT argument, which is the file to which the output is written. The fileref will be assigned if it is not pre-assigned. See [“OUT=fileref-to-response-data” in Base SAS Procedures Guide](#) for more information.

RESPHDR_FILEREF=fileref-to-response-header-file

Default resphdr

specifies the name of the fileref to use in the PROC HTTP HEADEROUT argument, which is text file to which the response headers are written in the format *key:value*. The fileref will be assigned if it is not pre-assigned. See [“HEADEROUT=fileref-to-response-header-file” in Base SAS Procedures Guide](#) for more information.

Details

The %JOBEXEC_RUN macro uses PROC HTTP and the SAS Job Execution Web Application execution URL to run a job.

The %JOBEXEC_RUN macro creates the following macro variables:

_JOBRUN_RC

returns a nonzero value if an error was encountered.

_STATUS_MESSAGE

returns a non-blank value if an error was encountered.

Examples

Example 1: Running a Job Using a POST Request and the IN String Syntax

PROC HTTP supports specifying a string value for the IN argument. The POST method is used when the IN argument is specified. In this example, only the required arguments are specified for the %JOBEXEC_RUN macro.

You must URL encode parameter values and use %NRSTR to prevent unwanted resolution macro triggers when using this form of the PROC HTTP IN argument. These techniques are illustrated in the QUERY_VALUE macro variable, which specifies the job definition to run in addition to other URL parameters. See the program comments for more information.

You can submit the following code interactively using SAS Studio or as a job:

```
* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global DSNAME
        MYJOBID
        QUERY_VALUE
        SERVICESBASEURL;

* Initialize the program variables;

%let DSNAME=class;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

*;
* Specify that the J1 job is executed in the background. Progress
* can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

%let QUERY_VALUE=%nrstr(_action=background&_program=%2FPublic%2FJob%20Chaining
%2FJ1&sleep=30&dsname=)&DSNAME;

*;
* The macro does not wait for the job to finish execution because _action=background is
* specified. You can use the value of the MYJOBID macro variable with the
* %JOBEXEC_WAITFOR macro to wait for the job to complete before executing further
* statements, or with the %JOBEXEC_CANCEL macro to cancel the job.
*;

%jobexec_run(baseurl=&SERVICESBASEURL,
             in=&QUERY_VALUE,
             jobid_varname=MYJOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
```

```

%put NOTE: You can view the job execution object at /jobExecution/jobs/&MYJOBID;
%end;
%else %do;
  %put ERROR: Problem running the job. &=_JOBRUN_RC  &=_STATUS_MESSAGE;
%end;

```

Example 2: Running a Job Using a POST Request and the IN FORM Syntax

PROC HTTP supports specifying the FORM keyword with a string value for the IN argument. The POST method is used when the IN argument is specified. In this example, only the required arguments are specified for the %JOBEXEC_RUN macro.

This form of the IN argument is more convenient than the previous example because you do not need to URL encode parameter values; PROC HTTP encodes the values by default. You can often avoid using %NRSTR by using single quotation marks around portions of the query value that contain macro triggers that you do not want to resolve. These techniques are illustrated in the QUERY_VALUE macro variable, which specifies the job definition to run in addition to other URL parameters. See the program comments for more information.

You can submit the following code interactively using SAS Studio or as a job:

```

* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global DSNAME
        MYJOBID
        QUERY_VALUE
        SERVICESBASEURL;

* Initialize the program variables;

%let DSNAME=class;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

*;
* Specify that the J1 job is executed in the background. Progress
* can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

%let QUERY_VALUE=FORM('_action'='background' '_program'='/Public/Job Chaining/J1'
'sleep'='30' 'dsname'='&DSNAME');

*;
* The macro does not wait for the job to finish execution because _action=background is
* specified. You can use the value of the MYJOBID macro variable with the
* %JOBEXEC_WAITFOR macro to wait for the job to complete before executing further
* statements, or with the %JOBEXEC_CANCEL macro to cancel the job.
*;

```

```

%jobexec_run(baseurl=&SERVICESBASEURL,
             in=&QUERY_VALUE,
             jobid_varname=MYJOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
  %put NOTE: You can view the job execution object at /jobExecution/jobs/&MYJOBID;
%end;
%else %do;
  %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;

```

Example 3: Running a Job Using a POST Request and the IN FILEREF Syntax

PROC HTTP supports specifying a fileref statement that contains the message body for the IN argument. The POST method is used when the IN argument is specified. In this example, only the required arguments are specified for the %JOBEXEC_RUN macro.

The following sample DATA step code shows one way to create the message body file. You do not need to URL encode parameter values in the CARDS4 block; PROC HTTP encodes the values by default. You can often avoid using %NRSTR by using single quotation marks around portions of the query value that contain macro triggers that you do not want to resolve. These techniques are illustrated in the QUERY_VALUE macro variable, which specifies the job definition to run in addition to other URL parameters. See the program comments for more information.

The following line is written to the file after the DATA step code executes:
 _action=background&_program=/Public/Job Chaining/J1&dsname=class&sleep=30

You can submit the following code interactively using SAS Studio or as a job:

```

* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global DSNAME
        MYJOBID
        SERVICESBASEURL;

* Initialize the program variables;

%let DSNAME=class;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

* Assign a fileref for use with the PROC HTTP IN=FILEREF option;

filename indata temp;

* Create the query value and store it in the file;

data _null_;
length name value $256 str $32767;

```

```

infile cards4 truncover eof=eof;
file indata termstr='';
retain str '';
input;
name = dequote(scan(_infile_, 1, '='));
if (_n_ gt 1) then name = cats('&', name);

value = scan(_infile_, 2, '=');
if (substr(value, 1, 1) eq '"')
  then value = dequote(resolve(value));
  else value = strip(dequote(value));

str = cats(str, name, '=', value);
RETURN;
eof:
put str;
cards4;
'_action'='background'
'_program'='/Public/Job Chaining/J1'
'dsname'='&DSNAME'
'sleep'='30'
;;;
run;

*;
* The macro does not wait for the job to finish execution because _action=background is
* specified. You can use the value of the MYJOBID macro variable with the
* %JOBEXEC_WAITFOR macro to wait for the job to complete before executing further
* statements, or with the %JOBEXEC_CANCEL macro to cancel the job.
*
* Progress can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

%jobexec_run(baseurl=&SERVICESBASEURL,
             in=indata,
             jobid_varname=MYJOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
  %put NOTE: You can view the job execution object at /jobExecution/jobs/&MYJOBID;
%end;
%else %do;
  %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;

```

Example 4: Running a Job Using a GET Request and the QUERY Syntax

PROC HTTP supports the QUERY argument. The GET method is used when the QUERY argument is specified. In this example, only the required arguments are specified for the %JOBEXEC_RUN macro.

Note: You must specify the `_ACTION=BACKGROUND` job parameter when using this argument.

The syntax of the `QUERY` argument is like the `IN` argument with the `FORM` keyword. You do not need to URL encode parameter values because `PROC HTTP` encodes the values by default. You can often avoid using `%NRSTR` by using single quotation marks around portions of the query value that contain macro triggers that you do not want to resolve. These techniques are illustrated in the `QUERY_VALUE` macro variable, which specifies the job definition to run in addition to other URL parameters. See the program comments for more information.

You can submit the following code interactively using SAS Studio or as a job:

```
* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global DSNAME
        MYJOBID
        QUERY_VALUE
        SERVICESBASEURL;

* Initialize the program variables;

%let DSNAME=class;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

*;
* Specify that the J1 job is executed in the background. Progress
* can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

%let QUERY_VALUE='_action'='background' '_program'='/Public/Job Chaining/J1'
'sleep'='30' 'dsname'='&DSNAME';

*;
* The macro does not wait for the job to finish execution because _action=background is
* specified. You can use the value of the MYJOBID macro variable with the
* %JOBEXEC_WAITFOR macro to wait for the job to complete before executing further
* statements, or with the %JOBEXEC_CANCEL macro to cancel the job.
*;

%jobexec_run(baseurl=&SERVICESBASEURL,
             query=&QUERY_VALUE,
             jobid_varname=MYJOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: You can view the job execution object at /jobExecution/jobs/&MYJOBID;
%end;
%else %do;
    %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;
```

Example 5: Running Multiple Jobs

You can use background processing to run multiple jobs in parallel. This example assumes that you have three jobs named P1, P2, and P3 that run in parallel. When running the jobs, you must specify different job ID macro variable names and different filerefs for the response and response headers of each execution to prevent overwriting data.

The technique described in “[Example 2: Running a Job Using a POST Request and the IN FORM Syntax](#)” on page 204 is used to run the jobs.

You can submit the following code interactively using SAS Studio or as a job:

```
* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global P1JOBID
        P2JOBID
        P3JOBID
        SERVICESBASEURL;

* Initialize the program variable;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

*;
* The macro does not wait for the jobs to finish execution because
* _action=background is specified. You can use the value of the P*JOBID
* macro variables with the %JOBEXEC_WAITFOR macro to wait for the job to
* complete before executing further statements, or with the %JOBEXEC_CANCEL
* macro to cancel the job.
*
* FILENAME statements define the files used for the PROC HTTP response
* and response headers for each job. These filerefs are passed as arguments
* to the %JOBEXEC_RUN macro.
*
* Progress can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

filename plresp    temp;
filename plrespHdr temp;

%jobexec_run(baseurl=&SERVICESBASEURL,
              in=FORM('_action'='background' '_program'='/FolderPath/P1'),
              jobid_varname=P1JOBID,
              resp_fileref=plresp,
              resphdr_fileref=plrespHdr)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: You can view the job execution object at /jobExecution/jobs/&P1JOBID;
%end;
%else %do;
    %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
```

```

%end;

filename p2resp temp;
filename p2resphd temp;

%jobexec_run(baseUrl=&SERVICESBASEURL,
              in=FORM('_action'='background' '_program'='/FolderPath/P2'),
              jobid_varname=P2JOBID,
              resp_fileref=p2resp,
              resphdr_fileref=p2resphd)

%if (&_JOBRUN_RC eq 0) %then %do;
  %put NOTE: You can view the job execution object at /jobExecution/jobs/&P2JOBID;
%end;
%else %do;
  %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;

filename p3resp temp;
filename p3resphd temp;

%jobexec_run(baseUrl=&SERVICESBASEURL,
              in=FORM('_action'='background' '_program'='/FolderPath/P3'),
              jobid_varname=P3JOBID,
              resp_fileref=p3resp,
              resphdr_fileref=p3resphd)

%if (&_JOBRUN_RC eq 0) %then %do;
  %put NOTE: You can view the job execution object at /jobExecution/jobs/&P3JOBID;
%end;
%else %do;
  %put ERROR: Problem running the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;

```

%JOBEXEC_WAITFOR Autocall Macro

Polls the job status and waits until execution is complete.

Syntax

%JOBEXEC_WAITFOR(BASEURL="*URL-to-target*",JOBID=*job-ID*<*optional-arguments*>)

Required Arguments

BASEURL="URL-to-target"

specifies the fully qualified base URL path for the SAS Viya platform server, used in the PROC HTTP URL argument. See ["URL="URL-to-target" in Base SAS Procedures Guide](#) for more information.

JOBID=job-ID

specifies the ID of the job to wait for execution to complete. Use the macro variable that you specified for the JOBID_VARNAME argument in the %JOBEXEC_RUN macro when you ran the job.

Optional Arguments

CLEAR_FILEREFs=true | false | blank

Default blank (filerefs are not cleared)

specifies whether filerefs used by the macro should be cleared after execution. Filerefs are cleared only if **true** is specified.

DEBUG_LEVEL=0 | 1 | 2 | 3

Default blank (the DEBUG statement is not included)

specifies the value used with the PROC HTTP DEBUG statement, which writes debugging information to the SAS log. See ["DEBUG Statement" in Base SAS Procedures Guide](#) for more information about valid values.

DEBUG=blank | 1 | 2

Default blank (no debugging messages are displayed)

specifies the level of debugging messages that are displayed by the macro.

FOLLOWLOC=true | false | blank

Default blank (neither argument is included)

specifies whether the PROC HTTP FOLLOWLOC or NOFOLLOWLOC arguments are included. The FOLLOWLOC argument enables write methods to automatically follow URL redirections. The NOFOLLOWLOC argument prevents the GET method from following URL redirections. See ["FOLLOWLOC" in Base SAS Procedures Guide](#) for more information.

HEADERS="HeaderName"="HeaderValue"

Default blank (the statement is not included)

specifies whether the PROC HTTP HEADERS statement is included. See ["HEADERS Statement" in Base SAS Procedures Guide](#) for more information and valid values.

HTTP_OPTIONS=<options>

Default blank (the options are not included)

You can use this argument to specify additional PROC HTTP options that are not already handled by other arguments. See [“PROC HTTP Statement” in Base SAS Procedures Guide](#) for more information about options.

OAuth_Bearer=token

Default sas_services

specifies the value to use with the PROC HTTP OAUTH_BEARER argument. Specify blank to omit this argument. See [“OAuth_Bearer=token” in Base SAS Procedures Guide](#) for more information.

Poll_Seconds=interval-in-seconds

Default 10

specifies the interval in seconds to poll the job status.

Resp_Fileref=fileref-to-response-data-file

Default resp

specifies the name of the fileref to use in the PROC HTTP OUT argument, which is the file to which the output is written. The fileref will be assigned if it is not pre-assigned. See [“OUT=fileref-to-response-data” in Base SAS Procedures Guide](#) for more information.

Resphdr_Fileref=fileref-to-response-header-file

Default resphdr

specifies the name of the fileref to use in the PROC HTTP HEADEROUT argument, which is text file to which the response headers are written in the format *key:value*. The fileref will be assigned if it is not pre-assigned. See [“HEADEROUT=fileref-to-response-header-file” in Base SAS Procedures Guide](#) for more information.

Details

This macro uses PROC HTTP and the SAS REST APIs to poll the job status until the job has finished running. The polling interval is controlled by the POLL_SECONDS macro argument. You can use this macro to cause SAS to wait for job definitions that are submitted in the background to finish running before executing more statements.

The %JOBEXEC_WAITFOR macro creates the following macro variables:

_JOBRUN_ERRMSG

returns a non-blank value if an error was encountered while retrieving the job status.

_JOBRUN_RC

returns a nonzero value if an error was encountered.

_JOBRUN_STATE

returns the final status of the job.

_STATUS_MESSAGE

returns a non-blank value if an error was encountered.

For more information about SAS Viya platform REST APIs, see <https://developer.sas.com>.

Examples

Example 1: Waiting for a Single Job to Finish Execution

This example runs sample job J1 in the background, and then waits for it to finish running before executing sample job J2. The technique described in “[Example 2: Running a Job Using a POST Request and the IN FORM Syntax](#)” on page 204 is used to run the jobs.

```
* Make the utility macros available;

%jesutil()

* Declare the program variables;

%global DSNAME
        J1JOBID
        J2JOBID
        QUERY_VALUE
        SERVICESBASEURL;

* Initialize the program variables;

%let DSNAME=class;

%let SERVICESBASEURL=%sysfunc(getoption(servicesbaseurl));

*;
* Specify that the J1 job is executed in the background. Progress
* can be monitored using the SAS Job Execution Web Application
* Jobs page (/SASJobExecution/jobs).
*;

%let QUERY_VALUE=FORM('_action'='background' '_program'='/Public/Job Chaining/J1'
'sleep'='30' 'dsname'="&DSNAME");

*;
* The macro does not wait for the job to finish execution because _action=background is
* specified. You use the value of the J1JOBID macro variable with the
* %JOBEXEC_WAITFOR macro to wait for the job to complete before executing further
* statements.
*;

%jobexec_run(baseurl=&SERVICESBASEURL,
             in=&QUERY_VALUE,
             jobid_varname=J1JOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
```

```

    %put NOTE: You can view the job execution object at /jobExecution/jobs/&J1JOBID;
%end;
%else %do;
    %put ERROR: Problem running the job. &=_JOBRUN_RC   &=_STATUS_MESSAGE;
%end;

*   Wait for the J1 job to finish before executing more statements;

%jobexec_waitfor(baseurl=&SERVICESBASEURL,
                 jobid=&J1JOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: Job finished executing. Final job state is "&_JOBRUN_STATE".;
%end;
%else %do;
    %put ERROR: Problem waiting for the job. &=_JOBRUN_RC   &=_STATUS_MESSAGE
&=_JOBRUN_ERRMSG;
%end;

*   Execute the second job;

%let QUERY_VALUE=FORM('_program='/Public/Job Chaining/J2' 'dsname'="&DSNAME");

*;
*   The macro waits for the job to finish execution because _action=background is not
*   specified.
*;

%jobexec_run(baseurl=&SERVICESBASEURL,
             in=&QUERY_VALUE,
             jobid_varname=J2JOBID)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: You can view the job execution object at /jobExecution/jobs/&J2JOBID;
    %put NOTE: You can view the job output at /SASJobExecution/?_jobexec=/jobExecution/
jobs/&J2JOBID;
%end;
%else %do;
    %put ERROR: Problem running the job. &=_JOBRUN_RC   &=_STATUS_MESSAGE;
%end;

```

Example 2: Waiting for Multiple Jobs to Finish Running

This example executes three jobs in parallel by specifying `_ACTION=BACKGROUND`, and then waits for all jobs to finish before executing additional code. The code in [“Example 5: Running Multiple Jobs” on page 208](#) is used here, with additional statements added to the end. The complete code from that example must precede the code for this example.

```

*   Complete code from "Example: Running Multiple Jobs" here;

*   Wait for the P1 job to finish before executing more statements;

filename plwait1 temp;
filename plwait2 temp;

```

```

%jobexec_waitfor(baseurl=&SERVICESBASEURL,
                  jobid=&P1JOBID,
                  resp_fileref=p1wait1,
                  resphdr_fileref=p1wait2)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: Job P1 finished executing. Final job state is "&_JOBRUN_STATE".;
%end;
%else %do;
    %put ERROR: Problem waiting for the job P1. &=_JOBRUN_RC &=_STATUS_MESSAGE
    &=_JOBRUN_ERRMSG;
%end;

*   Wait for the P2 job to finish before executing more statements;

filename p2wait1 temp;
filename p2wait2 temp;

%jobexec_waitfor(baseurl=&SERVICESBASEURL,
                  jobid=&P2JOBID,
                  resp_fileref=p2wait1,
                  resphdr_fileref=p2wait2)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: Job P2 finished executing. Final job state is "&_JOBRUN_STATE".;
%end;
%else %do;
    %put ERROR: Problem waiting for the job P2. &=_JOBRUN_RC &=_STATUS_MESSAGE
    &=_JOBRUN_ERRMSG;
%end;

*   Wait for the P3 job to finish before executing more statements;

filename p3wait1 temp;
filename p3wait2 temp;

%jobexec_waitfor(baseurl=&SERVICESBASEURL,
                  jobid=&P3JOBID,
                  resp_fileref=p3wait1,
                  resphdr_fileref=p3wait2)

%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: Job P3 finished executing. Final job state is "&_JOBRUN_STATE".;
%end;
%else %do;
    %put ERROR: Problem waiting for the job P3. &=_JOBRUN_RC &=_STATUS_MESSAGE
    &=_JOBRUN_ERRMSG;
%end;

%put NOTE: All jobs have finished executing.;

```

%JOBEXEC_CANCEL Autocall Macro

Cancels a job that is running.

Syntax

%JOBEXEC_CANCEL(BASEURL=*"URL-to-target"*,JOBID=*job-ID*<*optional-arguments*>)

Required Arguments

BASEURL=*"URL-to-target"*

specifies the fully qualified base URL path for the SAS Viya platform server, used in the PROC HTTP URL argument. See ["URL=*"URL-to-target"*" in Base SAS Procedures Guide](#) for more information.

JOBID=*job-ID*

specifies the ID of the job to cancel. Use the macro variable that you specified for the JOBID_VARNAME argument in the %JOBEXEC_RUN macro when you ran the job.

Optional Arguments

CLEAR_FILEREFs=true | false | blank

Default blank (filerefs are not cleared)

specifies whether filerefs used by the macro should be cleared after execution. Filerefs are cleared only if **true** is specified.

DEBUG_LEVEL=0 | 1 | 2 | 3

Default blank (the DEBUG statement is not included)

specifies the value used with the PROC HTTP DEBUG statement, which writes debugging information to the SAS log. See ["DEBUG Statement" in Base SAS Procedures Guide](#) for more information about valid values.

DEBUG=blank | 1 | 2

Default blank (no debugging messages are displayed)

specifies the level of debugging messages that are displayed by the macro.

HEADEROUT_OVERWRITE=true | false | blank

Default true

specifies whether the PROC HTTP HEADEROUT_OVERWRITE argument should be included. The argument is included only if **true** is specified. See

“[HEADEROUT_OVERWRITE](#)” in *Base SAS Procedures Guide* for more information.

HTTP_OPTIONS=<options>

Default blank (the options are not included)

You can use this argument to specify additional PROC HTTP options that are not already handled by other arguments. See “[PROC HTTP Statement](#)” in *Base SAS Procedures Guide* for more information about options.

OAUTH_BEARER=token

Default sas_services

specifies the value to use with the PROC HTTP OAUTH_BEARER argument. Specify blank to omit this argument. See “[OAUTH_BEARER=token](#)” in *Base SAS Procedures Guide* for more information.

RESP_FILEREF=fileref-to-response-data-file

Default resp

specifies the name of the fileref to use in the PROC HTTP OUT argument, which is the file to which the output is written. The fileref will be assigned if it is not pre-assigned. See “[OUT=fileref-to-response-data](#)” in *Base SAS Procedures Guide* for more information.

RESPHDR_FILEREF=fileref-to-response-header-file

Default resphdr

specifies the name of the fileref to use in the PROC HTTP HEADEROUT argument, which is text file to which the response headers are written in the format *key:value*. The fileref will be assigned if it is not pre-assigned. See “[HEADEROUT=fileref-to-response-header-file](#)” in *Base SAS Procedures Guide* for more information.

Details

This macro uses PROC HTTP and the SAS REST APIs to cancel a job that is running. You can view the final status of the job on the Jobs page. See “[Managing Jobs](#)” on page 9 for more information about the Jobs page.

The %JOBEXEC_CANCEL macro creates the following macro variables:

_JOBRUN_RC

returns a nonzero value if an error was encountered.

_STATUS_MESSAGE

returns a non-blank value if an error was encountered.

For more information about SAS Viya platform REST APIs, see <https://developer.sas.com>.

Example: Cancel a Job

This example executes sample job J1 in the background and then cancels it. The technique described in [“Example 2: Running a Job Using a POST Request and the IN FORM Syntax” on page 204](#) is used to run the jobs. The complete code from that example must precede the code for this example. DATA step code is added after the %JOBEXEC_RUN macro to allow time for PROC HTTP to submit the job for execution.

* Complete code from "Example: Running a Job Using a POST Request and the IN FORM Syntax" here;

* Wait for PROC HTTP to submit the job for execution;

```
data _null_;
length seconds 8;
seconds = sleep(30, 1);
run;
```

* Cancel the job;

```
%jobexec_cancel(baseUrl=&SERVICESBASEURL,
                jobId=&MYJOBID)
```

```
%if (&_JOBRUN_RC eq 0) %then %do;
    %put NOTE: You can view the job execution state (should be "canceled") at /
    jobExecution/jobs/&MYJOBID/state;
%end;
%else %do;
    %put ERROR: Problem canceling the job. &=_JOBRUN_RC &=_STATUS_MESSAGE;
%end;
```

* Verify that the job was canceled;

```
%jobexec_waitfor(baseUrl=&SERVICESBASEURL,
                 jobId=&MYJOBID)
```

```
%put NOTE: Final job state is "&_JOBRUN_STATE".;
```

