SAS® Event Stream Processing: Using SAS® Event Stream Processing Studio

2026.08

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

SAS Event Stream Processing Studio Overview

What Is SAS Event Stream Processing Studio

SAS Event Stream Processing Studio is a web-based client that enables you to create, edit, import, publish, and test event stream processing models using SAS Event Stream Processing Studio Modeler. SAS Event Stream Processing Studio Modeler displays a model as a data flow diagram, enabling you to see and control how windows relate and flow into one another.

Accessing SAS Event Stream Processing Studio

If you are using another SAS Event Stream Processing client or another application in the SAS Viya platform, you can access SAS Event Stream Processing Studio by clicking  and selecting **Design Streaming Projects**. The list of applications that appears in the applications menu depends on how your system was installed and on your access permissions.

Otherwise, to access SAS Event Stream Processing Studio:

1 Open the following URL:

`https://name-of-ingress-host/SASEventStreamProcessingStudio`

The variable *name-of-ingress-host* specifies the name of the host that is configured for your Kubernetes cluster Ingress.

2 Enter your user ID and password and click **Sign In**.

Note: If standalone SAS Event Stream Processing was deployed, your system might not be configured to require users to sign in. In this case, the page for signing in is not displayed. Skip this step.

For information about supported web browsers and whether the device that you use to access SAS Event Stream Processing Studio meets requirements, see "Client Requirements" in *System Requirements for the SAS Viya Platform*.

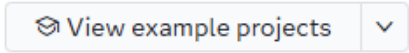
SAS Event Stream Processing Studio requires the use of cookies to maintain the session state.

Using the Welcome Page and Other Onboarding Resources

When you access SAS Event Stream Processing Studio or SAS Event Stream Manager for the first time, the **Welcome to SAS Event Stream Processing** page appears. Use the resources on the **Welcome to SAS Event Stream Processing** page to learn about SAS Event Stream Processing and the web-based clients.

When you first navigate from the **Welcome to SAS Event Stream Processing** page to SAS Event Stream Processing Studio, the Welcome to SAS Event Stream Processing Studio window appears. Use the links in this window to access further onboarding resources:

- a tour of the user interface, including an example project
- further example projects

You can also access the tour and the examples by clicking 

on the **Projects** page.

The user menu on the application bar provides access to other resources, such as information about what's new in each release. For more information, see "Toolbars".

Installing SAS Event Stream Processing Studio as a Progressive Web App

You can install SAS Event Stream Processing Studio as a Progressive Web App (PWA). For more information, see "Using SAS Event Stream Processing Web-based Clients as Progressive Web Apps" in *SAS Event Stream Processing: Overview*.

The ability to install the PWA is not available from within SAS Event Stream Processing Studio. However, you can install the PWA from any other web app and access SAS Event Stream Processing Studio from within the PWA.

To install the PWA:

1 Open any other SAS web app in a Chromium-based web browser (such as Chrome).

2 Do one of the following:

- Click the appropriate icon at the end of the address bar, and then select **Install SAS**.
- Open the web browser's **More** menu and select **Install SAS**.

> **Note:** The **Install SAS** option might be located under another menu option. For example, within the Microsoft Edge browser, it is located under **Apps**.

The web app is now installed as a desktop program named SAS. After one web application has been installed as a PWA, all other SAS applications are accessible in the PWA.

To uninstall the PWA:

1 Open the PWA.

2 In the PWA menu, select **Uninstall SAS**.

Understanding the User Interface

Pages

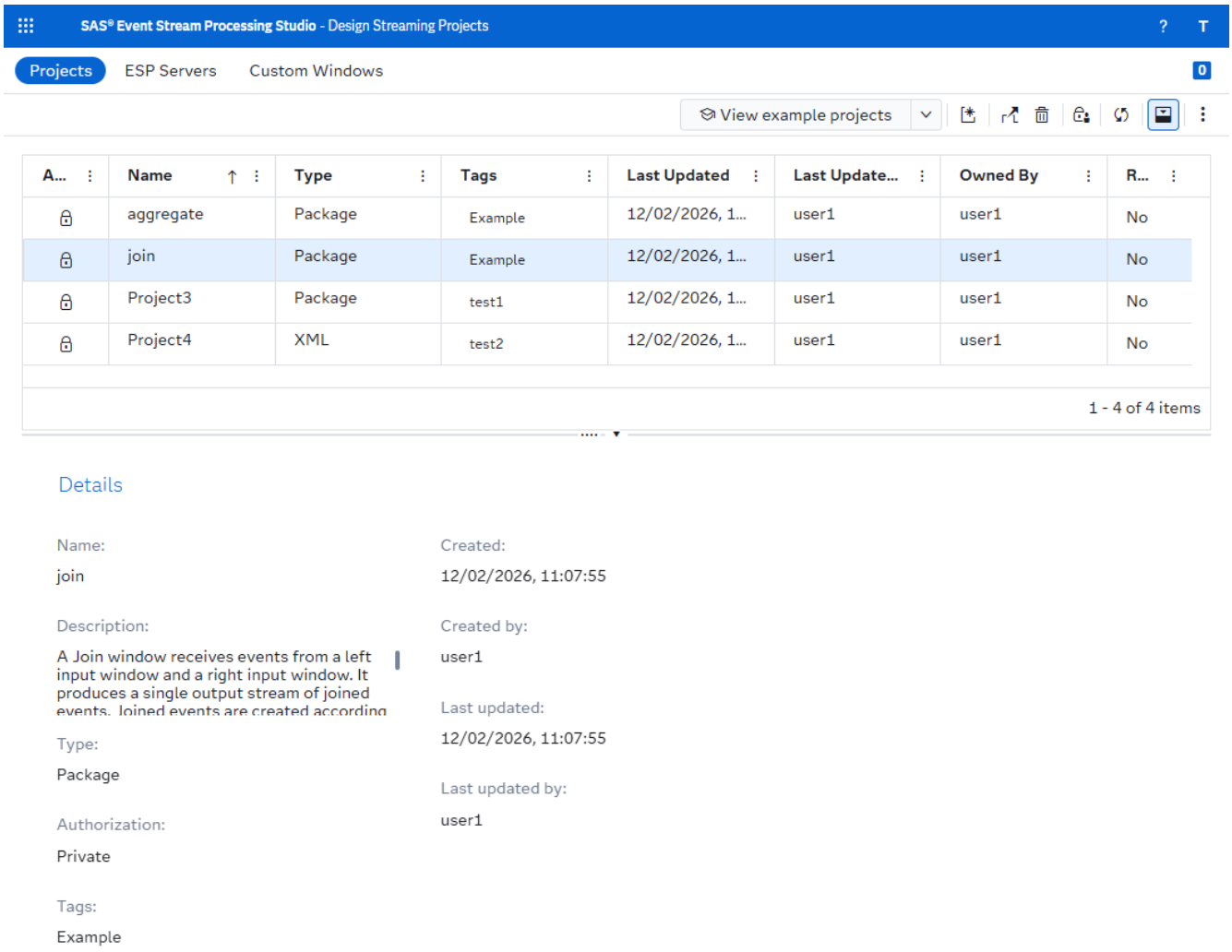
A *page* is the highest-level container in the user interface. All other user interface elements are contained within a page.

SAS Event Stream Processing Studio contains the following main pages:

- The **Projects** page enables you to create, edit, import, export, or delete the projects that contain your models.
- The **ESP Servers** page enables you to add, edit, import, export, or delete ESP servers.
- The **Custom Windows** page enables you to create and manage custom windows.

In addition, when you access SAS Event Stream Processing Studio for the first time, the **Welcome** page appears. For more information, see "Using the Welcome Page and Other Onboarding Resources".

Figure 1 *The Projects Page*

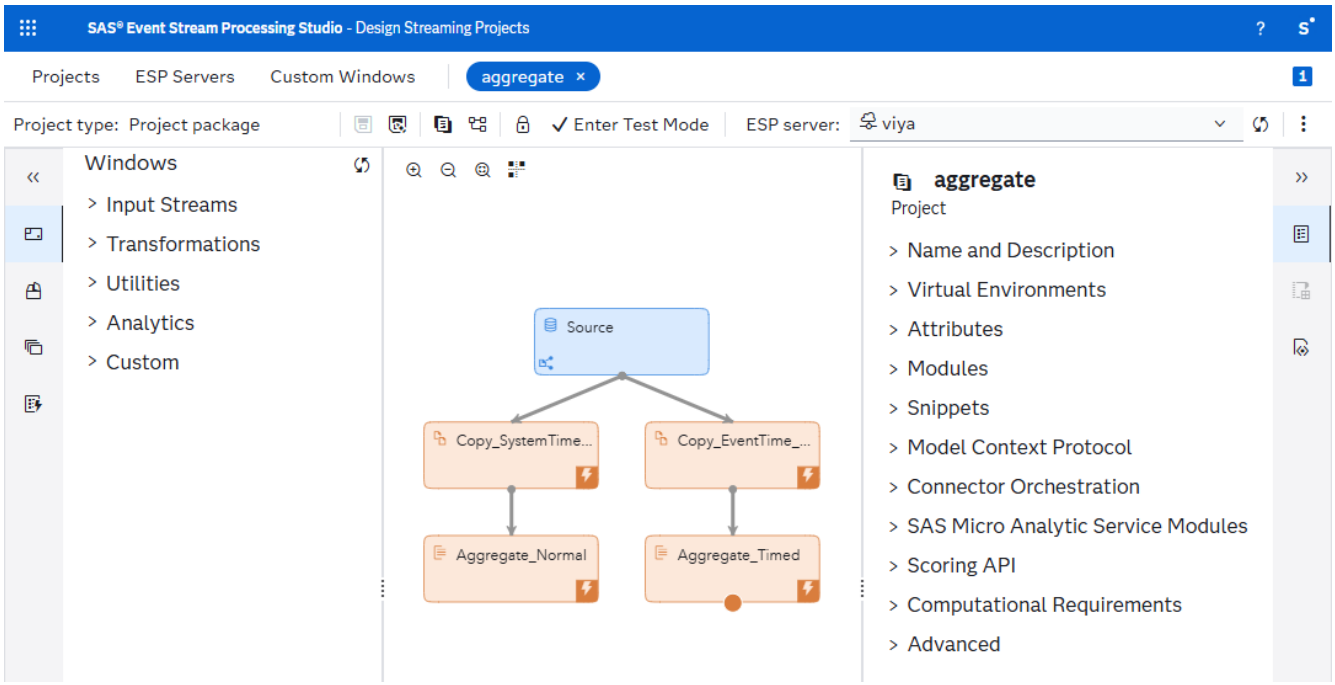


For more information about the tasks that you can perform on the **Projects** page, see "Overview of Working with Projects".

Panes

Panes that enable you to view different types of information within the same page. The following figure displays an open project in SAS Event Stream Processing Studio Modeler. The **Windows** pane on the left contains windows that you can add to your project. In the middle is the workspace. The right pane displays properties of the currently selected item in the project.

Figure 2 *Example of Two Vertical Panes*



For more information about SAS Event Stream Processing Modeler, see "Using SAS Event Stream Processing Studio Modeler".

For an example of a horizontal pane, see "Tiles".

To resize a pane, drag a border in the appropriate direction. To resize a horizontal pane, drag a border upward or downward. To resize a vertical pane, drag the border left or right.

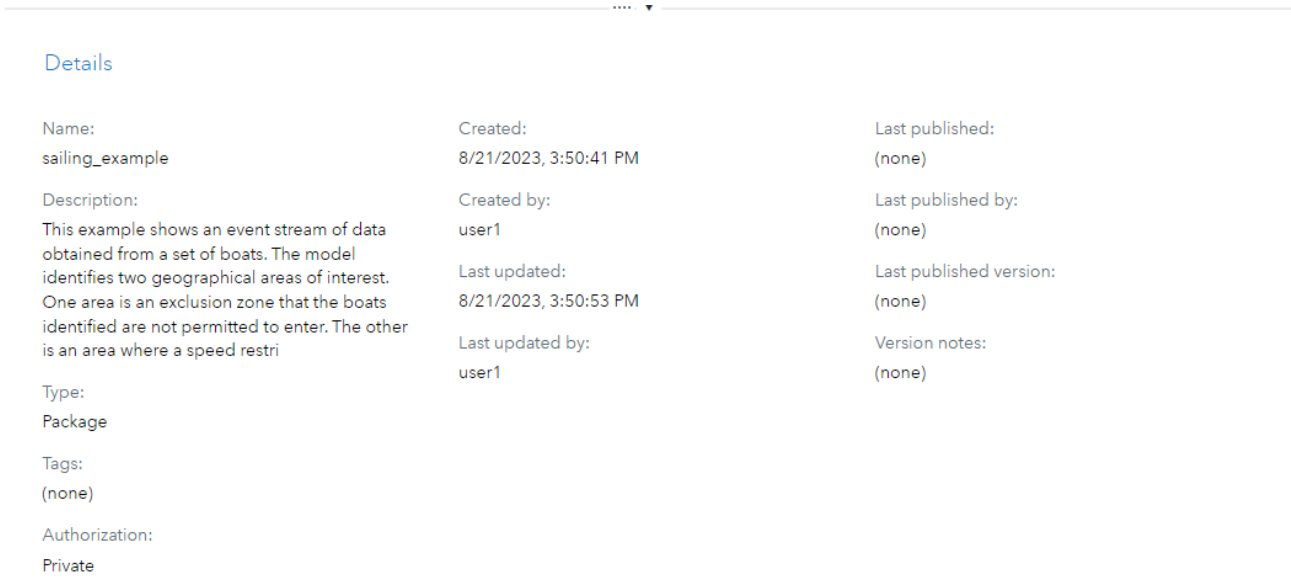
The **Projects**, **ESP Servers**, and **Custom Windows** pages contain horizontal panes that you can hide.

To hide a pane, click on the toolbar. Alternatively, click ▾ on the border to hide the pane. To display the pane again, click . Alternatively, click ▴ at the bottom of the user interface.

Tiles

A *tile* is a self-contained block of information that resides within a pane or sometimes directly on a page.

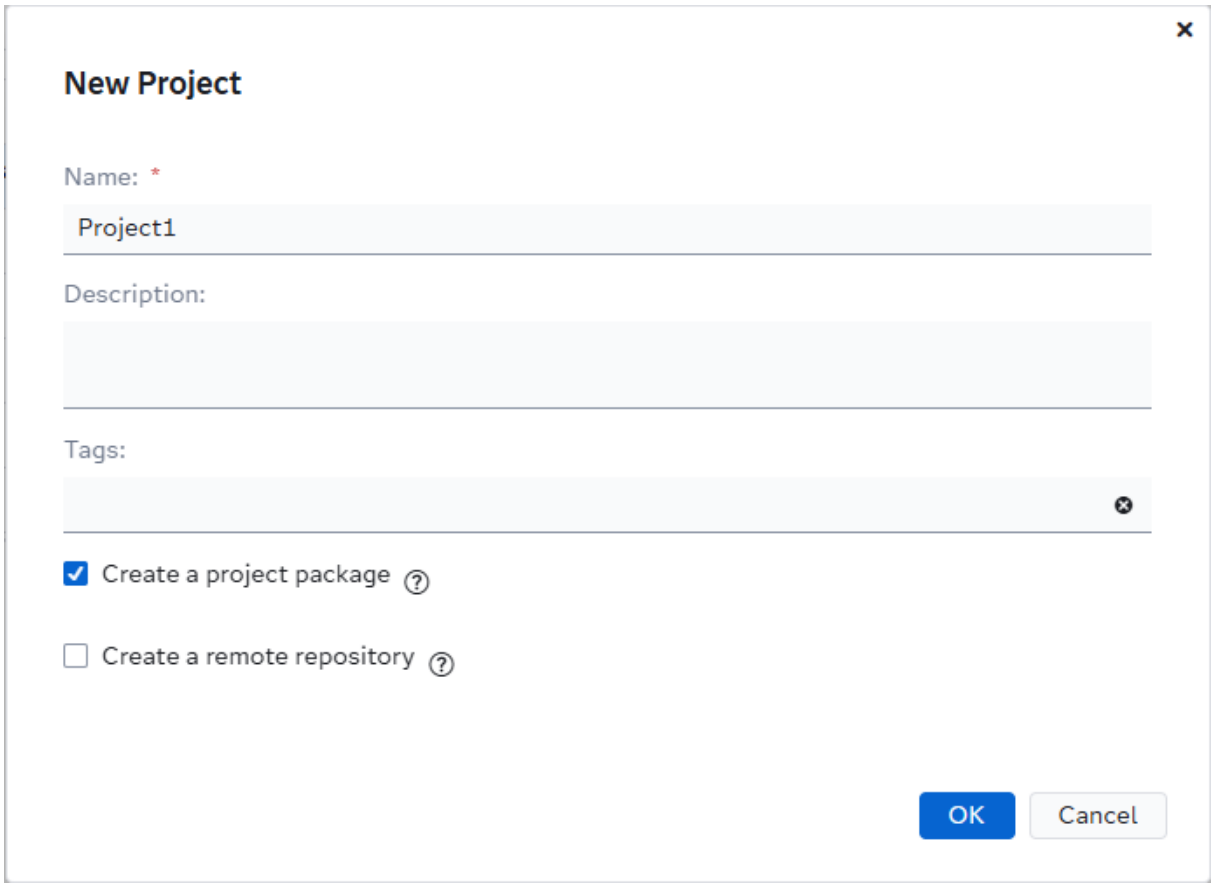
Figure 3 *Example of a Single Tile inside a Horizontal Pane*



Windows

A *window* is a floating user interface element that often appears as a result of a user action. Windows generally provide a means by which to perform an action and can be closed to return you to the page from which the window was launched. The following figure shows a window that is used to create a new project in SAS Event Stream Processing Studio.

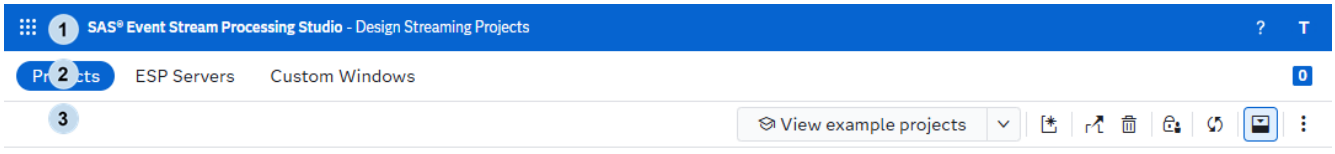
Figure 4 *The New Project Window*



Note: The user interface element *window* in SAS Event Stream Processing Studio does not have the same meaning as a SAS Event Stream Processing window. In SAS Event Stream Processing, windows are components of a continuous query. A continuous query contains a source window and one or more derived windows. SAS Event Stream Processing windows are connected by edges, which have an associated direction. SAS Event Stream Processing Studio contains both user interface element windows and SAS Event Stream Processing windows.

Toolbars

Figure 5 *The Main Toolbars*

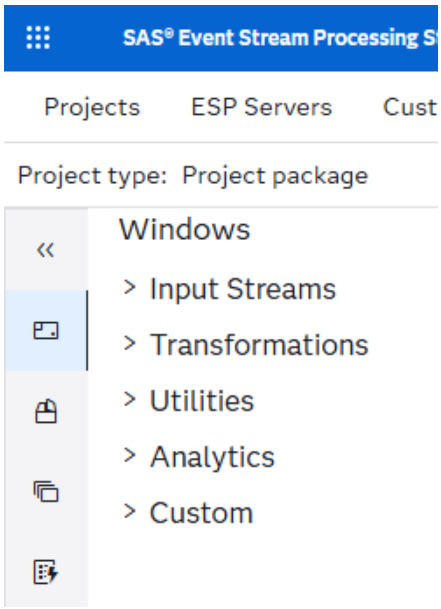


There are three main toolbars in SAS Event Stream Processing Studio, as shown in the following table:

Item Number	Description	Name
1	Application bar	■ Displays the applications menu, which enables you to access other applications on the SAS Viya platform. The presence of the applications menu and the list of applications within it depends on how your system was installed and on your access permissions. ■ Provides access to Help. ■ Displays your user icon, which shows the first character of your name or user ID. Click the user icon to access the following functionality: □ View your display name or your user ID in full. If a display name has not been set, your user ID is displayed by default. If SAS Event Stream Processing Studio has been configured so that you do not need to sign in with a user name and password, your user ID is not displayed. □ View information about what's new in each release, settings, and product information. □ Sign out of SAS Event Stream Processing Studio (if applicable).
2	Navigation bar	■ Provides access to the main SAS Event Stream Processing Studio pages: **Projects**, **ESP Servers**, and **Custom Windows**. ■ Provides access to each project, project version or model test that is currently open. The navigation overflow menu button displays the total number of these pages that are currently open. For example, if one page is open, the following icon appears: 1.
3	Toolbar	■ Includes buttons or tabs associated with the open item ■ Enables you to perform actions associated with a project. For example, you can import a project by clicking ⋮ on the toolbar and selecting **Import project**.

In addition to horizontal toolbars, SAS Event Stream Processing Studio contains vertical toolbars. The following figure shows an example of a vertical toolbar.The toolbar in this example is visible when you open a project. This toolbar enables you to switch between the **Windows** pane, the **Project Package** pane, and the **Project Versions** pane.

Figure 6 *Example of a Vertical Toolbar*



Sorting and Filtering

To make it easier to work with a large amount of information, you can sort and filter items displayed in tables. You can also show, hide, and reorder columns.

You can sort lists of items by ascending or descending order. To sort in ascending order, click the heading of the column that you want to sort. To sort in descending order, click the column again. To remove sorting, click the column a third time.

You can create filter criteria by which to display only a subset of information for a column. To create filter criteria, click ⋮ for the column that you want to apply filter criteria to, select **Filter**, and enter your filter criteria.

You can configure the columns that you want to display. To do this, click ⋮ in any column, select **Columns**, and deselect the columns that you do not want to appear.

You can re-order columns. To do this, click and hold the column heading, and drag it to a different location.

SAS Event Stream Processing Studio Modeler

When you create a new project or open an existing project, a separate page that contains the project content appears. This project page displays SAS Event Stream Processing Studio Modeler, which enables you to design models in a visual way and to test them. SAS Event Stream Processing Studio Modeler also includes the XML Editor, which you can use as an alternative way to construct your model.

For more information about the modeler, see “Using SAS Event Stream Processing Studio Modeler”.
For more information about the XML Editor, see “Using the XML Editor”.

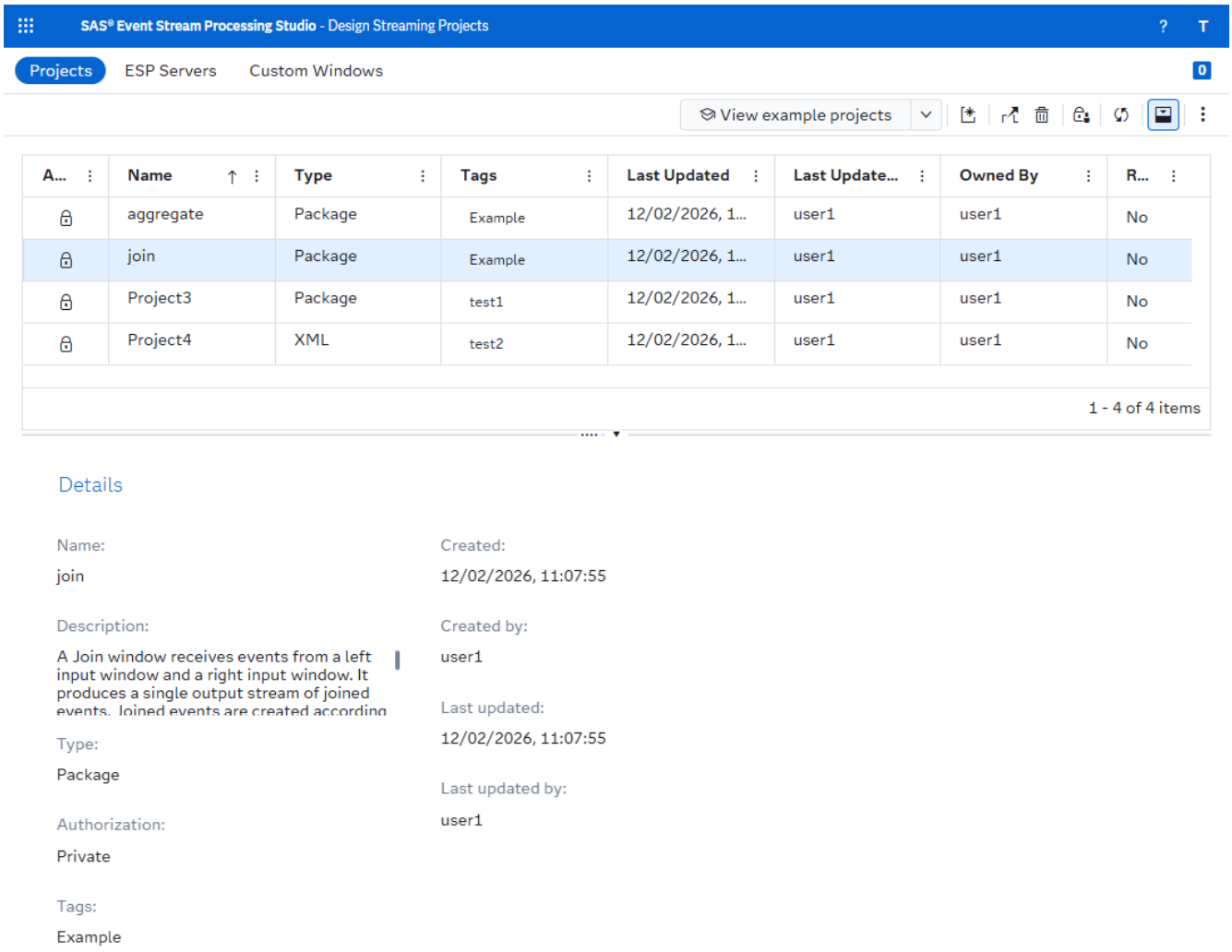
Working with Projects

Overview of Working with Projects

A *project* consists of one or more continuous queries. You can use SAS Event Stream Processing Studio to create, import, export, and delete projects. The **Projects** page enables you to view the projects in your deployment, along with their identification details.

The following figure shows an example:

Figure 7 *The Projects Page*



Note: Projects that are being edited in SAS Event Stream Processing Studio are automatically locked. This ensures that changes to a project cannot be unintentionally overwritten by another user. For more information, see "Project Locking".

The table on the **Projects** page displays the following information for each project:

- The project's access level. This column is visible when SAS Event Stream Processing is deployed with other SAS products.
- The project's name.
- The project's type. **Package** indicates that a project package exists. **XML** indicates that the project XML file is not part of a project package.
- Identifying tags assigned to the project.
- The date and time that the project was last updated.
- The user name of the user who last updated the project.

- The user name of the user who owns the project. The owner is the user who created or imported the project. The owner cannot be changed. The owner and members of the SASAdministrators group can change the project's access level.
- Whether the project is stored in a remote repository. This column is relevant when file-based versioning is used. For more information, see "File-Based Versioning and Git".

Here is information about some of the buttons on the toolbar:

- To view and install example projects, click View example projects. For more information, see "Install Example Projects".
- To open a project for editing, select the project from the main table and click . Alternatively, you can open a project for editing by double-clicking the project on the main table.
- When you create or import a project, it is restricted by default and marked as private. To change a project's access level, select the project from the main table, click , and select **Private project** or **Public project**. To change the access level of multiple projects simultaneously, press and hold Ctrl when selecting projects in the main table. For more information, see "Project Access Controls".
- To refresh the main table, click .
- To view a project's additional information, select the relevant project on the **Projects** page. A tile appears that contains this information, as shown in the previous figure. To hide this information, click .

Create a Project

1 On the **Projects** page, click .

The New Project window appears.

2 In the New Project window, complete these tasks:

a In the **Name** field, enter a unique name for the project.

Note: You must enter a unique project name. Duplicate project names are not supported.

If a project has been published and then subsequently deleted, you cannot reuse the deleted project's name.

b If required, in the **Description** field, enter a description for the project.

c If required, in the **Tags** field, enter any identifying keywords that describe the project.

d Use the **Create a project package** check box to indicate whether you want the project to be part of a project package. For more information, see "Working with Project Packages".

e (Optional) Select the **Use Large File Storage (LFS)** check box if you want to use Git LFS for storing large files that are associated with this project. For more information, see "File-Based Versioning and Git Large File Storage".

Note: Git LFS can be used when a project package is stored in one of the following ways:

- in the Git repository that is internal to SAS Event Stream Processing
- in a remote repository in Gitea

f (Optional) When file-based versioning is used, you can select the **Create a remote repository** check box to store your project in a remote Git repository. Use the **Remote server connection** drop-down list to select the remote Git server on which you want to create the repository. For more information, see "Overview of File-Based Versioning".

Note: Here is additional information about the **Remote server connection** drop-down list:

- If a message informs you that no remote server connections have been configured, contact your system administrator.
- The Git server's type is displayed in parentheses after the connection's name. Here is an example: **MyGitServer (GitLab)**.
- For more information, see "Specify Connections to Remote Git Servers".

g Click **OK**.

- If you are using a Kubernetes environment, skip to Step 5. In a Kubernetes environment, an ESP server is created on demand and you do not need to manually create one.
- If you are using an edge environment, an ESP server must be available and ready to configure. For more information about how to start an ESP server in an edge environment, see "Starting the ESP Server in an Edge Environment" in *SAS Event Stream Processing: Using the ESP Server in an Edge Environment*. In an edge environment, if ESP servers are not currently configured, you are prompted to decide whether you want to configure an ESP server now.

h Click **Yes** to configure an ESP server now or click **No** to configure an ESP server later.

If you selected **Yes**, the Edit ESP Server Properties window appears.

3 If you selected **No** to configure an ESP server later, skip this step. If you selected **Yes** to configure an ESP server now, complete these tasks:

a In the **Name** field, enter or update the name that identifies the ESP server.

b If required, in the **Description** field, enter or update the description of the ESP server.

c In the **Host** field, enter or update the ESP server's host name or IP address.

d In the **HTTP port** field, enter or update the ESP server's administration port number.

e If required, click **Edit** to change the setting for the **Authentication** field:

The Authentication window appears.

- **None**: This is the default option.
- **Kerberos**: This option is relevant only if the ESP server is configured to require authentication using Kerberos.

f If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the ESP server is configured to require SSL encryption.

g (Optional) Click **Test connection**. Testing the connection prevents the creation of an invalid ESP server connection and subsequent time-outs when the invalid connection is refreshed.

h If required, in the **Tags** field, enter or update any keywords that describe the ESP server and then press Enter.

i If required, select the **Enable server logging** check box to enable logging on the ESP server.

j If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the ESP server log. The default is 10,000 messages.

k Click **OK**.

4 Click **OK**.

5 SAS Event Stream Processing Studio Modeler appears. The workspace contains a Source window. Your project is created as a private project with a set of default properties.

Click .

Note: To create a copy of the project with a different file name, click . Enter the relevant information into the Save As window and click **OK**.

Before you start creating your model, review the default project properties and update them if necessary. For more information, see "Update Project Properties".

For more information about creating a model, see "Using SAS Event Stream Processing Studio Modeler".

Import a Project

Note: Project XML files imported to SAS Event Stream Processing Studio must be encoded in UTF-8 format. Importing project XML files that are not encoded in UTF-8 format can cause invalid characters to be displayed in SAS Event Stream Processing Studio. The project XML file must be encoded in UTF-8 format regardless of whether you import a project package or a project XML file that is not part of a project package.

1 On the **Projects** page, click .

2 Select the relevant option. The options that are available in your environment depend on how SAS Event Stream Processing was deployed.

- **Import XML files**: From the local file system, import a project XML file that is not part of a project package.
- **Import project**: From the local file system, import a ZIP file that contains a project package.
- **Import project from remote server**: From a repository on a remote Git server, import a project package.

> **IMPORTANT** The project package that you import from a remote Git server must have been originally published to the remote repository from SAS Event Stream Processing Studio.

You can also use the **Import project from remote server** option to import an empty repository, rather than a repository that already contains a project package. Importing an empty repository is useful if you want to store a project package in a repository that belongs to an organization rather than to an individual user. You can use this approach when the server type is GitHub or Gitea. To use this approach, create two empty repositories in the Git remote server with your organization rather than you as the owner. The repository names must match the desired name of the project package, and one of the repository names must end with **-config** or **_config**. For example, if you want the project package to be called `myproject`, you must create an empty repository called `myproject`and an empty repository called `myproject-config` or `myproject_config`.

3 If you are importing a ZIP file that contains a project package from the local file system, complete the following steps:

a Select the file that you want to import in the Import Project window.

b (Optional) Select the **Use Large File Storage (LFS)** check box if you want to use Git LFS for storing large files that are associated with this project. For more information, see "File-Based Versioning and Git Large File Storage".

Note: Git LFS can be used when a project package is stored in one of the following ways:

- in the Git repository that is internal to SAS Event Stream Processing
- in a remote repository in Gitea

c (Optional) Select the **Create a remote repository for each project** check box. Creating a remote repository during importing is similar to creating a remote repository for a project that is already in SAS Event Stream Processing Studio. For more information, see "Create a Remote Repository for an Existing Project".

4 If you are importing an XML file from the local file system, complete the following steps:

a Select the file that you want to import in the Import Project window. You can import several files at the same time.

b (Optional) Select the **Create a project package for each project** check box. For more information, see "Project Package".

c (Optional) Select the **Use Large File Storage (LFS)** check box if you want to use Git LFS for storing large files that are associated with this project. For more information, see "File-Based Versioning and Git Large File Storage".

Note: Git LFS can be used when a project package is stored in one of the following ways:

- in the Git repository that is internal to SAS Event Stream Processing
- in a remote repository in Gitea

d (Optional) Select the **Create a remote repository for each project** check box. Creating a remote repository during importing is similar to creating a remote repository for a project that is already in SAS Event Stream Processing Studio. For more information, see "Create a Remote Repository for an Existing Project".

5 If you are importing a project from a remote Git server, select the remote server connection and the project that you want to import in the Import Project from Remote Git Server window.

Note: Topics are labels that are used to organize repositories. If a topic was specified when the remote Git server connection was created, only projects with that topic are displayed in the Import Project from Remote Git Server window.

If a message informs you that no remote server connections have been configured, contact your system administrator.

For more information, see "Specify Connections to Remote Git Servers".

6 Click **Import**.

The project that you imported appears on the **Projects** page and the project is opened.

When you import multiple projects, the projects appear on the **Projects** page but are not opened.

If you import a project package that includes a `requirements.txt` file or a `modules.txt` file, when you open the project, a message informs you that a virtual environment might be required for the project to run correctly. Click **Yes** to select a virtual environment. Similarly, if the project package includes custom windows that require specific Python packages or Lua modules, a message might inform you to resolve a mismatch between required libraries and the selected virtual environment. Click **Resolve** to select a matching virtual environment. For more information about selecting a virtual environment, see "Update Project Properties".

When you import an empty repository, SAS Event Stream Processing Studio prompts you to create a project package for the project and optionally use Git LFS for storing large files that are associated with the project. Then a project is created with a name that reflects the name of the empty repository that you imported. The project appears on the **Projects** page and the project is opened.

Note: You cannot import a project that has the same name as a project version that has previously been published. This also applies to a project that has the same name as a project version that has been subsequently deleted from SAS Event Stream Processing Studio.

Update Project Properties

To access project-level properties rather than properties of a specific window, click  on the toolbar.

Before you start creating your model, review the default project properties and update them if necessary. You can update properties for functionality such as virtual environments, connector orchestration, SAS Micro Analytic Service modules, Scoring API, computational requirements, and error handling. The following sections discuss some functionality in more detail.

Select Virtual Environments

When your project is part of a project package, you can click on the toolbar and use the **Virtual Environments** section to select virtual environments that specify sets of Python packages or Lua modules. You can select one Python virtual environment and one Lua virtual environment. If your project depends on Python packages or Lua modules that are not in your environment, an error is reported when you run the project.

Note: If your project uses a Python or Lua virtual environment and you want to deploy the project to an edge device, you must use a project container to deploy your project. Project containers include the files that pertain to selected virtual environments, which can help ensure that required dependencies are available in the edge environment. For more information, see "When Is it Useful to Deploy Project Containers?" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Select a Python Virtual Environment

Here is information about selecting Python virtual environments:

- When you select the Python **Built-in environment** option, your project uses a default set of Python packages. To view the packages that are included in the built-in environment, click .
- When you select the **No Python dependencies** option, the built-in environment is not used. You can run basic Python code but no additional Python packages are included. If you use SAS Event Stream Manager to create a project container for your project, selecting this option can result in faster image creation as well as a smaller container image. For more information, see "Project Containers" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.
- When you select a different virtual environment, your project uses core Python functionality from the built-in environment and the Python packages that are defined in the selected virtual environment. To view the Python packages that are included in the environment, click . For more information about how a system administrator can create virtual environments, see "Specify Virtual Environments".
- When the Python **Built-in environment** option is used or you select a Python virtual environment, the `home` folder, the `home/analytics/` folder, and the `home/files/` folder of the project package are included in the search path for Python. When the **No Python dependencies** option is selected, only the `home` folder is included in the search path.

Select a Lua Virtual Environment

Here is information about selecting Lua virtual environments:

- When you select the Lua **Built-in environment**, you can run basic Lua code but no Lua modules are included.
- When you select the **No Lua dependencies** option, the built-in environment is not used. You can run basic Lua code but no Lua modules are included. This option has the same effect as the Lua **Built-in environment** option.
- When you select a different virtual environment, your project uses core Lua functionality from the built-in environment and the Lua modules that are defined in the selected virtual

environment. To view the Lua modules that are included in the environment, click . For more information about how a system administrator can create virtual environments, see "Specify Virtual Environments".

- The `home` folder is included in the search path for Lua, regardless of the virtual environment option that is selected.

Select a Matching Environment When Virtual Environment Files Have Been Detected

When you open a project package that includes a `requirements.txt` file or a modules.txt file but a virtual environment has not been selected for that project, the Virtual Environment Files Detected window is displayed.

A `requirements.txt` file specifies the configuration for a Python virtual environment. A modules.txt file specifies the configuration for a Lua virtual environment.

When the Virtual Environment Files Detected window is displayed:

1. Click **Yes** to indicate that you want to select a virtual environment.
2. In the Virtual Environments window, use the **Existing matching environments** drop-down list to select a suitable virtual environment.

   When there are no suitable virtual environments and you are a system administrator, you can use the Virtual Environments window as a quick way to create a virtual environment. Additional functionality for creating and updating virtual environments is available to system administrators on the **Virtual Environments** section of the **Settings** page. For more information, see "Specify Virtual Environments".

   Note: Only members of the administrator group can use the Virtual Environments window to create a virtual environment. Other users can use this window only to select existing virtual environments. By default, the administrator group is SASAdministrators.

3. Click **OK**.

Expose a Project as an MCP Server

The Model Context Protocol (MCP) provides an interface between large language models (LLMs) and external resources such as APIs, databases, and remote procedures. For more information, see MCP documentation.

MCP integration in SAS Event Stream Processing means that MCP can be used as an interface between LLMs and running ESP projects. When a project runs, it can expose some of its functionality as an *MCP server* that LLMs can interact with. That is, a project can become a resource that an LLM can contact—for example, to use a window to analyze data or to insert data into a window.

Note: Your organization must have an LLM that can interact with a remote MCP server.

To configure how a project interacts with LLMs, you use SAS Event Stream Processing Studio to create *MCP tools* and *MCP apps*:

- MCP tools are invoked by LLMs to perform operations. Tools can receive input, perform tasks on the ESP server, and send results back to the LLM.
- MCP apps provide an interactive user interface. An MCP app can return a mini-application such as a dashboard, form, or visualization that renders directly in a conversation. This type of output is useful when users need to explore or interact with results instead of reading a text-only response.

As you create tools and apps, you decide which capabilities of your project LLMs can interact with. You also create prompts that can be displayed to the user who is interacting with the LLM. For more information about MCP integration in SAS Event Stream Processing, see "Using the Model Context Protocol" in *SAS Event Stream Processing: Using Source and Derived Windows*.

For an example project that uses MCP, install the Sailing with MCP example. For more information, see "Install Example Projects".

Configure a Project as an MCP Server

To configure a project for use as an MCP server:

1. Click on the toolbar.
2. Expand the **Model Context Protocol** section.
3. Create MCP tools. For more information, see "Create MCP Tools".
4. Create MCP apps. For more information, see "Create MCP Apps".
5. Create prompts. For more information, see "Create Prompts".
6. If required, use the **Code** section to specify Python or Lua code to support tools that you added. You can enter the code by using an editor, specify a snippet that exists in the project, or select a file that contains the code.
7. Save the project and deploy it. For example, run the project in test mode. When the project runs, an ESP server is created on demand and the MCP server is exposed.
8. Use your LLM to connect to the MCP server at the ESP server's Ingress, at the `/eventStreamProcessing/v2/mcp` endpoint.
9. Use your LLM to run the tools.

Create MCP Tools

1. In the **Tools** section, click .
2. In the Add Model Context Protocol Tool window, enter a name, a title, and a description for the tool. LLMs use the information in titles and descriptions to determine whether they can use tools to accomplish tasks that the user wants to accomplish.
3. In the **Type** field, select the tool type:
   - The Score tool enables an LLM to score input events and return scored events. The results are delivered as event data from an output window that you specify in the tool configuration. For more information, see "Using the Scoring API" in *SAS Event Stream Processing: Using Source and Derived Windows*.

- The Query tool enables an LLM to query the contents of a window. The tool returns the contents of a specified output window to the LLM directly.
- The Subscribe tool enables an LLM to subscribe to an ESP window in the model and receive notifications in the form of event data. The difference between the Query tool and the Subscribe tool is that the Subscribe tool does not return events directly to the requesting LLM. Instead, the Subscribe tool sets up a subscription to the window and listens for events. When events are received by the listener they are sent as Server-Sent Events (SSE) over the persistent SSE connection.
- The Publish tool enables an LLM to publish data into the ESP model. The data can be events in CSV format, or string format if it is to be published into a specific field.
- The Custom tool enables a model designer to write Lua or Python code to accept input events, and to generate results that are returned to the LLM.

4 When you are creating a Score tool, complete these actions:

a If required, use the **Pre** field to enter the name of a function to run before publishing data into the model. You enter the Python or Lua code in Step 6 in "Configure a Project as an MCP Server".

b If required, use the **Post** field to enter the name of a function to run after the scoring has completed. You enter the Python or Lua code in Step 6 in "Configure a Project as an MCP Server".

c If required, use the **Event** field to enter the name of a function to run when an event is created from the input event block. You enter the Python or Lua code in Step 6 in "Configure a Project as an MCP Server".

d Use the **Input** tab to select the input window and the relevant fields. For binary fields, you can set the MIME type to indicate what type of binary data is contained in the event field.

You can also select the **Return fields as array** check box and enter the name for the array.

e Use the **Output** tab to select the output window and the relevant fields. For binary fields, you can set the MIME type to indicate what type of binary data is contained in the event field.

5 When you are creating a Query tool or a Subscribe tool, complete these actions:

a Use the **Output** tab to select the output window that you want to query or subscribe to, and the relevant fields. For binary fields, you can set the MIME type to indicate what type of binary data is contained in the event field.

b If required, use the **Filter** tab to limit the response to events that match certain criteria. That is, the data from the output window can be filtered prior to being returned. On the **Filter** tab, enter the name of a function for filtering data and add the relevant fields. The fields are populated from the LLM prompt and are provided to the function as properties of an object. For example, if you want the tool to filter out values that exceed a maximum value, you add a field for the maximum value. The field is given a value such as "get all values under 100" from the LLM prompt.

6 When you are creating a Publish tool, complete these actions:

a Use the **Input window** field to select the Source window into which you want to publish events.

b If required, use the **Field** field to select a field in the Source window. The data content is published into this field.

c If required, use the **Data** field to specify the contents of the data to publish. The LLM can specify this data in the request as well.

d If required, use the **URL** field to specify a URL that contains the contents of the data to publish. The LLM can specify this data in the request as well.

e If required, use the **Block size** field to specify the number of events that you want to put into each event block when data is published. The LLM can specify this data in the request as well.

7 When you are creating a Custom tool, complete these actions:

a In the **Call** field, enter the name of a Python or Lua function to be called. You enter the Python or Lua code in Step 6 in "Configure a Project as an MCP Server".

b If required, use the **Input** tab to add input fields.

You can also select the **Return fields as array** check box and enter the name for the array.

c Use the **Output** tab to add output fields.

8 If required, create another tool.

Create MCP Apps

1 In the **Apps** section, click  .

The Configure Model Context Protocol App window is a wizard with four pages.

2 On the **Name and Description** page, enter a name, a title, and a description for the app. LLMs use the information in titles and descriptions to determine whether they can use apps to accomplish tasks that the user wants to carry out.

3 In the **Type** field, select the app type:

- The Visual app type displays a single visual component such as a chart, gauge, or map.
- The Dashboard app type displays multiple visual components at the same time. You specify the width and height of each component to control the layout.
- The Navigation app type displays multiple visual components but shows them one at a time. Navigation between components uses tabs that are displayed at the bottom of the app.
- The Custom app type enables you to define an app by using embedded HTML or by referencing an external HTML file.

4 On the **Event Sources** page, add one or more event sources. Each event source references a window from which the app receives data. For each event source, specify the following items:

a A name for the event source.

b The type of event source. Select **collection** to receive the entire set of events that is currently in the window (that is, a snapshot). **collection** is suitable for components that show the current state, such as tables, gauges, or bar charts. Select **stream** to receive events as they are created. **stream** is required for time series charts.

c The window from which to receive data.

d (Optional) The interval in milliseconds at which to deliver data. The default is 1000 milliseconds.

e (Optional) The maximum number of events to deliver.

5 On the **Components** page, add one or more components. If you are creating a Visual app, you can add only one component. Each component displays data from an event source. For each component, specify the following items.

Note: When you are creating a Navigation app, the **Components** page contains two tables rather than one:

- Use the **Tabs** table to define tabs for the Navigation app.
- Use the **Components** table to specify each component's configuration, as described in the following steps. The Components table for a Navigation app contains an additional Tabs column: use it to select a tab that you specified in the **Tabs** table.

a A name for the component.

b The event source from which the component receives data. However, when the component type is **modelviewer**, **logviewer**, or **metricsviewer**, you do not need to select an event source because these components receive their data directly from the server.

c The component type. The options include a table, a bar chart, a line chart, a pie chart, a time series chart, a compass, a gauge, and a map. You can also select a viewer for an image, a SAS Event Stream Processing model, SAS Event Stream Processing log data, or SAS Event Stream Processing metrics. You can also create a custom component. When you are creating a custom component, you define it by entering HTML on the **Configuration** page, in the **Properties** table.

d (Optional) Width, height, text, and an icon for the component. Width and height are used in dashboard apps to control the layout. Specify width as a percentage (example: `40%`) and height in pixels (example: `250px`). Icon and text are used in Navigation apps to label the component.

6 On the **Configuration** page, you can configure visual elements for displaying event data from an event source. When you have created multiple components, the **Configuration** page contains a tab for configuring each component.

a You can add properties that control the appearance and behavior of the component. The available properties depend on the component type. For example, when the component type is a table, you can add a property named `title` and set its value to a suitable table title. Then you can add a property named `values` and set its value to a comma-separated list of column titles for the table. To edit a property's value in a code editor, select a row and click $x=\frac{y}{z}$.

When you are creating a time series chart, you must add a property named `time` and set its value to a date field or a time field.

When you are creating a custom component, you define it by entering HTML code in the **Properties** table. Use the property name `html` to enter HTML code inline in the Value column, or use the property name `html-url` to specify the URL of an external HTML file that is loaded each time the app runs.

For more information about the properties of different components, see "Visual Components" in *SAS Event Stream Processing: Using Source and Derived Windows*.

b When you are creating a Custom component, you can specify domains for network requests and static resources, to load content from external sources. Example: **`https://exampledomain.com`**.

c You can use the **Configure click options** table to define a pop-up menu for the component. When a user clicks an item in the component, the pop-up menu is displayed. Each row in the table defines one menu option. For each option, enter a label for the menu item and either a message or a URL. For example, a message might ask the LLM to look up more information about the selected item. A URL is opened in a new browser window when the user selects the menu option. In both the message and the URL, you can use `$data` as a placeholder. It is replaced with the key of the selected event when the menu option is selected.

7 If required, create another app.

Create Prompts

1 In the **Prompts** section, click .

2 In the Add Model Context Protocol Prompt window, enter a name, a title, and description for the prompt.

3 In the **Text** field, enter the text of the prompt, such as text that is displayed by a chat client in a chat window. Any values in the text that are preceded by a `$` character are treated as variables and the user is prompted to enter a value.

4 Variables that you enter in the **Text** field are automatically added to the adjacent table. Enter a description for each variable. The descriptions are displayed to users and should contain text that guides users to enter values.

5 If required, create another prompt.

Enable the Scoring API

When you want to use the Scoring API to score input events and return scored events on demand:

1 Click on the toolbar.

2 Expand the **Scoring API** section and select **Enable scoring API**.

3 Use the following fields to specify which windows and fields are used for the input events and output events:

- Use the **Input window** field to select the Source window that receives the event data that is passed in the body of a score request.
- Use the **Output window** field to select the window that is monitored for events that are created in response to a score request.
- If required, use the **Input field** field to select a field in the Source window to receive the request body. This field is available only when the project is part of a project package.

- If the response is to be returned as raw data (for example, an image), use the **Output field** field to specify a field in the output window. The content for this field is returned in response to a score request.

For general information about the Scoring API, see "Using the Scoring API" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about using SAS Event Stream Processing Studio to send events for scoring and to view scored events, see "Use the Scoring API When Testing a Model".

Specify Computational Requirements

Click  on the toolbar and use the **Computational Requirements** section to specify the recommended minimum amount of memory, CPU resource, and NVIDIA graphics processing unit (GPU) resource that the project requires.

Specifying these details lets other users of the project know what resources are needed. When you or other users run the project in SAS Event Stream Processing Studio or SAS Event Stream Manager, if the computational requirements are not met, then the Load and Start Project in Cluster window displays a warning message.

Note: Your system administrator can set limits for the amount of memory and the amount of CPU resource that is allocated.

For more information, see "Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster".

Select an Error Handling Strategy

1. Click  on the toolbar.
2. Expand the **Advanced** section.
3. Use the **Error handling strategy when fatal errors are encountered** field to select either **Log an error and continue project execution** or **Stop and remove project**.

Note: Error handling settings that you specify at project level can be overridden by error handling settings in Lua windows, Python windows, and custom windows. For more information, see the following topics:

- "Working with Lua Windows in SAS Event Stream Processing Studio"
- "Working with Python Windows in SAS Event Stream Processing Studio"
- "Add a Custom Window to a Project"

Delete a Project

Projects can be deleted by the project's owner or a member of the SASAdministrators group.

To delete a project:

1 Select the project that you want to delete from the table on the **Projects** page.

> **TIP** To delete multiple projects simultaneously, press and hold Ctrl, and then select the projects that you want to delete.

2 Click 🗑.

3 Confirm the deletion.

When a user who is not an administrator uses SAS Event Stream Processing Studio to delete a project XML file, SAS Micro Analytic Service stores that are associated with the project are not automatically deleted. The stores are orphaned.

When the following conditions are true, the confirmation window includes an option to also delete the project from SAS Event Stream Manager. Deleting the project from SAS Event Stream Manager also deletes SAS Micro Analytic Service stores that are associated with the project.

- SAS Event Stream Processing is deployed with other SAS products and SAS Viya platform services are used to publish project versions.
- You are deleting a published project.
- You are a member of the SASAdministators group and when you signed in to SAS Event Stream Processing Studio, you accepted administrator privileges in the Assumable Groups window.

The project is permanently deleted. If the project is part of a project package, the entire project package is deleted, including any files in the project package. For more information, see "Project Package".

Export a Project

1 On the **Projects** page, select the project that you want to export.

2 Click ⋮.

3 Select the relevant option. The project's type determines the format in which the project is exported:

- When the project that you selected is part of a project package, you can export the project package as a ZIP file. The `output` and `temp` folders are not exported.

- When the project that you selected is not part of a project package, you can export the project as an XML file.

The project is exported to your computer.

Note: The location of the project that you exported might vary depending on your browser's configuration.

Create a Remote Repository for an Existing Project

When file-based versioning is used, you can create a remote Git repository for an existing project. For more information about remote repositories, see "File-Based Versioning and Git".

To create a remote repository for an existing project:

1. On the **Projects** page, double-click the project for which you want to create a remote repository.
2. Click ⋮.
3. Select **Create a remote repository**.
4. In the Create a Remote Repository window, select a remote repository connection.

   Note:

   - If a message informs you that no remote server connections have been configured, contact your system administrator.
   - The Git server's type is displayed in parentheses after the connection's name. Here is an example: **MyGitServer (GitLab)**.
   - For more information, see "Specify Connections to Remote Git Servers".

5. Click **Create**.

A remote repository is created and the `home` folder of the project package is synchronized with the remote repository.

When you view your project in the table on the **Projects** page, the Remote Repository column indicates that your project is in a remote repository.

Link a Project to an Empty Remote Repository

Linking a project to an empty remote repository can be useful if you want to store a project package in a repository that belongs to an organization rather than to an individual user. You can use this approach when the server type is GitHub or Gitea. For more information about remote repositories, see "File-Based Versioning and Git".

To link a project to an empty remote repository:

1 Create two empty repositories on the remote server with your organization rather than you as the owner. The repository names must match the name of your project, and one of the repository names must end with **-config** or **_config**. For example, if your project is called `myproject`, you must create an empty repository called `myproject` and an empty repository called `myproject-config` or `myproject_config`.

2 On the **Projects** page, double-click the project that you want to link to an empty remote repository.

3 Click ⋮.

4 Select **Link to a remote repository**.

5 In the Link Project to Remote Git Server window, select a remote repository connection.

Note: If a message informs you that no remote server connections have been configured, contact your system administrator. For more information, see "Specify Connections to Remote Git Servers".

6 In the **Remote repository** field, select the empty repository that you created on the remote Git server.

7 Click **Link**.

View the Details of a Remote Repository

When file-based versioning is used, a project can be stored in a repository on a remote Git server. For more information, see "File-Based Versioning and Git".

When a project is stored in a remote repository, you can view the details of the remote repository:

1 On the **Projects** page, double-click the relevant project to open it.

> **TIP** You can use the Remote Repository column to identify projects that are stored in a remote repository.

2 Click ⋮.

3 Select **View repository details**.

A window appears and displays information about the connection that this project has to the relevant remote Git server. For example, if your project is stored in GitLab, clicking the **View repository** link enables you to view the remote repository in GitLab.

CAUTION

You can view the remote repository but do not attempt to interact with it. Use only SAS Event Stream Processing to interact with remote Git repositories. Interacting directly with a

repository that stores a SAS Event Stream Processing project on a remote Git server could result in data loss.

Project Access Controls

When you create or import a project in SAS Event Stream Processing Studio, you become the project's owner. When a project is created or imported, it is restricted by default and marked as private, and can be accessed, updated, deleted, or published only by the owner and users in the SASAdministrators group.

A public project can be accessed, updated, or published by any user.

A private project can be made public only by the owner or a member of the SASAdministrators group. Similarly, a public project can be made private only by the owner or a member of the SASAdministrators group.

If a user leaves the organization, that user's private projects do not disappear from the system. Members of the SASAdministrators group can make those private projects public.

When members of the SASAdministrators group log on to SAS Event Stream Processing Studio, they must accept administrator privileges in the Assumable Groups window. Failure to do so results in not having administrator privileges for project access.

When SAS Event Stream Processing is deployed with other SAS products and is upgraded from a release that does not contain project access controls, any existing projects automatically become private after the upgrade. When standalone SAS Event Stream Processing is upgraded from a release that does not contain project access controls, any existing projects remain public after the upgrade.

There are two ways to change the access level of a project:

- on the **Projects** page. This method enables you to change the access level of multiple projects simultaneously. For more information, see "Overview of Working with Projects".
- when you are viewing an opened project. For more information, see "Using SAS Event Stream Processing Studio Modeler".

Project Locking

Projects that are being edited in SAS Event Stream Processing Studio are automatically locked. This ensures that changes to a project cannot be unintentionally overwritten by another user. If you open a project, it is assumed that you intend to edit it. If a project has not been locked by another user, the project is locked immediately after you open it. If you are editing a project, the lock is released automatically if you close the tab that contains the project in the application. A lock is also released approximately two minutes after you close the application. For example, if you turn off your computer or close the browser, other users must wait two minutes until they can lock a project.

If you attempt to open a project that is locked by another user, you are informed that another user is editing the project. You are prompted to decide whether you want to continue. If you click **Yes**, the

project opens in SAS Event Stream Processing Studio Modeler in Read-Only mode. If you click **No**, you return to the **Projects** page.

Read-Only mode enables you to view the model and, if necessary, rearrange the position of the model's windows. However, you cannot save your changes.

Note: Projects are locked against your user name. If you are editing a project, it is not recommended that you open the project in another browser tab or window.

A banner appears when you open a read-only copy of a project. The banner specifies that the user that is editing the project and that a read-only copy of the project has been opened.

The **Windows** pane and magnification icons are unavailable in Read-Only mode. Therefore, you cannot add windows to your model using the **Windows** pane or adjust your model's magnification.

Project Metadata

When you create a project, the following unique information that identifies a project is created automatically:

- The user ID of the user who created the project
- The user ID of the user who last modified the project
- The date on which the project was created, in UNIX epoch time
- The date on which the project was last modified, in UNIX epoch time

The project information is displayed within the `<metadata>` element in your project's XML code, but it is stored in the SAS Event Stream Processing Studio database.

Metadata is also created if you perform one of the following actions:

- Apply a tag to the project
- Make changes to your model in the workspace

  Note: When you make a change to your model in the workspace, a `<meta id="layout">` element is added. This element specifies the name of your model's continuous query, the names of the windows in your model, and each window's X and Y coordinates in the workspace.

- Import a model that contains a SAS Micro Analytic Service module that has been created in SAS Model Manager into SAS Event Stream Processing Studio. For more information, see "Overview".

When you publish a version of a project that uses SAS Viya platform services, the following unique information that identifies the version is created automatically:

Note: The following elements are not displayed in the XML code of a working model. However, these elements are displayed in the model's XML code if you have published the model and you are viewing the model in Read-Only mode.

- The project's unique ID

- The project version's major version number
- The project version's minor version number

Here is an example of the metadata for a published project version that was published using SAS Viya platform services:

```
<metadata>
    <meta id="studioUploadedBy">espuser</meta>
    <meta id="studioUploaded">1614859578600</meta>
    <meta id="studioModifiedBy">espuser</meta>
    <meta id="studioModified">1614860116540</meta>
    <meta id="layout"><![CDATA[{"contQuery":{"aggregateWindow":
{"x":50,"y":175},"sourceWindow":{"x":50,"y":50}}}]]></meta>
    <meta id="studioVersionMajor">1</meta>
    <meta id="studioVersionMinor">0</meta>
    <meta id="studioId">c835319d-85bf-48e6-aaa5-c27b40bb1eee</meta>
</metadata>
```

In this example, the project's version number is 1.0.

When you publish a project version that uses file-based versioning, the project's XML code does not include metadata about project versions.

Working with Project Packages

Project Package

The XML file for a project contains elements such as windows and edges that are required to process streaming data. However, a project XML file can also reference additional resources that are located outside the XML file, such as input files, analytic stores, custom connectors, and plug-ins. For a project to be executed successfully, these additional resources must be available to the ESP server on which the project runs.

A project package contains a project XML file as well as a standard set of folders that contain the files that the project references. The project XML file and the additional files are stored on a persistent volume (PV). A PV is a piece of storage in a Kubernetes cluster.

IMPORTANT Using project packages requires that when SAS Event Stream Processing is deployed in a Kubernetes environment, it is configured to access persistent volumes. This configuration involves applying overlays. For more information, see "Managing Persistent Volumes" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

A project package makes it easier to manage the files that the project references:

- The functionality for project packages in the SAS Event Stream Processing Studio user interface enables you to perform tasks such as creating folders on the PV and uploading files to the PV, without using PV administration tools.
- When you reference a file from a window, in many cases SAS Event Stream Processing Studio enables you to navigate to a file in the project package rather than enter a path manually. Navigating to a file is less prone to error than manually entering a path.
- If a file that you want to reference from a window is not yet present in the project package, you can upload the file to the project package as part of the process of setting up the file reference, rather than uploading files separately before you can reference them. Furthermore, SAS Event Stream Processing Studio suggests an appropriate folder in which to upload your file, depending on the part of the project where you are adding the file reference. Placing files in different folders depending on the purpose of each file helps to keep the files organized within the project package. Here are some examples:
  - When you configure a file-and-socket publisher connector for a window and upload a CSV file as part of the process of referencing the file, SAS Event Stream Processing Studio suggests that you place the CSV file in the `test_files` folder.
  - When you reference a model from a Model Reader window and upload the model file as part of the process of referencing the file, SAS Event Stream Processing Studio suggests that you place the model file in the `analytics` folder.
  - When you configure a file-and-socket subscriber connector for a window, you can specify the name of an output file that does not exist yet. When you run the project in test mode and the output file is created, it is placed in the folder that you select. SAS Event Stream Processing Studio suggests the `output` folder as the location for the output file.
- When you browse for files in the project package or you upload files to the project package, SAS Event Stream Processing Studio restricts your view to the project package. You cannot navigate to a file that is located elsewhere on the PV (that is, outside your project package), and you cannot upload a file to another part of the PV.

Project packages can be published when file-based versioning is used. When you publish a project package, all files in the project package are versioned. For more information, see "Overview of File-Based Versioning".

Create a Project Package

Create a Project Package When Creating a New Project

You can create a project package as part of the process for creating a new project. For more information, see "Create a Project".

Create a Project Package for an Existing Project

Before you create a project package for an existing project, consider whether a project package is suitable for your particular project. A project package is not suitable if your project needs to reference files that are stored outside the project package (that is, in another location on the PV).

For example, your project might reference files that are shared by several projects. In a project package, the scope of navigating to files is restricted to files that are located within the same project package. This restriction helps ensure that all required files are located within the project package. Using a project package involves moving the referenced files into the project package and adjusting references to those files.

When project versions are published using SAS Viya platform services, you cannot create a project package for a project that has been published.

To create a project package for an existing project:

1 On the **Projects** page, locate the relevant project and check whether a project package already exists: when the **Type** column in the table has the value **XML**, a project package does not exist.

2 Double-click the project to open it.

SAS Event Stream Processing Studio Modeler appears.

3 Click on the left toolbar.

The **Project Package** pane appears.

4 Click **Create package**.

The project package, with a standard set of folders, is created on the persistent volume (PV). The **Project Package** pane displays the folders.

The project is saved in the `model` folder of the project package as model.xml. That is, the project now exists on the PV rather than only in the internal database of SAS Event Stream Processing Studio.

5 Adjust existing file references in the project. For an example, see "Example of Creating a Project Package for an Existing Project".

6 Click to save the project.

Example of Creating a Project Package for an Existing Project

The following steps show how to create a project package for one of the example projects in https://github.com/sassoftware/esp-studio-examples. The steps include adjusting file references in a publisher connector and a subscriber connector, and then adjusting a reference to an analytic store model.

1 Obtain the `/Analytics/analytics_modelReader_RPCA` directory from GitHub and save it on your computer.

2 Import the `model.xml` file to SAS Event Stream Processing Studio. For more information, see "Import a Project".

3 On the **Projects** page, right-click the project_01 project and select **Open project**.

4 Click on the left toolbar.

5 In the **Project Package** pane, click **Create package**.

6 Expand the **model** folder (a subfolder of the **home** folder) to observe that a **model.xml** file is now present there. The project now exists on the PV rather than only in the internal database of SAS Event Stream Processing Studio.

Note: When you create a project package, the project is always saved with the name model.xml. In this example, the original file name of the project also happened to be model.xml.

7 Update a publisher connector in the w_data window:

a Select the w_data window in the workspace. This window is a Source window.

b In the right pane, expand **Input Data (Publisher) Connectors**.

c Select the pub connector and click .

d In the Connector Configuration window, observe that the **Fsname** field contains an existing file reference: **input.csv**.

e Click .

f In the Select File window, **home > test_files** is displayed in the top left corner.

g Click to upload the **input.csv** file to the **test_files** folder in the project package.

h Click **OK**.

i In the Connector Configuration window, observe the new path in the **Fsname** field: **@ESP_PROJECT_HOME@/test_files/input.csv**.

j Click **OK**.

8 Update a reference to the analytic store model in the w_reader window:

a Select the w_reader window in the workspace. This window is a Model Reader window.

b Observe that the **Path to analytic store model** field contains an existing file reference: **RPCA.sasast**.

c Click .

d In the Select File window, observe that **home > analytics** is displayed in the top left corner.

e Click to upload the **RPCA.sasast** file to the **analytics** folder in the project package.

f Click **OK**.

g In the **Path to analytic store model** field, observe the new path in the **Fsname** field: **@ESP_PROJECT_HOME@/analytics/RPCA.sasast**.

9 Observe that a reference to the analytic store model in the w_score1 window has been updated automatically:

a Select the w_score1 window in the workspace. This window is a Score window.

b Click **Generate input and output maps from analytic store (ASTORE) file**.

c In the Generate Input and Output Maps window, observe that the **Path to file** field has been automatically updated to reference the `RPCS.sasast` file in the `analytics` folder of the project package: `@ESP_PROJECT_HOME@/analytics/RPCA.sasast`.

   In the original `model.xml` file that you imported to SAS Event Stream Processing Studio in step 2, this field referenced the `RPCA.sasast` file without specifying a path.

d Click **Cancel**.

10 Update a subscriber connector in the w_score1 window:

a In the right pane, expand **Subscriber Connectors**.

b Select the `sub` connector and click .

c In the Connector Configuration window, observe that the **Path** field contains an existing reference to `result.out`. The w_score1 window writes output to this file.

d In the **Fsname** field, delete the existing value and enter `result.out` (the same value). Observe that the path in the **Path** field changes to `@ESP_PROJECT_OUTPUT@/result.out`. When you test the project, the `result.out` file is written to this location within the project package.

e Click **OK**.

11 Select the root folder of the project package, `project01`, and click . Expand the folders within the `home` folder to observe that the RCPA.sasast and input.csv files that you uploaded are present in the project package.

12 Click to save the project.

13 Test the project. For more information, see "Running a Test".

Note: When you test the project, the **w_reader** tab does not show any data. This is expected behavior.

14 Return to the **project_01** page.

15 In the **Project Package** pane, select the root folder of the project package, `project01`, and click . Expand the `output` folder to observe that the `result.out` file has been created.

16 Select the `result.out` file and click . The `result.out` file is downloaded to your computer. The location of the file might vary depending on your browser's configuration.

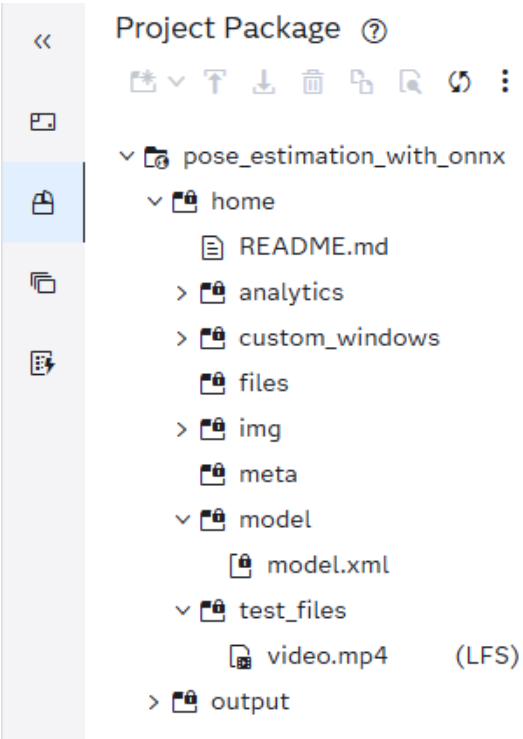
17 Use a text editor to open the `result.out` file. The file contains the output events for the w_score1 window.

Manage a Project Package

User Interface Controls

The following figure shows an example of a project package in the **Project Package** pane of SAS Event Stream Processing Studio Modeler:

Figure 8 *Example of a Project Package*



The root folder of the project package has the same name as your project. In this example, the root folder is **`pose_estimation_with_onnx`**.

The text **(LFS)** adjacent to a file name indicates that a file is stored in Git Large File Storage. For more information, see "File-Based Versioning and Git Large File Storage".

Use the buttons in the **Project Package** pane to manage files and folders that belong to a project package, as described in the following table.

Button	Description
	Create an additional file or folder within the selected folder. You can create files and folders within the subfolders of the **`home`** folder. For example, if your project references many analytics files, you might like to create additional folders within the **`analytics`** folder to organize your files.

Button	Description
	When you create a file, selected file formats are supported. Examples include CSV, LUA, MD, PY, and TXT. If the new file or folder does not appear, click  .
	Upload files to the selected folder.
	Download the selected file. The file is downloaded to your computer. The location of the file that you downloaded might vary depending on your browser's configuration. If you download the **`model.xml`** file, the resulting file is the same as when you export a project, apart from the file name. You cannot download the entire project package. However, you can export the project to your local file system and you can download each of the files in the project package to your local file system. For more information about exporting a project, see "Export a Project".
	Delete a file or folder. To delete multiple items simultaneously, press and hold Ctrl, and then select the files and folders that you want to delete. You cannot delete the root folder or the standard folders within the root folder. You cannot delete the model.xml file. When you delete a file or folder, you should remove all references to the file from the project.
	Copy the path to the selected file or folder. You can then paste the path elsewhere in SAS Event Stream Processing Studio.
	Open the selected file. This functionality is available for selected file types. Viewing of images is available for selected image file types such as GIF, JPG, and PNG. Depending on the file type, in addition to viewing the file, you might be able to edit and validate the file. For example, when you open a PY file, you can edit the file and check that the syntax of the Python code is valid. When you open an MD file (for example, a project's **`README.md`** file), you can edit the file and view an HTML rendering of it. For more information, see "Using the README.md File".
	When the selected file is a video, the button is displayed instead of the  button. Click to play the video. The MP4 file format and the H.264 codec are supported. The WEBM file format is also supported.
	Refresh all folders and their contents. Use this button to ensure that the folders and files that are displayed in the **Project Package** pane reflect the folders and files that exist on the persistent volume.
	This menu enables you to complete the perform the following tasks:

Button	Description
	■ Import a ZIP file that you have exported from SAS Model Manager. For more information, see "Import a SAS Model Manager ZIP file". ■ Clear the **temp** folder, when it contains temporary files that you want to delete. ■ Create a **README.md** file. For more information, see "Using the README.md File".

To rename a file or a folder, click the file or folder to select it, and then click it again to make the name editable. Renaming a file or a folder updates all references to it in the project's XML code. You cannot rename the model.xml file and certain other files and folders.

Using the Folders That Are Included in a Project Package

The root folder typically contains two folders, **home** and **output**. In some circumstances, a folder called **temp** can also be present.

Table 1 *Folders That Are Included within the home Folder*

Folder	Description
analytics	Use this folder to store analytics files, such as analytic stores. When you import models from SAS Model Manager, relevant files are stored in this folder. For example, when you import a DS2 model, the SAS Micro Analytic Service module is stored in the **/home/analytics/mas-modules/*moduleName*** folder. When you import a Python model, model files are stored in the **/home/analytics/model_manager** folder. For more information, see "Overview of SAS Model Manager Integration".
custom_windows	When you add a custom window to your project and save the project, the custom window's configuration file is stored in this folder. For more information, see "Add a Custom Window to a Project".
files	Use this folder to store other files that are needed by your project.
img	Use this folder to store image files such as images referenced by the project's **README.md** file. For more information, see "Using the README.md File".
meta	This folder can be used by other SAS Event Stream Processing functionality. When you specify that your project uses a virtual environment and save the project, a **requirements.txt** file that specifies the configuration of that virtual environment is stored in this folder. For more information, see "Create a Project".
model	When you save a project package, SAS Event Stream Processing Studio saves the project XML file in this folder. That is, SAS Event Stream Processing Studio stores the project not only in its internal database but also on the PV. The project's file name on the PV is always model.xml.

Folder	Description
`test_files`	Use this folder to store input files that are required for testing your project. This folder is intended for storing transient files rather than files that you need for production purposes. This folder is also used for storing scoring configuration files. For more information, see "Use the Scoring API When Testing a Model".

In addition to these folders, the **home** folder contains a **README.md** file. For more information, see "Using the README.md File".

While you are running a project, you cannot write files to the **home** folder or its subfolders. You can write files to the **output** folder.

The **output** folder is used for storing files that are generated when you run your project in test mode. This folder is intended for storing transient files rather than files that you need for production purposes.

If the windows in your project contain subscriber connectors that write output to files, those files are stored in the **output** folder. Select an output file and click ⭳ to download it. You can then open the file for viewing. The location of the file that you downloaded might vary depending on your browser's configuration.

Table 2 *Folders That Are Included within the output Folder*

Folder	Description
`luaroot`	This folder is relevant when you want to use persistent properties. For more information, see "Persistent Properties" in *SAS Event Stream Processing: Using Source and Derived Windows*.
`scoring_output`	When you use the **Scoring API** tab in test mode and a file that contains raw data is returned, the raw data file is stored in the **scoring_output** folder. For more information, see "Use the Scoring API When Testing a Model".

When there is an error during the uploading of files to the project package, a folder called **temp** might appear at the bottom of the list of files in the **Project Package** pane. The presence of the **temp** folder enables you to check and, if appropriate, delete temporary files, without needing to ask your system administrator to delete the files for you. To delete all files in the **temp** folder, click ⋮ on the toolbar and select **Clear "temp" folder**.

Using the README.md File

When you create a project package, a **README.md** file is included in the **home** folder of the package. When you open an existing project package that does not have a **README.md** file, a **README.md** file is added. You can use a **README.md** file to make notes about the project, for your reference and for the benefit of other users.

When you open the **README.md** file for viewing, its Markdown code is rendered as HTML. Click **Edit** to edit the file's Markdown code.

The **README.md** file contains sample text that you can replace with your own text. The file references an image called **example.png** that is located in the **img** folder within the **home** folder. You can add your own images to the **img** folder and reference those images from the **README.md** file.

If you delete the **README.md** file, you can create a **README.md** file that contains sample text. Click ⋮ in the **Project Package** pane, and then select **Create "README.md" file**.

When you install an example project, the **README.md** file in the project package contains information about how to use the project. For more information, see "Install Example Projects".

Reference Files That Are Included in a Project Package

The easiest way to reference a file within the same project package is to navigate to the file rather than manually enter the file path. Use this functionality whenever it is available in the user interface. Navigating to a file is less prone to error than manually entering a path. When you enter a path manually, you might accidentally refer to a file that is located outside the project package.

When you need to reference a file within the same project package by manually entering the path, you can use the ESP_PROJECT_HOME environment variable in the project's XML code to refer to the **home** folder of the project package. For example, you can refer to a file called **model.astore** that is located in the **analytics** folder like this: **@ESP_PROJECT_HOME@/analytics/model.astore**. Similarly, you can use the ESP_PROJECT_OUTPUT environment variable to refer to the **output** folder of the project package. Using an environment variable in a path means that you do not need to know the file's path on the PV.

Here is another example. A file and socket connector with a file path such as **/mnt/data/mydata.csv** would not work because the referenced CSV file is outside the project package. However, placing the CSV file in the **test_files** folder of the project package and referencing it as **@ESP_PROJECT_HOME@/test_files/mydata.csv** does work.

IMPORTANT

- You cannot use the ESP_PROJECT_HOME and ESP_PROJECT_OUTPUT environment variables to refer to locations within a project package from outside the project's XML code.
- When a project is part of a project package, the project must not reference files that are outside the project package. Otherwise, the project might not function:
  - When you use SAS Event Stream Processing Studio or SAS Event Stream Manager to run a project, the Kubernetes pod for that project does not have access to files that are outside the project package and the project might not run.
  - Project containers that you create from the project might not function. The files that are built into the project container include the files in the project package and the files that pertain to virtual environments that are selected for the project. When you run a project in test mode in SAS Event Stream Processing Studio, only these files are used. Using only these files ensures that the content that is used in test mode is the same as the content that is built into the project container.

Import or Export a Project Package

For information about importing a project package, see “Import a Project”. For information about exporting a project package, see “Export a Project”.

Unpack a Project Package

You cannot "unpack" a project that is part of a project package. However, you can complete the following steps:

1. Export the project XML file to your local file system. For more information, see “Export a Project”.
2. If you want to keep any files that are included in the project package, then download the files to your local file system. Use the button in the **Project Package** pane.
3. Delete the project from SAS Event Stream Processing Studio. The entire project package is deleted. For more information, see “Delete a Project”.
4. Import the project XML file back into SAS Event Stream Processing Studio. For more information, see “Import a Project”.
5. File paths within the project's XML code might include references to the ESP_PROJECT_HOME and ESP_PROJECT_OUTPUT environment variables. If required, adjust such references.

The XML code for a project that is part of a project package does not contain any elements or attributes that designate it as a project that is part of a project package.

Rename a Project Package

You cannot rename a project that is part of a project package. However, you can save the project package with a different name:

1. Click to save a copy of the project with a different name. The copied project includes the project package. That is, the copied project includes all the files and folders in the original project package.
2. Delete the original project.

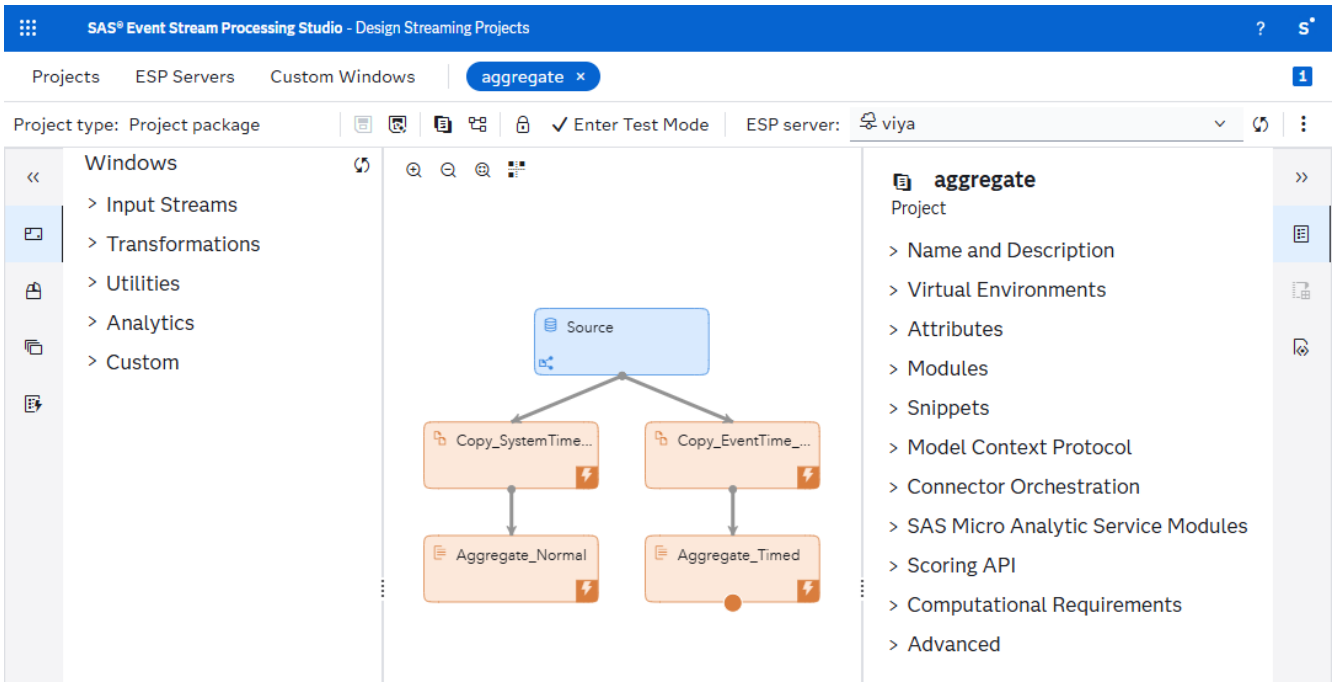
Using SAS Event Stream Processing Studio Modeler

Using SAS Event Stream Processing Studio Modeler

SAS Event Stream Processing Studio Modeler enables you to construct, change, and test SAS Event Stream Processing models. A *model* specifies how SAS Event Stream Processing analyzes and then transforms input event streams into meaningful results. From SAS Event Stream Processing Modeler, you can access context-sensitive help for a given subject, if it is available. To do this, click ⓘ if the icon is visible in the user interface.

SAS Event Stream Processing Studio Modeler appears when you create a new project or open an existing project.

Figure 9 *SAS Event Stream Processing Modeler*



The **Project type** field on the toolbar indicates the project's type:

- **Project package** means that the project XML file is part of a project package. For more information, see "Working with Project Packages".
- **Project package with LFS enabled** means that large files in the project package can be stored in Git Large File Storage. For more information, see "File-Based Versioning and Git Large File Storage".
- **XML file** means that the project XML file is not part of a project package.

The toolbar also displays information about the access level of the project:

- If the following button is displayed on the toolbar, the project is currently private: 🔒.
- If the following button is displayed, the project is currently public: 🔓.

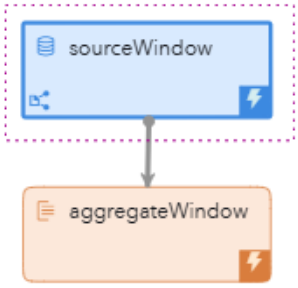
If you are the project's owner or a member of the SASAdministrators group, you can change the project's access level by clicking the button. For more information, see "Project Access Controls".

Use the following buttons to adjust the magnification and layout of your model:

- Click to increase the magnification.
- Click to decrease the magnification.
- Click to adjust the magnification of your model so that the entire model appears in the workspace.
- Click to automatically adjust the layout of the windows in the workspace.

To select a window, click the window in the workspace. The window that you have selected is shown in the workspace with a dashed bounding box:

Figure 10 *An Example of a Selected Window in the Workspace*



To select multiple windows, click in the workspace, and drag the dashed bounding box around the windows that you want to select.

To pan your model, press and hold Shift, click anywhere in the workspace, and then drag the cursor in the appropriate direction.

The modeler displays one continuous query at a time. When you create a new model, a continuous query named `cq1` is created by default. To construct your model, you must configure at least one continuous query. For more information about configuring continuous queries, see "Configuring Continuous Queries in SAS Event Stream Processing Studio".

To copy all or part of your model, select one or more windows in the workspace and press Ctrl + C. To paste the model elements that you copied, position the cursor in the workspace and press Ctrl + V. Copied windows retain their properties and schema fields. You can copy an edge on the condition that you also copy the edge's connecting windows. You can also copy and paste all or part of your model between continuous queries across one or more projects. When copying all or part of your model between continuous queries in separate projects, any project-level configuration, such as connector orchestration, is not copied. To replicate the configuration in the source project, you must manually update the project-level configuration in the target project.

Configure a Model's ESP Server

There are two ways in which an ESP server can be created: in a Kubernetes cluster or on an edge server.

ESP Servers That Run in a Kubernetes Cluster

If SAS Event Stream Processing Studio is running on Kubernetes, you can load and run projects in the Kubernetes cluster. When a project is run in a Kubernetes cluster, an ESP server is created on demand. When the project is stopped, the ESP server is deleted from the Kubernetes cluster. You cannot edit the properties of the ESP server that was created for you. Otherwise, an ESP server in a Kubernetes cluster behaves in the same way as an ESP server that is on an edge server. For more information, see "Working with a Kubernetes Cluster in SAS Event Stream Processing Studio".

When you open your project, the **ESP server** drop-down list displays the ESP server that is associated with your project. ESP servers in a Kubernetes cluster are identified in the drop-down list with the following icon: . Here is an example:

For more information about managing ESP servers, see "Managing ESP Servers in SAS Event Stream Processing Studio".

ESP Servers That Run on an Edge Server

For an ESP server that is to run on an edge server, the edge server does not form part of a Kubernetes cluster. Therefore, you must add details of ESP servers manually. For more information, see "Add or Edit an ESP Server to Run on an Edge Server". Available ESP servers are listed on the **ESP Servers** page.

When you open a project for the first time, the ESP server that is listed first in the table on the **ESP Server** page is assigned to the project. The **ESP server** drop-down list displays the name of that ESP server. Here is an example:

You can use the drop-down list to assign a different ESP server.

When an ESP server has been assigned to your project, the association is saved in your browser's local storage. That is, when you close the project and open it again, the same ESP server is still associated with the project.

If no ESP servers have been added in SAS Event Stream Processing Studio, the functionality that is available to your project is limited. For example, connector properties are unavailable to projects for which ESP servers are not assigned.

Add a Window

To add windows to the continuous query that is currently displayed, drag a window from the **Windows** pane on the left to the workspace.

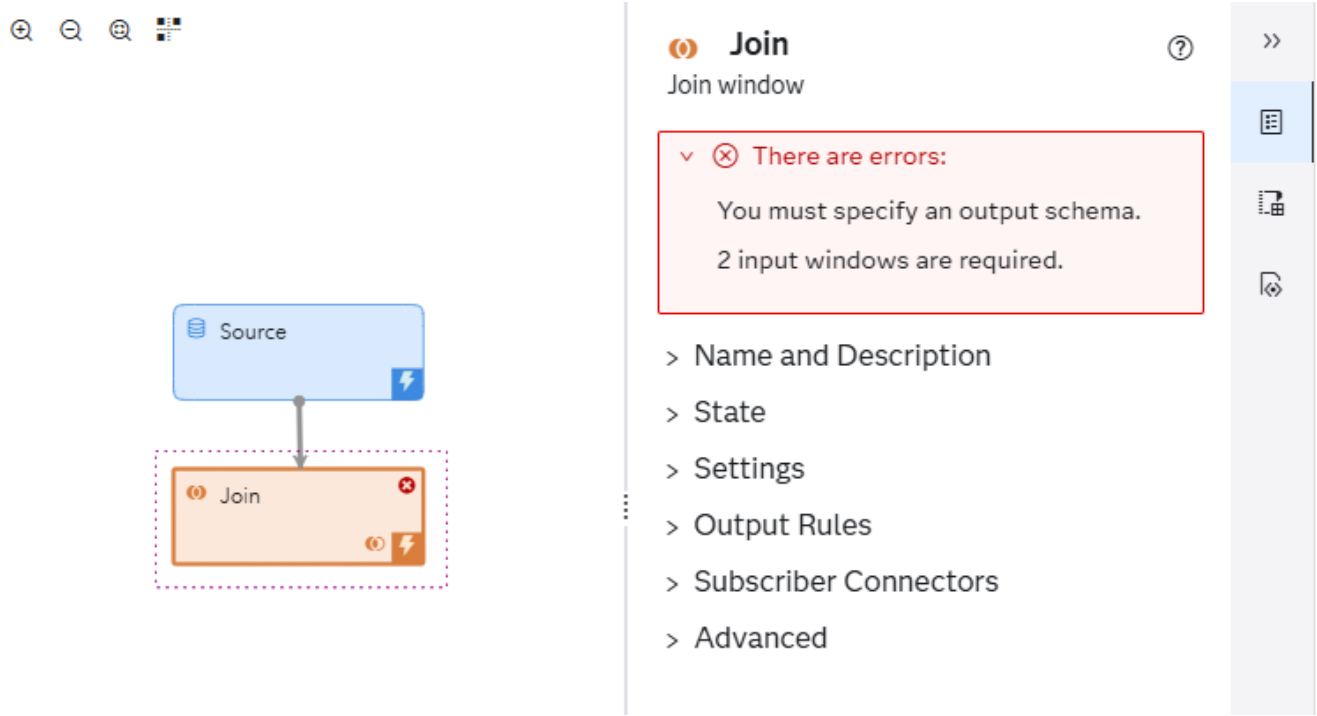
Note: Alternatively, you can add a window to the continuous query by double-clicking the relevant window in the **Windows** pane.

The windows are grouped into the following categories:

- Input Streams
- Transformations
- Utilities
- Analytics
- Custom

You must ensure that you enter valid properties for each window. Entering invalid window properties causes the window to display an error icon: . Here is an example of a model where the Join window contains invalid properties:

Figure 11 *A Window Validation Error Icon and Corresponding Message*



If a window requires further changes for the model to run successfully, the window displays a warning icon: . The right pane that displays the window's properties shows the corresponding warning message.

Connecting Windows

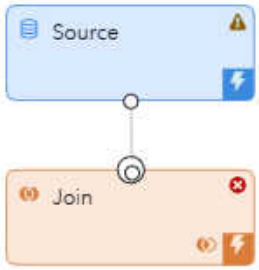
To connect a window to another window with an edge:

1 Position the cursor over the anchor point at the bottom of the window so that the anchor point color changes to white:

Figure 12 *Cursor over the Anchor Point*

2 Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point of another window:

Figure 13 *An Edge Connecting Two Windows*



The edge automatically connects to the window.

Note: You cannot change a connection by moving an edge from one window to another. Instead, you must establish a new connection by creating a new edge. If you have created a connection between two windows in error, you must delete the edge. To do this, select the edge that you want to delete and press **Delete**.

Edge Display Types

Connecting edges can appear differently depending on the type of connection between windows.

For information about edge display types, see the following table:

Edge Display Type	Connecting Edge	Details
Event stream	➡	Event stream edges connect input windows that contain event stream data.
Supporting data	→	Supporting data edges connect input windows that contain geometric data (that is, where the edge role is set to `Geometry`). They can also connect secondary input windows to a Join window.

Edge Display Type	Connecting Edge	Details
Non-data edges		Non-data edges connect windows that do not contain event stream data (that is, where the edge role is set to `Model` or `Request`).
Event forwarding		Event forwarding edges can connect Lua windows or Python windows to Source windows. When an event forwarding edge is inactive, the icon  is displayed on the edge. For more information, see "Forward Events to Source Windows".

Note: An invalid edge appears as a red dashed line in SAS Event Stream Processing Studio Modeler.

Edge Roles

You can use SAS Event Stream Processing Studio Modeler to configure the edge roles of connecting edges in your model. Edge roles must be specified for edges that connect streaming analytics windows and for edges that connect Geofence and Join windows. Each edge is assigned a role by default.

To change an edge's default role:

1 In the workspace, select the edge whose role you want to change.

2 In the right pane, in the **Role** field, change the default selection.

For information about edge roles, see the following table:

Window Name	Default Edge Role	Available Edge Roles	Dependencies
Join	Left table	Left table or Right table	If you assign an edge role to a Join window's connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.
Geofence	Position	Position or Geometry	If you assign an edge role to a Geofence window's connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.
Model Reader	Request	Request	Not applicable
Model Supervisor	Request	Request or Model	Not applicable

Window Name	Default Edge Role	Available Edge Roles	Dependencies
Calculate	Data	Data or Request	Not applicable
Train	Data	Data or Request	Not applicable
Score	Data	Data or Model	If you assign an edge role to a connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.

Delete a Window or an Edge

To remove a selected window or an edge from the model, press **Delete**.

Note: Deleting a window from a model automatically deletes all its connecting edges.

Window Icons

Each window in your model can display icons that represent its current state. For example, a Source window that contains a subscriber connector displays  . For information about each icon, see the following table:

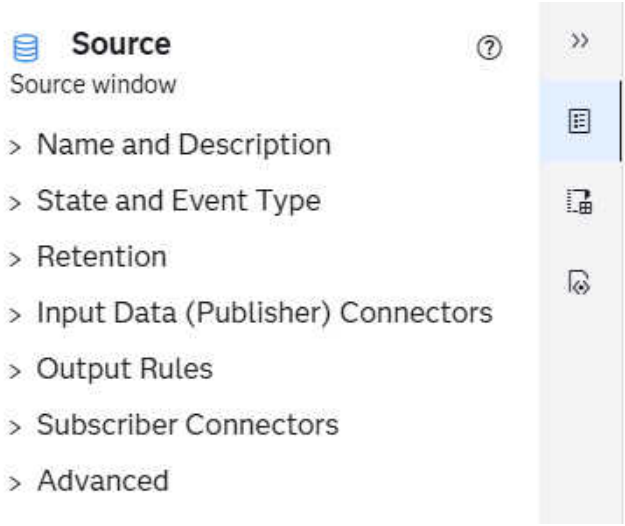
Icon	Description
	Indicates that the window contains an error message that will cause the model to fail to run.
	Indicates that the window requires further changes for it to be in a valid state.
	Indicates that the window contains a fully stateful index that is stored in memory. **Note:** If the window contains a non-stateful index, this icon is not displayed. This color of this icon changes depending on the window that contains it. This example shows an icon on a Source window.

Icon	Description
	Indicates that the window contains a fully stateful index that is stored on disk. **Note:** If the window contains a non-stateful index, this icon is not displayed. This color of this icon changes depending on the window that contains it. This example shows an icon on a Source window.
	Indicates that the Source window contains a publisher connector.
	Indicates that the Source window contains a subscriber connector.
	Indicates that the Join window contains an inner join type.
	Indicates that the Join window contains a full outer join type.
	Indicates that the Join window contains a right outer join type.
	Indicates that the Join window contains a left outer join type.
	Indicates that the window is used by the Scoring API as an input window.
	Indicates that the window is used by the Scoring API as an output window.
	Indicates that this window forwards events to a window in another continuous query.
	Indicates that this window receives forwarded events from a window in another continuous query.

Configure the Properties of a Window

To configure the properties of a window, click the window in the workspace. The right pane displays the properties for that window. Edit the properties as required.

Figure 14 *Properties of the Source Window*



Note: If the right pane displays XML code or an output schema, you can view the window's properties by clicking in the right pane to display the properties instead.

Configure the Output Schema of a Window

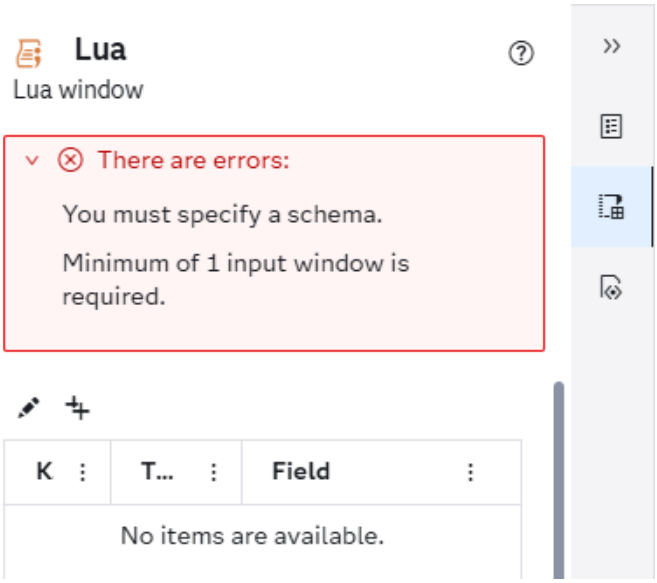
The output schema of a window defines the structure of the data that is passed from that window to the next window.

To configure the output schema for a window:

1 Select the window in the workspace.

2 Click on the right toolbar.

Figure 15 *An Example of an Empty Output Schema*



3 Click .

4 In the Output Schema window, configure the output schema as required:

- Add new schema fields or import existing schema fields from the parent window.
- Specify which fields are key fields.
- Specify the field order, to determine the sequence in which data appears in output events.

5 Click **OK**.

TIP For a quick way to import all schema fields from the parent window, instead of clicking in step 3, click . The button is available for selected window types.

Configuring Connectors

Connectors enable you to stream events into or out of SAS Event Stream Processing. You can use a publisher connector to stream events into a Source window. You can use a subscriber connector to stream events from a window to a destination that is outside the project. For more information, see "Using Connectors to Publish and Subscribe Event Streams" in *SAS Event Stream Processing: Overview*.

An alternative way to pass events to a project is to use the Scoring API. For more information, see "Using the Scoring API" in *SAS Event Stream Processing: Using Source and Derived Windows*.

Configure a Connector

1 Open a project.

2 In the workspace, select the window for which you want to configure a connector.

3 In the right pane, expand **Input Data (Publisher) Connectors** or **Subscriber Connectors**.

4 Click  .

5 Use the Connector Configuration window to set properties of the connector.

The properties that are available depend on the connector that you select in the **Connector type** field. For more information about each type of connector, see "What Are Connectors and Adapters?" in *SAS Event Stream Processing: Connectors and Adapters*.

6 Click **OK**.

Configure a File and Socket Connector

Here is an example of how to configure a file and socket connector to publish data from a CSV file into a Source window. The steps assume that you are using SAS Event Stream Processing Studio in a Kubernetes environment and that a persistent volume has been set up to enable file storage. The steps include setting a date format for the connector.

1 Open a project.

2 In the workspace, select the Source window for which you want to configure a publisher connector.

3 In the right pane, expand **Input Data (Publisher) Connectors**.

4 Click .

5 In the **Name** field of the Connector Configuration window, adjust the connector's default name as required.

6 In the **Fsname** field, click  and use the Select File window to select the CSV file that contains the events that you want to publish into the Source window. Here are some things to consider when you are selecting a file:

- Navigating to a file is less prone to error than manually entering a path.
- In a Kubernetes environment, the base location for files is always **`/mnt/data`**. However, the corresponding location where the input files reside depends on how your file storage has been set up. If you require assistance, contact your system administrator.
- When the project is part of a project package, you can use the Select File window not only to select an existing file but also to upload a file from your computer to the project package (that is, upload a file to the PV). For more information, see "Working with Project Packages".

7 Observe that the **Fstype** field is automatically set to **csv** because you selected a CSV file.

8 Click **All properties**.

9 In the All Properties window, locate the `dateformat` row and enter the date format in the Value column. Here is an example of a date format: `%d/%b/%Y:%H:%M:%S`. The appropriate date format depends on how the data in the CSV file is formatted.

Note: When the events in the CSV file do not include opcodes and flags, the `addcsvopcode` property is automatically set to `true` and the `addcsvflags` property is automatically set to `normal`. When the CSV file includes a header on the first row, the `header` property is automatically set to `1`.

10 Click **OK** to return to the Connector Configuration window.

11 Click **OK**.

TIP Alternatively, to quickly create a Source window with a file and socket publisher connector, click and drag a file from the **Project Package** pane to an empty part of the workspace. If you drag the file to an existing Source window instead, a connector is added to that window. You can make further adjustments to the Source window:

- If you want to adjust the connector's default properties, in the right pane expand **Input Data (Publisher) Connectors** and double-click the connector to edit it.
- When the file that you used is a CSV file, fields from it are added to the window's output schema. You must edit the output schema to select key fields, and you might want to make other adjustments to the output schema. For more information, see "Configure the Output Schema of a Window".

For more information about the file and socket connector, see "Using the File and Socket Connector and Adapter" in *SAS Event Stream Processing: Connectors and Adapters*.

When you use a file and socket connector to configure a publisher connector, the SAS Event Stream Processing Studio user who runs the project must have Read access to the input file. When you use a file and socket connector to configure a subscriber connector, the SAS Event Stream Processing Studio user must have Write access to the directory that you want to write the file to. For more information, see "Understanding File Access" in *SAS Event Stream Processing: Troubleshooting*. Using a project package can help ensure that you have access to locations of input files and output files. For more information, see "Project Package".

TIP You can use a WebSocket rather than a file and socket connector to stream events from a CSV file into a Source window. In some cases, this alternative might be more suitable to test a project. For more information, see "Publish Events from a CSV File".

Encrypt a Password in a Connector

Some connectors and adapters require you to enter a password in the connection's configuration details. SAS Event Stream Processing Studio enables you to encrypt passwords in connectors and adapters.

- To encrypt the password in a connector, use the Connector Configuration window. For more information, see the following example.
- To encrypt the password in an adapter, use the password encryption tool on the **Settings** page. For more information, see "Encrypt a Password in an Adapter".

Here is an example of using the Connector Configuration window to add an MQTT connector and encrypt the MQTT password:

1 Open a project.

2 In the workspace, select the Source window for which you want to configure an MQTT connector.

3 In the right pane, expand **Input Data (Publisher) Connectors**.

4 Click  .

5 In the **Name** field of the Connector Configuration window, adjust the connector's default name as required.

6 In the **Connector type** field, select **MQTT**.

7 Click **Encrypt password**.

8 In the Encrypt Password window, enter the user ID.

9 Enter the password that you want to encrypt and click **Encrypt password**.

10 Click **Copy** to copy the encrypted password to the clipboard.

11 Click **Close**.

12 In the Connector Configuration window, click **All properties**.

13 In the All Properties window, paste the encrypted password into the **mqttpassword** field.

14 Set the **mqttpasswordencrypted** field to **true**, and then click **OK**.

15 Complete the rest of the fields in the Connector Configuration window to suit your needs, and then click **OK**. For more information, see "Using the MQTT Connector" in *SAS Event Stream Processing: Connectors and Adapters*.

Here is another example. For the Database connector, use the encrypted password to create the system data source name (DSN) that you enter in the **Connectstring** field in the Connector Configuration window.

```
DSN=Oracle Wire Protocol;uid=exampleuserid;pwd=pkrrtO8hAlzcgkUUrO5zRhgZjePLR0HgqQYWtyLr5W
70yUjT3Wf1Eo72VGd4i6s3;hostname=example_DB_server_name.com;servicename=example.process_name"
```

For more information, see "What Are Connectors and Adapters?" in *SAS Event Stream Processing: Connectors and Adapters*.

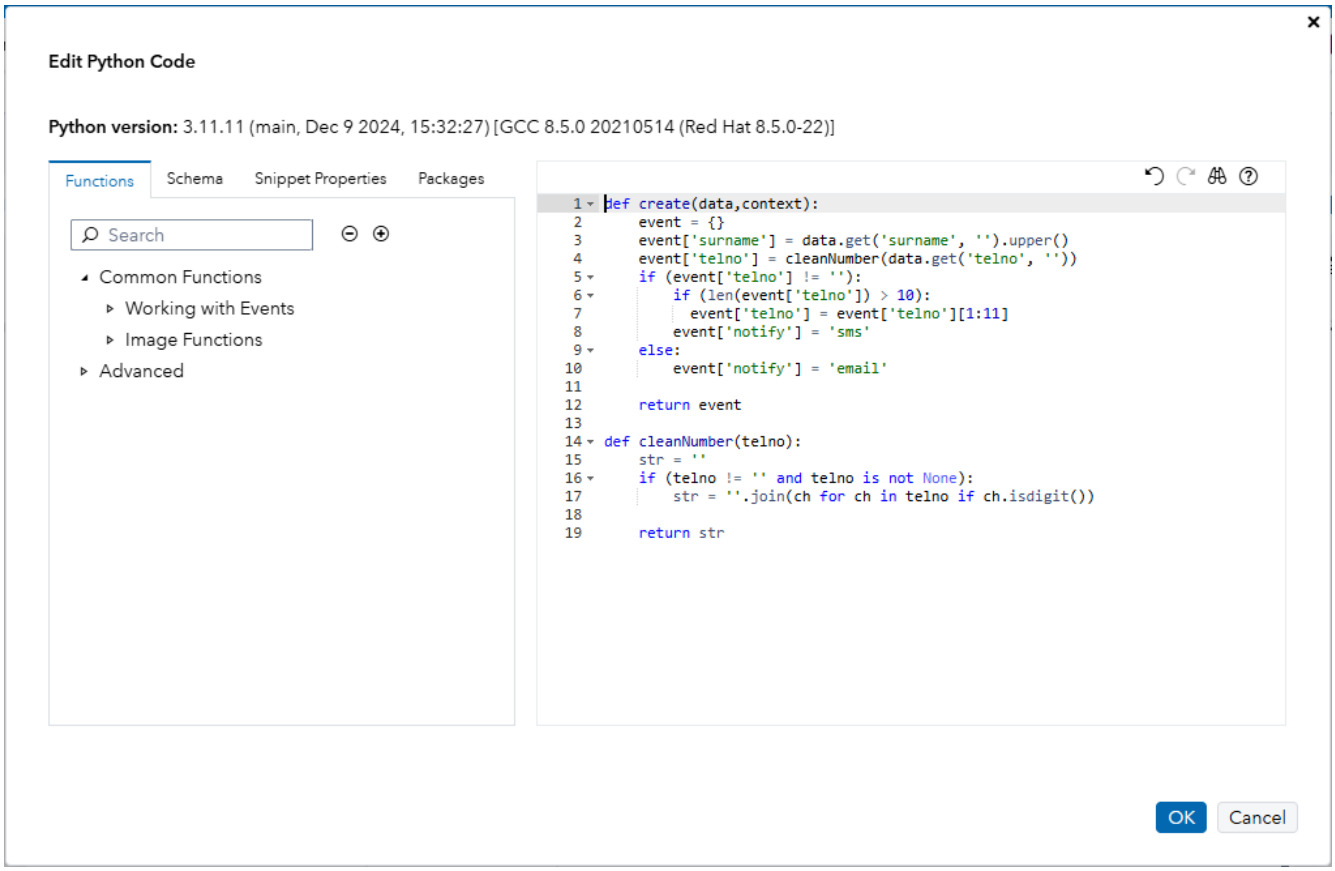
Edit Code and Expressions

Overview of Editing Code and Expressions

When you are entering large blocks of code, it is recommended that you use the editor that is available in SAS Event Stream Processing Studio for editing code and expressions.

In some cases, you access the editor by clicking the $x=\frac{y}{z}$ icon next to the relevant field. In such cases, the editor window often has a title such as Edit Lua Code, Edit Python Code, or Expression Editor.

Figure 16 *An Example of Editing Python Code*



Sometimes the editor is embedded within another window. For example, when you edit Lua code in a Lua-based Pattern window, the editor is embedded within the New Pattern or Edit Pattern window.

The functionality in the editor depends on the language of the code that you are editing. The editor can contain two panes.

Use the Left Pane to Add Items to Code

The left pane enables you to search for and add items to your code. Depending on the language of the code that you are editing, you can search for and add items such as functions, schema fields, snippets, templates, events, and variables.

When you edit Python code, the Python version is displayed above the left pane.

To add an item, position the cursor in the relevant location in the code in the right pane and double-click the item in the left pane.

The following information is specific to editing Lua and Python code:

- The **Functions** tab lists SAS Event Stream Processing functions that you can invoke within Lua or Python code. For more information about functions that you can use in Lua code, see "Using SAS Event Stream Processing Functions with Lua" in *SAS Event Stream Processing: Using Source and*

Derived Windows. For more information about functions that you can invoke from Python code, see "Using SAS Event Stream Processing Functions with Python" in *SAS Event Stream Processing: Using Source and Derived Windows*.

- The **Schema** tab lists fields that can be used in your Lua or Python code.
  - When you edit Lua code in a Lua window or a Lua-based Filter window, or you edit Python code in a Python window or a Python-based Filter window, the **Schema** tab lists a subset of fields from the input windows. The list reflects the settings in the **Fields to use in Lua code** field or the **Fields to use in Python code** field:
    - If you use the **Fields to use in Lua code** field or the **Fields to use in Python code** field to include specific fields, then those fields are listed on the **Schema** tab.
    - If you use the **Fields to use in Lua code** field or the **Fields to use in Python code** field to exclude specific fields, then all other fields are listed on the **Schema** tab.
    - If the **Fields to use in Lua code** field or the **Fields to use in Python code** field is empty, then all fields are listed on the **Schema** tab.
  - When you edit Lua code in a Pattern window, the **Schema** tab lists all fields from the input windows.
- When the **Snippet Properties** tab is available, it lists Lua snippets or Python snippets, and properties that relate to those snippets. Double-clicking a property in the left pane adds code that relates to that property to the right pane, rather than copying snippet code.

  For example, when a snippet sets a property every five minutes, the code that is added to the right pane gets that property, in order to read the relevant variable. Similarly, when a snippet is set to run periodically to refresh a token and then set that token, the code that is added to the right pane gets that token. When you are working with Lua code, only a Lua window or Lua snippet can set property values, whereas any Lua entity can get these values. When you are working with Python code, only a Python window or Python snippet can set property values, whereas any Python entity can get these values.

  When you hover over a snippet on the **Snippet Properties** tab, the description for that snippet is displayed. Providing descriptions when you create snippets can help you select the appropriate snippet. For more information about creating Lua snippets, see "Working with Lua Snippets in SAS Event Stream Processing Studio". For more information about creating Python snippets, see "Working with Python Snippets in SAS Event Stream Processing Studio".
- The **Packages** tab is available when you edit Python code. This tab provides a quick way to check which Python packages are available in your SAS Event Stream Processing environment. The list of packages is provided for information only: you do not select packages in the left pane and add them to the right pane.

Use the Right Pane to Manually Enter Code

The right pane contains a code area that enables you to manually enter code. The editor highlights the syntax by using a color scheme. Use the following buttons in the editor to perform specific operations on your code:

Icon	Description
	Reverts your previous change.
	Reverts the effects of the undo action.
	Validates your code in the editor. After the validation has completed, a message informs you of the code's validity.
	Provides access to Help. For example, when you are editing Lua code in a Lua window and you click this button, a Help topic is displayed with information about Lua code in Lua windows.
	Indicates that your code contains an error. The editor displays a message that specifies the error and its location. For a more detailed description of the error, position the cursor over the icon.

The editor enables you to select and insert existing code elements to complete code that you have partially entered. After partially entering code in the editor, you are presented with a list of suggestions for automatic completion. To accept the first suggestion from the list, press Tab. To accept one of the alternative suggestions, press the down arrow key until the relevant suggestion is selected, and then press Tab.

When you edit EEL expressions, the editor enables you to include operators in combination with numbers, strings, functions, and functions that use other functions. The editor contains a toolbar that enables you to access the operators. The toolbar is highlighted with a red box in the following example.

Figure 17 *An Example of an EEL Expression in the Editor*

For a description of an operator, position the cursor over the relevant operator on the toolbar. To insert an operator, position the cursor in the relevant location in the code and click the operator on the toolbar. For more information about operators that are available when you edit EEL code, see "Operators" in *Expression Language: Reference Guide*.

Disable or Enable a Window

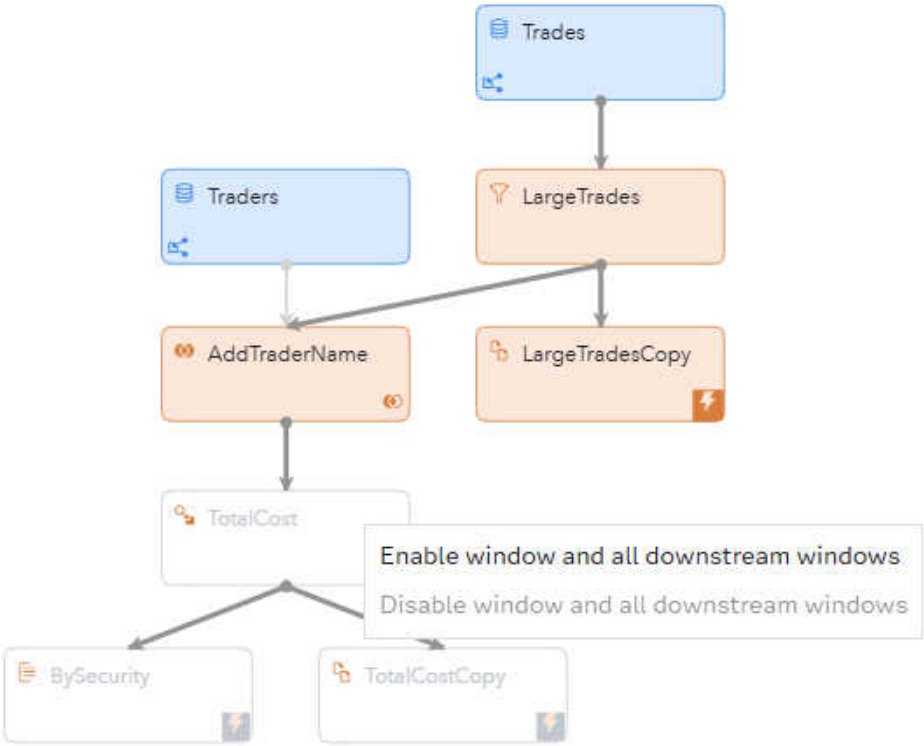
Disabling a window can be helpful when you are debugging a project. For example, when you are debugging memory issues or complex logic that includes `esp.publish()` functions, disabling some windows can help you figure out which windows are causing the problem. Disabling rather than deleting windows means that after you have identified and addressed the problem, you can quickly enable the relevant windows.

Disabling a window also disables any downstream windows. Likewise, enabling a window enables any downstream windows.

When you run a project, disabled windows are treated as if they were not part of the project and their connectors do not run. When a disabled window is included in connector orchestration, that window's connectors are treated as having reached their target state (running, stopped, or finished).

To disable or enable a window, right-click the window in the SAS Event Stream Processing Studio Modeler workspace and select **Enable window and all downstream windows** or **Disable window and all downstream windows**. When you disable a window, its color changes. For example, when the Light application theme is used, a disabled window changes to white.

Figure 18 *Example of Disabling and Enabling Windows*



Using the XML Editor

Using the XML Editor

SAS Event Stream Processing Studio Modeler includes the XML Editor. You can use it as an alternative way of creating models, compared to the visual modeling capabilities in SAS Event Stream Processing Studio Modeler. The workspace displays a snapshot of your model's XML code. You can use the XML Editor to rename a window. To do this, select the window that you want to rename in the workspace and then change the window's name in the XML Editor.

CAUTION

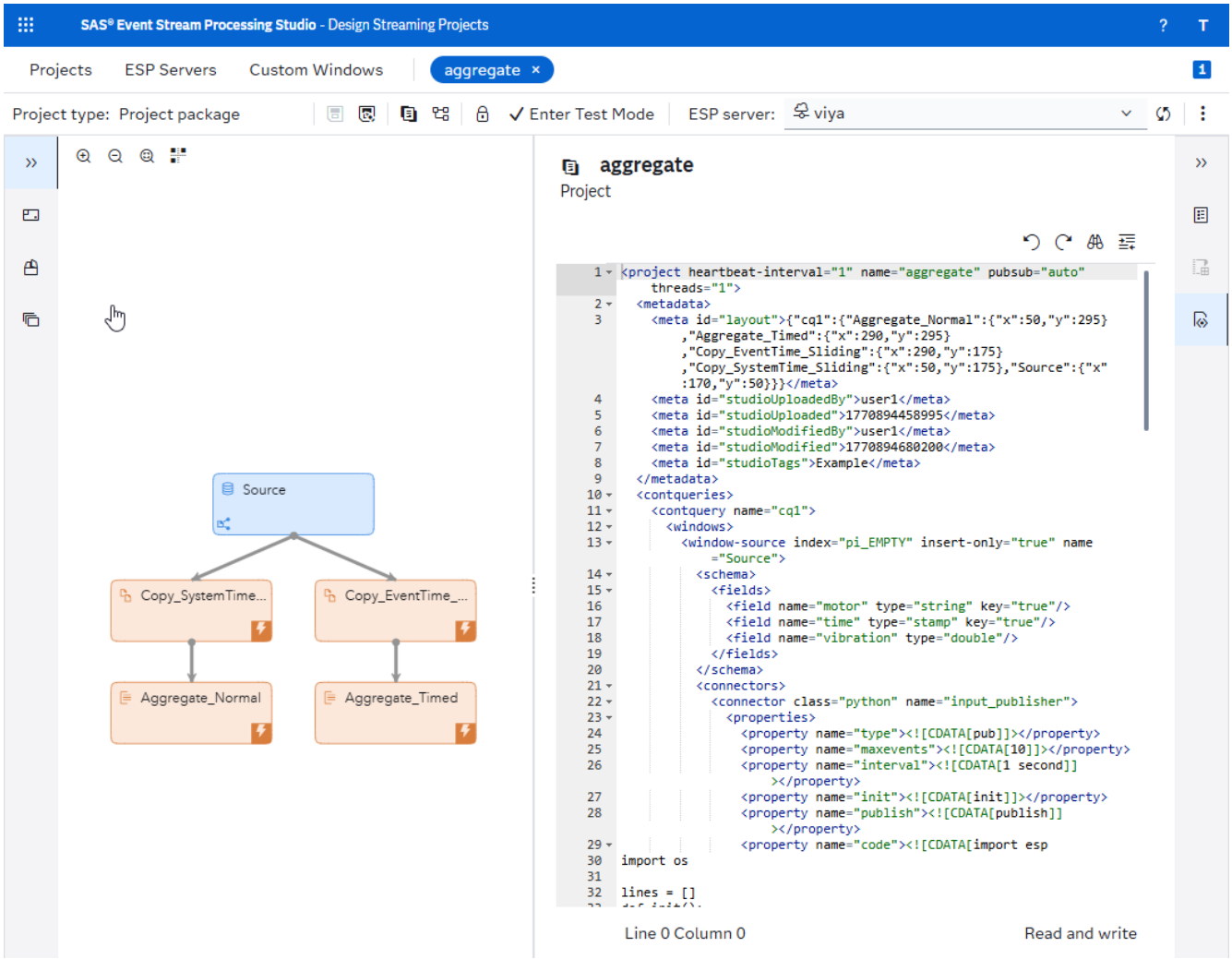
Manually editing your model's XML code using the XML Editor can result in an invalid model. Using SAS Event Stream Processing Studio Modeler to construct your model limits the possibility of your model containing invalid XML code. You must correct any invalid XML in the XML Editor before you can switch back to the Properties pane. Changes that you make manually in the XML Editor are not always reflected in the workspace. Using the XML Editor to rename a window, without first selecting it, results in the window being replaced by a new window in a default position on the workspace. This invalidates your

model. Any connections to or from the redundant window must then be deleted and then re-created in the workspace. Alternatively, you can manually edit the edges in the XML Editor.

To open the XML Editor, open a project and click on the right toolbar.

The right pane displays the XML Editor.

Figure 19 *The XML Editor*



Selecting a specific element in your workspace reloads the XML Editor to display only the corresponding section of XML code. To display the entire project's XML again, click .

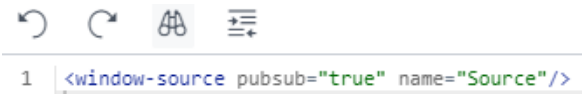
If you selected a specific ESP window, clicking an area of white space in your workspace reloads the XML Editor to display the XML relating to your model's continuous query.

If you add a comment to your XML code that is not enclosed within its relevant XML element, the comment is automatically moved within the XML element. This occurs when the XML code is reordered. For example, the XML code is reordered if you save your model or if you move away from the XML editor.

When a model is created, optional attributes are not included in its XML code. Models also contain settings that are not directly specified in the model's XML code, but they are represented in the user

interface. For example, if a Source window has a default window state of **(inherit from query) pi_HASH**, the implied setting is not displayed in the XML code. See the example shown here:

Figure 20 *The XML Editor Displaying a Source Window's XML Code without the Window's Default State*



Changing the window's state from its default value includes the attribute in the model's XML code, as shown here:

Figure 21 *The XML Editor Displaying a Source Window's XML Code with the Window's Default State*

Unique identifying project information is displayed within the `<metadata>` element in your project's XML code. Although the metadata is displayed in your project's XML code, it is located in the SAS Event Stream Processing Studio database. For more information, see "Project Metadata".

Using Editing Tools and Keyboard Shortcuts

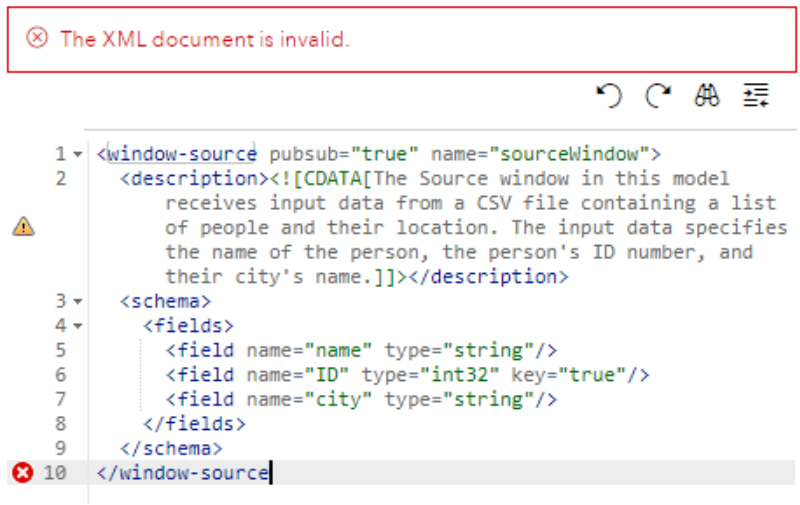
The XML Editor includes a toolbar that contains editing tools. These tools are also accessible using keyboard shortcuts.

Icon	Action	Keyboard Shortcut
	Reverts your previous change	Ctrl + Z
	Reverts the effects of the undo action	Ctrl + Y
	Prompts you to search for specific text. Pressing Ctrl + F again prompts you to replace the text that you have searched for.	Ctrl + F
	Formats the XML code that you have manually entered	Not available
No icon	Removes the code that you have selected from its original position	Ctrl + X
No icon	Copies the code that you have selected to the clipboard	Ctrl + C
No icon	Pastes the code on the clipboard at the cursor's position	Ctrl + V
No icon	Selects all of the code in the XML editor.	Ctrl + A

Validation

The XML Editor automatically validates the syntax of the code that you enter. If you enter invalid code, the XML Editor displays the following error message:

Figure 22 *Invalid XML Warning*



The XML error message also indicates the error's location with a breadcrumb trail. Each section of the breadcrumb trail represents an XML tag in your XML code. The top-level XML tag in your XML code is not included in the breadcrumb trail.

Position the cursor over the warning icon ⚠ to view a generic description of the error in your XML code. The icon ⊗ also displays the location of the error in your XML code.

For a more detailed description of the error, position the cursor over the icon ⊗.

Efficiency Tips

For some window types, you can copy schema fields between windows if there are windows in your model that use the same fields. In the Output Schema window, click ⇲ to open the Copy Fields from Input Schema window. Select the schema fields that you want to copy and click **OK**. Alternatively, you can use the XML Editor to copy and paste the fields between the windows.

Note: This functionality is not available for window types where it is not appropriate for schema fields to be copied from another window. For example, you cannot copy schema fields to or from windows that contain schemas that are implied or have been internally generated.

Continuous Queries

Add a Continuous Query

To add a new continuous query to your model, click ⋮ on the toolbar and select **Add continuous query**. You can also delete continuous queries from your model by selecting **Delete continuous query**.

To switch between each continuous query in your model, select the continuous query that you want to view from the **Continuous Query** drop-down list on the toolbar.

Configuring Continuous Queries in SAS Event Stream Processing Studio

Continuous queries allow SAS Event Stream Processing to analyze and manipulate data. *Continuous queries* are queries that run automatically and periodically on data in real time.

Models must contain at least one continuous query. SAS Event Stream Processing Studio Modeler creates a continuous query `cq1` by default. You can then add and configure windows within this continuous query. Your model can contain many continuous queries.

Your continuous query must contain at least one Source window. Source windows connect to one or more derived windows (for example, a Pattern or Join window). After you have created a Source window, you can then add derived windows to your model.

Configure the Properties of a Continuous Query

1 On the **Projects** page, right-click the project that contains the continuous query that you want to configure, and select **Open Project**.

   SAS Event Stream Processing Studio Modeler appears. The right pane displays the project's properties.

2 Click on the toolbar.

   The right pane displays the properties of the continuous query that you selected when the project was last saved.

3 Configure the fields as required.

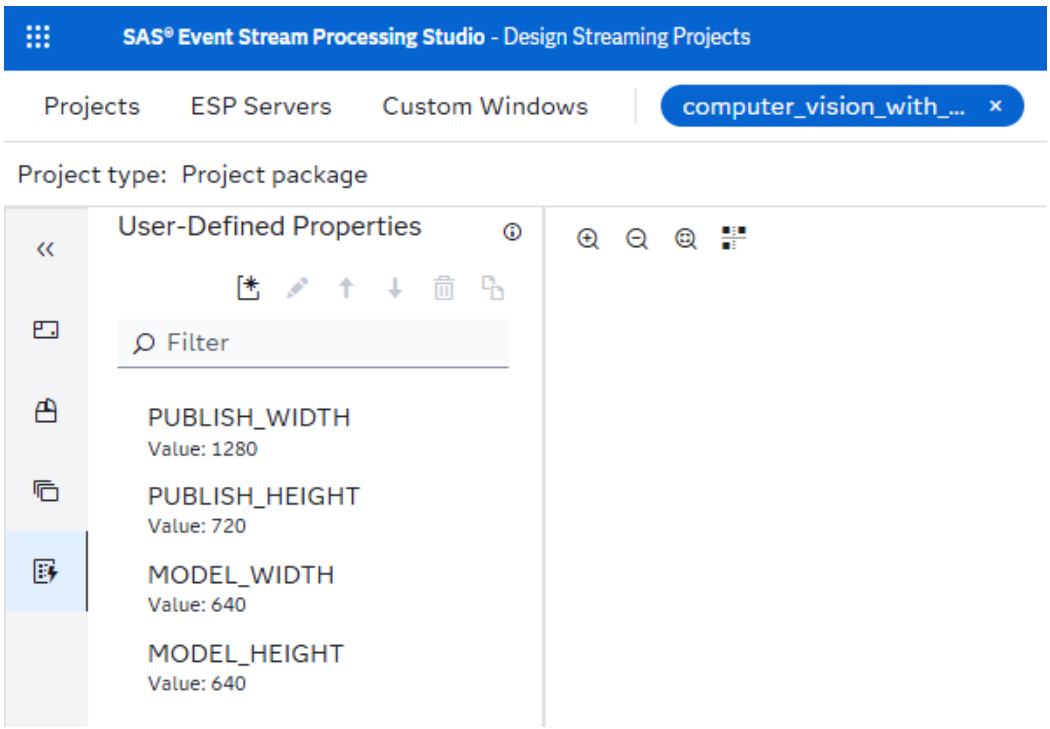
Working with User-Defined Properties

Overview of User-Defined Properties

User-defined properties enable you to define certain values once and reuse them throughout a project, instead of entering the same value repeatedly. User-defined properties can help you maintain consistency in a complex project. For example, you might define a property for a window state, an environment variable, or a file path. You can then refer to that property wherever the value is needed in the project.

User-defined properties are managed in SAS Event Stream Processing Studio Modeler, in the **User-Defined Properties** pane. The following figure shows what the **User-Defined Properties** pane looks like for the Computer Vision with ONNX example.

Figure 23 *User-Defined Properties Pane*



In this example, user-defined properties are used for resizing images to define the resolution of the published image frames. For more information about how to access the full example, see “Install Example Projects”.

Add a User-Defined Property

1 Open a project.

2 Click on the left toolbar.

3 In the **User-Defined Properties** pane, click .

4 Use the Property window to specify the user-defined property and click **OK**.

The property appears in the **User-Defined Properties** pane. You can use the ↑ and ↓ buttons to change the order in which the properties are listed, which can be helpful if the project contains a large number of user-defined properties. However, this order has no effect on how user-defined properties work when you run the project.

Reference a User-Defined Property

After you add a user-defined property, you can reference it from another location within the same project by copying the property from the **User-Defined Properties** pane and pasting it to the desired location.

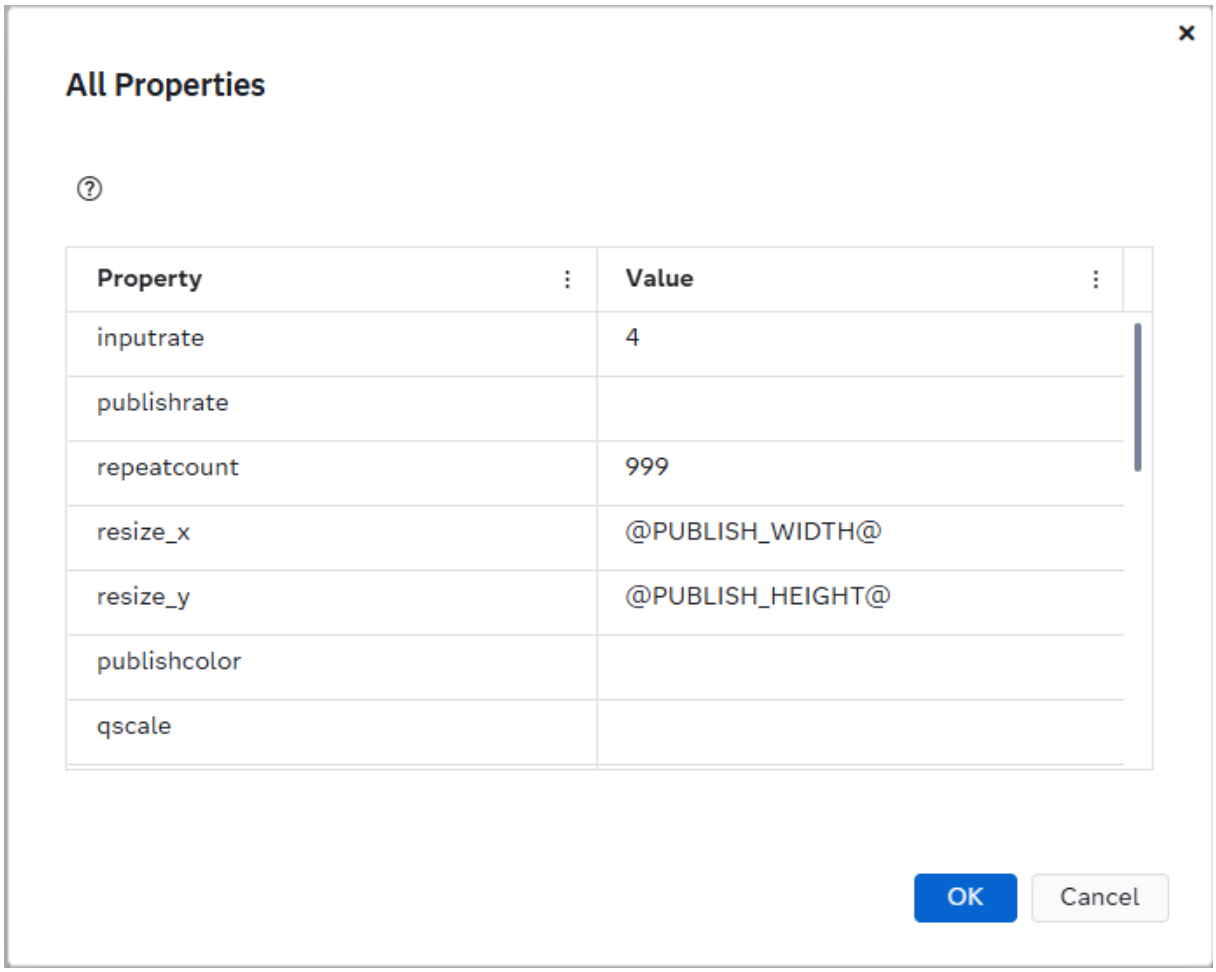
To copy and paste a user-defined property:

1 In the **User-Defined Properties** pane, select a property.

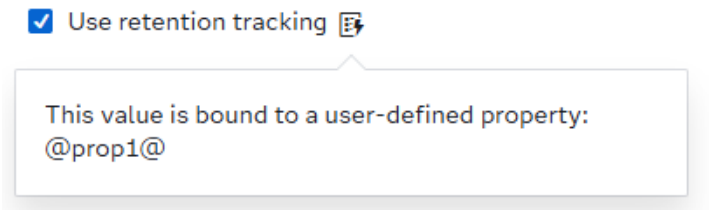
2 Click to copy the user-defined property. The property is enclosed in @ symbols. That is, the format for the copied text is `@property_name@`.

3 Paste the copied reference to another location within SAS Event Stream Processing Studio Modeler.

- You can paste a copied reference into a field that accepts text input. The following example shows user-defined properties being used in a connector configuration.



- You can paste the copied reference into XML code within the XML Editor. This approach enables advanced use cases where user-defined properties control values that you cannot edit as text in the user interface, such as the value of a check box. When this approach is used, is displayed adjacent to the user interface control to indicate that its value is bound to a user-defined property. The following example shows what the user interface looks like when the XML code includes `retention-tracking=@prop1@`:



Testing Models in SAS Event Stream Processing Studio

Running a Test

You can use SAS Event Stream Processing Studio to verify that your model operates as intended. You can analyze how incoming data is transformed into meaningful event streams that can be consumed by subscribers.

1 On the **Projects** page, right-click the project that contains the model that you want to test.

2 Select **Open project** from the menu.

The project appears in a new page.

Note: Ensure you have saved any changes that you have made to the project. Projects that contain unsaved changes cannot be opened in test mode.

3 Click ✓ Enter Test Mode.

A page appears, enabling you to test your model.

4 In the **ESP server** drop-down list, verify that an ESP server has been selected to perform the test on.

Note: When the test is run in a Kubernetes cluster, an ESP server is created on demand in the Kubernetes cluster. When the project is stopped, the ESP server is deleted from the Kubernetes cluster. For more information, see "Working with a Kubernetes Cluster in SAS Event Stream Processing Studio". If you are using an edge server, you must register an ESP server before you can continue testing your model. You can register a new ESP server on the **ESP Servers** page.

5 When your project contains several continuous queries, the **Continuous query** drop-down menu is available in the left pane. Use it to select the relevant continuous query.

6 Use the left pane to select the windows and fields whose results you want to view. By default, the first 15 fields for each window are selected. Blob fields are an exception: they are never selected by default.

> **TIP** An icon appears next to each window name and field name. Position the cursor over the icon to identify a window type or a field type.

Note: The left pane reflects the output schemas of windows at the time when you entered test mode. If you subsequently edit the output schema of a window and save the project, you can click ↻ Refresh schema to refresh the left pane.

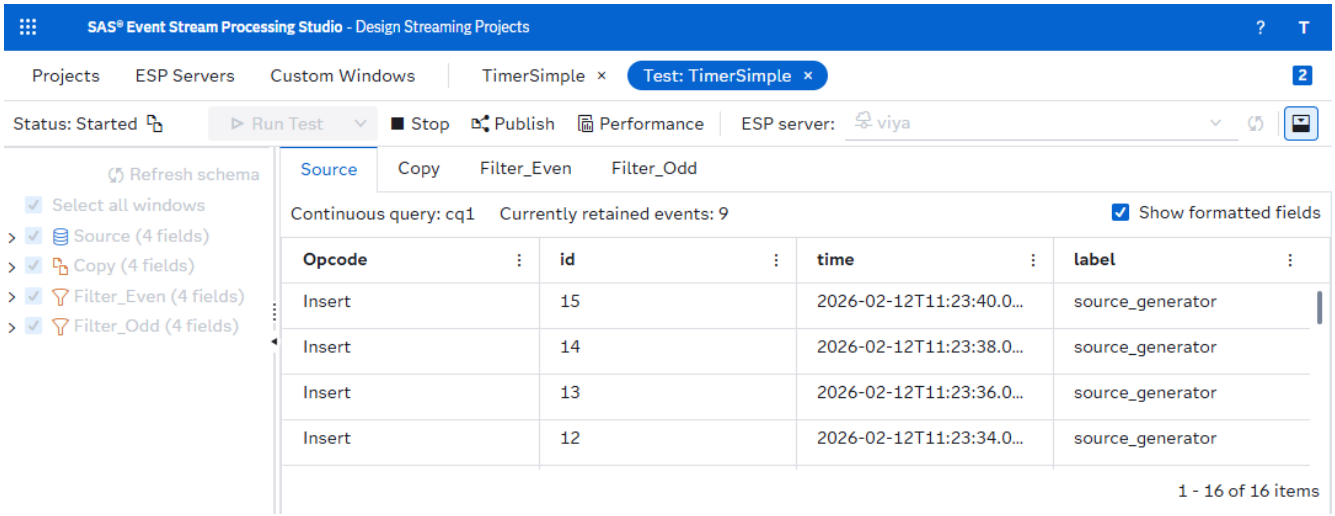
7 To run the test, click ▷ Run Test.

When the test is run in a Kubernetes cluster, you can change the settings that are used for creating the ESP server in the Kubernetes cluster.

- Clicking ▷ Run Test runs the test with the same settings that were used last time when you ran a test. If you have not adjusted the settings previously, default settings are used.
- To change the settings, click ▾ next to ▷ Run Test, and then click ⚙ Configure and Run Test. The Load and Start Project in Cluster window appears. Adjust the settings if required and click **OK**. For more information about the settings that are available in this window, see "Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster".

8 Test results appear. Each window's results appear in their corresponding tab.

Figure 24 *Test Mode*



The Status indicator on the toolbar informs you of your test's current status. The following table shows some possible test statuses:

Table 3 *Test Statuses*

		Environment in Which the Status Appears	
Status	**Description**	**Kubernetes**	**Edge**
Initial	The test is in an initial state and has not started.	✓	✓
Starting	The test is starting.	✓	✓
Started	The test has been started and is running.		✓
Containers are not initialized	The Kubernetes cluster is processing the request to start the test.	✓	

Status	Description	Environment in Which the Status Appears	
		Kubernetes	Edge
Pending creation	The Kubernetes cluster is processing the request to start the test.	✓	
Container is running	The test is running in the Kubernetes cluster.	✓	
Loading SAS Micro Analytic Store module `count` of `total`	The value for `total` indicates the total number of SAS Micro Analytic Store modules to be loaded and `count` indicates how many modules have been loaded thus far.	✓	✓
Waiting for the project to be loaded	A project with a large amount of SAS Micro Analytic Store code is being loaded.	✓	
Stopping	The test is stopping.	✓	✓
Stopped	The test has stopped.	✓	✓
Test failed	The test failed. A window appears, with more information about why the test failed.	✓	✓
Unknown	The pod status is unknown. For more information, select **Pod events** in the **Log** pane.	✓	

If a window in your model contains more than 1,000 results, only the last 1,000 results are displayed in test mode.

You can group information by column. The result table contains a horizontal bar at the top of the table, with the text **Drag a column header here to group by that column**. To group information by column, drag a column heading to the bar. If required, you can drag additional columns to the bar.

Alternatively, you can view your test results by opening the output file that you specified in your subscriber connector properties.

You can use the **Show formatted fields** check box to choose whether data appears exactly as it was received from the ESP server or with additional formatting. Here are some examples of additional formatting that is applied when the check box is selected:

- Dates and timestamps are shown as Coordinated Universal Time (UTC) in ISO 8601 format (for example, 2018-11-30T13:33:47.000Z). If you clear the check box, dates and timestamps appear in UNIX Epoch time, as this is the format in which the data is received from the ESP server.
- A dot is used as a separator in certain types of numerical data, rather than another separator, such as a comma. If you clear the check box and your locale is set to a locale that uses another separator, that separator is displayed instead of a dot.
- Opcodes are displayed using their localized names if your locale is not set to an English-language locale. If you clear the check box, opcodes are always shown in English, as this is how the data is received from the ESP server.

When a cell in the test mode results contains JSON or XML code and you hover over the cell, the button appears in the cell. Click the button to open a window where the code is formatted and syntax highlighting makes the code easier to read.

9 You can publish events from a CSV file to a model running in test mode through a WebSocket connection. This enables you to send events to a Source window in your model without configuring a publisher connector. This enables you to place the CSV file in a convenient location on your computer without having to upload it to a server. If you do not want to publish events from a CSV file to the model, skip this step.

To publish events from a CSV file, click Publish. For more information, see "Publish Events from a CSV File".

10 You can use the **Scoring API** tab in the bottom pane to send events for scoring and to view scored events. For more information, see "Use the Scoring API When Testing a Model".

11 To view performance information for the windows in the model, click Performance. For more information, see "View Performance Information for a Model".

Note: Performance information is not available when you run a test on an ESP server that runs on an edge server.

12 If your environment includes Grafana and the SAS Event Stream Processing Data Source Plug-in for Grafana, you can use them to visualize events. For more information, see https://github.com/sassoftware/grafana-esp-plugin.

13 To stop the test, click Stop.

14 When you have finished testing your model, close the page.

Note: If the test fails and the resulting error message does not explain what caused the failure, you can troubleshoot the problem by checking the test logs in the Log pane. For more information, see "Viewing a Model's Test Logs".

When the project is part of a project package and the project uses subscriber connectors to write output to files, you can access the output files in the `output` folder of the project package. For more information, see "Manage a Project Package".

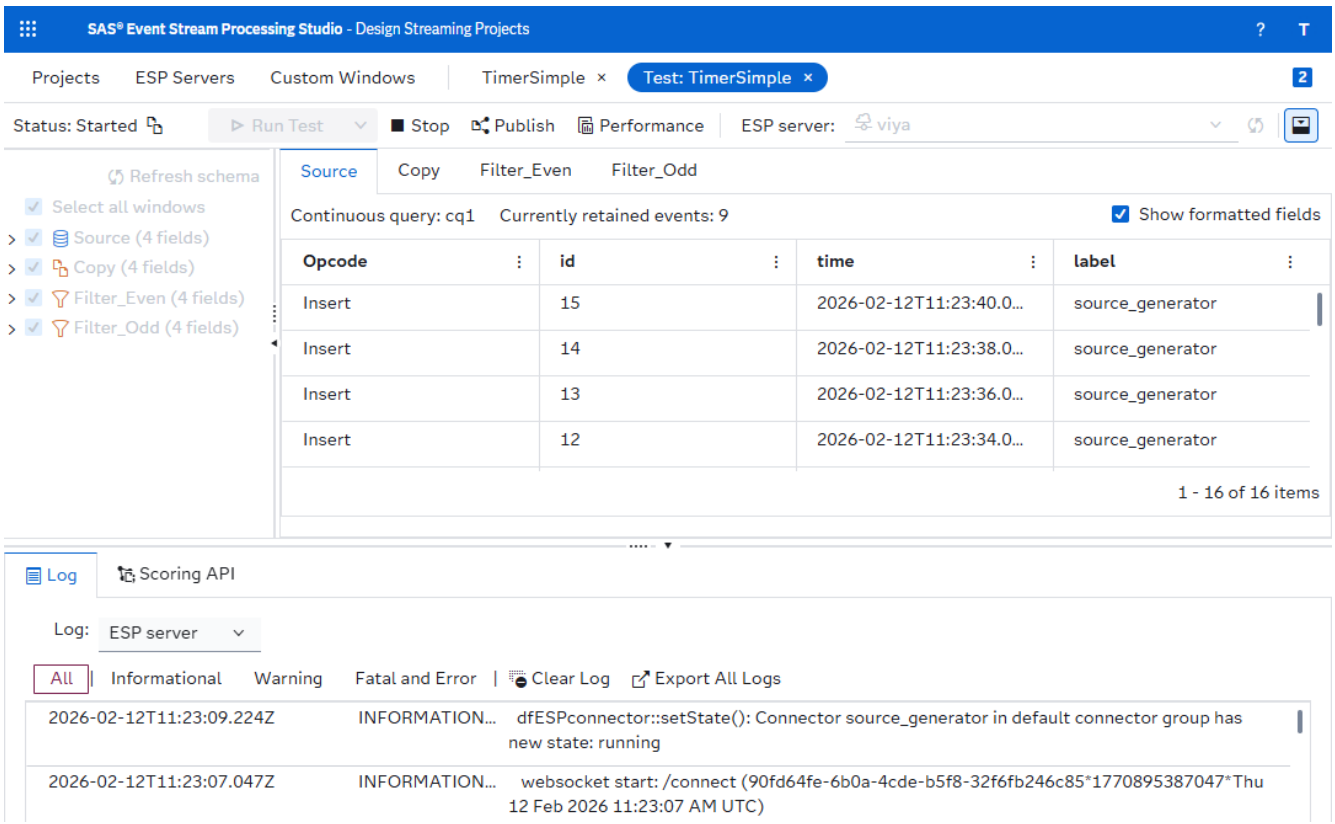
Viewing a Model's Test Logs

Overview to Viewing Test Logs

You can monitor your model in real-time by viewing the model's logs in test mode. Log messages appear in the Log pane, a horizontal pane located at the bottom of test mode.

Historical log messages are not displayed. SAS Event Stream Processing Studio displays messages that were logged after you ran the model in test mode. If you select another ESP server or close test mode for the specific model, log messages that were displayed in SAS Event Stream Processing Studio are not retained.

Figure 25 *Test Mode with Logging Enabled*



The following topics explain how to use the controls in the Log pane.

Select the Source of the Log Messages

Use the **Log** drop-down list to select the source of the log messages:

- **ESP server** – Messages from the ESP server log. These messages relate to the ESP server that is running the model. TRACE-level messages are not displayed. Some message details are not

displayed in the user interface. To view the complete message text, including TRACE-level messages, export the log for the Kubernetes pod and view its contents. For more information, see "Export the Logs".

- **ESP operator** – Messages from the ESP operator log. The ESP operator is a Kubernetes operator, a software extension that uses custom resources to manage applications. For more information about Kubernetes operators, see the relevant Kubernetes documentation. This option is not available for an ESP server that runs on an edge server. When this option is available, it enables you to view Kubernetes messages that might otherwise need to be accessed by a system administrator by using the `kubectl` command. If the pod does not start and you cannot access log messages in the Log pane, you can access the ESP operator log on the **Settings** page instead. For more information, see "ESP Operator".
- **Console** – Standard output (stdout) messages that are created by the ESP server but are not included in the ESP server log. This option is not available for an ESP server that runs on an edge server.
- **Pod events** – Messages that are created by the Kubernetes cluster for a specific pod. This option enables you to view Kubernetes messages that might otherwise need to be accessed by a system administrator by using the `kubectl` command.

Filter Log Messages

To filter log messages by message type, select one or more of the following options in the Log pane:

- All – Shows all log message types, except TRACE-level messages from the ESP server log.
- Informational – Shows general log messages that are not warnings or errors.
- Warnings – Shows only warning messages.
- Fatal and Error – Shows only error messages, including fatal errors and normal errors.

If you select **Console** or **Pod events** from the **Log** drop-down list, the ability to filter log messages is not available.

Clear the Log Pane

To clear the contents of the Log pane, click Clear Log.

Export the Logs

To export all logs to text files, click Export All Logs. A ZIP file that contains the log files is exported to your computer. The location into which the ZIP file is exported might vary depending on your browser's configuration.

The following log files are available:

- **`pod.log`** – The full log for the Kubernetes pod. The log contains the complete text of all ESP server messages, including TRACE-level messages (if TRACE-level logging is enabled), and all console messages.

> **TIP** If the exported Kubernetes pod log does not contain the level of logging detail that you want to view, you can change the logging properties on the **User Session** section of the **Settings** page. For example, you could change the logging level for a particular logger from INFO to TRACE. For more information, see "Settings for the User Session".

- `previous pod.log` – This file is present only when the Kubernetes pod has been restarted. The file contains log entries for the original pod.
- `operator.log` – The log for the ESP operator.

Hide or Display the Log Pane

To hide the **Log** pane, click  on the toolbar. To display the pane again, click  .

Use the Scoring API When Testing a Model

When you run a test in SAS Event Stream Processing Studio, you can use the **Scoring API** tab in the bottom pane to send events for scoring and to view scored events.

For an example of a project that uses the Scoring API, install the Image Processing with Scoring API example. For more information, see "Install Example Projects".

Ensure that a scoring input window and a scoring output window have been specified at the project level. For more information, see "Enable the Scoring API".

To send events for scoring and to view scored events:

1. Click the **Scoring API** tab.
2. When the **Input field** field has been specified at the project level, you can perform the following tasks:
   - When the project is part of a project package, you can click  to select a file that contains raw data that you want to score. For example, you can select an image file. The file must be located in the `test_files` folder of the project package.
   - Instead of selecting a file that is already in the project package, you can click  to upload a file that contains raw data that you want to score. For example, you can select an image file.
   - Click on the left toolbar to view or edit the event as JSON code.
3. When the **Input field** field has not been specified at the project level, you can perform the following tasks:
   - Click  to add an event to the table. You can then use the newly added table row to specify the details of the event that you want to send for scoring.
   - Click  to remove events from the table.

- Click to save this scoring configuration for later use. The file is saved to the project package's `test_files` folder, within the `scoring_configurations` subfolder. Saving a scoring configuration is possible only when your project is part of a project package.
- Click to import a previously saved scoring configuration.
- When the project is part of a project package, you can select a table row and click to select a file that contains raw data that you want to score. For example, you can select an image file. The file must be located in the `test_files` folder of the project package.
- Instead of selecting a file that is already in the project package, you can select a table row and click to upload a file that contains raw data that you want to score. For example, you can select an image file.
- Click on the left toolbar to view or edit the events as JSON code.

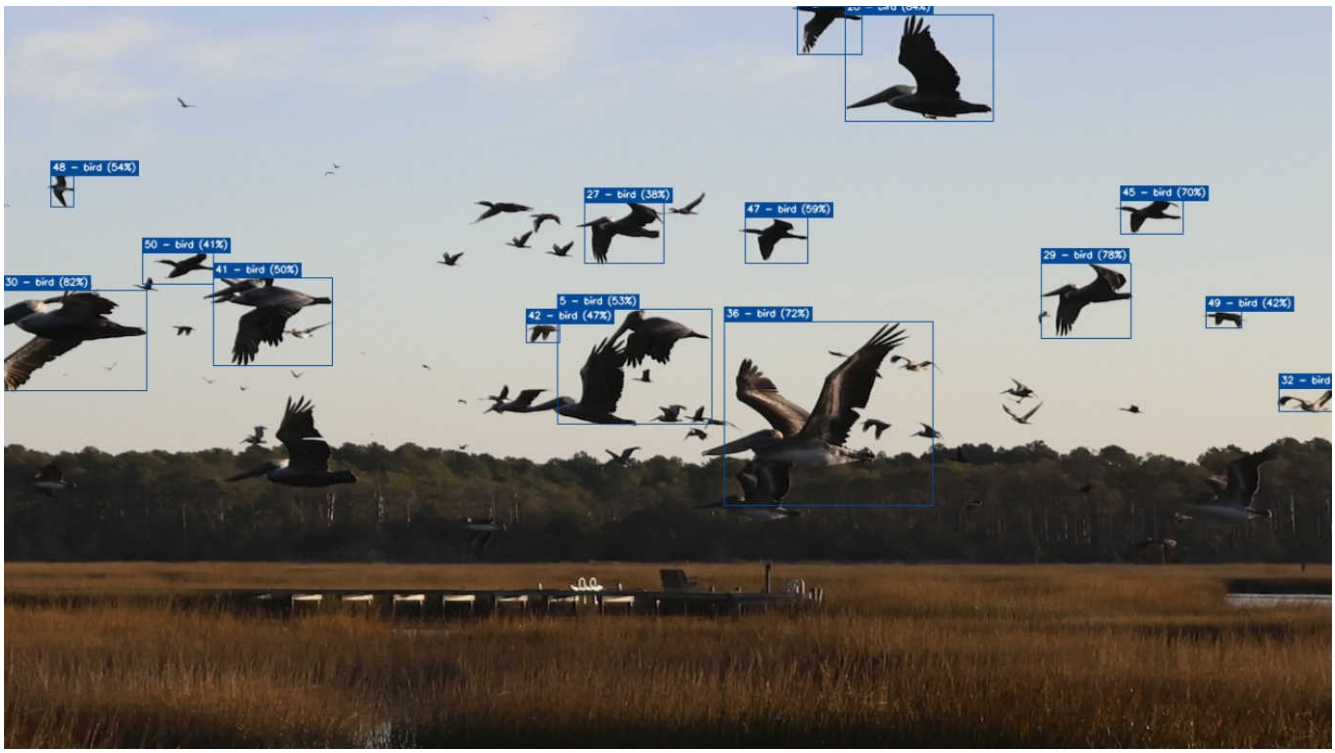
4 Click **Send request**.

5 When the **Output field** field has not been specified at the project level, the response events are displayed in the **Output** table. Click to view the events represented as JSON code.

6 When the **Output field** field has been specified at the project level, the response contains raw data, such as an image. The Save Output File window is displayed.

- When the response is a single event, the Save Output File window attempts to detect the file type and display a preview of the file contents. Preview is available for selected file types, including GIF, JPG, MP4, and PNG. If the file type is not detected, enter a file name and a file extension. Click **Save** to save the output file. The file is saved in the project package, in the `/output/scoring_outputs` folder. When the project is not part of a project package, the file is downloaded to your computer instead.

Figure 26 *Example of a Preview for a Scored Image File*



- When the response contains multiple events, one JSON response that contains information about all returned files is returned. The Save Output File window displays the JSON code. Click **Save** to save the output file. The file is saved in the project package, in the `/output/scoring_outputs` folder. When the project is not part of a project package, the file is downloaded to your computer instead.

View Performance Information for a Model

When you suspect that your model has performance issues, viewing performance information can help you identify the windows that are causing the issues.

When you run a test in SAS Event Stream Processing Studio, click Performance on the toolbar to view performance information.

You can also access performance information from the **ESP Servers** page. For more information, see "Managing ESP Servers in SAS Event Stream Processing Studio".

Note: Performance information is not available when a project runs on an ESP server that runs on an edge server.

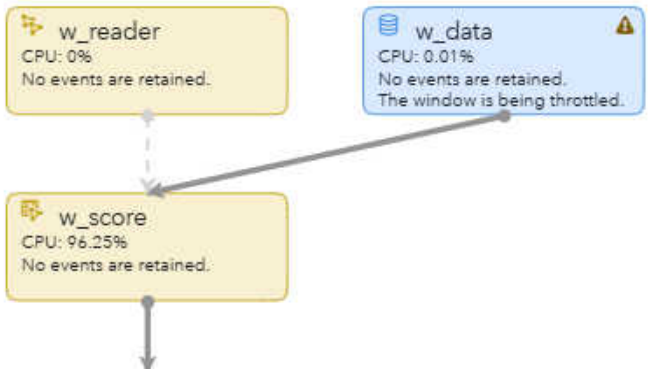
The following fields are displayed at the top of the page:

- The **System memory** field displays how much memory is available on the Kubernetes node.
- The **Virtual memory** field displays how much memory is allocated to the project (that is, the Kubernetes pod). This is not a hard limit. As resident size grows, the ESP server requests more virtual memory.
- The **Resident memory** field displays how much memory the project (the Kubernetes pod) actually consumes.
- The **Limit for memory** field displays how much memory you specified the project (the Kubernetes pod) is allowed to use. You can specify a limit for memory in the Deployment Settings window, which is accessed from the Load and Start Project in Cluster window.

The read-only project diagram on this page displays summary information for each window in the model, including CPU usage and event count. When a window is stateless, no events are retained.

The and icons can be displayed in the windows, depending on the severity of issues. For example, the icon is displayed when a window is throttled (a connector is blocked because data arrives faster than the ESP server can process it) or when a connector fails.

Figure 27 *Example of Performance Information in the Diagram*



When you select a window in the diagram, the tabs below the diagram display further information for that window:

- The **Health** tab displays various metrics for the window, such as the amount of time the window has been throttled and the current state of the window's connectors. For more information about these metrics, see "Understanding Project Health Metrics" in *SAS Event Stream Processing: Troubleshooting*.
- The **Events** tab displays events in the latest event block that has passed through the model. For example, if each event block contains one event, then this tab displays one event at a time.
- The **ESP Server Log** tab displays messages from the ESP server log. The controls on this tab are similar to the **Log** pane in test mode, except that only the ESP server log is displayed. For more information about the controls, see "Viewing a Model's Test Logs" on page 73.

The CPU usage information for each window is created by inspecting a time period and checking what percentage of that time the CPU is busy. For example, if a one-second period is inspected and the CPU is busy for the entire one-second period, then 100% CPU usage is reported. If a one-second period is inspected and the CPU is busy for half of that time, then 50% CPU usage is reported.

When a window uses more CPU resource than other windows in the model, this does not necessarily indicate a problem in the model's design. The windows that are designed to handle the most events in your model are also the windows that are likely to use more CPU resource than windows that handle fewer events. Similarly, when a window is throttled, the project is managing its intake of events. Throttling that occurs in a Source window can prevent throttling from occurring in subsequent windows.

When you use the Load and Start Project in Cluster window to specify the deployment settings for a test, take care to specify an appropriate amount of CPU resource. Specifying an insufficient amount of CPU resource for a resource-intensive model might result in warnings about CPU usage and throttling. The warnings might indicate an insufficient CPU resource allocation rather than a problem with in the model's design.

Publish Events from a CSV File

You can publish events from a CSV file to a model running in test mode through a WebSocket connection. This enables you to send events to a Source window in your model without configuring a

publisher connector. This enables you to place the CSV file in a convenient location on your computer without having to upload it to a Kubernetes persistent volume or to a server.

Note: Ensure that the input data you want to publish does not contain a header row. Input data that contains a header row will fail to publish to a model.

To do this:

1 In test mode, start the model that you want to publish events to. For more information, see "Running a Test".

2 Click  .

The Publish Data from File window appears.

3 In the **Source window** drop-down list, select the Source window that receives events from the CSV file.

Note: If your model contains multiple Source windows, you must repeat this procedure for each Source window in your model. You cannot use this functionality to publish events to multiple Source windows simultaneously.

4 In the **CSV file** field, click **Browse** and navigate to the location of the CSV file.

5 Navigate to the CSV file that you want to publish events from and click **Open**.

6 In the **Date format** drop-down list, select the date format used when events that contain formatted dates are sent.

7 In the **Default opcode** drop-down list, select the default opcode to use when an input event does not include the opcode. Valid values are insert, update, upsert, and delete. The default value is insert.

If an opcode has been provided in the input data, the default opcode that you have selected is ignored.

8 In the **Default flag** drop-down list, select the default flag to use when an input event does not include a flag. Valid values are normal and partial update. The default value is normal.

If a flag has been provided in the input data, the default flag that you have selected is ignored.

9 In the **Block size** field, enter the number of events to put into an event block for publishing. The default is 1 event. Therefore, the publisher injects each event as it is received by default.

10 In the **Maximum event rate to be maintained (events per second)** field, enter the maximum event rate to maintain in events per second. If the publisher attains this rate before 1 second has elapsed, it sleeps for the remainder of that second and then resumes publishing. When you do not set the rate, the publisher publishes events as fast as possible. The default value is 0.

11 In the **Pause between injection of events (milliseconds)** field, enter the number of milliseconds to pause between the injection of events. For each event received, when that event causes an event block to be injected into the ESP server, the publisher pauses for the specified number of milliseconds. The default value is 0.

12 Click **OK**.

You are returned to test mode. Events are published through the WebSocket connection.

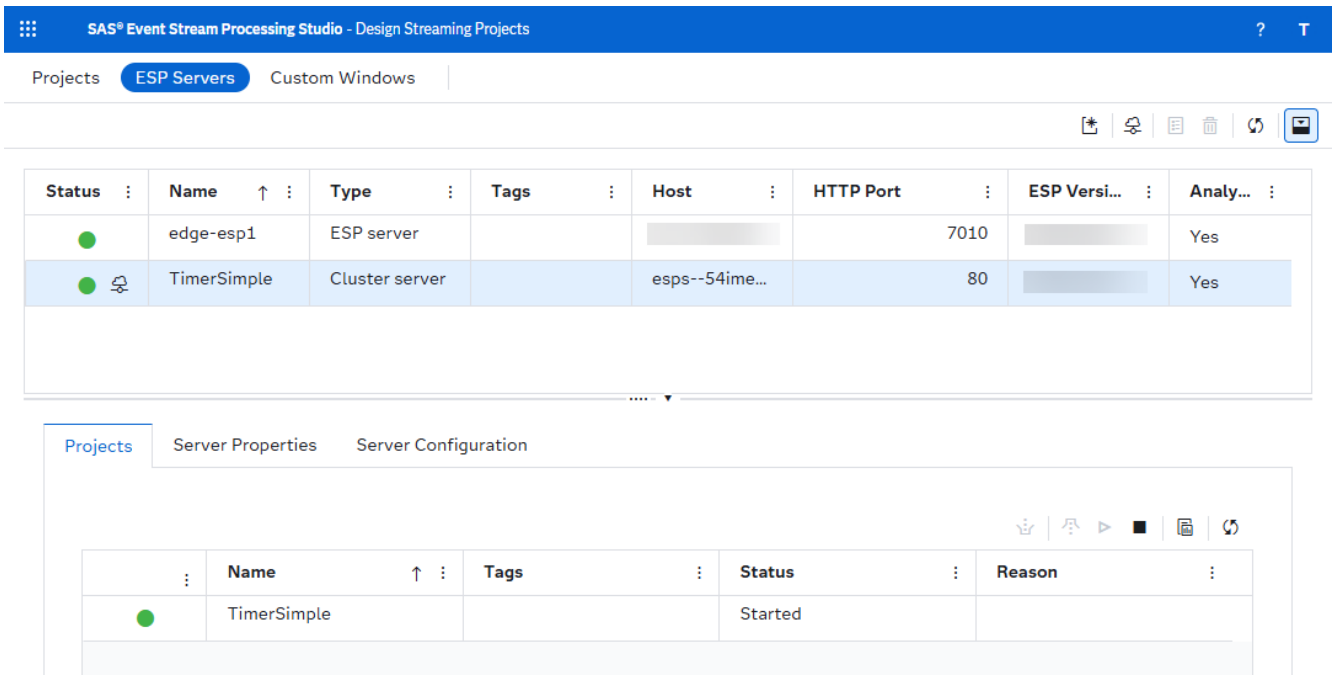
Managing ESP Servers in SAS Event Stream Processing Studio

Managing ESP Servers in SAS Event Stream Processing Studio

You can use the **ESP Servers** page to view details of existing ESP servers in your deployment. You can also use the controls on this page to add, edit, and delete ESP servers.

The following figure shows an example:

Figure 28 *The ESP Servers page*



The **ESP Servers** page displays the following information for each ESP server:

- The ESP server's status.
- The ESP server's name.
- The ESP server type:
  - ESP server: The ESP server is on an edge server

  - Cluster server: The ESP server is in a Kubernetes cluster
- Identifying tags assigned to the ESP server.
- The host on which the ESP server is running.

  Note: If a project is started on an ESP server that is in a Kubernetes cluster, the host name consists of the prefix `esps` followed by the name of the project that was started in the cluster and a unique identifier for the pod. These three parts are separated by hyphens. If the project name contains characters that do not conform to Kubernetes naming conventions, the resulting pod name contains changed characters. No characters are changed if the project name contains only lowercase letters (without diacritical marks), numbers, hyphens, or periods, and the name starts with a lowercase letter.

- The port that is used for HTTP administration requests.
- The SAS Event Stream Processing version installed on the host on which the ESP server is running.
- Whether streaming analytics is available on the ESP server.

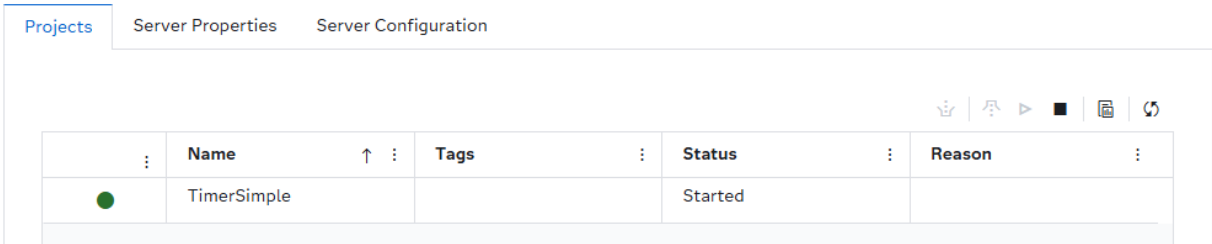
The Status column displays an icon summarizing the ESP server's condition. The condition of the ESP server is determined by the ESP server's connectivity and availability. This information helps you focus on the ESP servers that have problems. The following icons can appear in the Status column:

Icon	Description
	Undefined – The ESP server status has yet to be established.
	Available – The ESP server is on an edge server, available, and operating normally.
	Available in cluster – The ESP server is in a Kubernetes cluster, available, and operating normally.
	Unavailable – The ESP server is not available.

To view additional information for an ESP server, select the relevant ESP server on the **ESP Servers** page.

To hide this additional information, click  .

Figure 29 *The Projects Tab*



The **Projects** tab appears in the bottom pane by default and specifies the projects that have been uploaded to the ESP server selected.

You can use the **Projects** tab to load, unload, start, and stop a project on the ESP server.

To load a project onto the ESP server that is not in a Kubernetes cluster, click . In the Load Project window, select the project that you want to load from the table and click **OK**. This process uploads a working copy of the project to the ESP server. For information about loading and starting a project on an ESP server that is in a Kubernetes cluster, see "Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster". For information about stopping a project on an ESP server that is in a Kubernetes cluster, see "Delete an ESP Server That Is in a Kubernetes Cluster".

Note: Uploading projects that have already been published is not permitted.

After you have loaded the project onto the ESP server, you can run the project by clicking . The **Running Projects** field on the corresponding row in the table on the **ESP Servers** page is updated with the running project's name. To stop the running project, click and confirm that you want to stop the project. To unload the project from its ESP server, click .

To view performance information for the windows in the model, click . For more information, see "View Performance Information for a Model".

Note: Performance information is not available when a project runs on an ESP server that runs on an edge server.

The **Server Properties** tab in the bottom pane contains information about the ESP server's properties, such as the authentication method used on the ESP server. This tab also displays information such as the ESP server's host name, HTTP port, and whether Transport Layer Security (TLS) is enabled on the ESP server. For more information, see "Add or Edit an ESP Server to Run on an Edge Server". If the ESP server is on an edge server, you can click **Edit properties** to edit the ESP server's properties. The Edit ESP Server Properties window appears, enabling you to do this.

The **Server Configuration** tab contains information about connector types, streaming analytics, and whether SAS Event Stream Processing has been enabled to meter the number of events that are processed on the ESP server.

Working with a Kubernetes Cluster in SAS Event Stream Processing Studio

SAS Event Stream Processing Studio includes the ability to load and run projects on a Kubernetes cluster. When a project is loaded and started within a cluster, an ESP server is created on demand in the Kubernetes cluster. When the project is stopped, the ESP server is deleted from the Kubernetes cluster. Otherwise, an ESP server in a Kubernetes cluster behaves in the same way as an ESP server on an edge server.

From the **ESP Servers** page, you can load and start a project on an ESP server in a Kubernetes cluster. For more information about starting a project on an ESP server in a Kubernetes cluster, see "Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster". You can also stop a

project on an ESP server in a Kubernetes cluster. The ESP server is then automatically deleted from the Kubernetes cluster. For more information, see "Delete an ESP Server That Is in a Kubernetes Cluster".

Add or Edit an ESP Server to Run on an Edge Server

This topic applies to ESP servers that are to run on an edge server. Adding an ESP server adds the ESP server's details to the table on the ESP Servers page. This makes SAS Event Stream Processing Studio aware of that ESP server. The ESP server itself must exist for the ESP server's details to be added successfully to the table. For more information about how to start an ESP server in an edge environment, see "Starting the ESP Server in an Edge Environment" in *SAS Event Stream Processing: Using the ESP Server in an Edge Environment*. For information about ESP servers in a Kubernetes cluster, see "Working with a Kubernetes Cluster in SAS Event Stream Processing Studio".

1 If you want to add a new ESP server, on the **ESP Servers** page, click  .

The ESP Server Properties window appears.

Or

If you want to edit an existing ESP server's details, on the **ESP Servers** page, select the ESP server that you want to open and click  .

The Edit ESP Server Properties window appears.

2 Do the following to add or edit an ESP server:

a In the **Name** field, enter or update the name that identifies the ESP server.

b If required, in the **Description** field, enter or update the description of the ESP server.

c In the **Host** field, enter or update the ESP server's host name or IP address.

d In the **HTTP port** field, enter or update the ESP server's administration port number.

e If required, click **Edit** to change the setting for the **Authentication** field:

The Authentication window appears.

- **None**: This is the default option.
- **Kerberos**: This option is relevant only if the ESP server is configured to require authentication using Kerberos.

f If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the ESP server is configured to require SSL encryption.

g (Optional) Click **Test connection**. Testing the connection prevents the creation of an invalid ESP server connection and subsequent time-outs when the invalid connection is refreshed.

h If required, in the **Tags** field, enter or update any keywords that describe the ESP server and then press Enter.

i If required, select the **Enable server logging** check box to enable logging on the ESP server.

j If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the ESP server log. The default is 10,000 messages.

k Click **OK**.

Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster

For an introduction to using a Kubernetes cluster with SAS Event Stream Processing Studio, see "Working with a Kubernetes Cluster in SAS Event Stream Processing Studio".

1 On the **ESP Servers** page, click  and select **Start project**.

The Load and Start Project in Cluster window appears.

2 In the **Project** field, select the project.

3 When your project is not part of a project package, ensure that input files are available in the expected location. If your project uses connectors that require a Kubernetes persistent volume, you must place input files in the location that has been configured for the Kubernetes persistent volume. The base location where SAS Event Stream Processing expects input files to be located is `/mnt/data`, but the corresponding location where you need to place your input files depends on how your file storage has been set up.

If you require assistance, contact your system administrator.

4 (Optional) Adjust the values in the **Deployment settings** section. The **Deployment settings** section displays the settings that are used for creating an ESP server in the cluster. If the application can access previously used settings for the selected project, then those settings are displayed as default values. Otherwise, the project's computational requirements are used as default values. For more information, see "Specify Computational Requirements". Contact your system administrator for advice.

a Click **Edit deployment settings**.

The Deployment Settings window appears.

b Specify the requested (initial) amount and the maximum amount to be used by the ESP server for each of the following resources:

- Amount of memory.
- Amount of CPU resource.
- Amount of NVIDIA graphics processing unit (GPU) resource. The SAS Event Stream Processing Studio user interface contains one field for setting both the requested amount and the maximum amount because these amounts must be the same.

Note:

- The requested amount and maximum amount that you should set for memory and CPU resource depend on your circumstances. In many cases, the requested amount and the maximum amount can be set to the same value. However, to optimize the use of resources,

you can set different values. Setting different values means that the Kubernetes pod that runs the project can be started with a lower amount of resource than the maximum amount that the pod might need with peak usage. However, if you expect to leave the project running for a long time, having a significant difference between the requested amount and the maximum amount might result in the Kubernetes cluster starting the pod before the maximum amount is reached. That is, the Kubernetes cluster might start a pod when the cluster can meet the minimum amount that you requested and you might not get the maximum amount of resource that you specify. Lack of sufficient resource might then result in the project stopping unexpectedly.

- Your system administrator can set limits for the amount of memory and the amount of CPU resource that is allocated. If a value that you specify exceeds such a limit, then the value specified by your system administrator is used instead. The user interface displays an error message if you try to set a value that exceeds the value set by your system administrator.

c Click **OK**.

5 Click **OK**.

A new ESP server is created. The project that you loaded is started on the ESP server.

If your environment includes Grafana and the SAS Event Stream Processing Data Source Plug-in for Grafana, you can use them to visualize events. For more information, see https://github.com/sassoftware/grafana-esp-plugin.

Remove an ESP Server That Runs on an Edge Server from SAS Event Stream Processing Studio

You can remove an ESP server's details from the table on the **ESP Servers** page. This means that SAS Event Stream Processing Studio is no longer aware of that ESP server. However, the ESP server itself continues to exist. Removing an ESP server's details can be useful if the table contains ESP servers that are no longer used. You can remove an ESP server that is still running from the table. To remove a specific ESP server from the table:

1 On the **ESP Servers** page, select the ESP server that you want to remove from the table and click  .

The Remove ESP Server window appears.

2 Confirm the removal of the ESP server from the table.

Note: To remove multiple ESP servers simultaneously, press and hold Ctrl, select the ESP servers that you want to remove from the table, and click  . Then confirm the removal.

Delete an ESP Server That Is in a Kubernetes Cluster

1 On the **ESP Servers** page, select the project that you want to delete from the table.

2 Click and select **Stop project**.

> **TIP** Alternatively, in the **Projects** tab in the bottom pane, select the project that you want to delete and click ■.

The Stop Project and Delete ESP Server from Cluster window appears.

3 Click **Delete**.

The project is stopped and the ESP server is deleted from the Kubernetes cluster.

Overview to Versioning

You can use SAS Event Stream Processing Studio and SAS Event Stream Manager to manage multiple versions of a project.

The versioning functionality that is available in SAS Event Stream Processing Studio and SAS Event Stream Manager depends on how SAS Event Stream Processing was deployed.

When standalone SAS Event Stream Processing is deployed, file-based versioning is used to publish project versions. Project versions are stored in a Git repository that is internal to SAS Event Stream Processing and can be synchronized with an external Git repository. For more information, see "Overview of File-Based Versioning".

When SAS Event Stream Processing is deployed with other SAS products, project versions can be published using file-based versioning or SAS Viya platform services, depending on each project's type:

- When a project is part of a project package, project versions are published using file-based versioning.
- When the project is a project XML file, project versions are published using SAS Viya platform services. You can subsequently use SAS Event Stream Processing Studio to convert a project XML file to a project package that uses file-based versioning. For more information, see "Convert a Project to Use File-Based Versioning".

  When SAS Viya platform services are used to publish project versions, the Files service and the File Store service are used to store project versions in a database. For more information, see "Overview of Publishing Project Versions by Using SAS Viya Platform Services".

Publishing projects by using SAS Viya platform services is not available when standalone SAS Event Stream Processing is deployed.

Publishing Project Versions by Using File-Based Versioning

Overview of File-Based Versioning

To use file-based versioning, your project must be part of a project package. For more information, see "Project Package".

Benefits of File-Based Versioning

When file-based versioning is available and a project is part of a project package, the project's XML code and all other files within the project package are versioned. This is one of the benefits of file-based versioning.

Another benefit of file-based versioning is the ability to use promotion levels to manage project versions. There are three promotion levels: Development, Test, and Production. The purpose of promotion levels is to prevent non-production project versions from being deployed into a production environment.

- Development – This promotion level is intended for working on the design of your project in SAS Event Stream Processing Studio Modeler. You can save several versions of the project.
- Test – This promotion level is intended for testing project versions. When the SAS Event Stream Processing environment contains SAS Event Stream Processing Studio and a single instance of SAS Event Stream Manager, promoting a project version in SAS Event Stream Processing Studio from Development to Test makes the project version available to SAS Event Stream Manager. When the SAS Event Stream Processing environment contains multiple instances of SAS Event Stream Manager, you promote a project version in SAS Event Stream Processing Studio from Development to Test, and then import the project to the SAS Event Stream Manager test instance. In both cases, you can then use SAS Event Stream Manager to deploy a project version to a deployment that you are using for testing purposes.
- Production – When you are satisfied that a project version is ready for use in a production environment, you can use SAS Event Stream Manager to promote the project version from Test to Production. When the SAS Event Stream Processing environment contains multiple instances of SAS Event Stream Manager, you must also import the project to the SAS Event Stream Manager production instance. You can then use SAS Event Stream Manager to deploy the project version to a deployment that you are using for production purposes.

For more information about using SAS Event Stream Manager to promote and import projects, see "Managing Project Versions That Are Published Using File-Based Versioning" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

File-Based Versioning and Git

When file-based versioning is used, by default project versions are stored in a Git repository that is internal to SAS Event Stream Processing.

Project versions can also be synchronized with repositories on remote Git servers. Each project is stored in its own repository.

CAUTION
Use only SAS Event Stream Processing to interact with the internal Git repository and remote Git repositories. Using third-party tools to interact with the Git repositories could result in data loss.

CAUTION
Ensure that backup processes are in place. When the internal Git repository is used, backup processes must be in place for project data on the persistent volume that is used by SAS Event Stream Processing Studio. When project data is stored on a remote Git server, backup processes must be in place for the remote Git server. Not having backups in place could result in data loss.

To enable users to store projects in remote repositories, a system administrator must first specify a connection to a remote Git server. For more information, see "Specify Connections to Remote Git Servers".

You can then create a remote repository as follows:

- When you create a project, select the option to create a remote repository. For more information, see "Create a Project".
- Create a remote repository for an existing project. For more information, see "Create a Remote Repository for an Existing Project".

You can import an existing project that is stored on a remote Git server. For more information, see "Import a Project".

When the server type is GitHub or Gitea, you can store a project in a remote repository that is owned by your organization rather than by an individual user. For more information, see the following topics:

- "Import a Project"
- "Link a Project to an Empty Remote Repository"

TIP You can find out whether an existing project is stored in the internal Git repository or in a remote Git repository by viewing the Remote Repository column on the **Projects** page.

You can also find out the details of the remote Git server on which a project is stored. For more information, see "View the Details of a Remote Repository".

File-Based Versioning and Git Large File Storage

Git Large File Storage (LFS) is an extension to Git that is designed to manage large files efficiently. Instead of storing large files directly in the Git repository, Git LFS replaces those files with

references in Git. The actual file content is stored separately and retrieved only when it is needed. Git LFS is commonly used for large files that do not compress well or that change frequently.

Although Git LFS is often provided by the same platform as Git, it is not part of Git itself. Normal Git files (such as text files) are stored in the Git repository, whereas large files are stored separately by the Git LFS service. Even when you are using the same Git provider, Git LFS is distinct from standard Git storage: for example, Gitea LFS storage is distinct from standard Gitea storage.

Without Git LFS, storing large files directly in a Git repository can cause several problems:

- Many Git hosting products enforce a maximum file size. If you attempt to add a file that exceeds this limit, the file is rejected.
- In addition to per-file limits, Git hosting services often apply fair-use policies or overall repository size limits.
- Repository size grows quickly, as each change to a large file significantly increases repository size.
- Git is optimized for text files and you might experience poor performance with large binary files. Large binary files do not efficiently compare changes, which can significantly slow down common operations such as saving.

In SAS Event Stream Processing Studio, Git LFS can be used as part of file-based versioning to manage large files:

- When you create a project package, you can choose to use Git LFS for that project. Afterward, if you upload a binary file to the project package, then the file is automatically stored in Git LFS. If you upload a text file to the project package, then SAS Event Stream Processing Studio prompts you to choose whether you want to store the file in Git LFS.
- When you import a project package, you can choose to use Git LFS for that project. Binary files in the project package are automatically stored in Git LFS.

  Afterward, if you upload a binary file to the project package, then the file is automatically stored in Git LFS. If you upload a text file to the project package, then SAS Event Stream Processing Studio prompts you to choose whether you want to store the file in Git LFS.

Here are examples of file types that SAS Event Stream Processing Studio stores in Git LFS by default: JPG, MP4, PDF, PNG, and ONNX.

When Git LFS is used in SAS Event Stream Processing Studio, it provides the following benefits:

- efficient versioning of large files. Large files that are part of a project package are versioned without bloating the Git repository.
- improved performance, as Git transfers file references instead of full file content.
- consistent project versions. Large files remain logically part of the project version. When a project version is retrieved, the correct versions of large files are retrieved automatically.

Note: Git LFS can be used when a project package is stored in one of the following ways:

- in the Git repository that is internal to SAS Event Stream Processing
- in a remote repository in Gitea

IMPORTANT SAS Event Stream Processing does not include a Git LFS service. To use Git LFS functionality in SAS Event Stream Processing Studio:

- Your organization must have a Git LFS service, and the Git LFS service must be enabled in your organization's Git provider.
- Your system administrator must configure access to the Git LFS service when creating a remote repository connection. For more information, see "Specify Connections to Remote Git Servers".

The option to use Git LFS for a project is available when you complete these tasks:

- "Create a Project"
- "Import a Project"
- "Install Example Projects"

In the **Project Package** pane, the text **(LFS)** adjacent to a file name indicates that a file is stored in Git Large File Storage. For an example, see "User Interface Controls".

When you have enabled Git LFS for a project, you cannot later disable it for that project. However, you can click  and in the Save As window clear the **Enable Git Large File Storage (LFS)** check box before saving the project with another name.

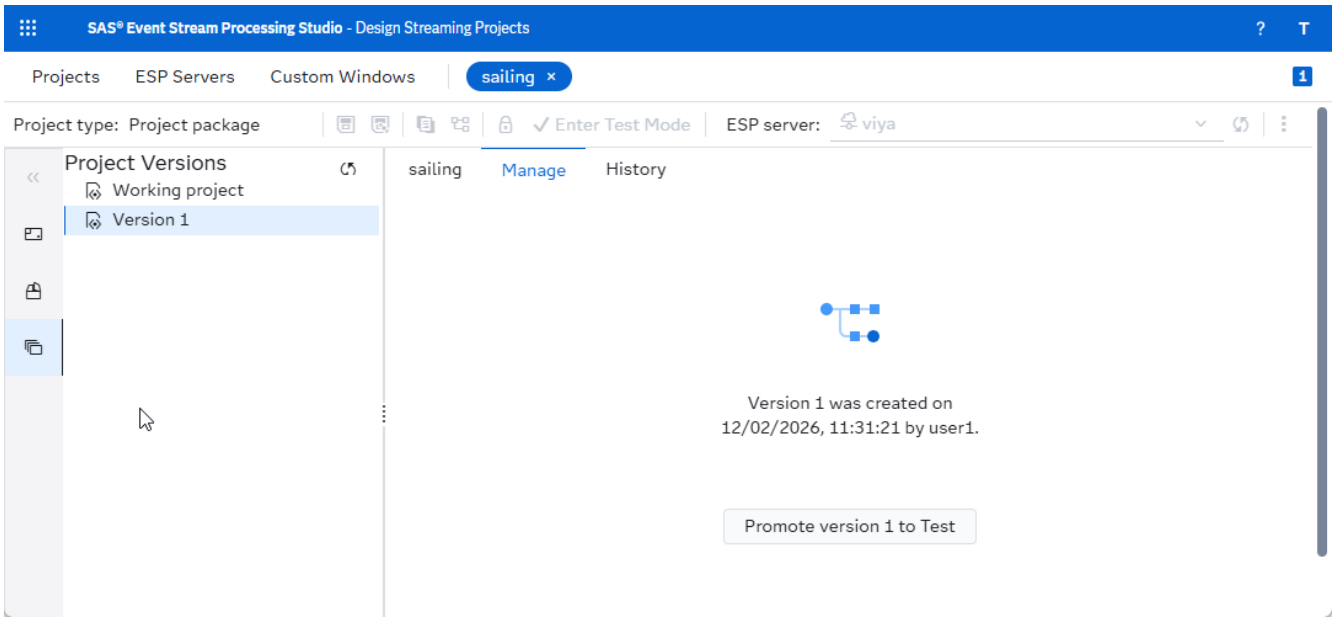
Manage Project Versions

This topic is relevant when project versions are published using file-based versioning. For an introduction to project versions, see "Overview of File-Based Versioning".

User Interface Controls

To view project versions, open a project and click  on the left toolbar. The following figure shows an example of information about project versions.

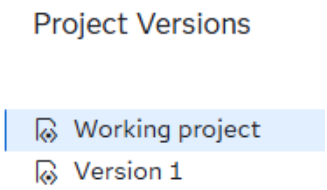
Figure 30 *Example of Viewing Project Versions*



Use the **Project Versions** pane to select the project version that you are interested in:

- The *working project* appears at the top of the pane.

  Figure 31 *Working Project*

  This is the project version whose design you can modify when you click on the left toolbar.

- Other project versions are also shown in the pane. They cannot be modified after they have been created. However, you can revert the working project to one of the other versions so that the specified version becomes the new working project.

Project versions are created only when you use the versioning functionality that is described in this topic. When you work on the design of your project (for example, by adding windows and configuring connectors), those changes are saved to the XML code of the working project. A project version is not created for those changes.

The right pane displays details for the project version that is selected in the **Project Versions** pane:

- Use the **Manage** tab to create a project version, to promote a project version to a different promotion level, or to revert the working project to a specific project version. The actions that you take on this tab can result in new versions appearing in the **Project Versions** pane.

> **TIP** The **Manage** tab also displays messages about the status of the selected project version. Here is an example: `Version 1 was promoted to Test on <time> by <user>.` You can use this information to learn which promotion level a project version has been promoted to.

- Use the **History** tab to view the version history for the project.

To test a specific project version, promote the project version to Test and use SAS Event Stream Manager to deploy that project version to a deployment that you are using for testing purposes, as described in the following example.

Example of Promoting a Project Version from Development to Test

The following steps use the Sailing example to demonstrate a basic scenario for creating and promoting project versions. You learn the following tasks:

- Saving a project version.
- Making a change to the working project.
- Reverting a change.
- Promoting a project version from Development to Test.
- In SAS Event Stream Manager: testing the project version, promoting it from Test to Production, and deploying it to a production deployment.

> **Note:** This example assumes that the SAS Event Stream Processing environment contains a single SAS Event Stream Manager instance. When there are multiple SAS Event Stream Manager instances, after you promote the project to each promotion level you also need to import the project to the relevant SAS Event Stream Manager instance. Only the SAS Event Stream Manager instance that has access to the relevant promotion level can import the project. For more information, see "Deploying a Project in a Separated System" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Complete the following steps:

1 Install the Sailing example:

   a In SAS Event Stream Processing Studio, on the **Projects** page, click **View example projects**.

   b In the SAS Event Stream Processing Examples window, find the Sailing example and click **Install example**.

   SAS Event Stream Processing Studio Modeler appears and the project diagram is displayed.

> **TIP** For more information about the project diagram for the Sailing example and how events flow through the windows, see the README in https://github.com/sassoftware/esp-studio-examples/tree/main/EndtoEndExamples/sailing.

2 Click on the left toolbar to display the **Project Versions** pane.

3 Suppose that you want to save the current state of the working project as a new project version.

   a Select **Working project**.

   b On the **Manage** tab, click **New version**.

   c In the New Version window, enter the tag `priority` and the comment `First version of my project`, and then click **OK**.

   The new version appears in the **Project Versions** pane as **Version 1**. Hover over this new version in the pane and observe that the tag and the comment that you entered are displayed.

4 Suppose that you now want to make a change to the working project:

   a Click on the left toolbar.

   b Add a new window to the workspace and click .

5 Suppose that you change your mind and decide that the previous project version, without the extra window, is more suitable for your purposes. To revert to the previous version:

   a Click on the left toolbar.

   b Select **Working project**.

   c On the **Manage** tab, click **Revert to last version**.

   d In the Revert window, click **Revert**.

   e Click on the left toolbar. Observe that the window that you added is not present. The reversion has been successful.

   f Click .

   g Click to return to project versioning.

6 Suppose that you decide to promote Version 1 to Test.

   a Select **Version 1**.

   b On the **Manage** tab, click **Promote version 1 to Test**.

   c In the Promote Version window, click **OK**.

   The **Manage** tab displays the following message: `Version 1 was promoted to Test on <time> by <user>.`

7 You can now use SAS Event Stream Manager to test Version 1 by deploying it to a deployment that you use for testing purposes, and later to promote Version 1 from Test to Production. Follow the steps in "Example of Promoting a Project Version from Test to Production" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Convert a Project to Use File-Based Versioning

File-based versioning was made available beginning in release 2024.08. For more information, see "Overview of File-Based Versioning".

This topic explains how to manually convert projects that were previously published using SAS Viya platform services, one project at a time. However, administrators can perform a bulk conversion of projects. For more information, see "Convert Projects to File-Based Versioning in Bulk".

If the project did not previously have a project package, a project package is created as part of the conversion.

IMPORTANT You cannot undo the conversion.

To convert a project that has been published using SAS Viya platform services to use file-based versioning:

1 Open the project in SAS Event Stream Processing Studio.

A message is displayed, informing you that the project has been published using SAS Viya platform services and that you can use file-based versioning instead.

Note: If the message does not appear, the project has not been previously published.

2 Click **Convert project**.

3 In the Use File-Based Versioning window, click **Yes** to convert the project.

4 Click on the left toolbar and observe that a project package is present. When your project contains references to files (for example, files that contain test data), consider adding those files to the project package and adjusting the references to point to the new location of the files. For more information, see "Manage a Project Package".

5 Click on the left toolbar and observe that the **Project Versions** pane displays the project's versions. For more information, see "Manage Project Versions".

Publishing Project Versions by Using SAS Viya Platform Services

Overview of Publishing Project Versions by Using SAS Viya Platform Services

For information about circumstances in which it is possible to publish project versions by using SAS Viya platform services, see "Overview to Versioning".

When SAS Viya platform services are used to publish project versions, the Files service and the File Store service are used to store project versions in a database.

You cannot edit project versions after they have been published. A project that you can edit is designated as the working copy of the project. It does not become a project version until you publish it. The working copy of the project enables you to make changes to the project without affecting any project versions that you previously published.

When you publish a project version, the version's XML code is updated to display its unique ID number. Project versions that are published for the first time are assigned a version number of 1.1. The number to the left of the decimal point is the project's major version number. The number to the right of the decimal point is the project's minor version number. Publishing subsequent versions of a project increments the major version number. In SAS Event Stream Processing Studio, you can publish major project versions, but you cannot publish minor project versions. Minor versions are created when you make a change to the project version in SAS Event Stream Manager (for example, when you use SAS Event Stream Manager to accept an update to a SAS Micro Analytic Service module that your project references).

You can view a project's major versions and minor versions in the version hierarchy on the **Versioning** page.

Publish a Version of a Project

1 On the **Projects** page, right-click the relevant project and select **Open Project**.

SAS Event Stream Processing Studio Modeler appears.

Note: To create a new version of a project, the version must exist in a valid state.

2 Click .

The **Versioning** page appears. This page contains a version hierarchy that displays the versions of the project that you are working on.

3 Click .

The Publish — Version window appears.

a In the **Version notes** field, enter any notes that relate to the version of the project that you want to publish. This enables you to maintain a record of a project's version history.

Note: You cannot modify the version notes of a project version that has already been published.

b Click **OK**.

The **Versioning** page displays your published project version in the version hierarchy.

4 To view information relating to the project version that you published, select the relevant version in the version hierarchy on the left.

When you publish a project version, the version's XML code is updated to display its unique ID number. The project's ID number, major version number, and minor version number are specified in the version's XML code as metadata. For more information, see "Overview of Working with Projects".

View a Published Version of a Project

1 On the **Projects** page, right-click the relevant project and select **Open Project**.

SAS Event Stream Processing Studio Modeler appears.

2 Click .

The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.

3 In the version hierarchy on the left, select the version of the project that you want to view in SAS Event Stream Processing Studio Modeler.

4 Click .

The published version is displayed in Read-Only mode.

Note: You cannot make changes to a version of a project that has already been published. If you want to make more changes to a project, a new working copy is made available for you to edit in SAS Event Stream Processing Studio.

Revert to a Previous Version

1 On the **Projects** page, right-click the relevant project and select **Open Project**.

SAS Event Stream Processing Studio Modeler appears.

2 Click .

The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.

3 In the version hierarchy on the left, select the version of the project that you want to revert to.

4 Click .

The Revert to Version window appears.

5 Click **Yes** to confirm the reversion.

The working version of the project reverts to the published version that you selected in the version hierarchy. You cannot undo this operation.

Export a Previously Published Version

1 On the **Projects** page, right-click the relevant project and select **Open Project**.

SAS Event Stream Processing Studio Modeler appears.

2 Click .

The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.

3 In the version hierarchy on the left, select the version of the project that you want to export.

4 Click .

The project version exports to your computer. The location of the project version that you exported might vary depending on your browser's configuration.

Quarantined Project Versions

The Files service for the SAS Viya platform can be configured to scan files for malicious content and absolute URLs that are forbidden by policy. If such a configuration is in place, then when you publish a project version in SAS Event Stream Processing Studio, the project version is scanned for malicious content and disallowed URLs. When suspicious content is detected, a message on the **Versioning** page in SAS Event Stream Processing Studio informs you that the project version is quarantined. Quarantined project versions are not made available in SAS Event Stream Manager.

Only an administrator can revert or export a quarantined project version in SAS Event Stream Processing Studio. For more information about how administrators use the Files service to manage quarantined files and about the properties that are used to enable scanning, see "Files Service" in *SAS Viya Platform: Configuration Properties*.

Examples

Install Example Projects

The SAS Event Stream Processing Examples window enables you to install some examples from the SAS Event Stream Processing Examples GitHub.

To install an example:

1 On the **Projects** page, click View example projects.

2 Select an example in the left pane.

For detailed information about how the selected example works, click **View README in GitHub** in the right pane. When you have installed an example, you can also access its README file from the **Project Package** pane.

3 (Optional) Select the **Use Large File Storage (LFS)** check box if you want to use Git LFS for storing large files that are associated with this project. For more information, see "File-Based Versioning and Git Large File Storage".

Note: Git LFS can be used when a project package is stored in one of the following ways:

- in the Git repository that is internal to SAS Event Stream Processing
- in a remote repository in Gitea

4 (Optional) You can select the **Create a remote repository** check box to store the example project in a remote Git repository. Use the **Remote server connection** drop-down list to select the remote Git server on which you want to create the repository. For more information, see "Overview of File-Based Versioning".

For more information, see Using the Examples in the GitHub's main README file.

Processing Trades

Overview

This example processes stock market trades. The model identifies large securities transactions and the traders who were involved in those trades.

The following instructions teach you how to use SAS Event Stream Processing Studio to create a project from scratch and test the project.

> **TIP** Alternatively, you can install and test a ready-made version of the project. Follow the steps in "Install Example Projects" to install the Trades example. Then follow the steps in "Run the Model in Test Mode".

The example uses files that are available in https://github.com/sassoftware/esp-studio-examples/tree/main/EndtoEndExamples/trades. Obtain the following files from GitHub and save them on your computer:

- **`trades.csv`** is an input file. This file contains events relating to securities transactions.
- **`traders.csv`** is an input file. This file contains events relating to the traders involved in the securities transactions.

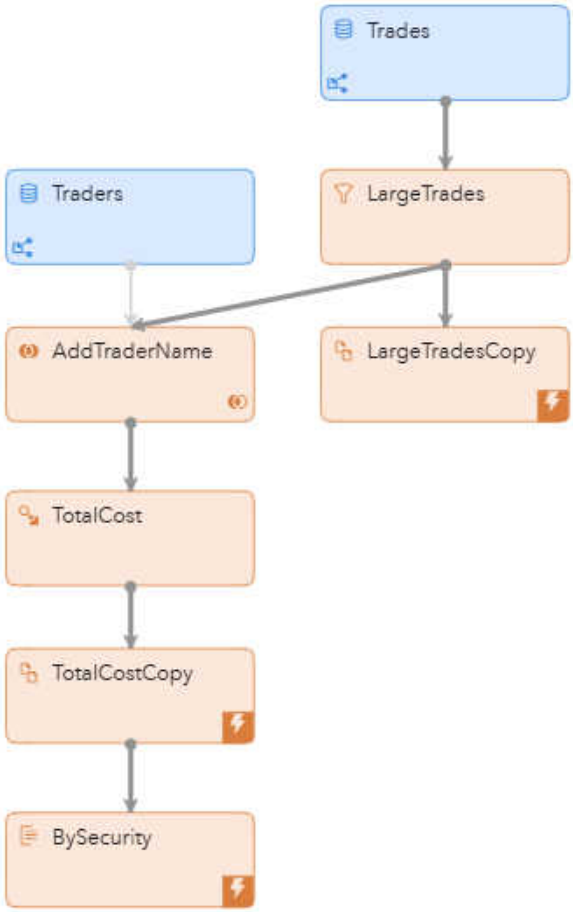
There are several ways to feed input data into a project. The instructions explain how to configure file and socket connectors to point to input files, and how to upload those input files from your computer to a project package.

The instructions assume that you are using a Kubernetes environment. That is, you are not testing the project on an ESP server that runs on an edge server.

Project Details

The following figure shows the diagram of the project:

Figure 32 *Diagram of the Processing Trades Project*



The project contains eight windows:

- Trades is a Source window. This is where the securities from the **`trades.csv`** file enter the model.
- Traders is a Source window. This is where the trader names from the **`traders.csv`** file enter the model.
- LargeTrades is a Filter window. This window filters out all trades not in the specified range.
- LargeTradesCopy is a Copy window. This window retains a list of all large trades.
- AddTraderName is a Join window. This window combines the large trades with their corresponding trader names.
- TotalCost is a Compute window. This window shows the total cost of each transaction. You can use this information to identify high-value transactions.
- TotalCostCopy is a Copy window. This window retains a list of the total costs of each transaction in accordance with the retention policy.
- BySecurity is an Aggregate window. This window shows all the inserts, deletes, and update blocks for the large trades.

Example Steps

Create a Project

1 On the **Projects** page, click  .

The New Project window appears.

2 In the New Project window, do the following:

a In the **Name** field, enter `trades_lua`.

b In the **Description** field, enter a description: `This model can be used to identify large securities transactions and the traders who were involved in those trades.`

c Select the **Create a project package** check box if it is not already selected. Later in this example, you upload input files to the project package. For more information about the purpose of a project package, see "Project Package".

d Click **OK**.

Configure the Project's Threading Level and Continuous Query

1 The right pane displays project-level properties. In the right pane, expand **Attributes**.

2 In the **Threads** field, change the thread pool size to `4`.

This means that four threads are used from the available thread pool. Using a pool of threads in a project enables more efficient processing.

3 Click  to view the settings for the project's continuous query.

4 In the right pane, in the **Name** field, change the continuous query's default name `cq1` to `trades_cq`.

Configure a Source Window to Receive Input Events

When the project was created, a Source window was automatically added to the workspace.

To configure the Source window to receive events about securities transactions:

1 Change the name of the Source window. In the right pane, in the **Name** field, change the default name to `Trades`.

2 Specify an output schema for the Trades window:

a In the right pane, click  .

b Click  .

The Output Schema window appears.

c The schema table already contains one row, for a key field. Click to add more rows to the schema table.

Enter the values shown in the following table in the rows. That is, adjust the values in the row that was present by default and add values to the new rows that you added.

Key	Field Name	Type
Y	tradeID	string
N	security	string
N	quantity	int32
N	price	double
N	traderID	int64
N	time	stamp

d Click **OK**.

e In the right pane, click .

3 Configure the Trades window to stream events from the `trades.csv` file, which contains securities transactions:

a Expand **Input Data (Publisher) Connectors**.

b Click .

The Connector Configuration window appears.

c In the **Name** field, enter `input_trades`.

d Observe that, by default, the connector type is set to the file and socket connector. This example uses CSV files to feed data into the example, so this connector type is suitable.

e In the **Fsname** field, click .

The Select File window appears.

f Click to upload the `trades.csv` file.

The file is uploaded to the `test_files` folder in the project package.

g Click **OK**.

h In the **Fstype** drop-down list, select **csv**.

i Configure the input_trades connector's properties:

1 Click **All properties**.

The All Properties window appears.

2 Enter `%d/%b/%Y:%H:%M:%S` in the **dateformat** property's **Value** field.

3 Enter `100` in the **rate** property's **Value** field.

4 Click **OK**.

j Click **OK**.

4 Specify a state and event type for the Trades window:

a Expand **State and Event Type**.

b In the **Window state and index** drop-down list, select **Stateless (pi_EMPTY)**.

A window that uses a primary index type pi_EMPTY is non-stateful or stateless. It acts as a pass-through for all incoming events. This index does not store events.

c Select the **Accept only "Insert" events** check box.

This option prevents the window from passing non-insert events to other windows.

Create Another Source Window to Receive Input Events

1 Expand **Input Streams** on the **Windows** pane on the left and drag another Source window to the workspace.

The right pane displays the Source window's properties.

This window receives information about stock market traders.

2 Specify a name for the Source window. In the right pane, in the **Name** field, change the default name to `Traders`.

3 Specify an output schema for the Traders window:

a In the right pane, click  .

b Click  .

The Output Schema window appears.

c Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Key	Name	Type
Y	ID	int64
N	name	string

d Click **OK**.

e In the right pane, click  .

4 Configure the Traders window to receive information from the **traders.csv** file, which contains details of stock market traders:

   a Expand **Input Data (Publisher) Connectors**.

   b Click  .

   The Connector Configuration window appears.

   c In the **Name** field, enter **input_traders**.

   d In the **Fsname** field, click  .

   The Select File window appears.

   e Click  to upload the **traders.csv** file.

   The file is uploaded to the **test_files** folder in the project package.

   f Click **OK**.

   g In the **Fstype** drop-down list, select **csv**.

   h Click **OK**.

5 Specify a state and event type for the Traders window:

   a Expand **State and Event Type**.

   b In the **Window state and index** drop-down list, select **Stateless (pi_EMPTY)**.

Add Connector Orchestration

Add connector orchestration to ensure that events from the Traders window enter the project before events from the Trades window. For more information, see "Orchestrating Connectors" in *SAS Event Stream Processing: Connectors and Adapters*.

1 Click  to view project-level properties rather than properties of a specific window.

2 In the right pane, expand **Connector Orchestration**.

3 Create a connector group for the Trades window:

   a In the **Connector groups** table, click .

   b In the Connector Group window, change the default name to **Trades**.

   c Click  .

   d In the Connector column, select **trades_cq/Trades/input_trades**.

   e In the Target state column, check that the target state is **Finished**.

   f Click **OK**.

4 Create a connector group for the Traders window:

   a In the **Connector groups** table, click  .

b In the Connector Group window, change the default name to **`Traders`**.

c Click  .

d In the Connector column, select **trades_cq/Traders/input_traders**.

e In the Target state column, check that the target state is **Finished**.

f Click **OK**.

5 Set the controlling group:

a In the **Dependency rules** table, click  .

b In the Controlling Group column, select **Traders**.

c In the Dependent Group column, select **Trades**.

Events from the Trades window now do not enter the project until the connector in the Traders window has the status **Finished**.

Create a Filter Window to Filter Large Trades

1 Expand **Transformations** on the **Windows** pane on the left and drag a Filter window to the workspace.

Configure this window to identify large trades. In this example, a trade is regarded as a large trade if the quantity of stock traded equals or exceeds 100.

2 In the right pane, specify a name for the Filter window. In the **Name** field, change the default name to **`LargeTrades`**.

3 Connect the Trades window to the LargeTrades window with an edge:

a Position the cursor over the anchor point at the bottom of the Trades window so that the anchor point color changes to white.

b Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the LargeTrades window.

The LargeTrades window now accepts trades from the Trades window.

4 Click the LargeTrades window on the workspace.

5 In the right pane, expand **Filter**.

6 Set the filter type to Lua.

7 In the **Fields to use in Lua code** field, in the drop-down list that is located below the **Include fields** radio button, select the **quantity** field.

8 Observe that the **Filter function** field is set to **`filter`**. This field identifies the function in the Lua code acts that acts as the filter function.

9 In the Lua code area, replace the default code with the following code:

```
function filter(data)
        return data.quantity >= 100
    end
```

10 Click to validate the Lua code.

A message appears, informing you of the code's validity.

11 Configure the LargeTrades window's state:

a Expand **State**.

b In the **Window state and index** field, select **Stateless (pi_EMPTY)** from the drop-down list.

Create a Join Window to Combine Large Trades with Their Corresponding Trader

1 Expand **Transformations** on the **Windows** pane on the left and drag a Join window to the workspace.

Configure this window to match large trades with the traders who made those trades.

2 Specify a name and description for the Join window. In the right pane, in the **Name** field, change the default name to `AddTraderName`.

3 Connect the LargeTrades window to the AddTraderName window with an edge.

The AddTraderName window now accepts trades from the LargeTrades window.

4 Connect the Traders window to the AddTraderName window with an edge.

The AddTraderName window now accepts trader names from the Traders window.

5 Click the AddTraderName window on the workspace.

The right pane displays the AddTraderName window's properties.

6 Observe the AddTraderName window's configuration settings:

a In the right pane, expand **Settings** and observe that the LargeTrades window is regarded as the left window and the Traders window is regarded as the right window. This is due to the order in which you added the edges.

b In the **Output field calculation method** field, observe that **Select fields** is selected from the drop-down list.

Selecting the **Select fields** option ensures that, as new input events arrive, join non-key fields are calculated using a join selection string. This selection string is a one-to-one mapping of input fields to join fields.

c Collapse **Settings**.

7 Configure the AddTraderName window's join conditions:

a Expand **Join Conditions**.

b In the **Join Conditions** section, click to add a row to the table.

c Click the cell in the Left: LargeTrades column, and select **traderID**.

d Click the cell in the Right: Traders column, and select **ID**.

e Collapse **Join Conditions**.

8 Observe the AddTraderName window's join criteria:

   a Expand **Join Criteria** if it is not already expanded.

   b Observe that the **Join Type** drop-down list has a default value of **Left outer**.

   c Select the **Set the "no-regenerates" option** check box.

   For more information about this option, see "Using Regeneration versus No Regeneration" in *SAS Event Stream Processing: Using Source and Derived Windows*.

9 Configure the AddTraderName window's state:

   a Expand **State**.

   b In the **Window state and index** field, select **Stateless (pi_EMPTY)** from the drop-down list.

10 Configure the output rules for the AddTraderName window:

   a Expand **Output Rules**.

   b Clear the **Collapse redundant updates** check box. Clearing the check box means that multiple update blocks are not collapsed into a single update block.

11 Specify a schema for the AddTraderName window:

   a In the right pane, click  .

   b Click .

   The Edit Output Schema – Non-Key Fields window appears. Use this window to configure the fields as shown in the following table. The schema fields that are required have already been defined previously. Click  to open the Copy Fields from Input Schema window. Select the following schema fields.

Window	Field	Type
LargeTrades	security	string
LargeTrades	quantity	int32
LargeTrades	price	double
LargeTrades	traderID	int64
LargeTrades	time	stamp
Traders	name	string

   c Click **OK**.

   You are returned to the Edit Output Schema – Non-Key Fields window.

   d Click **OK**.

12 In the right pane, click .

Create a Copy Window to Retain the Large Trades

1 Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace.

The right pane displays the Copy window's properties.

2 Specify a name and description for the Copy window. In the right pane, in the **Name** field, change the default name to `LargeTradesCopy`.

3 Configure the LargeTradesCopy window's retention properties:

a Expand **Retention**.

b Observe that the **Type** field is set to **By time, sliding**. This is the default value.

c In the **Time limit** field, enter `5` and select **Minutes** from the drop-down list.

4 Configure the output rules for the LargeTradesCopy window:

a Expand **Output Rules**.

b Clear the **Collapse redundant updates** check box. Clearing the check box means that multiple update blocks are not collapsed into a single update block.

5 Connect the LargeTrades window to the LargeTradesCopy window with an edge.

The LargeTradesCopy window now accepts trades from the LargeTrades window.

Create a Compute Window to Compute the Total Cost of Each Transaction

1 Expand **Transformations** on the **Windows** pane on the left and drag a Compute window to the workspace.

The right pane displays the Compute window's properties.

Configure this window to compute the total cost of the large trades.

2 Specify a name for the Compute window. In the **Name** field, change the default name to `TotalCost`.

3 Connect the AddTraderName window to the TotalCost window with an edge.

The TotalCost window now accepts trades from the AddTraderName window.

4 Click the TotalCost window to display the window's properties in the right pane again.

5 Configure the TotalCost window's state:

a In the right pane, expand **State**.

b In the **Window state and index** field, select **Stateless (pi_EMPTY)** from the drop-down list.

6 Specify an output schema for the TotalCost window:

a In the right pane, click .

b Click .

The Output Schema window appears.

c Click to add a row to the schema table.

Enter the following values:

Note: Alternatively, you can copy the schema fields that you have previously defined. Click to open the Copy Fields from Input Schema window. Select the schema fields that you want to copy and click **OK**.

If you copied schema fields you previously created, you must still manually enter all new fields and their values.

Key	Field Name	Type	Expression
Y	tradeID	string	(not applicable)
N	security	string	security
N	quantity	int32	quantity
N	price	double	price
N	totalCost	double	price*quantity
N	traderID	int64	traderID
N	time	stamp	time
N	name	string	name

d Click **OK**.

7 In the right pane, click .

Create a Copy Window to Retain the Computed Totals

1 Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace.

The right pane displays the Copy window's properties.

2 Specify a name and description for the Copy window. In the right pane, in the **Name** field, change the default name to `TotalCostCopy`.

3 Configure the TotalCostCopy window's retention properties:

a Expand **Retention**.

b Observe that the **Type** field is set to **By time, sliding**.

c In the **Time limit** field, enter `1` and select **Days** from the drop-down list.

4 Configure the output rules for the TotalCostCopy window:

a Expand **Output Rules**.

b Clear the **Collapse redundant updates** check box. Clearing the check box means that multiple update blocks are not collapsed into a single update block.

5 Connect the TotalCost window to the TotalCostCopy window with an edge.

The TotalCostCopy window now accepts trades from the TotalCost window.

Create an Aggregate Window to Aggregate the Large Trades and Their Total Costs

1 Expand **Transformations** on the **Windows** pane on the left and drag an Aggregate window to the workspace.

The right pane displays the Aggregate window's properties.

Configure this window to compute the total cost of the large trades.

2 Specify a name for the Aggregate window. In the **Name** field, change the default name to **`BySecurity`**.

3 Configure the output rules for the BySecurity window:

a Expand **Output Rules**.

b Clear the **Collapse redundant updates** check box. Clearing the check box means that multiple update blocks are not collapsed into a single update block.

4 Connect the TotalCostCopy window to the BySecurity window with an edge.

The BySecurity window now accepts trades from the TotalCostCopy window.

5 Click the BySecurity window to display the Aggregate window's properties in the right pane again.

6 Specify an output schema for the BySecurity window:

a In the right pane, click  .

b Click  .

The Output Schema window appears.

c Click to add a row to the schema table. Enter the following values:

Key	Field Name	Type	Aggregate function	Parameters
Y	security	string	(not applicable)	(not applicable)
N	quantityTotal	double	ESP_aSum	quantity
N	costTotal	double	ESP_aSum	totalCost

d Click **OK**.

Run the Model in Test Mode

1 The model is now complete. Click to save your model.

2 Click ✓ Enter Test Mode.

A new page called Test: trades_proj_lua appears.

3 Click ▷ Run Test.

> **TIP** If you want to adjust the settings that are used to create an ESP server in the Kubernetes cluster, instead of clicking ▷ Run Test, click ▾ next to ▷ Run Test, and then click Configure and Run Test. The Load and Start Project in Cluster window appears. Adjust the settings if required, and then click **OK**. For more information about the settings that are available in this window, see "Load and Start a Project on an ESP Server That Is in a Kubernetes Cluster".

The results for each window appear on separate tabs:

- The **Trades** tab lists the securities transactions
- The **Traders** tab lists the traders
- The **LargeTrades** tab lists the large trades
- The **AddTraderName** tab lists the large trades and includes an additional column that shows trader names
- The **LargeTradesCopy** tab lists the retained large trades in accordance with the retention policy.
- The **TotalCost** tab includes an additional column that shows the total cost of each transaction. You can use this information to identify high-value transactions.
- The **TotalCostCopy** tab lists the retained total cost of each transaction in accordance with the retention policy.
- The **BySecurity** tab shows all the inserts, deletes, and update blocks for the large trades. The newest event is shown at the top of the table.

4 To stop the test, click ■ Stop.

Importing Models Created in SAS Model Manager into SAS Event Stream Processing Studio

Overview of SAS Model Manager Integration

To import a model from SAS Model Manager into SAS Event Stream Processing Studio, choose one of the following ways:

- "Use the Models Pane to Import a Model from SAS Model Manager"
- "Import a SAS Model Manager ZIP file"
- "Publish a Model from SAS Model Manager into SAS Event Stream Processing"

Use the Models Pane to Import a Model from SAS Model Manager

This way of importing a model is available for DS2 and Python models.

You can use the **Models** pane in SAS Event Stream Processing Studio to import a model from SAS Model Manager. When the project is deployed, the model is retrieved from the SAS Model Manager common model repository and written to the Kubernetes pod that runs the ESP server.

- When you import a DS2 model, a SAS Micro Analytic Service module is created to accommodate the model. A Source window and a Calculate window are created. The SAS Micro Analytic Service module is referenced from a Calculate window's input handler. You can then use SAS Event Stream Processing Studio to check for new champion models and accept them. For more information, see "Update SAS Micro Analytic Service Modules".
- When you import a Python model, a Source window and a Python window are created. The model's score code is copied to the Python window.

  Note: When you import a Python model, SAS Event Stream Processing Studio cannot check for new champion models. When you identify a new champion model in SAS Model Manager, you can repeat the process of importing a model to SAS Event Stream Processing Studio.

IMPORTANT This way of importing models from SAS Model Manager requires the use of overlays when SAS Event Stream Processing is deployed with other SAS products. These overlays are used to configure SAS Event Stream Processing to access persistent volumes and analytic stores. For more information, see "Managing Persistent Volumes" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

The model that you want to import from SAS Model Manager must meet the following criteria:

- The model exists in a SAS Model Manager project.
- The model is a champion of the project.
- The model has a score code type of DATA step, DS2 package, DS2 multi-type, or Python that meets the following criteria:
  - The DATA step, DS package, or Python model contains a score file with a file role of score code.
  - The DS2 multi-type file contains analytic store content with a DS2 score code file named `dmcas_packagescorecode.sas`.
- In the case of DS2 models, the model contains a single DS2 code file (DS2 package) or a DS2 code file with one or more analytic store files.

To use the **Models** pane to import a model:

1. In SAS Event Stream Processing Studio, open the project into which you want to import the model. The project must be part of a project package. For more information, see "Create a Project Package".
2. Click on the left toolbar. If SAS Model Manager is not available or the project is not part of the project package, then this button is not displayed on the toolbar.
3. In the **Models** pane, use the search and filtering functionality to find the model that you want to import. For example, click to filter the models by repository and then limit the search further by entering the name of a project in SAS Model Manager. You can also double-click a model to view more information about it.
4. Drag the desired model from the **Models** pane to the workspace.
5. When the model contains a `requirements.txt` file and the project's virtual environment does not meet the requirements in the file, a warning is displayed on the toolbar. Click **Resolve** and use the Virtual Environment Files Detected window to select a suitable virtual environment. For more information, see "Select a Matching Environment When Virtual Environment Files Have Been Detected".
6. When the model is a DS2 model, a Source window and a Calculate window appear on the workspace. The Calculate window references a SAS Micro Analytic Service module that has been created to accommodate the model. To view the details of the module, click , expand **SAS Micro Analytic Service modules**, and double-click the name of the SAS Model Manager project.
7. When the model is a Python model, a Source window and a Python window appear on the workspace. The Python window contains the model's score code. To view the code, select the Python window, expand **Python Settings** in the right pane, and view the Python code and related settings.

Note: The score code of the imported model might require additional configuration, such as ensuring that the return values of the entry point function are dictionaries with the expected output variable names as keys.

Other model files are added to the project package. To view them, click on the left pane and expand the **analytics** folder on the **Project Package** pane. For more information, see "Using the Folders That Are Included in a Project Package".

8 Configure the rest of the project as required and click .

Import a SAS Model Manager ZIP file

This way of importing a model is available for DS2 models.

An ESP project can reference a model that you have exported from SAS Model Manager as a ZIP file and imported into a project package in SAS Event Stream Processing Studio. When the project is deployed, the model is written to the Kubernetes pod that runs the ESP server. SAS Micro Analytic Service modules are used to accommodate such models. The module is referenced from a Calculate window's input handler.

Note: When you use this way of importing a model, SAS Event Stream Processing Studio cannot check for new champion models. When you identify a new champion model in SAS Model Manager, you can repeat the process of importing a model to SAS Event Stream Processing Studio.

The model that you want to import from SAS Model Manager must meet the following criteria:

- The model exists in a SAS Model Manager project.
- The model is a champion of the project.
- The model has a score code type of DATA step, DS2 package, or DS2 multi-type that meets the following criteria:
  - The DATA step or DS2 package model contains a score file with a file role of score code.
  - The DS2 multi-type file contains analytic store content with a DS2 score code file named **dmcas_packagescorecode.sas**.
- In the case of DS2 models, the model contains a single DS2 code file (DS2 package) or a DS2 code file with one or more analytic store files.

To import a SAS Model Manager ZIP file:

1 If the model's score code type is DATA step, use SAS Model Manager to publish the model to a SAS Micro Analytic Service destination. Publishing the model converts DATA step code to DS2 code. For more information, see "Publish a Project Champion Model" in *SAS Model Manager: User's Guide*.

The following files are added to the model: **ScoreCodeGen-ds2Package_code.sas**, **ScoreCodeGen-ds2Package.log**, and **ScoreCodeGen-ds2Package.sas**.

2 Export your model from SAS Model Manager as a ZIP file. For more information, see "Export a Model" in *SAS Model Manager: User's Guide*.

3 In SAS Event Stream Processing Studio, open the project into which you want to import the model. The project must be part of a project package. For more information, see "Create a Project Package".

4 In the right pane, expand **SAS Micro Analytic Service Modules** and click .

> **TIP** Alternatively, you can access the Import from SAS Model Manager ZIP File window from the **Project Package** pane. Click ⋮ on the pane's toolbar and select **Import SAS Micro Analytic Service modules**.

5 In the Import from SAS Model Manager ZIP File window, complete the following steps:

a Select the ZIP file that you exported from SAS Model Manager.

b The Name field shows a default name for your SAS Micro Analytic Service module. You can edit this name.

c By default, the **Add a Source window** and the **Add a Calculate window** check boxes are selected. When these check boxes are selected, the output schemas of these windows are created on your behalf and an input handler is added to the Calculate window. If you do not want these windows to be added, clear the check boxes.

d Click **OK**.

6 Your module appears in the **SAS Micro Analytic Service Modules** section in the right pane.

a Double-click the module to view its details in the SAS Micro Analytic Service Module window. Your code file is referenced in the **External file** field and any analytic store files appear in the Module Members table. If you converted DATA step code to DS2 code, the **External file** field references a **`ScoreCodeGen-ds2Package_code.esp.sas`** file. This file is discussed in more detail later in this topic.

b Click **OK** to close the SAS Micro Analytic Service Module window.

7 If you kept the default options to add Source and Calculate windows, these windows are present in the workspace and are connected to each other with an edge.

Select a window and click to view its output schema. The schemas are populated with the fields that were specified in the **`inputVars.json`** and **`outputVars.json`** files in the ZIP file that you exported from SAS Model Manager. The variables that were specified in the ZIP file are converted to types that are compatible with SAS Event Stream Processing. For example, the `decimal` type is converted to the `double` type. Similarly, `boolean` and `integer` types are converted to `int32` types.

In the Source window, a key field called `id` is added automatically. The presence of the `id` field means that the project is ready to be run. If required, you can adjust the `id` field. You might need to adjust the schemas further for the SAS Micro Analytic Service module to function with other windows in your project. You might also need to add windows between the Source window and the Calculate window.

8 Click on the left toolbar.

9 In the **Project Package** pane, observe that your module is stored in the **/home/analytics/mas-modules/*moduleName*/1.0** folder. The module is always stored in this location, even when the `store-location` ESP server configuration property specifies another location.

   If you converted a DATA step code to DS2 code, the project package contains a **ScoreCodeGen-ds2Package_code.esp.sas** file. This file is needed because the DS2 code that was generated by SAS Model Manager might contain a placeholder for the package name: `package &SCOREPACK/overwrite=yes;`. This placeholder is intended but is not compatible with the ESP server run time. A modified copy of the DS2 score code is stored in the **ScoreCodeGen-ds2Package_code.esp.sas** file.

10 Click  .

Publish a Model from SAS Model Manager into SAS Event Stream Processing

You can publish a model from SAS Model Manager into SAS Event Stream Processing.

A project package that contains a Source window and a Calculate window is created in SAS Event Stream Processing Studio. A SAS Micro Analytic Service module is created to accommodate the model, and this module is referenced from the Calculate window's input handler.

In addition, a project container is created in SAS Event Stream Manager. For more information, see "Working with Project Containers" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Note: When you publish a model in this way, SAS Event Stream Processing Studio or SAS Event Stream Manager cannot check for new champion models. When you identify a new champion model in SAS Model Manager, you can repeat the process of publishing a model.

IMPORTANT This way of importing models from SAS Model Manager requires the use of overlays when SAS Event Stream Processing is deployed with other SAS products. These overlays are used to configure SAS Event Stream Processing to access persistent volumes and analytic stores. For more information, see "Managing Persistent Volumes" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

The following prerequisites must be met:

- The model that you want to publish from SAS Model Manager must meet the following criteria:
  - The model exists in a SAS Model Manager project.
  - The model has a score code type of DATA step, DS2 package, or DS2 multi-type that meets the following criteria:
    - The DATA step or DS2 package model contains a score file with a file role of score code.
    - The DS2 multi-type file contains analytic store content with a DS2 score code file named **dmcas_packagescorecode.sas**.

- The model contains a single DS2 code file (DS2 package) or a DS2 code file with one or more analytic store files.

- You must have access to SAS Event Stream Manager. For more information, see "Access SAS Event Stream Manager" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.
- SAS Event Stream Manager must contain a connection to a container registry in which a project container can be created. For more information, see "Add Container Registries" in *SAS Event Stream Processing: Using SAS Event Stream Manager*. The name of the container registry must match the name that is configured as part of an Event Stream Processing publishing destination. For more information, see "Configure Event Stream Processing Publishing Destinations" in *SAS Viya Platform: Publishing Destinations*. The configured Event Stream Processing publishing destination can be viewed in the Publish window within SAS Model Manager. For example, if the container registry name that is used in the configured publishing destination is `Model Publishing`, SAS Event Stream Manager must contain a connection to a container registry that has this name.

To publish a model:

1. Publish your model from SAS Model Manager. Ensure that you set the destination to **Event Stream Processing**. For more information, see "Publish a Project Champion Model" in *SAS Model Manager: User's Guide*.
2. When the model's status in SAS Model Manager changes to `Published successfully`, the model appears on the **Projects** page in SAS Event Stream Processing Studio. You can open the project and view the following items:
   a. The workspace in SAS Event Stream Processing Studio Modeler contains a Source window and a Calculate window.
   b. To view the SAS Micro Analytic Service module that was created, click and expand **SAS Micro Analytics Service Models** in the right pane.
   c. To view the files for the SAS Micro Analytic Service module, click on the left toolbar and view the **/home/analytics/mas-modules/*moduleName*** folder in the **Project Package** pane.
   d. To view project versions, click and select **Version 1** in the **Project Versions** pane. The information in the right pane shows that the project has already been promoted to the Test promotion level. Promoting a project to Test makes it available in SAS Event Stream Manager.
3. In SAS Event Stream Manager, click **Container Jobs** on the navigation bar. The Status column shows the progress of the job for creating a project container. When the status is , the project container has been created. For more information, see "Deploy and Manage Project Containers" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Working with SAS Micro Analytic Service Modules in SAS Event Stream Processing Studio

Overview

You can use SAS Micro Analytic Service modules to create input handler functions in SAS Event Stream Processing Studio by using Python, DS2, and C. You can embed code in the input handler or place the code in an external file.

Python code included in SAS Micro Analytic Service modules is controlled and executed separately from Python code elsewhere in ESP projects.

Instead of embedding code or using an external code file, you can reference models that were created in SAS Model Manager. For more information, see "Overview of SAS Model Manager Integration".

Create a SAS Micro Analytic Service Module

1 Open your project and click  .

2 In the right pane, expand **SAS Micro Analytic Service Modules**.

3 Click .

The SAS Micro Analytic Service Module window appears.

4 In the **Name** field, enter a name for the module.

5 In the **Language** drop-down list, select the language that you want to use to write the module.

Note: When you are working with Python code, you can use Python windows instead of placing the Python code in a SAS Micro Analytic Service module. For more information, see "Working with Python Windows in SAS Event Stream Processing Studio".

6 In the **Description** field, enter a description of the module.

7 In the **Function names** field, enter a comma-separated list of function names.

8 In the **Code source** field, select one of the following options:

- **Embedded code** to enter your own code
- **External file** to use code that is located in an external file
- **SAS Micro Analytic Service store** to use code in an analytic store file

Note: Using this option involves automatically importing a model from SAS Model Manager. This option is relevant only when SAS Event Stream Processing is deployed with other SAS Viya platform products. For more information, see "Overview of SAS Model Manager Integration".

9 If you selected **Embedded code** in the **Code source** field, enter your code in the **Embedded code** field. If you set the code language to C or DS2, the editor suggests keywords as you enter your code. If you set the code language to Python, the editor validates your Python code.

10 If you selected **External file** in the **Code source** field, enter the file path to the external file in the **External file** field.

11 If you selected **SAS Micro Analytic Service store** in the **Code source** field, complete these actions:

a In the **External file** field, enter the file path to the analytic store file.

b In the **SAS Micro Analytic Service store** field, enter the module store location.

c In the **SAS Micro Analytic Service store version** field, enter the version of the module store location.

d In the **Module Members** field, enter your module's member names. To add a new module member, click and fill in the applicable fields.

12 Click **OK**.

The module that you created appears in the SAS Micro Analytic Service Modules table.

Delete a SAS Micro Analytic Service Module

1 Open your project and click .

2 In the right pane, expand **SAS Micro Analytic Service Modules**.

3 Select the module that you want to delete and click .

4 Confirm the deletion.

If the project is part of a project package, and the SAS Micro Analytic Service module had been created as a result of manually importing a model into a project package, deleting the SAS Micro Analytic Service module also deletes associated files in the project package.

If the project is a project XML file, the module is deleted from the project. This means that the project's reference to a SAS Micro Analytic Service store is deleted. However, the SAS Micro Analytic Service store is not deleted. Members of the administrator group can delete SAS Micro Analytic Service stores by using the **Settings** page in SAS Event Stream Manager. For more information, see

"Delete SAS Micro Analytic Service Stores" in *SAS Event Stream Processing: Using SAS Event Stream Manager*.

Update SAS Micro Analytic Service Modules

This topic is relevant for models that were automatically imported from SAS Model Manager. For more information, see "Overview of SAS Model Manager Integration".

A SAS Event Stream Processing project that contains a SAS Micro Analytic Service module can be updated when a new champion model is declared. SAS Event Stream Processing Studio does not automatically check for new champion models.

To check for new champion models:

1 Open the project.

2 Click ⋮ and select **Check for new champion models**.

3 In the New Champion Model Available window, click **Yes** to accept the update.

Note: When the project contains several SAS Micro Analytic Service modules, the New Champion Model Available window appears again for each module for which there is a new champion model available.

Working with Input Handlers

Overview

You can register event stream input handlers for the Procedural and Calculate windows. *Input handlers* process incoming event streams in your model. You can import score code created in SAS Model Manager directly into a specific Calculate window in your model. You define a SAS Micro Analytic Service map in a Calculate window to bind a function to an input window. This binding acts as the input handler for the Calculate window.

Create Input Handler Functions in Calculate Windows

1 Open the relevant project.

2 Click the relevant Calculate window.

The right pane displays the properties of the Calculate window.

3 In the right pane, expand **Settings**.

4 In the **Calculation** drop-down list, select **User-specified**.

5 In the **Handlers** section, click the row for the input window that you want to link the import handler function to.

6 Click .

The Input Handler window appears.

7 In the **Handler type** field, select one of the following handler types:

- **SAS Micro Analytic Service** – enables you to reference a SAS Micro Analytic Service module.
- **Import a module from SAS Model Manager** – enables you to reference a SAS Micro Analytic Service module.

Note: Using this option involves automatically importing a model from SAS Model Manager. For more information, see "Overview of SAS Model Manager Integration".

The Input Handlers window reloads to display fields that relate to the handler type that you selected.

Note: In the **Function** drop-down list, the function that is automatically selected by default is the first function in the drop-down list. Alternatively, if the drop-down list does not contain any functions, a Score method is displayed.

8 Update any other fields as required.

9 Click **OK**.

The **Handlers** section refreshes to display the imported function.

Create Input Handler Functions in Procedural Windows

1 Open the relevant project.

2 Click the relevant Procedural window.

The right pane displays the properties of the Procedural window.

3 In the right pane, expand **Input Handlers**.

4 Click the row for the input window that you want to link the import handler function to.

5 Click .

The Input Handlers window appears.

6 Enter the name of the plug-in library and the function that you want to reference.

The function that you reference operates on a single event within an incoming event block by default. The function is repeatedly called for each event. Alternatively, you can run the function on an entire incoming event block, instead of on each event. To enable this functionality, select the **Run the function once on the incoming event block** check box.

CAUTION
Exercise caution when running a function on an event block instead of an event. As the function that you are running from the plug-in library is located outside of your SAS Event Stream Processing environment, this could cause your ESP server to stop processing events.

7 Click **OK**.

The **Input handler functions** section refreshes to display the imported function.

Working with ONNX Models in SAS Event Stream Processing Studio

You can use SAS Event Stream Processing Studio to reference an Open Neural Network Exchange (ONNX) model from a project:

1 Obtain an ONNX model.

2 Place your ONNX model in a location where SAS Event Stream Processing Studio can access it. This location must be on the persistent volume.

When your project is part of a project package, an easy way to place the ONNX model in an appropriate location is to upload it to the **`analytics`** folder of the project package. For more information, see "Manage a Project Package".

IMPORTANT In a Kubernetes environment, using ONNX models requires that when SAS Event Stream Processing is deployed, it is configured to access persistent volumes. This configuration involves applying overlays. For more information, see "Managing Persistent Volumes" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

3 Use the Model Reader and Score windows in your project to reference the ONNX model.

For examples of how to use SAS Event Stream Processing Studio to reference an ONNX model from a project, install the `Computer Vision with ONNX`, `Pose Estimation with ONNX` and `Voice Transcription with ONNX` examples. For more information, see "Install Example Projects".

For more information about support for ONNX models in SAS Event Stream Processing, including information about supported execution providers, see "Working with ONNX Models" in *SAS Event Stream Processing: Using Streaming Analytics*.

Forward Events to Source Windows

You can use event forwarding to send events from Lua windows or Python windows to Source windows. Event forwarding is particularly useful when one of the windows that is doing the forwarding produces several events from a single event. In this case, using event forwarding instead of a direct edge means that less memory is used, and there is more concurrency in the streaming model.

This topic explains how to set up event forwarding in SAS Event Stream Processing Studio. For more information about forwarding events from a Lua window, see "Event Forwarding" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about forwarding events from a Python window, see "Event Forwarding" in *SAS Event Stream Processing: Using Source and Derived Windows*.

For an example of a project that uses event forwarding, install the Forwarding example. For more information, see "Install Example Projects". The following discussion uses materials from the Forwarding example to illustrate how event forwarding is represented in the project diagram.

The Forwarding example contains two continuous queries: `internal` and `external`. The convertTemp window in the `internal` continuous query forwards events to the readingsFahrenheitExternal window in the `external` continuous query. When a Lua window or a Python window forwards events to a Source window that is in another continuous query, this action is indicated with the icon in the Lua window or the Python window, rather than with an edge. Similarly, when a Source window receives forwarded events from another continuous query, this action is indicated with the icon.

Figure 33 *Continuous Query Called Internal*

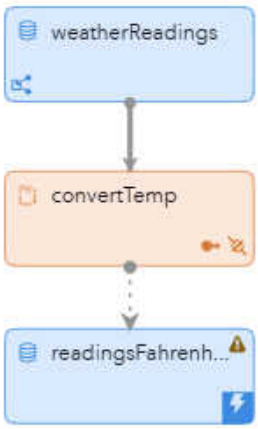
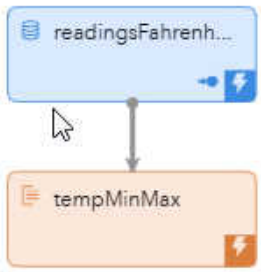


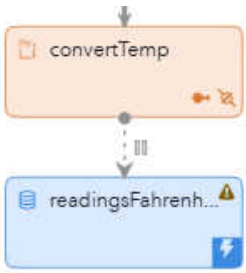
Figure 34 *Continuous Query Called External*



Event forwarding within the same continuous query is represented in the diagram by an edge: ····➤. In the Forwarding example, in addition to forwarding events to another continuous query, the convertTemp window also forwards events to a window within the same continuous query.

Connections for event forwarding can be active or inactive. When the project is running, an inactive connection for event forwarding does not send events to the destination Source window. When an event forwarding edge is inactive, the icon ⏸ is displayed on the edge. The following figure shows what the Forwarding example would look like if you were to make the edge between the convertTemp and the next window inactive.

Figure 35 *Example of an Event Forwarding Edge That Is Inactive*



To configure a Lua window or a Python window to send events to Source windows:

- "Working with Lua Windows in SAS Event Stream Processing Studio"
- "Working with Python Windows in SAS Event Stream Processing Studio"

Several Lua windows and Python windows can pass events to the same Source window. To view the incoming connections for event forwarding for a specific Source window:

1 Select the Source window.

2 In the right pane, expand **Event Forwarding**.

Working with Lua in SAS Event Stream Processing Studio

Working with Lua Virtual Environments

A Lua virtual environment specifies a set of Lua modules. When a user selects a virtual environment that an administrator has created, the project runs with the Lua modules specified in that virtual environment.

For information about how an administrator creates a virtual environment, see "Specify Virtual Environments". For information about how a user selects a virtual environment, see "Update Project Properties".

Working with Lua Windows in SAS Event Stream Processing Studio

The Lua window enables you to consume and generate SAS Event Stream Processing events with Lua programs. This topic explains how to configure a Lua window in SAS Event Stream Processing Studio. For more information about using Lua windows, see "Using Lua Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

For examples of projects with Lua windows, install the Lua Compute, Lua Parse, and Lua State examples. For more information, see "Install Example Projects".

To configure a Lua window in SAS Event Stream Processing Studio:

1. Open a project.
2. Expand **Transformations** in the **Windows** pane on the left, and drag a Lua window to the workspace.
3. In the right pane, expand **Lua Settings**.
4. Use the **Fields to copy** field to specify which fields you want to copy from input events to output events. Selecting fewer fields improves performance.
   a. To copy all fields except the fields that you specify in the next step, select the **Exclude fields** radio button. To copy only those fields that you specify in the next step, select the **Include fields** radio button.
   b. Expand the drop-down list and select the check boxes for the fields that you want to exclude or include.

> **TIP** If there is a large number of fields to choose from, you might want to enter field names manually. When you start entering the field name in the drop-down list, matching fields are displayed.

5 In the **Fields to use in Lua code** field, specify fields that you want to pass from the ESP server to Lua code. The **Exclude fields** and **Include fields** radio buttons, and the drop-down list, work in the same way as in the **Fields to copy** field.

> **TIP** You can select the **Expand parameters** check box to add these fields to the Lua code. The fields are added as parameters of the events function.
>
> If you select the **Expand parameters** check box, then the **Process the events in blocks** check box is not available.

6 In the **Events function** field, enter the name of the Lua function that you want to use to generate SAS Event Stream Processing events.

This function must exist in the Lua code that you enter later in these steps. Lua code can contain several functions. The **Events function** field specifies which function is the model's entry point into the Lua code.

7 If required, use the **Initialization function** field to specify which function in your Lua code acts as an initialization function. For more information, see "Initialization of Lua Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

8 If required, use the **Heartbeat function** field to specify the name of a Lua function that is called at every heartbeat.

The heartbeat interval is specified for the entire project, rather than for the Lua window. For more information about heartbeats in Lua windows, see "Configuring a Lua Window" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about heartbeats at the project level, see "XML Language Elements for the Structure of a Model" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

9 If required, use the **Cleanup function** field to specify which function in your Lua code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Lua cleanup tasks.

10 If required, select the **Process the events in blocks** check box. If you process event in blocks, then the events function receives an array of Lua tables that represent the events in the block. Otherwise, the events function receives a single event.

11 If required, select the **Persist properties** check box. For more information, see "Persistent Properties" in *SAS Event Stream Processing: Using Source and Derived Windows*.

12 If required, select the **Encode binary data** check box. This option uses Base64 encoding to encode binary data as text, before that data is used by your Lua code.

13 If required, use the **Required modules** field to select Lua modules that you want to reference in your Lua code. For more information, see "Working with Lua Modules in SAS Event Stream Processing Studio".

14 Select the code source:

- Select **Window** if you want to enter new Lua code in this window.
- Select **Snippet** if you want to reference shared code in an existing Lua snippet. When the Lua window uses shared code within a snippet, it shares all data, functions, and variables with any other Lua entity that points to that snippet. The shared code is always locked by the entity

that is currently using it. For more information, see "Working with Lua Snippets in SAS Event Stream Processing Studio".

- Select **File** if you want to reference Lua code that is in a file. Use the **Path to code file** field to specify the file. When your project is part of a project package, the referenced file should be within the project package too. For more information, see "Reference Files That Are Included in a Project Package".

15 If you want to enter new Lua code in this window:

a Click $x=\frac{y}{z}$.

The Expression Editor window appears.

b Enter your code. For more information about Lua code that is appropriate in a Lua window, see "Using Lua Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about the code editor, see "Edit Code and Expressions".

c Click **OK**.

16 If required, use the **Error handling strategy when fatal errors are encountered** field to select an error handling strategy.

Lua code can throw exceptions. Exceptions from the `init` function cause the project to fail to load. By default, exceptions from the `create` function, the `heartbeat` function, and snippets (when threaded) are handled based on project-level error handling settings. If required, you can select a different error handling strategy for your Lua window, to override the project-level setting. You can choose to log an error and continue project execution, or to log a fatal error and stop the project. To check what the project-level setting is, click , expand **Advanced** in the right pane, and check which error handling strategy is selected.

17 If required, you can forward events from the Lua window to a Source window. For more information, see "Forward Events to Source Windows".

a Expand **Event Forwarding**.

b If required, select the **Suppress event creation** check box. Selecting this option means that the Lua window only forwards events and does not generate events.

c Click .

d In the Event Forwarding Destinations window, select a continuous query and a Source window.

e If required, adjust the block size (that is, the number of events that you want to put into each event block that is published to the Source window).

f Connections for event forwarding can be active or inactive. If required, use the **Active** check box to change the status of the connection.

g Click OK.

18 Edit any other window properties in the right pane as required.

> **TIP** When a project is part of a project package, if you want to make your Lua window available to other users as a new window type, you can convert the window to a custom window. For more information, see "Convert a Lua Window or a Python Window to a Custom Window" on page 154.

Working with Lua Snippets in SAS Event Stream Processing Studio

Lua snippets are self-contained blocks of Lua code that can be used to support other Lua entities. You create Lua snippets at the project level. You can then reference properties that are contained in a Lua snippet from Lua code elsewhere in the project.

This topic explains how to create Lua snippets in SAS Event Stream Processing Studio. For general information, see "Using Lua Snippets" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you already have a project to which you want to add Lua snippets. The steps explain how to add Lua snippets. For an example of a complete project with a Lua snippet, install the Lua Snippet example. For more information, see "Install Example Projects".

1 Open your project.

2 Click .

3 In the right pane, expand **Snippets**.

4 In the **Lua Snippets** section, the **Lua root path** field displays the project's Lua root directory. This information is relevant when you want to use persistent properties. For more information, see "Persistent Properties" in *SAS Event Stream Processing: Using Source and Derived Windows*.

   If your project is part of a project package, the path is set to `@ESP_PROJECT_OUTPUT@/luaroot` and you cannot change the path. For more information about the ESP_PROJECT_OUTPUT environment variable that is included in the path, see "Manage a Project Package".

   If your project is not part of a project package, enter the project's Lua root directory. You must specify a writeable directory. In a Kubernetes environment, you must specify a location within the persistent volume. For example, you might set this field to an appropriate subdirectory of `/mnt/data`.

5 Click .

6 In the Edit Lua Code Snippet window, adjust the name of the snippet.

7 Enter a description for the Lua snippet. When you close the Edit Lua Code Snippet window, the description that you enter is visible in the **Lua Snippets** section in the right pane. The description is also available when you edit Lua code elsewhere in your project and use the code editor to add a reference to your snippet. Providing a description can help you select the appropriate snippet. For more information, see "Edit Code and Expressions".

8 Select the code source:

- Select **Code editor** if you want to enter new Lua code in this snippet.
- Select **File** if you want to reference Lua code that is in a file. Use the **Path to code file** field to specify the file. When your project is part of a project package, the referenced file should be within the project package too. For more information, see "Reference Files That Are Included in a Project Package".

9 If required, use the **Initialization function** field to specify which function in your Lua code acts as an initialization function. For more information, see "Initialization of Lua Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

10 If required, specify a Lua function that is repeated at an interval that you select.

- The **Repeating function** field specifies which function in your Lua code acts as the repeating function.
- The return code from the function is a Boolean that tells the server whether to keep repeating the function. When you return `true`, the function is repeated. When you return `false`, the function is not repeated.
- For more information about using a repeating function (also referred to as starting a thread), see "Using Lua Snippets" in *SAS Event Stream Processing: Using Source and Derived Windows*.

11 If required, use the **Cleanup function** field to specify which function in your Lua code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Lua cleanup tasks.

12 If required, use the **Required modules** field to select Lua modules that you want to reference in the Lua code for your snippet. For more information, see "Working with Lua Modules in SAS Event Stream Processing Studio".

13 If you set the **Code source** field to **Code editor**, enter your Lua code in the code editor. For more information, see "Edit Code and Expressions".

14 Click **OK**.

15 If required, add more snippets.

16 Check the order in which the snippets appear in the table in the **Lua Snippets** section. Lua snippets are loaded in the order in which they are listed in the table. A snippet that is declared before another one might require the variables in the subsequent snippet, but they will not be available.

17 Reference the properties in the snippets from Lua code in other parts of the project. Use the **Snippet Properties** tab in the code editor to add such references. For more information, see "Edit Code and Expressions". In a Lua window, you can also use the **Code source** field to reference a snippet. For more information, see "Working with Lua Windows in SAS Event Stream Processing Studio".

18 Configure any other desired settings for the rest of your project.

Working with Lua Modules in SAS Event Stream Processing Studio

Lua modules are self-contained blocks of Lua code that can be reused in other Lua entities. You create Lua modules at the project level. Lua modules enable you to write code one time and reference that code from Lua code elsewhere in the same project. For example, you could create a Lua module that defines useful common functions, and then reference those functions from other Lua code in the project.

This topic explains how to create Lua modules in SAS Event Stream Processing Studio. For more information, see "Using Lua Modules" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you already have a project to which you want to add Lua modules. The steps explain how to add Lua modules. For an example of a complete project with a Lua module, install the Lua Module example. For more information, see "Install Example Projects".

1 Open your project.

2 Click  .

3 In the right pane, expand **Modules**.

4 In the **Lua Modules** section, click  .

5 In the Edit Lua Code Module window, adjust the name of the module.

6 Enter a description for the Lua module.

7 In the **Required modules** drop-down field, specify any other Lua modules that you want to reference from this new module.

8 Enter your Lua code. For more information about the code editor, see "Edit Code and Expressions".

9 Click **OK**.

10 If required, add more modules.

11 Reference the modules from Lua code in other parts of the project. Use the right pane of the code editor to add such references. For more information, see "Edit Code and Expressions".

12 Configure any other desired settings for the rest of your project.

Working with Lua-Based Filter Windows in SAS Event Stream Processing Studio

The Filter window enables you to determine which input events are permitted to pass through. To define filter conditions, you can use Lua, Python, a plug-in function, or Expression Engine Language (EEL).

Note: The ability to use EEL in Filter windows is legacy functionality.

This topic explains how to create a Lua-based Filter window in SAS Event Stream Processing Studio. For general information, see "Using Filter Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you have a project that already contains a Source window with an output schema. The steps explain how to add a Filter window. For examples of complete projects with Lua-based Filter windows, install the Trades and Sailing examples. For more information, see "Install Example Projects".

1 Open your project.

2 Expand **Transformations** in the **Windows** pane on the left and drag a Filter window to the workspace.

3 In the right pane, expand **Filter**.

4 Set the filter type to Lua.

5 Use the **Fields to use in code** field to specify fields that you want to pass from the ESP server to Lua code. Selecting fewer fields improves performance.

   a To use all fields except the fields that you specify in the next step, select the **Exclude fields** radio button. To use only those fields that you specify in the next step, select the **Include fields** radio button.

   b Expand the drop-down list and select the check boxes for the fields that you want to exclude or include.

   > **TIP** You can select the **Expand parameters** check box to add the selected fields to the Lua code. The fields are added as parameters of the filter function.

6 Use the **Filter function** field to specify which function in your Lua code acts as a filter function. The default value is `filter`.

7 If required, use the **Initialization function** field to specify which function in your Lua code acts as an initialization function. For more information, see "Initialization of Lua Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

8 If required, use the **Heartbeat function** field to specify the name of a Lua function that is called at every heartbeat.

   The heartbeat interval is specified for the entire project, rather than for the Filter window. For more information about heartbeats at the project level, see "XML Language Elements for the Structure of a Model" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

9 If required, use the **Cleanup function** field to specify which function in your Lua code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Lua cleanup tasks.

10 If required, use the **Required modules** field to select Lua modules that you want to reference in the Lua code for your filter condition. For more information, see "Working with Lua Modules in SAS Event Stream Processing Studio".

11 Select the code source:

   - Select **Window** if you want to enter new Lua code for your filter condition.
   - Select **Snippet** if you want to reference shared code in an existing Lua snippet. When a Lua-based Filter window uses shared code within a snippet, it shares all data, functions, and variables with any other Lua entity that points to that snippet. The shared code is always locked by the entity that is currently using it. For more information, see "Working with Lua Snippets in SAS Event Stream Processing Studio".

12 If you want to enter new Lua code for your filter condition:

   a Click $x=\frac{y}{z}$.

   b In the Edit Lua Code window, enter your Lua code. You code must include a filter function that returns a Boolean value. The ESP server calls this function for each input event and uses the return code to determine whether to generate an output event. Ensure that the name of this function matches the name that is specified in the **Filter function** field.

      For more information about Lua code that is appropriate in a Filter window, see "Using Lua in a Filter Window" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about the code editor, see "Edit Code and Expressions".

   c Click **OK**.

13 Configure any other desired settings for the Filter window and for the rest of your project.

Working with Lua-Based Pattern Windows in SAS Event Stream Processing Studio

The Pattern window enables you to detect events of interest (EOIs) as they stream through a project and then generate output events as a result of applying pattern logic. To define EOIs, you can use Lua, Python, or the Expression Engine Language (EEL).

Note: It is recommended that you use Lua or Python in Pattern windows. The ability to use EEL in Pattern windows is legacy functionality.

This topic explains how to create a Lua-based Pattern window in SAS Event Stream Processing Studio. For general information about the purpose of Pattern windows, see "Using Pattern Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you have a project that already contains a Source window with an output schema. The steps explain how to add a Pattern window. For an example of a complete project with a Lua-based Pattern window, install the Lua Pattern example. For more information, see "Install Example Projects".

1 Open your project.

2 Expand **Utilities** in the **Windows** pane on the left, and drag a Pattern window to the workspace.

3 In the right pane, expand **Patterns**.

The pattern type is set to Lua by default.

4 Click .

The New Pattern window appears. The New Pattern window is a wizard with four pages.

5 Complete the **Initialize** page:

a Adjust the name of the pattern if required.

b If required, select index fields. These fields, from the input window, form an index generation function. The input window might be a Source window or another type of window. All incoming events are grouped by the specified index.

c If required, select the **Allow open patterns the possibility of timing out with each incoming event, regardless of the event's index value** check box.

d If required, specify one or more time fields. For each window, you can select a field to be used to specify the time. If a time field is not specified, system time is used.

e Click **Next**.

6 Complete the **Lua Code** page:

a If required, use the **Required modules** field to select Lua modules that you want to reference in the Lua code for your pattern. For more information, see "Working with Lua Modules in SAS Event Stream Processing Studio".

b Enter the Lua code for your pattern, including the EOI functions and an output function that are required for your pattern.

For more information about using the code editor, see "Edit Code and Expressions". For more information about EOI functions, see "Writing EOI Functions in Lua" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about output functions, see "Writing Output Functions in Lua" in *SAS Event Stream Processing: Using Source and Derived Windows*.

c Click **Next**.

7 Complete the **Output and Events** page:

a If required, use the **Initialization function** field to specify which function in your Lua code acts as an initialization function. For more information, see "Initialization of Lua Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

b In the **Output function** field, select the name of the output function. The output function must exist in the Lua code that you added on the **Lua Code** page.

c If required, use the **Cleanup function** field to specify which function in your Lua code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Lua cleanup tasks.

d In the **Events** subsection, click  to create an EOI.

A new row is added to the table.

e In the Name column, adjust the event's name as required.

f In the Included Fields column, select the Lua fields that you want to pass to the event. If you leave this column empty, all Lua fields are passed to the event.

g In the Lua Function column, select the EOI function that you want to associate with the event. All functions that you entered in the Edit Lua Functions window are available in this column. Select an appropriate EOI function, rather than an output function.

h In the Send Event column, select the check box if you want to send all previously matched events into the function.

i Add more EOIs if required.

j Click **Next**.

8 Use the Logic Expression page to create pattern logic. The logic expression does not use the Lua language. When you have finished, click **OK**.

For more information about using the code editor, see "Edit Code and Expressions". For more information about writing a logic expression, see "Applying Pattern Logic" in *SAS Event Stream Processing: Using Source and Derived Windows*.

9 If required, add more patterns.

10 Click  .

11 Click  .

The Output Schema window appears.

12 Use the Output Schema window to specify a key field, and to specify output fields and field types. When you have finished, click **OK**.

Note: All patterns in a Pattern window have the same output fields. If you require different output fields for certain patterns, you can add more Pattern windows to your project.

13 Configure any other desired settings for the Pattern window and for the rest of your project.

Working with Python in SAS Event Stream Processing Studio

Working with Python Virtual Environments

A Python virtual environment specifies a set of Python packages. When a user selects a virtual environment that an administrator has created, the project runs with the Python packages specified in that virtual environment.

For information about how an administrator creates a virtual environment, see "Specify Virtual Environments". For information about how a user selects a virtual environment, see "Update Project Properties".

Working with Python Windows in SAS Event Stream Processing Studio

The Python window enables you to consume and generate SAS Event Stream Processing events with Python programs. When you are working with Python code, you can use Python windows instead of placing the Python code in a SAS Micro Analytic Service module. For more information, see "Using Python Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

Note: When you use the Python window, your code must be compatible with Python 3.11.

For an example of a project with a Python window, install the Python Compute example. For more information, see "Install Example Projects".

To configure a Python window in SAS Event Stream Processing Studio:

1. Open a project.
2. Expand **Transformations** in the **Windows** pane on the left, and then drag a Python window to the workspace.
3. In the right pane, expand **Python Settings**.
4. Use the **Fields to copy** field to specify which fields you want to copy from input events to output events. Selecting fewer fields improves performance.

a To copy all fields except the fields that you specify in the next step, select the **Exclude fields** radio button. To copy only those fields that you specify in the next step, select the **Include fields** radio button.

b Expand the drop-down list and select the check boxes for the fields that you want to exclude or include.

> **TIP** If there is a large number of fields to choose from, you might want to enter field names manually. When you start entering the field name in the drop-down list, matching fields are displayed.

5 In the **Fields to use in Python code** field, specify fields that you want to pass from the ESP server to Python code. The **Exclude fields** and **Include fields** radio buttons, and the drop-down list, work in the same way as in the **Fields to copy** field.

> **TIP** You can select the **Expand parameters** check box to add these fields to the Python code. The fields are added as parameters of the events function.
>
> If you select the **Expand parameters** check box, then the **Process the events in blocks** check box is not available.

6 In the **Events function** field, enter the name of the Python function that you want to use to generate SAS Event Stream Processing events.

This function must exist in the Python code that you enter later in these steps. Python code can contain several functions. The **Events function** field specifies which function is the model's entry point into the Python code.

7 If required, use the **Initialization function** field to specify which function in your Python code acts as an initialization function. For more information, see "Initialization of Python Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

8 If required, use the **Heartbeat function** field to specify the name of a Python function that is called at every heartbeat.

The heartbeat interval is specified for the entire project, rather than for the Python window. For more information about heartbeats in Python windows, see "Configuring a Python Window" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about heartbeats at the project level, see "XML Language Elements for the Structure of a Model" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

9 If required, use the **Cleanup function** field to specify which function in your Python code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Python cleanup tasks.

10 If required, select the **Process the events in blocks** check box. If you process events in blocks, then the events function receives an array of Python tables that represent the events in the block. Otherwise, the events function receives a single event.

11 If required, select the **Encode binary data** check box. This option uses Base64 encoding to encode binary data as text, before that data is used by your Python code.

12 Select the code source:

- Select **Window** if you want to enter new Python code in this window.
- Select **Snippet** if you want to reference shared code in an existing Python snippet. When the Python window uses shared code within a snippet, it shares all data, functions, and variables with any other Python entity that points to that snippet. The shared code is always locked by the entity that is currently using it. For more information, see "Working with Python Snippets in SAS Event Stream Processing Studio".
- Select **File** if you want to reference Python code that is in a file. Use the **Path to code file** field to specify the file. When your project is part of a project package, the referenced file should be within the project package too. For more information, see "Reference Files That Are Included in a Project Package".

13 If you want to enter new Python code in this window:

a Click  .

The Expression Editor window appears.

b Enter your code. For more information about Python code that is appropriate in a Python window, see "Using Python Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about the code editor, see "Edit Code and Expressions".

c Click **OK**.

14 If required, use the **Error handling strategy when fatal errors are encountered** field to select an error handling strategy.

Python code can throw exceptions. Exceptions from the `init` function cause the project to fail to load. By default, exceptions from the `create` function, the `heartbeat` function, and snippets (when threaded) are handled based on project-level error handling settings. If required, you can select a different error handling strategy for your Python window, to override the project-level setting. You can choose to log an error and continue project execution, or to log a fatal error and stop the project. To check what the project-level setting is, click  , expand **Advanced** in the right pane, and check which error handling strategy is selected.

15 If required, you can forward events from the Python window to a Source window. For more information, see "Forward Events to Source Windows".

a Expand **Event Forwarding**.

b If required, select the **Suppress event creation** check box. Selecting this option means that the Python window only forwards events and does not generate events.

c Click  .

d In the Event Forwarding Destinations window, select a continuous query and a Source window.

e If required, adjust the block size (that is, the number of events that you want to put into each event block that is published to the Source window).

f Connections for event forwarding can be active or inactive. If required, use the **Active** check box to change the status of the connection.

g Click OK.

16 Edit any other window properties in the right pane as required.

> **TIP** When a project is part of a project package, if you want to make your Python window available to other users as a new window type, you can convert the window to a custom window. For more information, see "Convert a Lua Window or a Python Window to a Custom Window" on page 154.

Working with Python Snippets in SAS Event Stream Processing Studio

Python snippets are self-contained blocks of Python code that can be used to support other Python entities. You create Python snippets at the project level. You can then reference properties that are contained in a Python snippet from Python code elsewhere in the project.

This topic explains how to create Python snippets in SAS Event Stream Processing Studio. For general information, see "Using Python Snippets" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you already have a project to which you want to add Python snippets. The steps explain how to add Python snippets. For an example of a complete project with a Python snippet, install the Python Snippet example. For more information, see "Install Example Projects".

1 Open your project.

2 Click .

3 In the right pane, expand **Snippets**.

4 In the **Python Snippets** section, click .

5 In the Edit Python Code Snippet window, adjust the name of the snippet.

6 Enter a description for the Python snippet. When you close the Edit Python Code Snippet window, the description that you enter is visible in the **Python Snippets** section in the right pane. The description is also available when you edit Python code elsewhere in your project and use the code editor to add a reference to your snippet. Providing a description can help you select the appropriate snippet. For more information, see "Edit Code and Expressions".

7 Select the code source:

- Select **Code editor** if you want to enter new Python code in this snippet.
- Select **File** if you want to reference Python code that is in a file. Use the **Path to code file** field to specify the file. When your project is part of a project package, the referenced file should be within the project package too. For more information, see "Reference Files That Are Included in a Project Package".

8 If required, use the **Initialization function** field to specify which function in your Python code acts as an initialization function. For more information, see "Initialization of Python Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

9 If required, specify a Python function that is repeated at an interval that you select.

- The **Repeating function** field specifies which function in your Python code acts as the repeating function.
- The return code from the function is a Boolean that tells the server whether to keep repeating the function. When you return `true`, the function is repeated. When you return `false`, the function is not repeated.
- For more information about using a repeating function (also referred to as starting a thread), see "Using Python Snippets" in *SAS Event Stream Processing: Using Source and Derived Windows*.

10 If required, use the **Cleanup function** field to specify which function in your Python code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Python cleanup tasks.

11 If you set the **Code source** field to **Code editor**, enter your Python code in the code editor. For more information, see "Edit Code and Expressions".

12 Click **OK**.

13 If required, add more snippets.

14 Check the order in which the snippets appear in the table in the **Python Snippets** section. Python snippets are loaded in the order in which they are listed in the table. A snippet that is declared before another one might require the variables in the subsequent snippet, but they will not be available.

15 Reference the properties in the snippets from Python code in other parts of the project. Use the **Snippet Properties** tab in the code editor to add such references. For more information, see "Edit Code and Expressions". In a Python window, you can also use the **Code source** field to reference a snippet. For more information, see "Working with Python Windows in SAS Event Stream Processing Studio".

16 Configure any other desired settings for the rest of your project.

Working with Python Modules in SAS Event Stream Processing Studio

Python modules are self-contained blocks of Python code that can be reused in other Python entities. You create Python modules at the project level. Python modules enable you to write code one time and reference that code from Python code elsewhere in the same project. For example, you could create a Python module that defines useful common functions, and then reference those functions from other Python code in the project.

This topic explains how to create Python modules in SAS Event Stream Processing Studio. For more information, see "Using Python Modules" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you already have a project to which you want to add Python modules. The steps explain how to add Python modules. For an example of a complete project with a Python module, install the Python Module example. For more information, see "Install Example Projects".

1 Open your project.

2 Click  .

3 In the right pane, expand **Modules**.

4 In the **Python Modules** section, click  .

5 In the Edit Python Code Module window, adjust the name of the module.

6 Enter a description for the Python module.

7 Enter your Python code. For more information about the code editor, see "Edit Code and Expressions".

8 Click **OK**.

9 If required, add more modules.

10 Reference the modules from Python code in other parts of the project. Use the right pane of the code editor to add such references. For more information, see "Edit Code and Expressions".

11 Configure any other desired settings for the rest of your project.

Working with Python-Based Filter Windows in SAS Event Stream Processing Studio

The Filter window enables you to determine which input events are permitted to pass through. To define filter conditions, you can use Lua, Python, a plug-in function, or Expression Engine Language (EEL).

Note: The ability to use EEL in Filter windows is legacy functionality.

This topic explains how to create a Python-based Filter window in SAS Event Stream Processing Studio. For more information, see "Using Python Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

Note: When you use a Python-based Filter window, your code must be compatible with Python 3.11.

The following steps assume that you have a project that already contains a Source window with an output schema. The steps explain how to add a Filter window. For an example of a complete project

with a Python-based Filter window, install the Geofence example. For more information, see "Install Example Projects".

1 Open your project.

2 Expand **Transformations** in the **Windows** pane on the left and drag a Filter window to the workspace.

3 In the right pane, expand **Filter**.

The filter type is set to Python by default.

4 Use the **Fields to use in code** field to specify fields that you want to pass from the ESP server to Python code. Selecting fewer fields improves performance.

a To use all fields except the fields that you specify in the next step, select the **Exclude fields** radio button. To use only those fields that you specify in the next step, select the **Include fields** radio button.

b Expand the drop-down list and select the check boxes for the fields that you want to exclude or include.

> **TIP** If there is a large number of fields to choose from, you might want to enter field names manually. When you start entering the field name in the drop-down list, matching fields are displayed.

5 Use the **Filter function** field to specify which function in your Python code acts as a filter function. The default value is `filter`.

6 If required, use the **Initialization function** field to specify which function in your Python code acts as an initialization function. For more information, see "Initialization of Python Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

7 If required, use the **Heartbeat function** field to specify the name of a Python function that is called at every heartbeat.

The heartbeat interval is specified for the entire project, rather than for the Filter window. For more information about heartbeats at the project level, see "XML Language Elements for the Structure of a Model" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

8 If required, use the **Cleanup function** field to specify which function in your Python code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Python cleanup tasks.

9 Select the code source:

- Select **Window** if you want to enter new Python code for your filter condition.
- Select **Snippet** if you want to reference shared code in an existing Python snippet. When a Python-based Filter window uses shared code within a snippet, it shares all data, functions, and variables with any other Python entity that points to that snippet. The shared code is always locked by the entity that is currently using it. For more information, see "Working with Python Snippets in SAS Event Stream Processing Studio".

10 If you want to enter new Python code for your filter condition:

a Click $x = \frac{y}{z}$.

b In the Edit Python Code window, enter your Python code. You code must include a filter function that returns a Boolean value. The ESP server calls this function for each input event and uses the return code to determine whether to generate an output event. Ensure that the name of this function matches the name that is specified in the **Filter function** field.

For more information about Python code that is appropriate in a Filter window, see "Using Python in a Filter Window" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about the code editor, see "Edit Code and Expressions".

c Click **OK**.

11 Configure any other desired settings for the Filter window and for the rest of your project.

Working with Python-Based Pattern Windows in SAS Event Stream Processing Studio

The Pattern window enables you to detect events of interest (EOIs) as they stream through a project and then generate output events as a result of applying pattern logic. To define EOIs, you can use Lua, Python, or the Expression Engine Language (EEL).

Note: It is recommended that you use Lua or Python in Pattern windows. The ability to use EEL in Pattern windows is legacy functionality.

This topic explains how to create a Python-based Pattern window in SAS Event Stream Processing Studio. For general information about the purpose of Pattern windows, see "Using Pattern Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

The following steps assume that you have a project that already contains a Source window with an output schema. The steps explain how to add a Pattern window. For an example of a complete project with a Python-based Pattern window, install the Python Pattern example. For more information, see "Install Example Projects".

1 Open your project.

2 Expand **Utilities** in the **Windows** pane on the left, and drag a Pattern window to the workspace.

3 In the right pane, expand **Patterns**.

4 Set the pattern type to **Python**.

5 Click  .

The New Pattern window appears. The New Pattern window is a wizard with four pages.

6 Complete the **Initialize** page:

a Adjust the name of the pattern if required.

b If required, select index fields. These fields, from the input window, form an index generation function. The input window might be a Source window or another type of window. All incoming events are grouped by the specified index.

c If required, select the **Allow open patterns the possibility of timing out with each incoming event, regardless of the event's index value** check box.

d If required, specify one or more time fields. For each window, you can select a field to be used to specify the time. If a time field is not specified, system time is used.

e Click **Next**.

7 Complete the **Python Code** page:

a Enter the Python code for your pattern, including the EOI functions and an output function that are required for your pattern.

For more information about using the code editor, see "Edit Code and Expressions". For more information about EOI functions, see "Writing EOI Functions in Python" in *SAS Event Stream Processing: Using Source and Derived Windows*. For more information about output functions, see "Writing Output Functions in Python" in *SAS Event Stream Processing: Using Source and Derived Windows*.

b Click **Next**.

8 Complete the **Output and Events** page:

a If required, use the **Initialization function** field to specify which function in your Python code acts as an initialization function. For more information, see "Initialization of Python Entities" in *SAS Event Stream Processing: Using Source and Derived Windows*.

b In the **Output function** field, select the name of the output function. The output function must exist in the Python code that you added on the **Python Code** page.

c If required, use the **Cleanup function** field to specify which function in your Python code acts as a cleanup function. The cleanup function runs when the project is stopped. Your cleanup function should perform any necessary Python cleanup tasks.

d In the **Events** subsection, click to create an EOI.

A new row is added to the table.

e In the Name column, adjust the event's name as required.

f In the Included Fields column, select the Python fields that you want to pass to the event. If you leave this column empty, all Python fields are passed to the event.

g In the Python Function column, select the EOI function that you want to associate with the event. All functions that you entered in the Edit Python Functions window are available in this column. Select an appropriate EOI function, rather than an output function.

h In the Send Event column, select the check box if you want to send all previously matched events into the function.

i Add more EOIs if required.

j Click **Next**.

9 Use the Logic Expression page to create pattern logic. The logic expression does not use the Python language. When you have finished, click **OK**.

For more information about using the code editor, see "Edit Code and Expressions". For more information about writing a logic expression, see "Applying Pattern Logic" in *SAS Event Stream Processing: Using Source and Derived Windows*.

10 If required, add more patterns.

11 Click  .

12 Click  .

The Output Schema window appears.

13 Use the Output Schema window to specify a key field, and to specify output fields and field types. When you have finished, click **OK**.

Note: All patterns in a Pattern window have the same output fields. If you require different output fields for certain patterns, you can add more Pattern windows to your project.

14 Configure any other desired settings for the Pattern window and for the rest of your project.

Working with StateDB Windows in SAS Event Stream Processing Studio

Overview

The StateDB Writer and StateDB Reader windows enable you to use an externally deployed, high-performance data store or database to store and maintain the "state" that is required for joins and aggregations. Using an external store reduces the need to use RAM. For more information about the purpose of StateDB windows and an example, see "Using StateDB Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

A project that includes a StateDB Writer window might include a Source window that receives data, Functional windows that process that data, and a StateDB Writer window that writes the data to the database.

There can be only one edge leading to a StateDB Writer window. When a project receives data from multiple Source windows, use a Join window to pass the data to a StateDB Writer window.

In a project that includes a StateDB Reader window, the StateDB Reader window would read data from the database and could also perform aggregation on that data. You might then use other windows, such as Functional windows, to process the data further.

Configure a StateDB Writer Window

1 Open a project.

2 Expand **Utilities** on the **Windows** pane on the left, and drag a StateDB Writer window to the workspace.

3 Connect the StateDB Writer window to a parent window that provides data to the StateDB Writer window.

4 In the right pane, expand **Database Connection**.

5 Select the database type. You can select Redis or SingleStore.

6 Configure the settings for accessing the database.

   a When the database type is Redis, you must indicate whether you are using a single Redis node or a cluster.

   b Enter host and port information. If you are using a Redis cluster, use the **Hosts** table to enter multiple hosts and their ports.

   c Enter the user name and password for accessing the database.

   d (Optional) When the database type is Redis, you can specify the interval, in seconds, at which to send a ping command to keep the connection alive. The default value is 15 seconds. When you are using a single Redis node, the ping is sent to the Redis server. When you are using a Redis cluster, the ping is sent to all nodes in the Redis cluster.

   e (Optional) When the database type is Redis, you can specify time-out values in milliseconds:

      - Use the **Connect time-out** field to specify the amount of time that the client waits to connect to Redis. The time-out should be short, between `1000` and `3000` milliseconds. The default value is `1500` milliseconds. The time-out might need to be adjusted based on network conditions and responsiveness of the Redis server.
      - Use the **Command time-out** field to specify the amount of time that the client waits for a response from Redis. Set the value based on your slowest expected command. The default value is `0`, which means that the Redis client's default value is used.
      - Use the **TCP user time-out** field to set the Linux transmission control protocol (TCP) socket option TCP_USER_TIMEOUT. This property is for the Redis server only and is not supported for Redis clusters. The default value is `0`, which means that the Redis client's default value is used.

   f When the database type is SingleStore, enter the name of a SingleStore database.

   g (Optional) Enter a name for the database connection. This name is a label for the connection between the ESP server and the data store. The name has no other effect on the project.

   h (Optional) When the database type is Redis, you can select an option to use Transport Layer Security (TLS). For more information, see "Using Transport Layer Security (TLS) in StateDB Windows When Redis Is the External Data Store" in *SAS Event Stream Processing: Using*

Source and Derived Windows. When you select the option to use TLS, you then specify the following further details as required:

- Specify the Server Name Indication (SNI). SNI is an extension to the TLS protocol that enables a client to indicate which host name it is attempting to connect to.
- Specify either a certificate authority (CA) certificate directory or a CA certificate file.

  You can specify a CA certificate directory where trusted CA certificate files are stored in an OpenSSL-compatible structure. TLS uses whatever authentication artifacts are located that directory.

  Alternatively, you can specify a CA certificate file. The file can be a CA certificate or a bundle file that specifies a set of known CA certificates.
- When mutual authentication is required, select a client-side certificate and a private key file. These two files are used together. If you select a client-side certificate, then you must also select a private key file.

i When the database type is Redis, click **Test Connection** to check that your credentials work.

Note: When the option to use TLS is selected, the **Test Connection** button is not available.

7 Expand **Database Write Properties** and configure the settings for writing events to the database.

a When the database type is Redis, enter the Redis prefix. The prefix is used to group data and serves as the table name.

b When the database type is SingleStore, enter the name of the SingleStore table. If the specified table does not already exist, a table with this name is created.

c You can specify “time to live” settings:

- Use the **Time to live** field to indicate when records should expire. You can enter the name of a time field or a value in seconds.
- Use the **Time to live offset** field to specify, in seconds, an offset for delaying record deletion.
- Use the **Time field** drop-down list to select a field that is used to specify the time for the expiration of records. If a time field is not specified, system time is used.

d In the **Secondary Indexes** table, specify the fields that are used to form secondary indexes. Each row in the table relates to a different secondary index.

e When the database type is Redis, you can select an option to delete outdated keys in secondary indexes.

f In the **Output** table, specify output fields for the events that are to be written to the database. That is, use the table to map fields in the ESP project to fields in the database.

g When the database type is SingleStore, you can select an option to remove all rows from the SingleStore table before writing.

CAUTION

Be careful when you are using this option. You cannot recover any data that is lost. For example, this option might not be appropriate for a project that is used in a production environment.

For more information about these settings, see "Elements That Are Specific to window-statedb-writer" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

8 Configure any other desired settings for the StateDB Writer window.

Configure a StateDB Reader Window

1 Open a project.

2 Expand **Utilities** on the **Windows** pane on the left, and drag a StateDB Reader window to the workspace.

3 In the right pane, expand **Database Connection**.

4 Select the database type. You can select Redis or SingleStore.

IMPORTANT When a StateDB Reader window reads from a Singlestore table, that table must contain the following two fields:

- `esp_meta_timestamp`
- `esp_meta_ttd`

Those fields are used by SAS Event Stream Processing to manage deleting fields from the Singlestore table. They are automatically created when the table is created by a StateDB Writer window. When you use a table that was otherwise created, you must manually add those two fields as type BIGINT.

5 Configure the settings for accessing the database.

a When the database type is Redis, you must indicate whether you are using a single Redis node or a cluster.

b Enter host and port information. If you are using a Redis cluster, use the **Hosts** table to enter multiple hosts and their ports.

c Enter the user name and password for accessing the database.

d (Optional) When the database type is Redis, you can specify the interval, in seconds, at which to send a ping command to keep the connection alive. The default value is 15 seconds. When you are using a single Redis node, the ping is sent to the Redis server. When you are using a Redis cluster, the ping is sent to all nodes in the Redis cluster.

e (Optional) When the database type is Redis, you can specify time-out values in milliseconds:

- Use the **Connect time-out** field to specify the amount of time that the client waits to connect to Redis. The time-out should be short, between `1000` and `3000` milliseconds. The

default value is `1500` milliseconds. The time-out might need to be adjusted based on network conditions and responsiveness of the Redis server.

- Use the **Command time-out** field to specify the amount of time that the client waits for a response from Redis. Set the value based on your slowest expected command. The default value is `0`, which means that the Redis client's default value is used.
- Use the **TCP user time-out** field to set the Linux transmission control protocol (TCP) socket option TCP_USER_TIMEOUT. This property is for the Redis server only and is not supported for Redis clusters. The default value is `0`, which means that the Redis client's default value is used.

f When the database type is SingleStore, enter the name of a SingleStore database.

g (Optional) Enter a name for the database connection. This name is a label for the connection between the ESP server and the data store. The name has no other effect on the project.

h (Optional) When the database type is Redis, you can select an option to use Transport Layer Security (TLS). For more information, see "Using Transport Layer Security (TLS) in StateDB Windows When Redis Is the External Data Store" in *SAS Event Stream Processing: Using Source and Derived Windows*. When you select the option to use TLS, you then specify the following further details as required:

- Specify the Server Name Indication (SNI). SNI is an extension to the TLS protocol that enables a client to indicate which host name it is attempting to connect to.
- Specify either a certificate authority (CA) certificate directory or a CA certificate file.

  You can specify a CA certificate directory where trusted CA certificate files are stored in an OpenSSL-compatible structure. TLS uses whatever authentication artifacts are located that directory.

  Alternatively, you can specify a CA certificate file. The file can be a CA certificate or a bundle file that specifies a set of known CA certificates.
- When mutual authentication is required, select a client-side certificate and a private key file. These two files are used together. If you select a client-side certificate, then you must also select a private key file.

i When the database type is Redis, click **Test Connection** to check that your credentials work.

Note: When the option to use TLS is selected, the **Test Connection** button is not available.

6 Expand **Database Query** and configure the settings for reading events from the database.

a When the database type is Redis, enter the Redis prefix for the data that you want to query.

b When the database type is SingleStore, enter the name of the table that you want to query.

c Use the **Query** table to create the database query. The query specifies the condition on which records are fetched from the database.

- The input fields that are available for you to select in the **Query** table reflect the schema of the parent window of the StateDB Writer window.
- When the database type is Redis, the only available operator is `==`. That is, your database query matches an input field to a field in Redis.

- When the database type is SingleStore, additional operators that are supported by SingleStore are available in the **Query** table.

d You can select a time field to manage the expiration of records. If a time field is not specified, system time is used.

e You can choose to generate Delete events. Consider selecting this option if your project includes subsequent stateful windows such as Aggregate windows.

For more information, see "Elements That Are Specific to window-statedb-reader" in *SAS Event Stream Processing: XML Language Reference for Event Stream Processing Models*.

7 Click and then click . Use the **Output Schema** window to specify the output for your StateDB Reader window.

- Use the Source column to indicate whether the data for a specific field comes from a database query or whether the data comes from the input event.
- For data that comes from a database query, use the Source Field column to indicate the name of the database field. For data that comes from an input event, use the Source Field column to indicate the name of the input field.
- If you want to perform aggregation, use the Aggregate Function column to select an aggregate function. The aggregate functions that are available in the Aggregate Function column are a subset of the functions available in Aggregate windows. The behavior of the functions is the same. For more information about aggregate functions, see "Using Aggregate Functions" in *SAS Event Stream Processing: Using Source and Derived Windows*.

> **TIP** The aggregate functions in a StateDB Reader window are intended for use with data that comes from database queries. To perform aggregation on data that comes from another source, use an Aggregate window instead.

- If relevant, use the Parameter column to specify function parameters. Only the ESP_aCat and ESP_aLag functions have parameters.

8 Configure any other desired settings for the StateDB Reader window.

Working with Custom Windows

Overview of Custom Windows

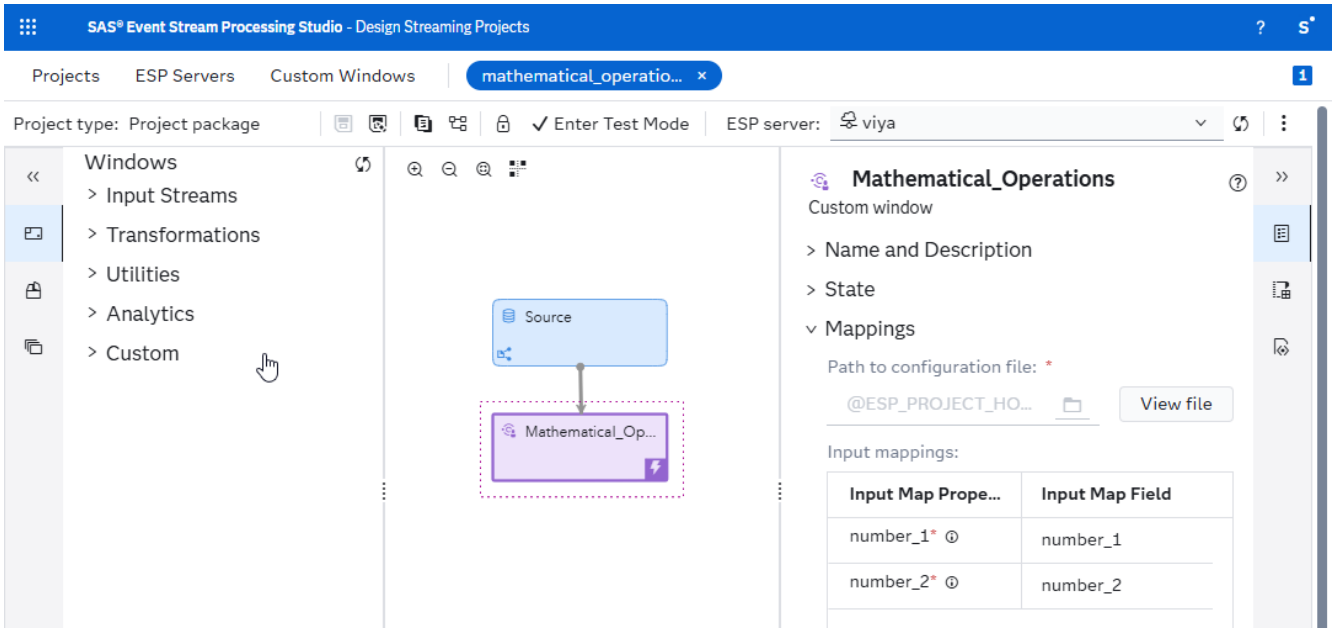
Custom windows enable a developer to define new window types that users can easily add to their projects. That is, a developer can create a window type that appears in the **Custom** section of the **Windows** pane, adjacent to the window types that are provided by SAS.

Custom windows enable reuse of code across projects. Creating a custom window involves a developer specifying the following configuration:

- the actions that are performed by the window on incoming and outgoing events
- the settings that users can configure in SAS Event Stream Processing Studio Modeler when they use the custom window

The code that specifies a custom window includes Python or Lua code, and JSON code. A developer creates a custom window by using a wizard. The wizard includes an option to upload a configuration file that the developer has created using a third-party code editor. The developer can also convert an existing Python window or a Lua window to a custom window.

Figure 36 *Example of Custom Window Settings Available to Users in the Right Pane*



For information about how a developer creates a custom window, see "Creating and Managing Custom Windows".

For information about how a user uses a custom window, see "Using a Custom Window".

Visit SAS Event Stream Processing Studio Custom Windows GitHub for a custom window example and to obtain custom windows that you can add to SAS Event Stream Processing Studio.

Creating and Managing Custom Windows

The following topics explain how to use the **Custom Windows** page. Creating and managing custom windows are tasks for developers rather than users.

Create a Custom Window

This topic provides general information about how to use the New Custom Window wizard.

> **TIP** For an example that demonstrates how to create a custom window that multiplies two numbers, see the Getting Started example in SAS Event Stream Processing Studio Custom Windows GitHub. You can also access this GitHub by clicking **View Custom Windows GitHub** on the **Custom Windows** page.

To create a custom window:

On the **Custom Windows** page, click  .

> **TIP** Rather than create a custom window from scratch, you can also convert an existing Lua window or a Python window to a custom window. For more information, see "Convert a Lua Window or a Python Window to a Custom Window" on page 154.

The New Custom Window window is a wizard with three pages.

1 Complete the **Initialization** page.

   a (Optional) Upload a configuration file that you created using your preferred code editor or that you obtained from SAS Event Stream Processing Studio Custom Windows GitHub.

   If you upload a configuration file, ensure that its format meets the requirements. For more information, see "Creating the Configuration File" in *SAS Event Stream Processing: Using Source and Derived Windows*.

   Uploading a configuration file populates the fields throughout the wizard. After uploading a configuration file, you can use the wizard to make further configuration changes to the custom window. These further changes are not saved to the original configuration file that you uploaded from your computer. However, when you finish using the New Custom Window Version window, you can export a configuration file that represents the current configuration of the custom windows. For more information, see "Export a Configuration File".

   b Enter a name for the custom window.

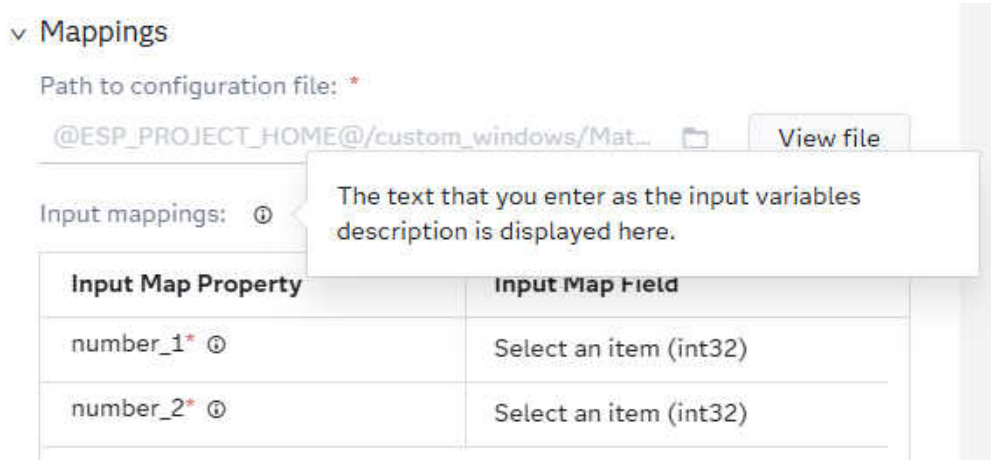
   c If required, enter a description, tags, and version notes for the custom window. This information is displayed to users when they hover over the custom window in the **Windows** pane and can help them select the appropriate window.

   d Click **Next**.

2 On the **Configuration** page, use the **Input Variables** tab to map input data to the custom window's `create` function.

   Note: When the **Enable variable mappings** check box is not selected, all fields from the parent window are passed in.

   - In the **Input variables description** field, enter a description that is displayed to the user of the custom window. Here is an example that shows where this description is displayed:



- In the Name column of the table, enter the name used to populate the object that is passed into the `create` function (when parameters are not expanded).
- In the Description column, enter a description of the field.
- In the Type column, if required, select the data type of the field that the user of the custom window is permitted to map to each input variable.
- If required, use the Optional column to mark the field as optional. When a field is optional, the user of the custom window is not required to enter a mapping entry for that field.

3 Use the **Output Variables** tab of the **Configuration** page to map output data from the `create` function to event fields in the output.

- In the **Output variables description** field, enter a description that is displayed to the user of the custom window.
- In the table, enter a name and a description for each field.
- In the Type column, if required, select the data type of the field that the user of the custom window is permitted to map to each output variable.

4 Use the **Settings** tab of the **Configuration** page to enable or disable settings.

- In the **Settings description** field, enter a description of how you are using these settings. This description is intended for the attention of developers who maintain this custom window and is not displayed to the user of the custom window.
- The **Expand parameters (expand-parms)** setting determines whether the `create` function receives individual parameters for each input variable or an object that contains fields that map to the names of the input variables. You specify the `create` function on the **Code** page of the wizard.

  Note: When the **Expand parameters (expand-parms)** setting is selected, you must use the **Input Variables** tab to define input variables.

- The **Process the events in blocks (process-blocks)** setting determines whether the `create` function receives events in blocks that contain multiple events or a single event at a time.

  Note: The **Expand parameters (expand_parms)** setting and the **process_blocks** setting cannot both be enabled at the same time.

- The **Encode binary data (encode-binary)** setting determines whether binary data that is passed into the `create` function is Base64 encoded.

5 Use the **Initialization** tab of the **Configuration** page to add information about the objects that are passed into the `init` function. The custom window puts the values into a single data object, which is passed into the function. If you add items to the **Initialization** tab, then you must provide an `init` function on the **Code** page of the wizard.

Note: When the **Enable variable mappings** check box is not selected, the window sets fields in the output event to match the object that is returned by the Python or Lua code.

- In the **Initialization description** field, enter a description that is displayed to the user of the custom window.
- In the table, enter a name and description for each property.
- In the Input Type column, select whether the initialization values are displayed to the user of the custom window as a text field or in a drop-down list.
- The Values column is enabled when the Input Type column is set to **Dropdown**. Use the Values column to specify initialization values that the user of the custom window can choose from.
- In the Default Value column, specify an optional default value for each property. This value is used when the model does not provide a value in the `<plugin>` element. If no default value is specified and the `<plugin>` element does not contain an entry, the property is not sent to the `init` function.

6 Use the **Code** page to add Python or Lua code. For more information about the required `create` function and other code that you might enter, see "Working with Custom Windows" in *SAS Event Stream Processing: Using Source and Derived Windows*.

You can also specify Python packages or Lua modules that are required for the code to run. When a user adds the custom window to a project or opens a project that contains the custom window, if a suitable virtual environment is not already selected the user is prompted to select a virtual environment that matches the requirements.

Table 4 *Operators for Specifying Versions*

Operator	Description	Relevant Virtual Environments
`==`	Use the exact package version or module version .	This operator is relevant to Python virtual environments and Lua virtual environments.
`~=`	Use an appropriate minor version. For example, if you select `~=` in the Operator column and enter `1.2` in the Installed Version column, SAS Event Stream Processing could use version 1.2.1 in	This operator is relevant to Python virtual environments.

Operator	Description	Relevant Virtual Environments
	the virtual environment but could not use version 1.3.	
>=	Use the specified minimum version or a newer version.	This operator is relevant to Python virtual environments.
<	Use an older version than the specified version.	This operator is relevant to Python virtual environments.

7 Click **Create**.

The window type that you created is listed in the table on the **Custom Windows** page and is available in the **Windows** pane in SAS Event Stream Processing Studio Modeler.

Convert a Lua Window or a Python Window to a Custom Window

When a project is part of a project package, and the project contains a Lua window or a Python window, you can convert that window to a custom window:

1 Open the project and select the window that you want to convert.

2 In the right pane, click  .

3 Use the New Custom Window wizard to adjust the configuration of the custom window as required. For more information about using the wizard, see "Create a Custom Window" on page 150.

When you click **Create** to complete the wizard, the Lua window or Python window in your project changes to a custom window.

The new custom window also appears in the **Windows** pane. You can manage the custom window on the **Custom Windows** page.

Create a Version of a Custom Window

Creating versions of a custom window enables you to make updates to your custom window while retaining earlier versions.

Consider creating multiple versions of the same custom window rather than creating separate custom windows. Each custom window that you create appears in the **Windows** pane in SAS Event Stream Processing Studio Modeler. The presence of many custom windows in the **Windows** pane could be confusing to users.

Note: When a project contains multiple instances of the same custom window, all of those custom windows must have the same version.

To create a version:

1 On the **Custom Windows** page, select the relevant row in the table and click .

2 In the New Custom Window Version window, select an updated configuration file or adjust the other fields as required.

3 Click **New version**.

For information about how a user can obtain the updates that you made, see "Manage a Custom Window's Version".

Export a Configuration File

You can export a custom window's configuration file:

1 On the **Custom Windows** page, select the relevant row in the table.

2 Click ⋮ on the toolbar and select **Export configuration file**.

The configuration file that contains the custom window definition is exported to your computer.

Note: The location of the file that you exported might vary depending on your browser's configuration.

Import a Configuration File

You can import a previously exported configuration file or a configuration file that you have obtained from SAS Event Stream Processing Studio Custom Windows GitHub.

1 On the **Custom Windows** page, click ⋮ and select **Import custom window**.

The New Custom Window window appears.

2 On the **Initialization** tab, use the **Configuration file** field to select a configuration file.

Selecting a configuration file populates the fields throughout the wizard. For more information about these fields, see "Create a Custom Window".

3 Click **Create**.

The custom window that is defined in the configuration file is listed in the table on the **Custom Windows** page and is available in the **Windows** pane in SAS Event Stream Processing Studio Modeler.

Delete a Custom Window

On the Custom Windows page, select the relevant row in the table and click .

All versions of that custom window are deleted and users can no longer select the custom window in the Windows pane in SAS Event Stream Processing Studio Modeler.

Using a Custom Window

The following topics are intended for users who want to use custom windows that were created by developers.

Add a Custom Window to a Project

1 Open the project into which you want to add a custom window.

2 In the **Windows** pane, expand the **Custom** section.

3 Drag the desired custom window to the workspace.

4 The custom window might require specific Python packages or Lua modules to be available in a virtual environment. If a message prompts you to resolve a mismatch between required libraries and the selected virtual environment, click **Resolve** and use the Virtual Environments window to select a matching virtual environment. For more information about how a system administrator can create virtual environments, see "Specify Virtual Environments".

5 Connect a parent window to the custom window to provide input events and an input schema for the custom window.

6 In the right pane, in the **Custom** section, configure the following settings.

Note:

- If you have questions about these settings, contact the developer who created the custom window.
- The path to the custom window's configuration file is a read-only field.

a Configure the input mappings. That is, select fields in the input schema that you want to map to properties that are passed into the `create` function in the window's Python or Lua code.

> **TIP** In the Input Map Property column, a red asterisk indicates that the developer of this custom window has set this property to be a required property. When ⓘ is displayed in this column, hover over it to view information from the developer of the custom window about that property.

b Configure the output mappings. That is, specify how data that is returned from the `create` function is mapped to fields in the output schema.

> **TIP** In the Output Map Property column, a red asterisk indicates that the developer of this custom window has set this property to be a required property. When ⓘ is displayed in this column, hover over it to view information from the developer of the custom window about that property.

You might need to add more fields to the window's output schema so that you can map the outputs of Python or Lua code to the window.

c Configure the initialization mapping. That is, select the values that are passed into the `init` function in the window's Python or Lua code.

7 If required, use the **Error handling strategy when fatal errors are encountered** field to select an error handling strategy.

Lua or Python code can throw exceptions. Exceptions from the `init` function cause the project to fail to load. By default, exceptions from the `create` function, the `heartbeat` function, and snippets (when threaded) are handled based on project-level error handling settings. If required, you can select a different error handling strategy for your custom window, to override the project-level setting. You can choose to log an error and continue project execution, or to log a fatal error and stop the project. To check what the project-level setting is, click , expand **Advanced** in the right pane, and check which error handling strategy is selected.

8 Edit any other window properties in the right pane as required.

9 Save the project.

When you save the project, a read-only folder with the same name as the custom window is added to the **`custom_windows`** folder of the project package. The folder contains the configuration file of the custom window. If you rename the window that you added to the workspace (in the **Name** field in the right pane), the folder in the project package still reflects that original name of the custom window. If you delete the custom window from your project, the folder and the configuration file are deleted too.

Manage a Custom Window's Version

When a developer has created a version of a custom window and you drag that custom window from the **Windows** pane to your project's workspace, the latest version of the custom window is used. For more information, see "Create a Version of a Custom Window".

In addition, when you open a project that already contains that custom window on its workspace, a message is displayed informing you that a new version is available. To update the custom window's version:

1 Open the project that contains the custom window and click **Version manager**.

2 Use the Custom Window Version Manager window to select the new version for any affected custom windows.

3 If you do not want to keep the existing input, output, and initialization mappings, clear the check box in the Keep Mappings column for each custom window whose mappings you do not want to keep. By default, this check box is selected.

4 Click **Update**.

You can access the Custom Window Version Manager window at any time by opening your project, clicking ⋮ on the toolbar, and selecting **Manage custom window versions**. For example, if you want to revert to an earlier version of a custom window, you can use the Custom Window Version Manager window to select an earlier version.

Note: When a project contains multiple instances of the same custom window, all of those custom windows must have the same version.

Managing Settings

Overview of Settings

Use the Settings window to edit user preferences or customize accessibility settings for all SAS web applications. For more information, see "Settings Window".

Use the **ESP Settings** page to adjust how certain features of SAS Event Stream Processing Studio and SAS Event Stream Manager work. For more information, see "ESP Settings Page".

Settings Window

Use the Settings window to edit user preferences or customize accessibility settings for all SAS web applications. Changing these settings does not impact other users.

To access these settings, click the user button on the application bar, and select **Settings**.

General

The **General** section includes settings that enable users to change the appearance of the web applications, 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 applications 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.

Select a theme from the drop-down list to change the look of the applications. The theme change takes effect immediately.

SAS themes:

High Contrast
Presents a black 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.

  Note: This setting relates to messages from other SAS products, rather than messages from SAS Event Stream Processing Studio or SAS Event Stream Manager.

- 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.

  Note: When standalone SAS Event Stream Processing is deployed, you cannot select a profile picture.

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

  To remove an existing profile picture, click the down arrow and select **Remove image**.

Region and Language

The **Region and Language** section enables users to specify the locale for regional formats and sorting.

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. For example, you might use this setting to display the user interface in English while using a different locale, such as Japanese, to sort and format dates, times, numbers, and currency.

Note: Select the *Browser locale* option only if your browser's locale is one of the supported locales listed in the **Locale for regional formats and sorting** setting. This avoids errors when sorting or formatting dates, times, numbers, and currency.

TIP Changes take effect after you sign out and sign back in.

Notifications

The **Notifications** section enables users to adjust how notifications from other SAS products are displayed.

Select **Show a pop-up message** to display a pop-up message when a notification arrives from another SAS product.

Note: When standalone SAS Event Stream Processing is deployed, the **Notifications** section is not available.

Accessibility

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

- 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** and adjusting its color, thickness, and opacity.

ESP Settings Page

The **ESP Settings** page enables you to complete tasks such as adjusting how certain features of SAS Event Stream Processing Studio and SAS Event Stream Manager work.

To access the **ESP Settings** page, click the user button on the application bar, and select **Settings**.

The content that is available on the **ESP Settings** page in SAS Event Stream Processing Studio is different from the content that is available on the **ESP Settings** page in SAS Event Stream Manager. The content in each application relates to tasks that you carry out in these applications.

The content that is available to you on the **ESP Settings** page depends on whether you are a member of the administrator group. By default, the administrator group is SASAdministrators.

Global Settings

Overview of Global Settings

The **Global** section of the **ESP Settings** page enables you to modify event stream processing settings that affect all users of the application.

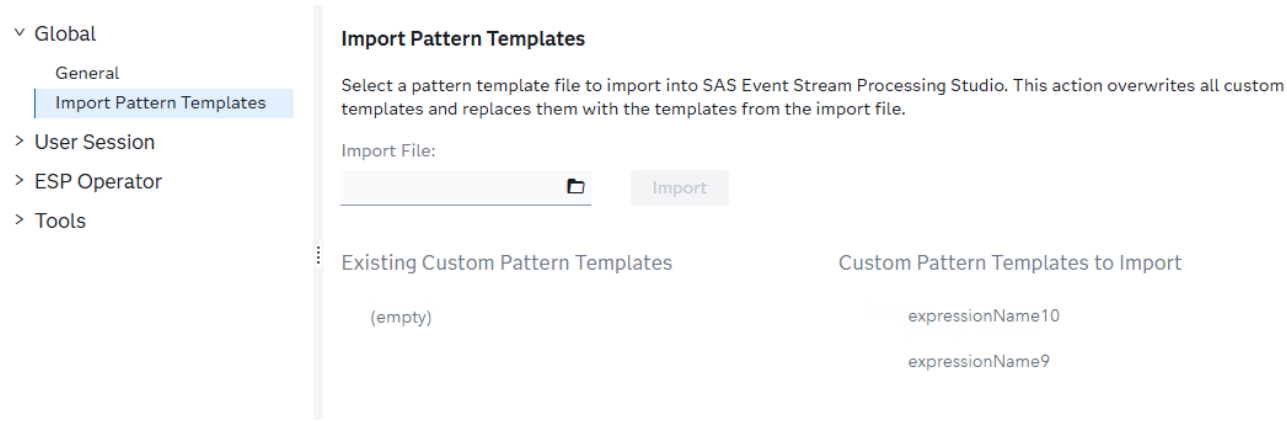
Import Pattern Templates into SAS Event Stream Processing Studio

You can import custom pattern templates into SAS Event Stream Processing Studio. Pattern templates specify a text string that represents the logical operator expression to combine events of interest (EOIs). The enclosed text specifies an expression that defines the pattern.

The pattern templates that you import must be located in an import file. Importing a set of custom pattern templates overwrites and replaces any existing custom pattern templates that were previously imported.

The following figure shows an example of a pattern template file that contains two custom pattern templates:

Figure 37 *Importing Pattern Templates*



Note: If you import an empty pattern template, all previously imported patterns are overwritten with the new, empty pattern template.

1 On the **ESP Settings** page, in the **Global** section, select **Import Pattern Templates**.

   The right pane displays the **Import Pattern Templates** page.

2 In the **Import File** field, click 🗀.

3 Navigate to the file that contains the pattern templates that you want to import and click **Open**.

The **Import Pattern Templates** page refreshes to show a list of the custom pattern templates present in the file. It also shows any existing custom pattern templates that you previously imported.

The following code provides an example of a pattern template:

```
<pattern-templates>
    <pattens-template name= "expression1">
        <pattern-expression><![CDATA[ and(fby{5 seconds}(A, B, C, D),
notoccur(K))]]></pattern-expression>
    </pattens-template>
    <pattens-template name= "expression2">
        <pattern-expression><![CDATA[and(fby{5 minutes}(A, B, C, D), notoccur(L))]]></
pattern-expression>
    </pattens-template>
</pattern-templates>
```

4 To clear your file selection, click .

5 Hover the pointer over a custom pattern template that you want to import.

Note: The pattern template's logical operator expression appears in a message box.

6 Click **Import**.

A warning message appears, informing you that this action overwrites all existing custom pattern templates and replaces them with pattern templates from the file.

7 To continue, click **Yes**.

A message appears, informing you whether your import is successful. The pattern templates that you imported are now available to use when defining a Pattern window's event logic.

Restart the Client-Configuration Server

Under certain circumstances, you might need to restart the client-configuration server. For more information, see "Understanding and Managing the Client-Configuration Server" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

IMPORTANT Restarting the client-configuration server affects all users of the application.

1 On the **ESP Settings** page, in the **Global** section, select **Global Settings for ESP Servers**.

Note: Only members of the administrator group can access the **Global Settings for ESP Servers** page. By default, the administrator group is SASAdministrators.

The right pane displays the **Global Settings for ESP Servers** page.

2 Click .

The **Status** field displays the following message: `Restarting the client-configuration server.`

The restart is complete when the **Status** field displays the following message: `The client-configuration server is running.`

Include or Exclude Connectors

You can use the **Connectors** tab on the **Global Settings for ESP Servers** page to modify ESP server properties for connectors. These ESP server properties control which connectors are available to users of SAS Event Stream Processing Studio in the Connector Configuration window.

Note: Only members of the administrator group can access the **Global Settings for ESP Servers** page. By default, the administrator group is SASAdministrators.

TIP To access the Connector Configuration window:

1. Open a project in SAS Event Stream Processing Studio. Select a Source window or add a new Source window.
2. In the right pane, expand **Input Data (Publisher) Connectors or Subscriber Connectors**.
3. Click . The **Connector Configuration** window appears. The **Connector type** field displays available connector types.

For information about alternative ways to control which connectors are displayed, see "Excluding Connectors" in *SAS Event Stream Processing: Connectors and Adapters*.

IMPORTANT Any changes to the **Global Settings for ESP Servers** page affect all users of the application. To change settings for the user session instead, on the **ESP Settings** page, in the **User Session** section, select **ESP Server Properties**.

The process for including or excluding connectors includes an automatic restart of the client-configuration server.

The **Connectors** tab enables you to change ESP server properties for connectors, but not other ESP server properties. To change other ESP server properties for all users of the application, select the **Properties** tab.

To include or exclude connectors:

1. On the **ESP Settings** page, in the **Global** section, select **Global Settings for ESP Servers**.

   The right pane displays the **Global Settings for ESP Servers** page.
2. Select the **Connectors** tab.
3. Select check boxes for the connectors that you want to be available in the Connector Configuration window, and clear check boxes for connectors that you do not want to be available.
4. Click .

   The Save Changes window appears.

5 Click **Save**.

The client-configuration server is restarted. Your changes are complete when the **Status** field displays the following message: `The client-configuration server is running.`

You can also reset all ESP server properties for connectors to their defaults.

1 Click .

The Reset ESP Server Properties for Connectors window appears.

2 Click **Reset**.

The client-configuration server is automatically restarted.

Specify Environment Variables and ESP Server Properties for All Users

You can use the **Environment Variables** tab and the **Properties** tab on the **Global Settings for ESP Servers** page to add, edit, or delete environment variables and ESP server properties for all users. The changes take effect immediately and all new ESP servers that are created in the Kubernetes cluster use your latest settings. If you want your changes to be applied to the client-configuration server, you must restart the client-configuration server.

Note: Only members of the administrator group can access the **Global Settings for ESP Servers** page. By default, the administrator group is SASAdministrators.

IMPORTANT Any changes to the **Global Settings for ESP Servers** page affect all users of the application. To change settings for the user session instead, on the **ESP Settings** page, in the **User Session** section, select **Environment Variables** or **ESP Server Properties**.

To specify environment variables and ESP server properties for all users:

1 On the **ESP Settings** page, in the **Global** section, select **Global Settings for ESP Servers**.

The right pane displays the **Global Settings for ESP Servers** page.

2 Select the **Environment Variables** tab or the **Properties** tab.

3 In the table, specify settings as required:

- To add a setting, click . A window appears, prompting you to specify the name of the setting and the value of the setting.
- To edit a setting, click the setting's **Value** field and specify a new value. Then click . A window appears, prompting you to confirm that you want to save changes.
- To delete a setting, click the row that contains the setting and then click . A window appears, prompting you to confirm the deletion.

The changes take effect immediately and all new ESP servers that are created in the Kubernetes cluster use your latest settings.

4 If you want your changes to be applied to the client-configuration server, click .

The **Status** field displays the following message: `Restarting the client-configuration server.`

The restart is complete when the **Status** field displays the following message: `The client-configuration server is running.`

Specify Connections to Remote Git Servers

The **Remote Git Server Connections** page enables you to add, update, and delete connections to remote Git servers. After remote Git servers have been defined, you can synchronize project data from the Git repository that is internal to SAS Event Stream Processing to remote Git repositories. For more information, see "Overview of File-Based Versioning".

Note: Only members of the administrator group can access the **Remote Git Server Connections** page. By default, the administrator group is SASAdministrators.

CAUTION

Specifying a remote Git server connection is a task that must be performed by a system administrator. This task involves specifying a Git access token and specifying a topic to organize repositories. Because project data is stored on a remote Git server, you must ensure that backup processes are in place for the remote Git server. Not having backups in place could result in data loss. Use only SAS Event Stream Processing to interact with remote Git repositories. Interacting directly with a repository that stores a SAS Event Stream Processing project on your remote Git server could result in data loss.

Note: To use external services such as GitHub, GitLab, and Gitea with SAS Event Stream Processing, the Kubernetes cluster administrator must ensure that there is access from the cluster to the appropriate IP addresses.

To specify a connection to a remote Git server:

1 On the **ESP Settings** page, in the **Global** section, select **Remote Git Server Connections**.

The right pane displays the **Remote Git Server Connections** page.

2 Click .

The Add Remote Git Server Connection window appears.

3 Enter a name to describe the connection. This name will be displayed elsewhere in the user interface. For example, when a user creates a project and chooses to store it in a remote repository, the name that you enter in the Add Remote Git Server Connection window is displayed as one of the remote Git servers on which the user can store the project.

4 Select the server type:

- **GitHub**
- **GitHub Enterprise Cloud**: GitHub Enterprise that is hosted by GitHub on GitHub.com.
- **GitHub Enterprise Server**: GitHub Enterprise that is self-hosted.

- **GitLab**
- **Gitea**

5 If you selected GitLab, GitHub, or Gitea as the server type, specify who has visibility of the repositories that are created on this Git server.

- If you are using GitLab, you can set visibility to **Private**, **Internal**, or **Public**. For more information about the meaning of these settings in GitLab, see GitLab documentation.
- If you are using GitHub or Gitea, you can set visibility to **Private** or **Public**. For more information about the meaning of these settings in GitHub, see GitHub documentation.

6 Enter the server URL. If you selected GitHub or GitHub Enterprise Cloud as the server type, the URL is entered for you.

7 Enter your user name.

8 Enter the value of your access token instead of the name of the access token. For example, an access token for GitLab might have the following format: **`glpat-xxxxxxxxxxxxxxxxxxxx`**. (In this example, some characters have been changed to **`x`**.)

- If you are using GitLab, the access token must be a personal access token. The access token's scope must include `api`. For more information about creating access tokens in GitLab, see GitLab documentation.
- If you are using GitHub, the access token must be a classic access token. The access token's scope must include `repo`. For more information about creating access tokens in GitHub, see GitHub documentation.
- If you are using Gitea, the access token's scope must include `repo` and `user`. For more information about access tokens in Gitea, see Gitea documentation.

9 When the server type is Gitea, you can enable Git Large File Storage (LFS). For more information, see "File-Based Versioning and Git Large File Storage". To configure your remote Git server connection to use Git LFS:

a Contact your Gitea administrator and obtain the value of the `LFS_JWT_SECRET` field from the Gitea **`app.ini`** file. For more information about the `LFS_JWT_SECRET` field in Gitea, see Gitea documentation.

b Enter the value in the **Gitea LFS secret** field in the Add Remote Git Server Connection window.

> **IMPORTANT** If Gitea restarts, then the value of the `LFS_JWT_SECRET` field is regenerated. You must update the value in the **Gitea LFS secret** field.

10 If you selected GitLab, GitHub, or Gitea as the server type, you can enter the name of a topic. Here is an example: **`esp-project-package`**. Topics are labels that are used to organize repositories. Specifying a topic is optional.

- The topic that you specify here is used when a project is stored in a remote repository. That is, when a remote repository is created to store a project, the repository has the topic that you specify here.

- The topic that you specify here is also used when a project is imported from a remote repository. That is, when you import a project from a remote Git server, only those repositories that have the specified topic are listed as available repositories.
- For more information about topics in GitLab, see GitLab documentation. For more information about topics in GitHub, see GitHub documentation.

11 Click **Test connection** to check that your credentials work.

12 Click **OK**.

When an access token expires, if a new access token is created you can edit the connection to update the value of the access token: click and enter the new access token. You cannot edit other connection properties.

To delete a connection, click .

When projects are synchronized with repositories on remote Git servers, projects can be deleted from the persistent volume to free up storage space. For more information, see "Clean Up Projects from the Persistent Volume".

Specify Virtual Environments

The **Virtual Environments** page enables you to view, add, update, delete, and export virtual environments. A virtual environment specifies either a set of Python packages or a set of Lua modules. When a user creates a project and selects virtual environments that an administrator has created, the project runs with the Python packages or Lua modules specified in those virtual environments. For more information, see "Update Project Properties".

Note: Only members of the administrator group can access the **Virtual Environments** page. By default, the administrator group is SASAdministrators.

To view the details of a virtual environment, select it in the **Virtual environment** field. For example, select **Built-in environment** to check what is included in the default set of Python packages or Lua modules.

- The adjacent fields and the **Packages** tab or Modules tab are updated to show details of the selected environment.
- When the packages or modules in the virtual environment depend on other packages or modules, you can click the **View dependent packages** or View dependent modules link at the bottom of the page to view these packages or modules. When there are no dependent packages or modules, this link is not displayed.

To specify a virtual environment:

1 On the **ESP Settings** page, in the **Global** section, select **Virtual Environments**. The right pane displays the **Virtual Environments** page.

2 Click in the upper right corner of the page.

3 In the Create Virtual Environment window, enter a name to describe the virtual environment. This name is displayed when a user creates a project and selects a virtual environment.

You cannot give the same name to a Python virtual environment and a Lua virtual environment. For example, if you have created a Python virtual environment called `ComputerVision`, you then cannot create a Lua virtual environment with that name. You could, however, create environments with names such as `ComputerVisionPython` and `ComputerVisionLua`.

4 Enter a description for the virtual environment.

5 (Optional) Select a previously exported **`requirements.txt`** file or a modules.txt file. A **`requirements.txt`** file specifies the configuration for a Python virtual environment. A modules.txt file specifies the configuration for a Lua virtual environment. Selecting such a file is an alternative to manually selecting packages or modules on the **Virtual Environments** page.

6 Click **OK**.

7 On the **Packages** or Modules tab, click  to add a package or module.

8 In the Python Package column or the Lua Module column, enter the name of the package or module.

9 (Optional) Use the Operator column and the Version column to specify the package or module version. If you do not use these columns, SAS Event Stream Processing determines an appropriate package version or module version and includes it in your virtual environment. After the virtual environment has been created, the Installed Version column displays the package version or module version that was installed.

Table 5 *Operators for Specifying Versions*

Operator	Description	Relevant Virtual Environments
==	Use the exact package version or module version .	This operator is relevant to Python virtual environments and Lua virtual environments.
~=	Use an appropriate minor version. For example, if you select **~=** in the Operator column and enter **`1.2`** in the Installed Version column, SAS Event Stream Processing could use version 1.2.1 in the virtual environment but could not use version 1.3.	This operator is relevant to Python virtual environments.
>=	Use the specified minimum version or a newer version.	This operator is relevant to Python virtual environments.

Operator	Description	Relevant Virtual Environments
<	Use an older version than the specified version.	This operator is relevant to Python virtual environments.

10 (Optional) Add more packages or modules.

11 Click .

The **Status** field shows the progress of building the virtual environment.

If the process fails, click the **Log** tab to view error messages and make appropriate changes. Next, click on the toolbar to rebuild the virtual environment.

> **TIP** Here is a quick way to specify a virtual environment: open a project, click , expand **Virtual Environments**, and click . Use the Virtual Environments window to create a virtual environment. This way of creating a virtual environment can be helpful when you have imported a project package that includes a `requirements.txt` file or a `modules.txt` file, and there are no suitable existing virtual environments. However, the Virtual Environments window does not enable you to view the logs that are available in the **Log** tab on the **Virtual Environments** section of the **ESP Settings** page. The Virtual Environments window also does not enable you to update existing virtual environments.
>
> Only members of the administrator group use the Virtual Environments window to create a virtual environment. Other users can use this window only to select existing virtual environments. By default, the administrator group is SASAdministrators.

Here is an example of Python libraries that might be included in a virtual environment for image processing:

```
scikit-learn
scikit-image
scipy
mahotas
pillow
opencv-python
```

To update a virtual environment:

1 In the **Virtual environment** field, select the virtual environment that you want to update.

2 Make changes on the **Packages** or Modules tab.

3 Click .

The **Status** field shows the progress of updating the virtual environment.

To delete a virtual environment, click in the upper right corner of the page.

You can export the configuration of your Python virtual environment into a `requirements.txt` file and you can export the configuration of your Lua virtual environment into a modules.txt file.

Exporting the virtual environment enables you to create the same virtual environment again. For example, you might want to add the same virtual environment to another deployment of SAS Event Stream Processing.

To export a virtual environment:

1 In the **Virtual environment** field, select the virtual environment that you want to export.

2 Click ⋮ and then click **Export configuration**.

A `requirements.txt` file or a modules.txt file is exported to your computer.

Note: The location of the file that you exported might vary depending on your browser's configuration.

To import a previously exported `requirements.txt` file or a modules.txt file into another virtual environment, click ⇲ on the **Packages** or Modules tab.

- Packages or modules specified in the file are added to the virtual environment.
- When the file specifies packages that were already included in the virtual environment, the details from the file overwrite the existing details. For example, if a virtual environment contains the Python package numpy 1.26.3 and you import a `requirements.txt` file that specifies numpy 1.26.4, then numpy 1.26.4 is used.

Settings for the User Session

Overview of Settings for the User Session

The **User Session** section of the **ESP Settings** page enables you to change event stream processing settings for your user session.

You can add, edit, reset, and delete the value of specific environment variables and specific ESP server properties (including logging properties) that are used when an ESP server is created in a Kubernetes cluster. An ESP server is created when you deploy a project to a Kubernetes cluster in SAS Event Stream Processing Studio or SAS Event Stream Manager, or test a project in SAS Event Stream Processing Studio.

IMPORTANT Any changes to these settings affect the current user only. To change settings for all users of the application, on the **ESP Settings** page, in the **Global** section, select **Global Environment Variables** or **Global ESP Server Properties**.

The changes that you make in the **User Session** sections on the **ESP Settings** pages of SAS Event Stream Processing Studio and SAS Event Stream Manager are independent of each other. To use the same settings in both applications, you must configure the settings in both applications.

Specify Environment Variables and ESP Server Properties for the User Session

To adjust environment variables and ESP server properties (including logging properties):

1 On the **ESP Settings** page, in the **User Session** section, select **Environment Variables** or **ESP Server Properties**.

The right pane displays the **Environment Variables** page or the **ESP Server Properties** page.

2 In the table, specify settings as required:

- To add a setting, click  . The Add Environment Variable or Add ESP Server Property window appears. Use this window to specify the name of the setting and the value of the setting.
- To edit a setting, click the setting's **Value** field and specify a new value. Then click  . The Save Changes window appears, prompting you to confirm that you want to save changes.
- To delete a setting, click the row that contains the setting and then click  . The Delete Environment Variable or Delete ESP Server Property window appears, prompting you to confirm the deletion.
- To reset all settings to default values, click  . The Reset Environment Variables or Reset ESP Server Properties window appears, prompting you to confirm the resetting.

Here is additional information about some properties that are present on the **ESP Server Properties** page by default:

- `server.disableTrace` – When the value of this setting is false, you can enable trace-level logging for specific windows in a project. Open a project in SAS Event Stream Processing Studio and click  to view settings for the continuous query. In the **Debugging** section, you can select the windows for which you want to enable trace-level logging.
- `server.maxws-queue` – Use this property to specify the maximum number of megabytes that the ESP server keeps in a WebSocket event queue before it starts to drop old events. You do not normally need to alter the value of this property. However, if you see warnings in the ESP server log about events being discarded, consider increasing the value of this property or slowing the input rate of events. For more information, see "Property to Set the Size of Event Queues" in *SAS Event Stream Processing: Using SAS Event Stream Processing in a Kubernetes Environment*.

Specify Whether to Run Projects as the Current User

When standalone SAS Event Stream Processing is deployed, projects always run in the Kubernetes cluster as the system user. You cannot change this.

When SAS Event Stream Processing is deployed with other SAS products, in some circumstances it is possible to specify how projects run in the Kubernetes cluster:

- When a project is part of a project package, the project always runs in the Kubernetes cluster as the system user.
- By default, project XML files run in the Kubernetes cluster with the access rights and permissions of the current user. You can adjust this setting to select which group's permissions are used for

the current user or to run project XML files as the system user instead. For more information about situations in which you might want to run projects as the system user, see "Understanding File Access" in *SAS Event Stream Processing: Troubleshooting*.

To specify how project XML files should run in the Kubernetes cluster:

1 On the **ESP Settings** page, in the **User Session** section, select **ESP Server Properties**.

The right pane displays the **ESP Server Properties** page.

2 Select the desired setting:

- **Run project XML files as the system user**. By default, the system user is the sas user account.
- **Run project XML files as the current user**. The group whose permissions are used is the group ID (GID) from the current user's LDAP record. When several groups are available, you can select the **Override group** check box, and then select a different group.

Note: These settings affect your user session only. They do not apply globally.

3 Click .

ESP Operator

Overview

The **ESP Settings** page provides an alternative way to access the ESP operator log. This can be useful if a pod does not start and you cannot access log messages in test mode in SAS Event Stream Processing Studio or on the page for a specific deployment in SAS Event Stream Manager.

Access the ESP Operator Log

On the **ESP Settings** page, in the **ESP Operator** section, select **ESP Operator Log**.

The right pane displays messages from the ESP operator log.

Filter Log Messages

To filter log messages by message type, select one or more of the following options:

- All – Shows all log message types.
- Informational – Shows general log messages that are not warnings or errors.
- Warnings – Shows only warning messages.
- Fatal and Error – Shows only error messages, including fatal errors and normal errors.

Clear the ESP Operator Log Page

To clear the contents of the **ESP Operator Log** page, click Clear Log.

Export the ESP Operator Log

You can export the ESP operator log to a text file. Click Export All Logs. The log is exported to your computer. The location of the log that you exported might vary depending on your browser's configuration.

Search within Log Messages

Use the search field to search the log messages for a specific term. The search field works in conjunction with the log filters. For example, if you click **Warning** and then search for a specific term, the search finds occurrences of that term within warning messages only. The search is not case sensitive.

Tools

Overview of Tools

The **Tools** section of the **ESP Settings** page enables you to encrypt a password in an adapter, clean up projects from the persistent volume, and convert project XML files to use file-based versioning.

Encrypt a Password in an Adapter

Some adapters require you to enter a password in the connection's configuration details. SAS Event Stream Processing Studio and SAS Event Stream Manager enable you to encrypt passwords in adapters.

- To encrypt the password in a connector, use the Connector Configuration window in SAS Event Stream Processing Studio. For more information, see "Encrypt a Password in a Connector".
- To encrypt the password in an adapter, including the alternative transports for RabbitMQ and Solace, use the password encryption tool on the **ESP Settings** page in SAS Event Stream Processing Studio or SAS Event Stream Manager.

To use the password encryption tool on the **ESP Settings** page:

1. On the **ESP Settings** page, in the **Tools** section, select **Encrypt Password**.
2. Select the adapter.
3. Enter the user ID.

4 Enter the password that you want to encrypt and click **Encrypt password**.

5 Click **Copy** to copy the encrypted password to the clipboard.

6 Use the copied password when you connect to the data source.

For example, when you are using the MQTT adapter, use the encrypted password as the value for the `mqttpassword` key. Depending on the adapter, you might need to set other relevant keys too. For example, for the MQTT adapter, you also need to set the `mqttpasswordencrypted` key to `true`.

Here is another example. For the Database adapter, use the encrypted password to create the system data source name (DSN).

```
DSN=Oracle Wire Protocol;uid=exampleuserid;pwd=pkrrtO8hAlzcgkUUrO5zRhgZjePLR0HgqQYWtyLr5W
70yUjT3Wf1Eo72VGd4i6s3;hostname=example_DB_server_name.com;servicename=example.process_name"
```

For more information about the connection details for adapters, see "Adapters by Source Language" in *SAS Event Stream Processing: Connectors and Adapters*.

For more information about the connection details for the alternative transports for RabbitMQ and Solace, see "Using Alternative Transports for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.

Clean Up Projects from the Persistent Volume

When projects are synchronized with repositories on remote Git servers, projects can be deleted from the persistent volume to free up storage space. The projects remain on the remote Git servers and are restored to the persistent volume if the projects are reopened.

Note: Only members of the administrator group can access the **Clean Up Persistent Volume** page. By default, the administrator group is SASAdministrators.

To delete projects from the persistent volume:

1 On the **ESP Settings** page, in the **Tools** section, select **Clean Up Persistent Volume**.

Only projects that are synchronized with repositories on remote Git servers and that are not running are listed on this page.

2 Select the projects that you want to delete and click .

3 In the Clean Up Persistent Volume window, click **Yes**.

Convert Projects to File-Based Versioning in Bulk

The **Convert Projects to File-Based Versioning** page enables administrators to perform a bulk conversion of all project XML files to use file-based versioning. For more information, see "Publishing Project Versions by Using File-Based Versioning".

Note: Only members of the administrator group can access the **Convert Projects to File-Based Versioning** page. By default, the administrator group is SASAdministrators.

A project package is created for each project XML file. For more information, see "Project Package".

When SAS Event Stream Processing is deployed with other SAS products, the version history of project XML files that were previously published using SAS Viya platform services (that is, the Files service and the File Store service) is retained. It is also possible for ordinary users to manually convert projects, one at a time. For more information, see "Convert a Project to Use File-Based Versioning".

To convert projects in bulk:

1 On the **ESP Settings** page, in the **Tools** section, select **Convert Projects to File-Based Versioning**.

   In the Status column, the icon indicates projects that are ready to be converted.

2 Click **Convert** to start the conversion.

3 Monitor the status of the conversion process:

   - When a project's status changes to , its conversion is complete.

     TIP You can click to remove projects whose conversion is complete from the table, enabling you to focus on projects whose conversion is not complete.

   - When a project's status is shown as , hover over the status icon to view an error message, and then address the issue. For example, if a project could not be converted because the project is running in SAS Event Stream Processing Studio, you can stop the project and try converting projects again. You can convert a project that is running in SAS Event Stream Manager.

4 When SAS Event Stream Processing is deployed with other SAS products, the published versions of the original project XML files are still present in the Files service and the File Store service. Click to delete them.

Each converted project now has a project package. When a project contains references to files (for example, files that contain test data), consider adding those files to the project package and adjusting the references to point to the new location of the files. For more information, see "Manage a Project Package".

Security Considerations for Uploading and Importing Files

When you upload or import files to SAS Event Stream Processing Studio, you must ensure that the content of the files is not harmful. For example, you might import a project or upload a file to a project package. SAS Event Stream Processing Studio does not check uploaded or imported files for harmful content.

To prevent path manipulation, SAS Event Stream Processing Studio restricts the characters that file names of uploaded or imported files can contain. These restrictions also apply to the names of files in ZIP files.