Logbot ETL User Guide

Contents

Introduction

Logbot ETL is a dataflow system based on the concepts of flow-based programming. It supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic. Logbot ETL has a web-based user interface for design, control, feedback, and monitoring of dataflows. It is highly configurable along several dimensions of quality of service, such as loss-tolerant versus guaranteed delivery, low latency versus high throughput, and priority-based queuing. Logbot ETL provides fine-grained data provenance for all data received, forked, joined cloned, modified, sent, and ultimately dropped upon reaching its configured end-state.

Terminology

DataFlow Manager: A DataFlow Manager (DFM) is a Logbot ETL user who has permissions to add, remove, and modify components of a Logbot ETL dataflow.

FlowFile: The FlowFile represents a single piece of data in Logbot ETL. A FlowFile is made up of two components: FlowFile Attributes and FlowFile Content. Content is the data that is represented by the FlowFile. Attributes are characteristics that provide information or context about the data; they are made up of key-value pairs. All FlowFiles have the following Standard Attributes:

- **uuid**: A Universally Unique Identifier that distinguishes the FlowFile from other FlowFiles in the system.
- **filename**: A human-readable filename that may be used when storing the data to disk or in an external service
- **path**: A hierarchically structured value that can be used when storing data to disk or an external service so that the data is not stored in a single directory

Processor: The Processor is the Logbot ETL component that is used to listen for incoming data; pull data from external sources; publish data to external sources; and route, transform, or extract information from FlowFiles.

Relationship: Each Processor has zero or more Relationships defined for it. These Relationships are named to indicate the result of processing a FlowFile. After a Processor has finished processing a FlowFile, it will route (or “transfer”) the FlowFile to one of the Relationships. A DFM is then able to connect each of these Relationships to other components in order to specify where the FlowFile should go next under each potential processing result.

Connection: A DFM creates an automated dataflow by dragging components from the Components part of the Logbot ETL toolbar to the canvas and then connecting the components together via Connections. Each connection consists of one or more Relationships. For each Connection that is drawn, a DFM can determine which Relationships should be used for the Connection. This allows data to be routed in different ways based on its processing outcome. Each connection houses a FlowFile Queue. When a FlowFile is transferred to a particular Relationship, it is added to the queue belonging to the associated Connection.

Controller Service: Controller Services are extension points that, after being added and configured by a DFM in the User Interface, will start up when Logbot ETL starts up and provide information for use by other components (such as processors or other controller services). A common Controller Service used by several components is the StandardSSLContextService. It provides the ability to configure keystore and/or truststore properties once and reuse that configuration throughout the application. The idea is that, rather than configure this information in every processor that might need it, the controller service provides it for any processor to use as needed.

Reporting Task: Reporting Tasks run in the background to provide statistical reports about what is happening in the Logbot ETL instance. The DFM adds and configures Reporting Tasks in the User Interface as desired. Common reporting tasks include the ControllerStatusReportingTask, MonitorDiskUsage reporting task, MonitorMemory reporting task, and the StandardGangliaReporter.

Funnel: A funnel is a Logbot ETL component that is used to combine the data from several Connections into a single Connection.

Process Group: When a dataflow becomes complex, it often is beneficial to reason about the dataflow at a higher, more abstract level. Logbot ETL allows multiple components, such as Processors, to be grouped together into a Process Group. The Logbot ETL User Interface then makes it easy for a DFM to connect together multiple Process Groups into a logical dataflow, as well as allowing the DFM to enter a Process Group in order to see and manipulate the components within the Process Group.

Port: Dataflows that are constructed using one or more Process Groups need a way to connect a Process Group to other dataflow components. This is achieved by using Ports. A DFM can add any number of Input Ports and Output Ports to a Process Group and name these ports appropriately.

Remote Process Group: Just as data is transferred into and out of a Process Group, it is sometimes necessary to transfer data from one instance of Logbot ETL to another. While Logbot ETL provides many different mechanisms for transferring data from one system to another, Remote Process Groups are often the easiest way to accomplish this if transferring data to another instance of Logbot ETL.

Bulletin: The Logbot ETL User Interface provides a sigLogbot ETLcant amount of monitoring and feedback about the current status of the application. In addition to rolling statistics and the current status provided for each component, components are able to report Bulletins. Whenever a component reports a Bulletin, a bulletin icon is displayed on that component. System-level bulletins are displayed on the Status bar near the top of the page. Using the mouse to hover over that icon will provide a tool-tip that shows the time and severity (Debug, Info, Warning, Error) of the Bulletin, as well as the message of the Bulletin. Bulletins from all components can also be viewed and filtered in the Bulletin Board Page, available in the Global Menu.

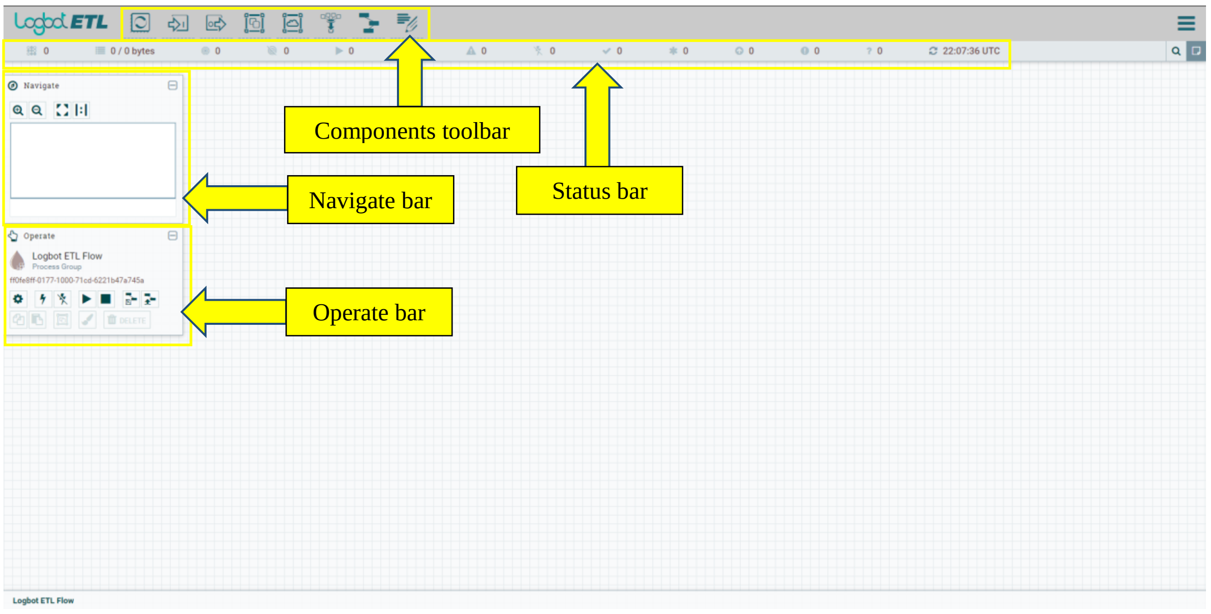
Template: Often times, a dataflow is comprised of many sub-flows that could be reused. Logbot ETL allows DFMs to select a part of the dataflow (or the entire dataflow) and create a Template. This Template is given a name and can then be dragged onto the canvas just like the other components. As a result, several components may be combined together to make a larger building block from which to create a dataflow. These templates can also be exported as XML and imported into another Logbot ETL instance, allowing these building blocks to be shared.

Logbot ETL User Interface

The Logbot ETL UI provides mechanisms for creating automated dataflows, as well as visualizing, editing, monitoring, and administering those dataflows. The UI can be broken down into several segments, each responsible for different functionality of the application. This section provides screenshots of the application and highlights the different segments of the UI. Each segment is discussed in further detail later in the document.

When the application is started, the user is able to navigate to the UI by going to the default address of `http://<hostname>:8080/` in a web browser. There are no permissions configured by default, so anyone is able to view and modify the dataflow. For information on securing the system, see the System Administrator's Guide.

When a DFM navigates to the UI for the first time, a blank canvas is provided on which a dataflow can be built:

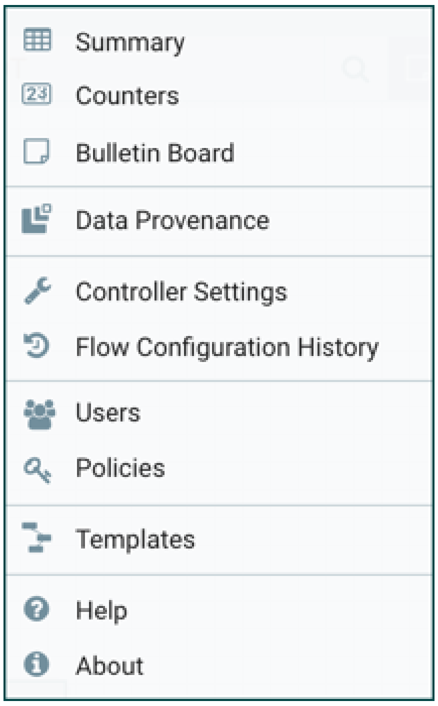


The Components Toolbar runs across the top left portion of your screen. It consists of the components you can drag onto the canvas to build your dataflow. Each component is described in more detail in Building a DataFlow.

The Status Bar is under the Components Toolbar. The Status bar provides information about the number of threads that are currently active in the flow, the amount of data that currently exists in the flow, how many Remote Process Groups exist on the canvas in each state (Transmitting, Not Transmitting), how many Processors exist on the canvas in each state (Stopped, Running, Invalid, Disabled), how many versioned Process Groups exist on the canvas in each state (Up to date, Locally modified, Stale, Locally modified and stale, Sync failure) and the timestamp at which all of this information was last refreshed. Additionally, if the instance of Logbot ETL is clustered, the Status bar shows how many nodes are in the cluster and how many are currently connected.

The Operate Palette sits to the left-hand side of the screen. It consists of buttons that are used by DFMs to manage the flow, as well as by administrators who manage user access and configure system properties, such as how many system resources should be provided to the application.

On the right side of the canvas is Search, and the Global Menu. For more information on search refer to Search Components in DataFlow. The Global Menu contains options that allow you to manipulate existing components on the canvas:



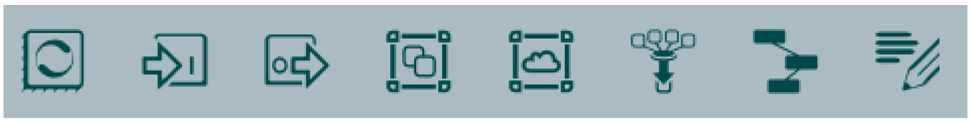
Additionally, the UI has some features that allow you to easily navigate around the canvas. You can use the Navigate Palette to pan around the canvas, and to zoom in and out. The "Birds Eye View" of the dataflow provides a high-level view of the dataflow and allows you to pan across large portions of the dataflow. You can also find breadcrumbs along the bottom of the screen. As you navigate into and out of Process Groups, the breadcrumbs show the depth in the flow, and each Process Group that you entered to reach this depth. Each of the Process Groups listed in the breadcrumbs is a link that will take you back up to that level in the flow.

Building a DataFlow

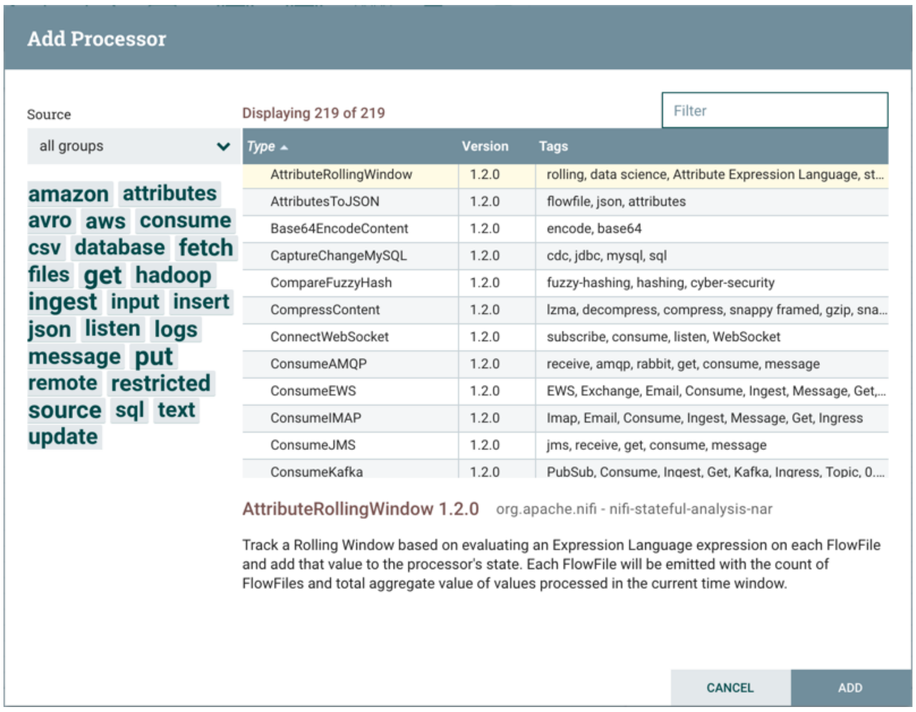
A DFM is able to build an automated dataflow using the Logbot ETL UI. Simply drag components from the toolbar to the canvas, configure the components to meet specific needs, and connect the components together.

Adding Components to the Canvas

The User Interface section above outlined the different segments of the UI and pointed out a Components Toolbar. This section looks at each of the Components in that toolbar:



Processor: The Processor is the most commonly used component, as it is responsible for data ingress, egress, routing, and manipulating. There are many different types of Processors. In fact, this is a very common Extension Point in Logbot ETL, meaning that many vendors may implement their own Processors to perform whatever functions are necessary for their use case. When a Processor is dragged onto the canvas, the user is presented with a dialog to choose which type of Processor to use:


Add Processor
Source
all groups
Filter
amazon attributes avro aws consume csv database fetch files get hadoop ingest input insert json listen logs message put remote restricted source sql text update
CANCEL
ADD

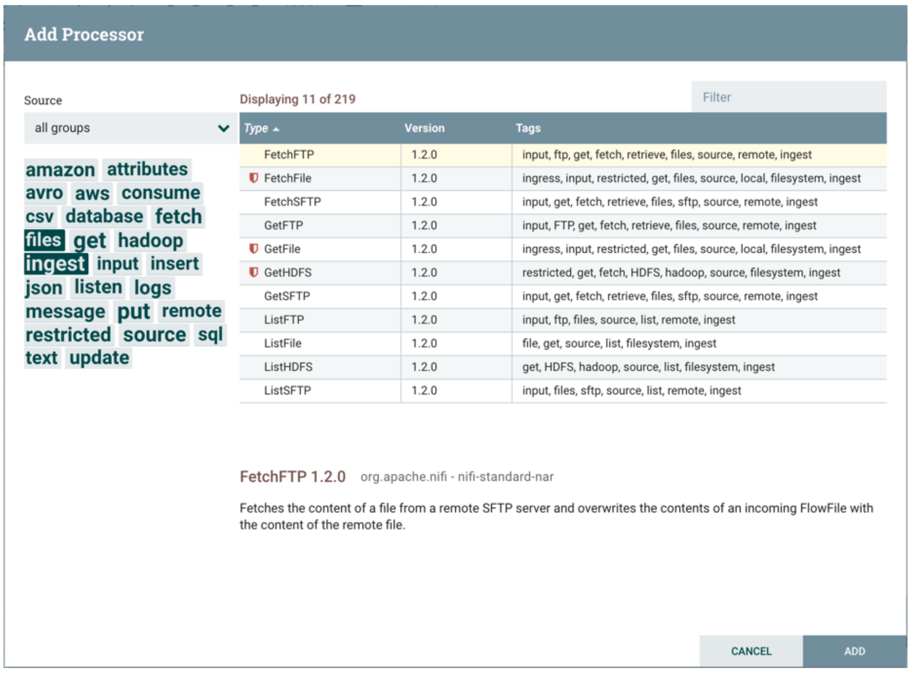
Displaying 219 of 219

Type	Version	Tags
AttributeRollingWindow	1.2.0	rolling, data science, Attribute Expression Language, st...
AttributesToJSON	1.2.0	flowfile, json, attributes
Base64EncodeContent	1.2.0	encode, base64
CaptureChangeMySQL	1.2.0	cdc, jdbc, mysql, sql
CompareFuzzyHash	1.2.0	fuzzy-hashing, hashing, cyber-security
CompressContent	1.2.0	lzma, decompress, compress, snappy framed, gzip, sna...
ConnectWebSocket	1.2.0	subscribe, consume, listen, WebSocket
ConsumeAMQP	1.2.0	receive, amqp, rabbit, get, consume, message
ConsumeEWS	1.2.0	EWS, Exchange, Email, Consume, Ingest, Message, Get,...
ConsumeIMAP	1.2.0	Imap, Email, Consume, Ingest, Message, Get, Ingress
ConsumeJMS	1.2.0	jms, receive, get, consume, message
ConsumeKafka	1.2.0	PubSub, Consume, Ingest, Get, Kafka, Ingress, Topic, 0....

AttributeRollingWindow 1.2.0 org.apache.nifi - nifi-stateful-analysis-nar

Track a Rolling Window based on evaluating an Expression Language expression on each FlowFile and add that value to the processor's state. Each FlowFile will be emitted with the count of FlowFiles and total aggregate value of values processed in the current time window.

In the top-right corner, the user is able to filter the list based on the Processor Type or the Tags associated with a Processor. Processor developers have the ability to add Tags to their Processors. These tags are used in this dialog for filtering and are displayed on the left-hand side in a Tag Cloud. The more Processors that exist with a particular Tag, the larger the Tag appears in the Tag Cloud. Clicking a Tag in the Cloud will filter the available Processors to only those that contain that Tag. If multiple Tags are selected, only those Processors that contain all of those Tags are shown. For example, if we want to show only those Processors that allow us to ingest files, we can select both the `files` Tag and the `ingest` Tag:


Add Processor
Source
all groups
amazon attributes avro aws consume csv database fetch files get hadoop ingest input insert json listen logs message put remote restricted source sql text update
Displaying 11 of 219
Filter
CANCEL
ADD

Type	Version	Tags
FetchFTP	1.2.0	input, ftp, get, fetch, retrieve, files, source, remote, ingest
FetchFile	1.2.0	ingress, input, restricted, get, files, source, local, filesystem, ingest
FetchSFTP	1.2.0	input, get, fetch, retrieve, files, sftp, source, remote, ingest
GetFTP	1.2.0	input, FTP, get, fetch, retrieve, files, source, remote, ingest
GetFile	1.2.0	ingress, input, restricted, get, files, source, local, filesystem, ingest
GetHDFS	1.2.0	restricted, get, fetch, HDFS, hadoop, source, filesystem, ingest
GetSFTP	1.2.0	input, get, fetch, retrieve, files, sftp, source, remote, ingest
ListFTP	1.2.0	input, ftp, files, source, list, remote, ingest
ListFile	1.2.0	file, get, source, list, filesystem, ingest
ListHDFS	1.2.0	get, HDFS, hadoop, source, list, filesystem, ingest
ListSFTP	1.2.0	input, files, sftp, source, list, remote, ingest

FetchFTP 1.2.0 org.apache.nifi - nifi-standard-nar

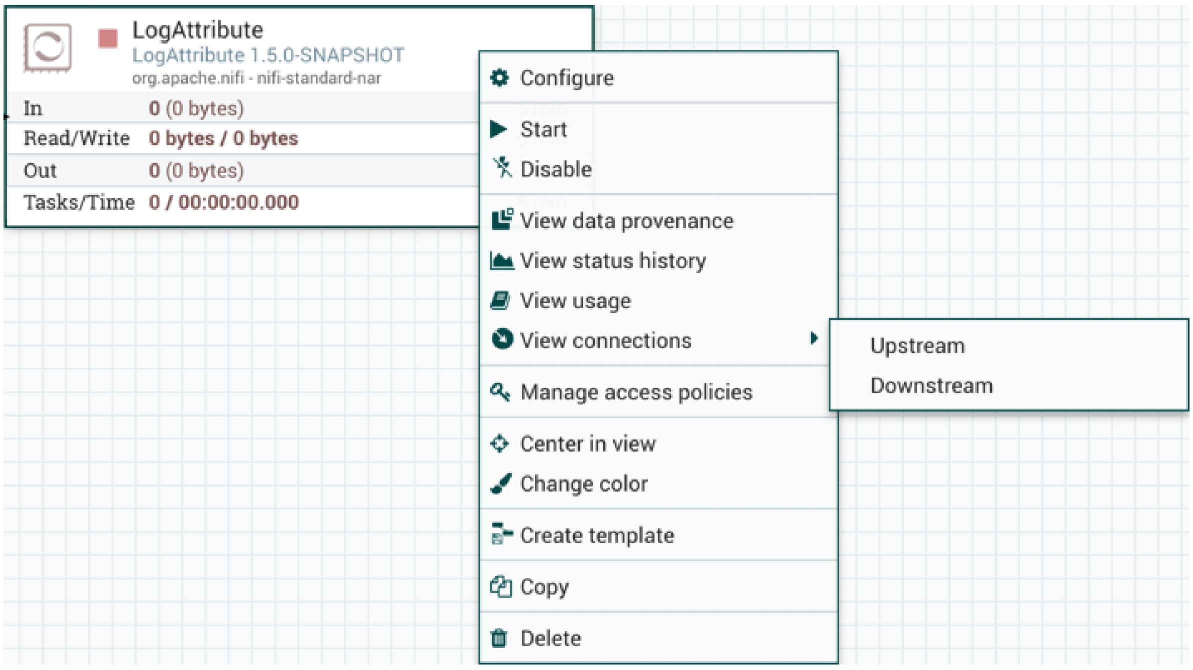
Fetches the content of a file from a remote SFTP server and overwrites the contents of an incoming FlowFile with the content of the remote file.

Restricted components will be marked with a icon next to their name. These are components that can be used to execute arbitrary unsanitized code provided by the operator through the Logbot ETL REST API/UI or can be used to obtain or alter data on the Logbot ETL host system using the Logbot ETL OS credentials. These components could be used by an otherwise authorized Logbot ETL user to go beyond the intended use of the application, escalate privilege, or could expose data about the internals of the Logbot ETL process or the host system. All of these capabilities should be considered privileged, and admins should be aware of these capabilities and explicitly enable them for a subset of trusted users. Before a user is allowed to create and modify restricted components they must be granted access. Hovering over the icon will display the specific permissions a restricted component requires. Permissions can be assigned regardless of restrictions. In this case, the user will have access to all restricted components. Alternatively, users can be assigned access to specific restrictions. If the user has been granted access to all restrictions a component requires, they will have access to that component assuming otherwise sufficient permissions.

Clicking the "Add" button or double-clicking on a Processor Type will add the selected Processor to the canvas at the location that it was dropped.

For any component added to the canvas, it is possible to select it with the mouse and move it anywhere on the canvas. Also, it is possible to select multiple items at once by either holding down the Shift key and selecting each item or by holding down the Shift key and dragging a selection box around the desired components.

Once you have dragged a Processor onto the canvas, you can interact with it by right-clicking on the Processor and selecting an option from the context menu. The options available to you from the context menu vary, depending on the privileges assigned to you.



While the options available from the context menu vary, the following options are typically available when you have full privileges to work with a Processor:

- **Configure**: This option allows the user to establish or change the configuration of the Processor (see Configuring a Processor).

For Processors, Ports, Remote Process Groups, Connections and Labels, it is possible to open the configuration dialog by double-clicking on the desired component.

- **Start** or **Stop**: This option allows the user to start or stop a Processor; the option will be either Start or Stop, depending on the current state of the Processor.
- **Enable** or **Disable**: This option allows the user to enable or disable a Processor; the option will be either Enable or Disable, depending on the current state of the Processor.
- **View data provenance**: This option displays the Logbot ETL Data Provenance table, with information about data provenance events for the FlowFiles routed through that Processor (see Data Provenance).
- **View status history**: This option opens a graphical representation of the Processor’s statistical information over time.
- **View usage**: This option takes the user to the Processor’s usage documentation.
- **View connections→Upstream**: This option allows the user to see and "jump to" upstream connections that are coming into the Processor. This is particularly useful when processors connect into and out of other Process Groups.
- **View connections→Downstream**: This option allows the user to see and "jump to" downstream connections that are going out of the Processor. This is particularly useful when processors connect into and out of other Process Groups.
- **Center in view**: This option centers the view of the canvas on the given Processor.
- **Change color**: This option allows the user to change the color of the Processor, which can make the visual management of large flows easier.
- **Create template**: This option allows the user to create a template from the selected Processor.
- **Copy**: This option places a copy of the selected Processor on the clipboard, so that it may be pasted elsewhere on the canvas by right-clicking on the canvas and selecting Paste. The Copy/Paste actions also may be done using the keystrokes Ctrl-C (Command-C) and Ctrl-V (Command-V).

- **Delete**: This option allows the DFM to delete a Processor from the canvas.

Input Port: Input Ports provide a mechanism for transferring data into a Process Group. When an Input Port is dragged onto the canvas, the DFM is prompted to name the Port. All Ports within a Process Group must have unique names.

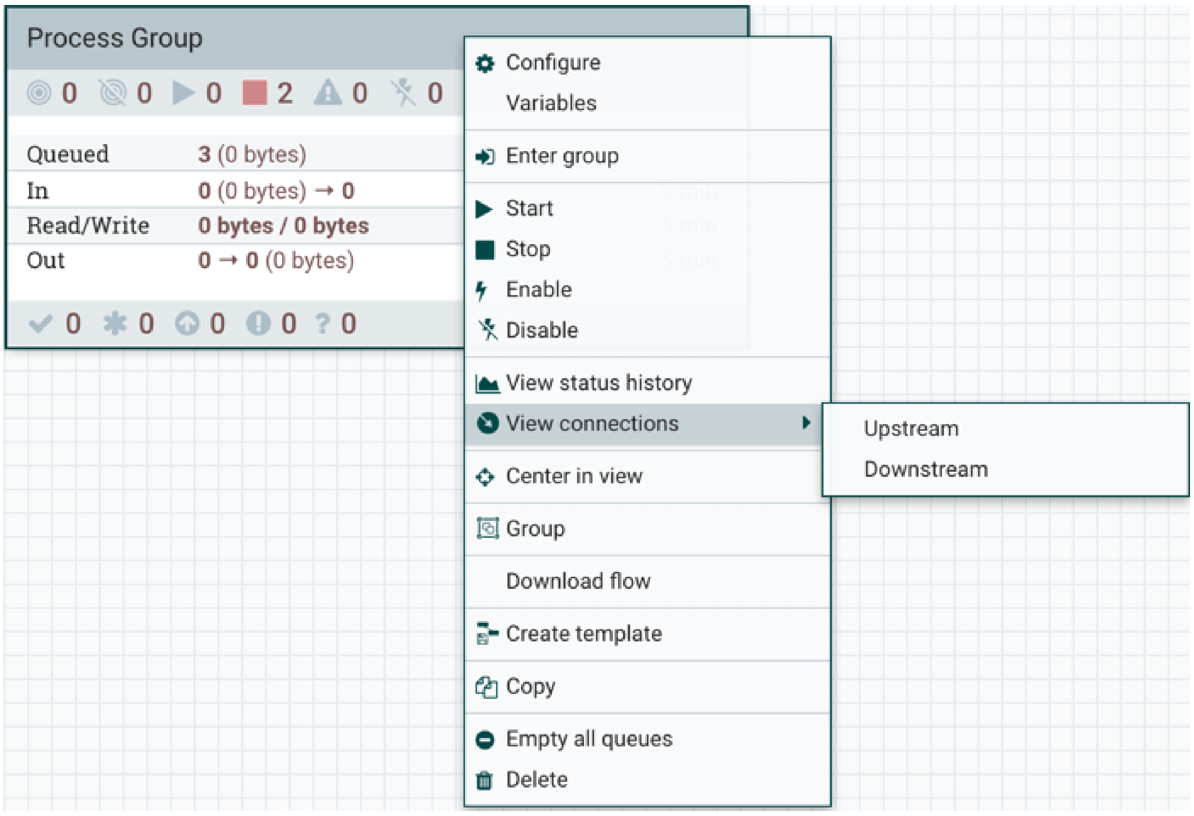
All components exist only within a Process Group. When a user initially navigates to the Logbot ETL page, the user is placed in the Root Process Group. If the Input Port is dragged onto the Root Process Group, the Input Port provides a mechanism to receive data from remote instances of Logbot ETL via Site-to-Site. In this case, the Input Port can be configured to restrict access to appropriate users, if Logbot ETL is configured to run securely. For information on configuring Logbot ETL to run securely, see the System Administrator’s Guide.

Output Port: Output Ports provide a mechanism for transferring data from a Process Group to destinations outside of the Process Group. When an Output Port is dragged onto the canvas, the DFM is prompted to name the Port. All Ports within a Process Group must have unique names.

If the Output Port is dragged onto the Root Process Group, the Output Port provides a mechanism for sending data to remote instances of Logbot ETL via Site-to-Site. In this case, the Port acts as a queue. As remote instances of Logbot ETL pull data from the port, that data is removed from the queues of the incoming Connections. If Logbot ETL is configured to run securely, the Output Port can be configured to restrict access to appropriate users. For information on configuring Logbot ETL to run securely, see the System Administrator’s Guide.

Process Group: Process Groups can be used to logically group a set of components so that the dataflow is easier to understand and maintain. When a Process Group is dragged onto the canvas, the DFM is prompted to name the Process Group. The Process Group will then be nested within that parent group.

Once you have dragged a Process Group onto the canvas, you can interact with it by right-clicking on the Process Group and selecting an option from the context menu. The options available to you from the context menu vary, depending on the privileges assigned to you.



While the options available from the context menu vary, the following options are typically available when you have full privileges to work with the Process Group:

- **Configure**: This option allows the user to establish or change the configuration of the Process Group.
- **Variables**: This option allows the user to create or configure variables within the Logbot ETL UI.
- **Enter group**: This option allows the user to enter the Process Group.

It is also possible to double-click on the Process Group to enter it.

- **Start**: This option allows the user to start a Process Group.
- **Stop**: This option allows the user to stop a Process Group.
- **Enable**: This option allows the user to enable all Processors in the Process Group.
- **Disable**: This option allows the user to disable all Processors in the Process Group.
- **View status history**: This option opens a graphical representation of the Process Group's statistical information over time.
- **View connections→Upstream**: This option allows the user to see and "jump to" upstream connections that are coming into the Process Group.
- **View connections→Downstream**: This option allows the user to see and "jump to" downstream connections that are going out of the Process Group.
- **Center in view**: This option centers the view of the canvas on the given Process Group.
- **Group**: This option allows the user to create a new Process Group that contains the selected Process Group and any other components selected on the canvas.
- **Download flow definition**: This option allows the user to download the flow definition of the process group as a JSON file. The file can be used as a backup or imported into a Logbot ETL Registry using the Logbot ETL CLI. (Note: If "Download flow definition" is selected for a versioned process group, there is no versioning information in the download. In other words,

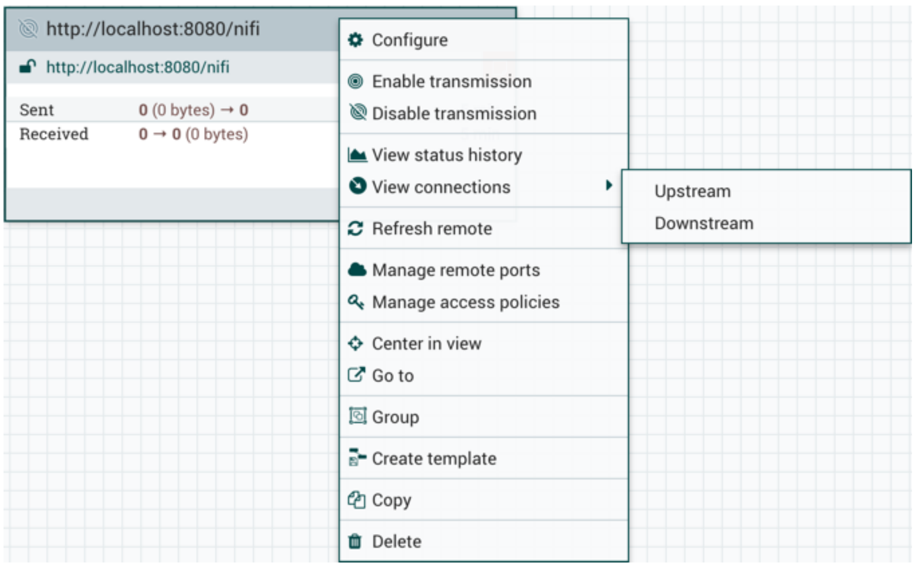
the resulting contents of the JSON file is the same whether the process group is versioned or not.)

- **Create template**: This option allows the user to create a template from the selected Process Group.
- **Copy**: This option places a copy of the selected Process Group on the clipboard, so that it may be pasted elsewhere on the canvas by right-clicking on the canvas and selecting Paste. The Copy/Paste actions also may be done using the keystrokes Ctrl-C (Command-C) and Ctrl-V (Command-V).
- **Empty all queues**: This option allows the user to empty all queues in the selected Process Group. All FlowFiles from all connections waiting at the time of the request will be removed.
- **Delete**: This option allows the DFM to delete a Process Group.

Remote Process Group: Remote Process Groups appear and behave similar to Process Groups. However, the Remote Process Group (RPG) references a remote instance of Logbot ETL. When an RPG is dragged onto the canvas, rather than being prompted for a name, the DFM is prompted for the URL of the remote Logbot ETL instance. If the remote Logbot ETL is a clustered instance, adding two or more cluster node URLs is recommended so that an initial connection can be made even if one of the nodes is unavailable. Multiple URLs can be specified in a comma-separated format.

When data is transferred to a clustered instance of Logbot ETL via an RPG, the RPG will first connect to the remote instance whose URL is configured to determine which nodes are in the cluster and how busy each node is. This information is then used to load balance the data that is pushed to each node. The remote instances are then interrogated periodically to determine information about any nodes that are dropped from or added to the cluster and to recalculate the load balancing based on each node's load. For more information, see the section on Site-to-Site.

Once you have dragged a Remote Process Group onto the canvas, you can interact with it by right-clicking on the Remote Process Group and selecting an option from the context menu. The options available to you from the menu vary, depending on the privileges assigned to you.

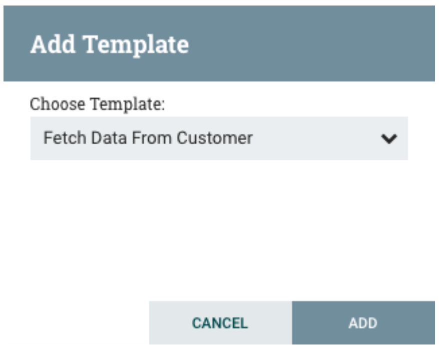


The following options are typically available when you have full privileges to work with the Remote Process Group:

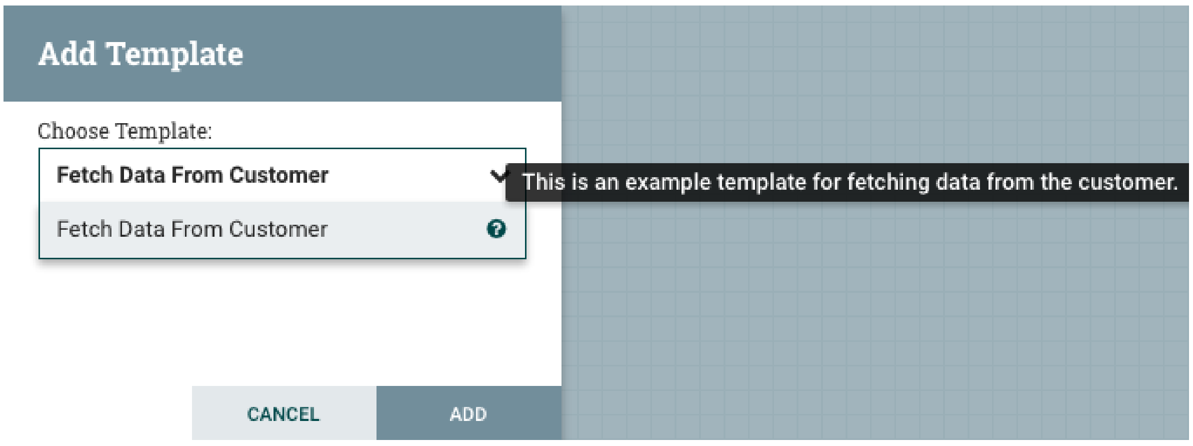
- **Configure**: This option allows the user to establish or change the configuration of the Remote Process Group.
- **Enable transmission**: Makes the transmission of data between Logbot ETL instances active (see Remote Process Group Transmission).
- **Disable transmission**: Disables the transmission of data between Logbot ETL instances.
- **View status history**: This option opens a graphical representation of the Remote Process Group’s statistical information over time.
- **View connections→Upstream**: This option allows the user to see and "jump to" upstream connections that are coming into the Remote Process Group.
- **View connections→Downstream**: This option allows the user to see and "jump to" downstream connections that are going out of the Remote Process Group.
- **Refresh remote**: This option refreshes the view of the status of the remote Logbot ETL instance.
- **Manage remote ports**: This option allows the user to see input ports and/or output ports that exist on the remote instance of Logbot ETL that the Remote Process Group is connected to. Note that if the Site-to-Site configuration is secure, only the ports that the connecting Logbot ETL has been given accessed to will be visible.
- **Center in view**: This option centers the view of the canvas on the given Remote Process Group.
- **Go to**: This option opens a view of the remote Logbot ETL instance in a new tab of the browser. Note that if the Site-to-Site configuration is secure, the user must have access to the remote Logbot ETL instance in order to view it.
- **Group**: This option allows the user to create a new Process Group that contains the selected Remote Process Group and any other components selected on the canvas.
- **Create template**: This option allows the user to create a template from the selected Remote Process Group.
- **Copy**: This option places a copy of the selected Process Group on the clipboard, so that it may be pasted elsewhere on the canvas by right-clicking on the canvas and selecting Paste. The Copy/Paste actions also may be done using the keystrokes Ctrl-C (Command-C) and Ctrl-V (Command-V).
- **Delete**: This option allows the DFM to delete a Remote Process Group from the canvas.

Funnel: Funnels are used to combine the data from many Connections into a single Connection. This has two advantages. First, if many Connections are created with the same destination, the canvas can become cluttered if those Connections have to span a large space. By funneling these Connections into a single Connection, that single Connection can then be drawn to span that large space instead. Secondly, Connections can be configured with FlowFile Prioritizers. Data from several Connections can be funneled into a single Connection, providing the ability to Prioritize all of the data on that one Connection, rather than prioritizing the data on each Connection independently.

Template: Templates can be created by DFMs from sections of the flow, or they can be imported from other dataflows. These Templates provide larger building blocks for creating a complex flow quickly. When the Template is dragged onto the canvas, the DFM is provided a dialog to choose which Template to add to the canvas:



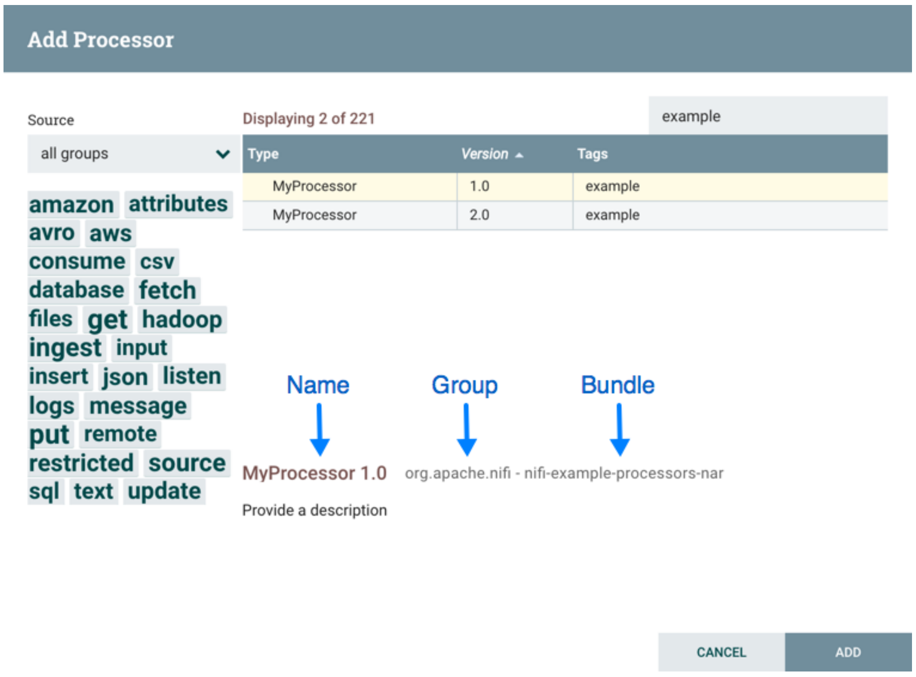
Clicking the drop-down box shows all available Templates. Any Template that was created with a description will show a question mark icon, indicating that there is more information. Hovering over the icon with the mouse will show this description:



Label: Labels are used to provide documentation to parts of a dataflow. When a Label is dropped onto the canvas, it is created with a default size. The Label can then be resized by dragging the handle in the bottom-right corner. The Label has no text when initially created. The text of the Label can be added by right-clicking on the Label and choosing `Configure`.

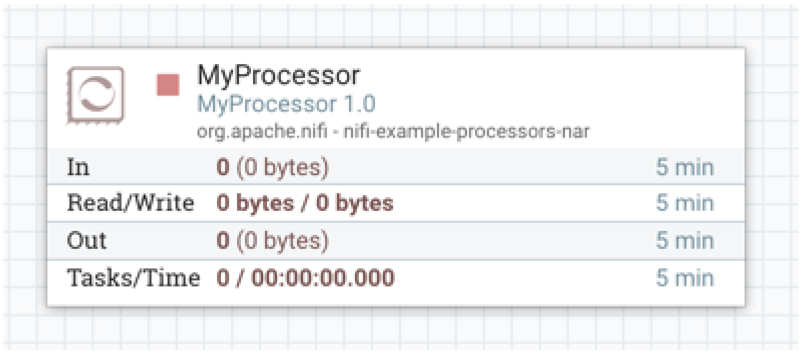
Component Versions

You have access to information about the version of your Processors, Controller Services, and Reporting Tasks. This is especially useful when you are working within a clustered environment with multiple Logbot ETL instances running different versions of a component or if you have upgraded to a newer version of a processor. The Add Processor, Add Controller Service, and Add Reporting Task dialogs include a column identifying the component version, as well as the name of the component, the organization or group that created the component, and the NAR bundle that contains the component.


Add Processor
Source
all groups
Displaying 2 of 221
example
amazon attributes avro aws consume csv database fetch files get hadoop ingest input insert json listen logs message put remote restricted source sql text update
Name
Group
Bundle
MyProcessor 1.0
org.apache.nifi - nifi-example-processors-nar
Provide a description
CANCEL
ADD

Type	Version ▲	Tags
MyProcessor	1.0	example
MyProcessor	2.0	example

Each component displayed on the canvas also contains this information.



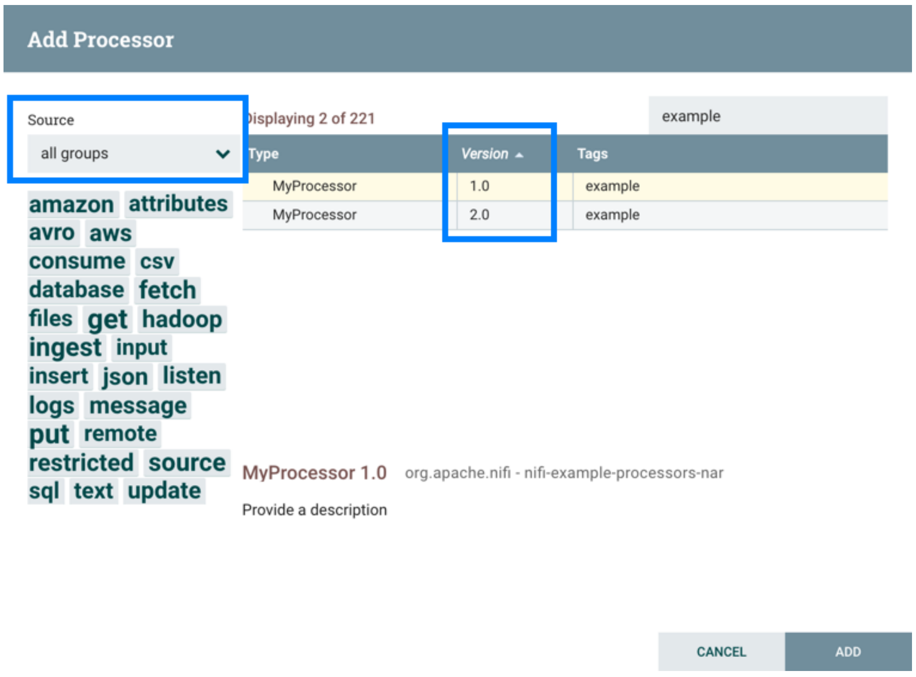
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Sorting and Filtering Components

When you are adding a component, you can sort on version number or filter based on originating source.

To sort based on version, click the version column to display in ascending or descending version order.

To filter based on source group, click the source drop-down in the upper left of your Add Component dialog, and select the group you want to view.


Add Processor
Source
all groups
amazon attributes avro aws consume csv database fetch files get hadoop ingest input insert json listen logs message put remote restricted source sql text update
example
CANCEL
ADD

Displaying 2 of 221

Type	Version	Tags
MyProcessor	1.0	example
MyProcessor	2.0	example

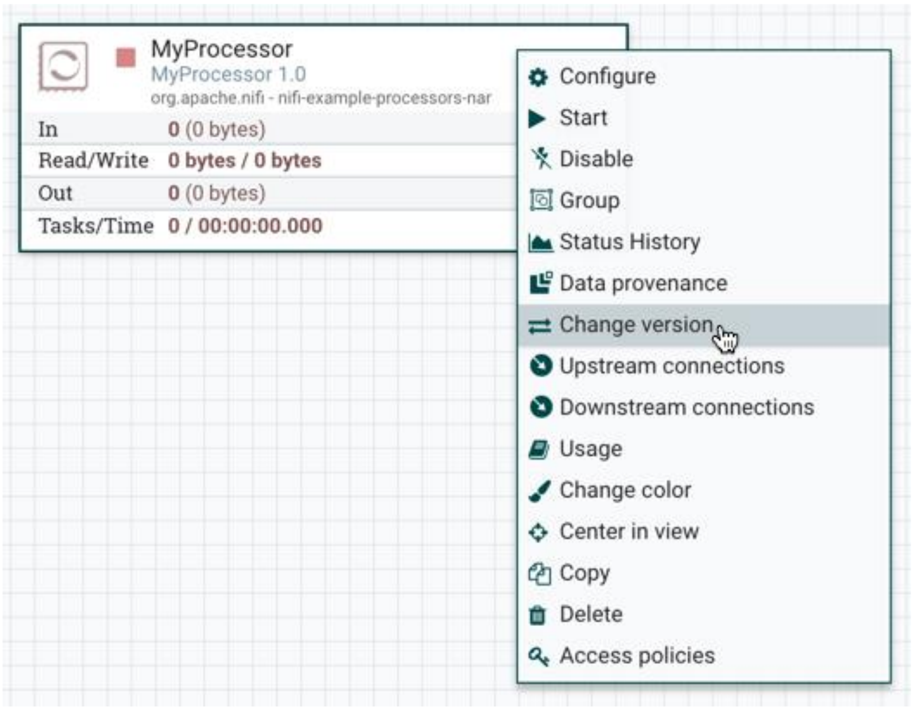
MyProcessor 1.0 org.apache.nifi - nifi-example-processors-nar

Provide a description

Changing Component Versions

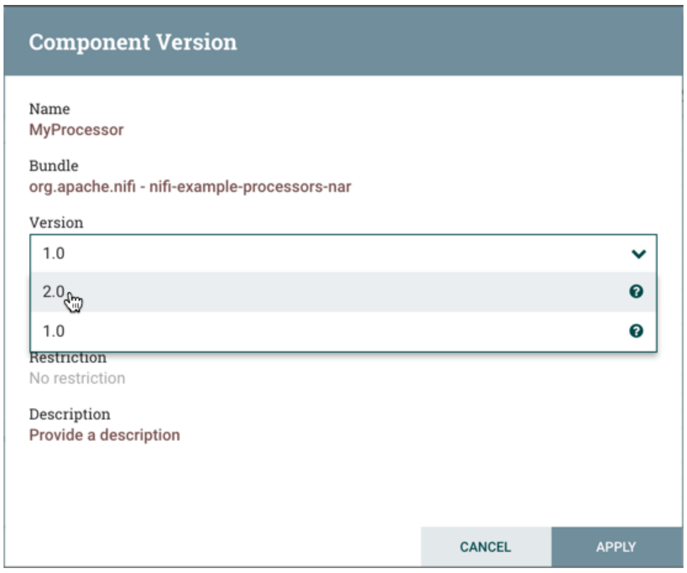
To change a component version, perform the following steps.

1. Right-click the component on the canvas to display configuration options.
2. Select Change version.



In	0 (0 bytes)
Read/Write	0 bytes / 0 bytes
Out	0 (0 bytes)
Tasks/Time	0 / 00:00:00.000

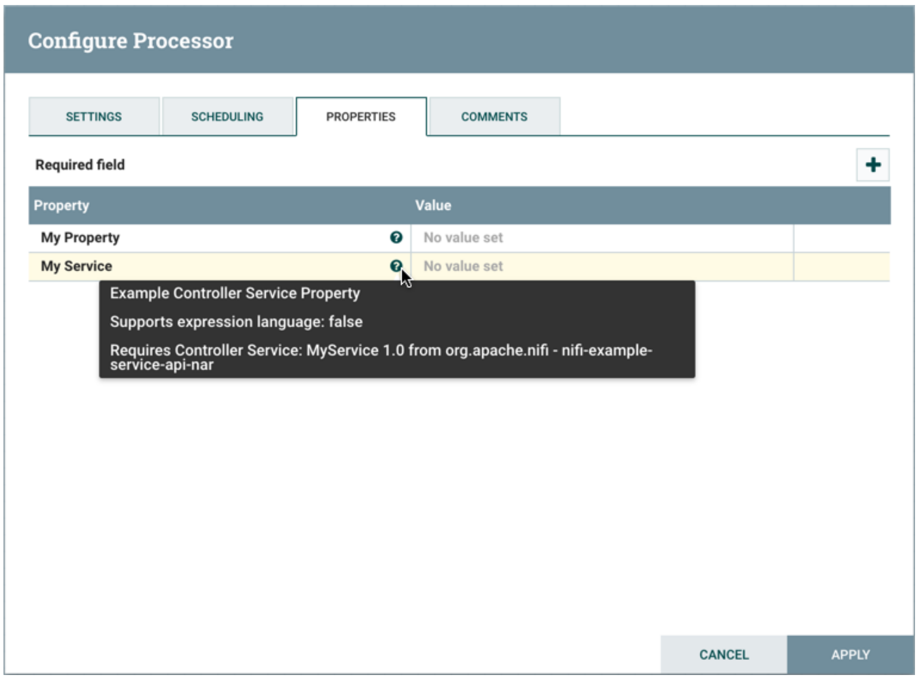
3. In the Component Version dialog, select the version you want to run from the Version drop-down menu.



Understanding Version Dependencies

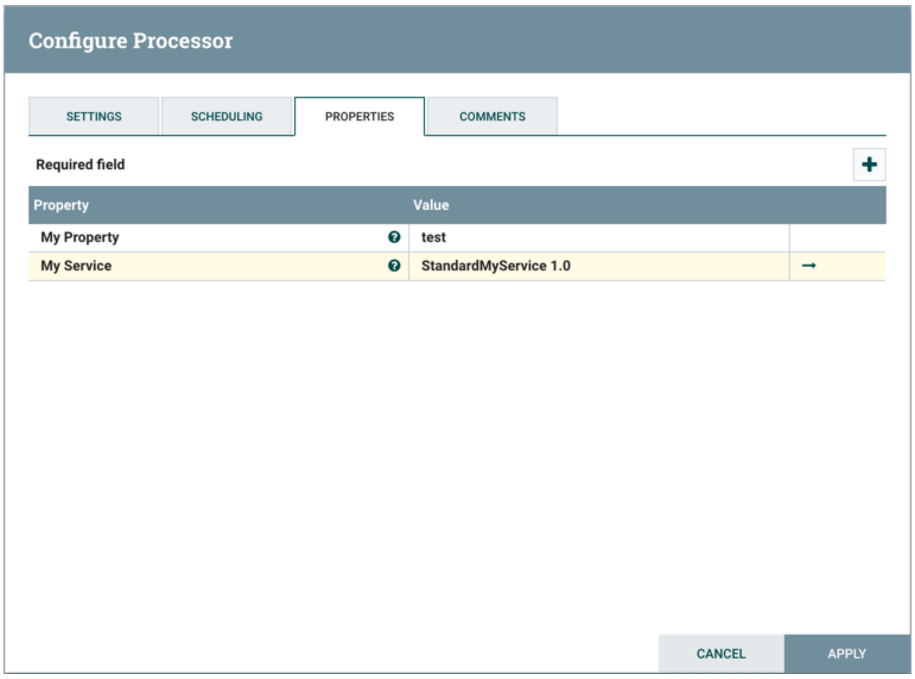
When you are configuring a component, you can also view information about version dependencies.

1. Right-click your component and select Configure to display the Configure dialog for your component.
2. Click the Properties tab.
3. Click the information icon to view any version dependency information.



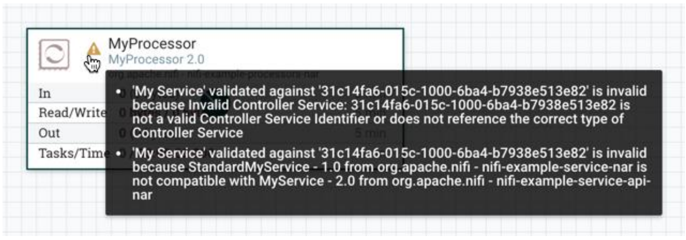
Example Controller Service Property
Supports expression language: false
Requires Controller Service: MyService 1.0 from org.apache.nifi - nifi-example-service-api-nar

In the following example, MyProcessor version 1.0 is configured properly with the controller service StandardMyService version 1.0:



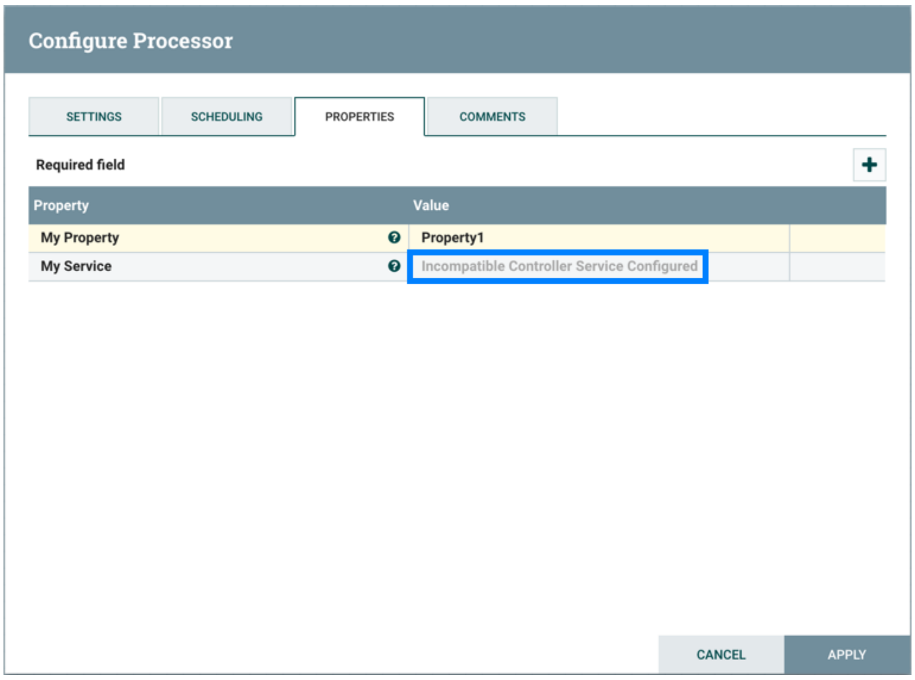
Property	Value
My Property	test
My Service	StandardMyService 1.0

If the version of MyProcessor is changed to an incompatible version (MyProcessor 2.0), validation errors will be displayed on the processor:



- 'My Service' validated against '31c14fa6-015c-1000-6ba4-b7938e513e82' is invalid because Invalid Controller Service: 31c14fa6-015c-1000-6ba4-b7938e513e82 is not a valid Controller Service Identifier or does not reference the correct type of Controller Service
- 'My Service' validated against '31c14fa6-015c-1000-6ba4-b7938e513e82' is invalid because StandardMyService - 1.0 from org.apache.nifi - nifi-example-service-nar is not compatible with MyService - 2.0 from org.apache.nifi - nifi-example-service-api-nar

and an error message will be displayed in the processor's controller service configuration since the service is no longer valid:



Property	Value
My Property	Property1
My Service	Incompatible Controller Service Configured

Configuring a Processor

To configure a processor, right-click on the Processor and select the `Configure` option from the context menu. Alternatively, just double-click on the Processor. The configuration dialog is opened with four different tabs, each of which is discussed below. Once you have finished configuring the Processor, you can apply the changes by clicking "Apply" or cancel all changes by clicking "Cancel".

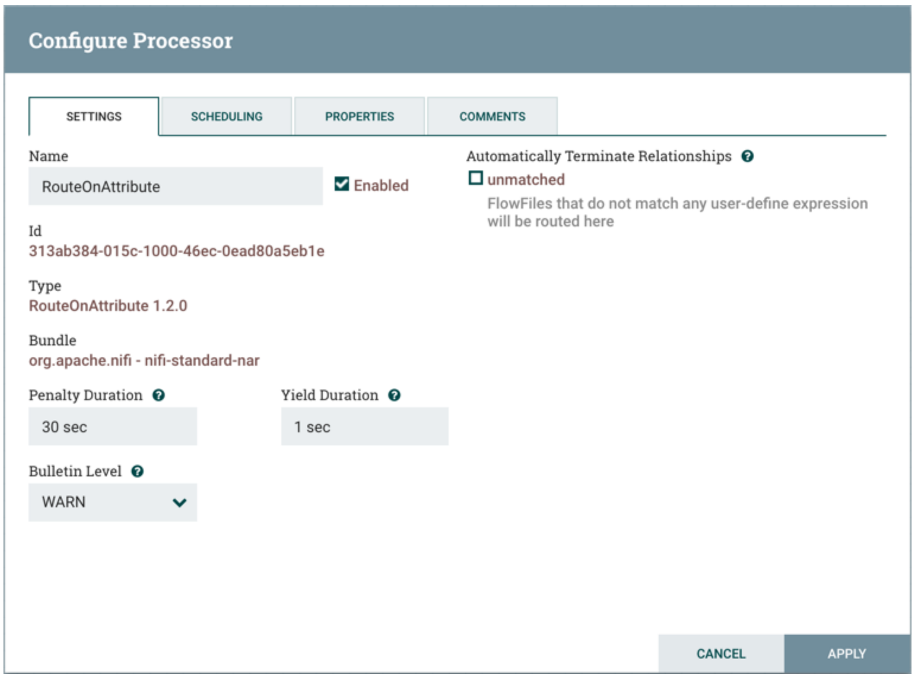
Note that after a Processor has been started, the context menu shown for the Processor no longer has a `Configure` option but rather has a `View Configuration` option. Processor configuration cannot be changed while the Processor is running. You must first stop the Processor and wait for all of its active tasks to complete before configuring the Processor again.

Note that entering certain control characters are not supported and will be automatically filtered out when entered. The following characters and any unpaired Unicode surrogate codepoints will not be retained in any configuration:

```
[#x0], [#x1], [#x2], [#x3], [#x4], [#x5], [#x6], [#x7], [#x8], [#xB], [#xC], [#xE],
[#xF], [#x10], [#x11], [#x12], [#x13], [#x14], [#x15], [#x16], [#x17], [#x18],
[#x19], [#x1A], [#x1B], [#x1C], [#x1D], [#x1E], [#x1F], [#xFFFE], [#xFFFF]
```

Settings Tab

The first tab in the Processor Configuration dialog is the Settings tab:



Name: RouteOnAttribute
☑ Enabled
Id: 313ab384-015c-1000-46ec-0ead80a5eb1e
Type: RouteOnAttribute 1.2.0
Bundle: org.apache.nifi - nifi-standard-nar
Penalty Duration: 30 sec
Yield Duration: 1 sec
Bulletin Level: WARN
Automatically Terminate Relationships
☐ unmatched
FlowFiles that do not match any user-define expression will be routed here

This tab contains several different configuration items. First, it allows the DFM to change the name of the Processor. The name of a Processor by default is the same as the Processor type. Next to the Processor Name is a checkbox, indicating whether the Processor is Enabled. When a Processor is added to the canvas, it is enabled. If the Processor is disabled, it cannot be started. The disabled state is used to indicate that when a group of Processors is started, such as when a DFM starts an entire Process Group, this (disabled) Processor should be excluded.

Below the Name configuration, the Processor’s unique identifier is displayed along with the Processor’s type and NAR bundle. These values cannot be modified.

Next are two dialogues for configuring 'Penalty Duration' and 'Yield Duration'. During the normal course of processing a piece of data (a FlowFile), an event may occur that indicates that the data cannot be processed at this time but the data may be processable at a later time. When this occurs, the Processor may choose to Penalize the FlowFile. This will prevent the FlowFile from being Processed for some period of time. For example, if the Processor is to push the data to a remote service, but the remote service already has a file with the same name as the filename that the Processor is specifying, the Processor may penalize the FlowFile. The 'Penalty Duration' allows the DFM to specify how long the FlowFile should be penalized. The default value is `30 seconds`.

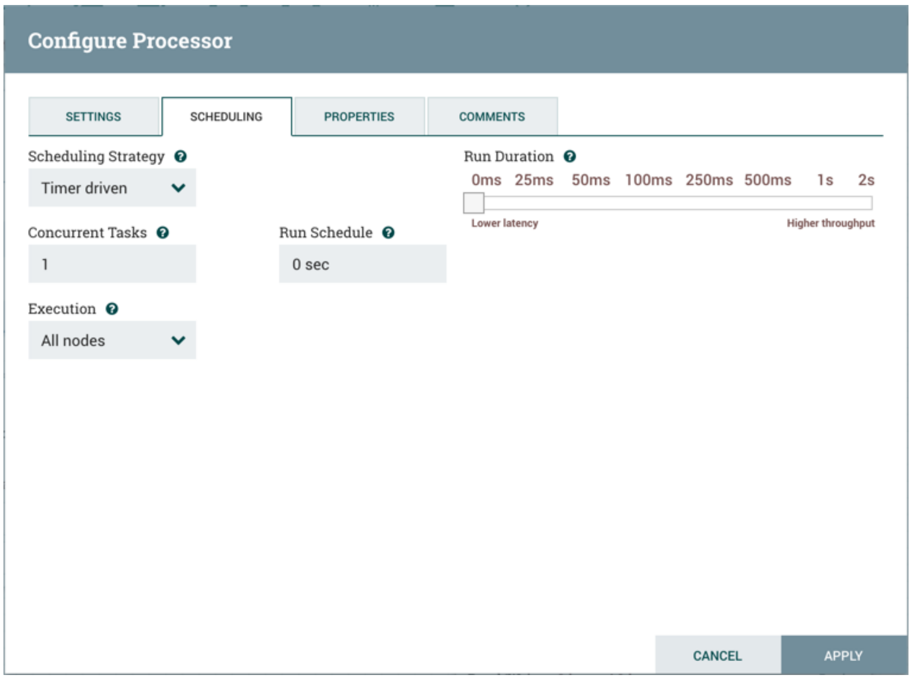
Similarly, the Processor may determine that some situation exists such that the Processor can no longer make any progress, regardless of the data that it is processing. For example, if a Processor is to push data to a remote service and that service is not responding, the Processor cannot make any progress. As a result, the Processor should 'yield', which will prevent the Processor from being scheduled to run for some period of time. That period of time is specified by setting the 'Yield Duration'. The default value is `1 second`.

The last configurable option on the left-hand side of the Settings tab is the Bulletin level. Whenever the Processor writes to its log, the Processor also will generate a Bulletin. This setting indicates the lowest level of Bulletin that should be shown in the User Interface. By default, the Bulletin level is set to `WARN`, which means it will display all warning and error-level bulletins.

The right-hand side of the Settings tab contains an 'Automatically Terminate Relationships' section. Each of the Relationships that is defined by the Processor is listed here, along with its description. In order for a Processor to be considered valid and able to run, each Relationship defined by the Processor must be either connected to a downstream component or auto-terminated. If a Relationship is auto-terminated, any FlowFile that is routed to that Relationship will be removed from the flow and its processing considered complete. Any Relationship that is already connected to a downstream component cannot be auto-terminated. The Relationship must first be removed from any Connection that uses it. Additionally, for any Relationship that is selected to be auto-terminated, the auto-termination status will be cleared (turned off) if the Relationship is added to a Connection.

Scheduling Tab

The second tab in the Processor Configuration dialog is the Scheduling Tab:


Configure Processor

Scheduling Strategy: Timer driven
Concurrent Tasks: 1
Run Schedule: 0 sec
Execution: All nodes
Run Duration: 0ms 25ms 50ms 100ms 250ms 500ms 1s 2s (Lower latency – Higher throughput)

Scheduling Strategy

The first configuration option is the Scheduling Strategy. There are three possible options for scheduling components:

Timer driven: This is the default mode. The Processor will be scheduled to run on a regular interval. The interval at which the Processor is run is defined by the 'Run Schedule' option (see below).

Event driven: When this mode is selected, the Processor will be triggered to run by an event, and that event occurs when FlowFiles enter Connections feeding this Processor. This mode is currently considered experimental and is not supported by all Processors. When this mode is selected, the 'Run Schedule' option is not configurable, as the Processor is not triggered to run periodically but as the result of an event. Additionally, this is the only mode for which the 'Concurrent Tasks' option can be set to 0. In this case, the number of threads is limited only by the size of the Event-Driven Thread Pool that the administrator has configured. (EXPERIMANTAL)

CRON driven: When using the CRON driven scheduling mode, the Processor is scheduled to run periodically, similar to the Timer driven scheduling mode. However, the CRON driven mode provides sigLogbot ETLcantly more flexibility at the expense of increasing the complexity of the configuration. The CRON driven scheduling value is a string of six required fields and one optional field, each separated by a space. These fields are:

Field	Valid values
Seconds	0-59
Minutes	0-59
Hours	0-23
Day of Month	1-31
Month	1-12 or JAN-DEC

Field	Valid values
Day of Week	1-7 or SUN-SAT
Year (optional)	empty, 1970-2099

You typically specify values one of the following ways:

- **Number**: Specify one or more valid value. You can enter more than one value using a comma-separated list.
- **Range**: Specify a range using the <number>-<number> syntax.
- **Increment**: Specify an increment using <start value>/<increment> syntax. For example, in the Minutes field, 0/15 indicates the minutes 0, 15, 30, and 45.

You should also be aware of several valid special characters:

- * — Indicates that all values are valid for that field.
- ? — Indicates that no specific value is specified. This special character is valid in the Days of Month and Days of Week field.
- L — You can append L to one of the Day of Week values, to specify the last occurrence of this day in the month. For example, 1L indicates the last Sunday of the month.

For example:

- The string `0 0 13 * * ?` indicates that you want to schedule the processor to run at 1:00 PM every day.
- The string `0 20 14 ? * MON-FRI` indicates that you want to schedule the processor to run at 2:20 PM every Monday through Friday.
- The string `0 15 10 ? * 6L 2011-2017` indicates that you want to schedule the processor to run at 10:15 AM, on the last Friday of every month, between 2011 and 2017.

For additional information and examples, see the cron triggers tutorial in the Quartz Documentation.

Concurrent Tasks

Next, the Scheduling tab provides a configuration option named 'Concurrent Tasks'. This controls how many threads the Processor will use. Said a different way, this controls how many FlowFiles should be processed by this Processor at the same time. Increasing this value will typically allow the Processor to handle more data in the same amount of time. However, it does this by using system resources that then are not usable by other Processors. This essentially provides a relative weighting of Processors — it controls how much of the system's resources should be allocated to this Processor instead of other Processors. This field is available for most Processors. There are, however, some types of Processors that can only be scheduled with a single Concurrent task.

Run Schedule

The 'Run Schedule' dictates how often the Processor should be scheduled to run. The valid values for this field depend on the selected Scheduling Strategy (see above). If using the Event driven Scheduling Strategy, this field is not available. When using the Timer driven Scheduling Strategy, this value is a time duration specified by a number followed by a time unit. For example, `1 second` or `5 mins`. The default value of `0 sec` means that the Processor should run as often as possible as long as it has data to process. This is true for any time duration of 0, regardless of the time unit (i.e., `0 sec`, `0 mins`, `0 days`). For an explanation of values that are applicable for the CRON driven Scheduling Strategy, see the description of the CRON driven Scheduling Strategy itself.

Execution

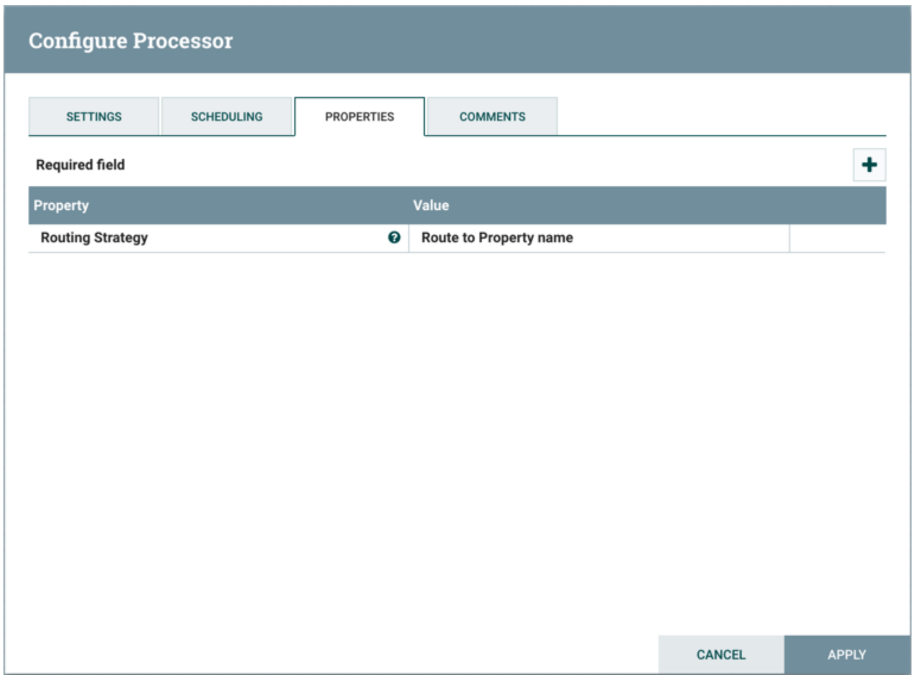
The Execution setting is used to determine on which node(s) the Processor will be scheduled to execute. Selecting 'All Nodes' will result in this Processor being scheduled on every node in the cluster. Selecting 'Primary Node' will result in this Processor being scheduled on the Primary Node only. Processors that have been configured for 'Primary Node' execution are identified by a "P" next to the processor icon. To quickly identify 'Primary Node' processors, the "P" icon is also shown in the Processors tab on the Summary page.

Run Duration

The right-hand side of the Scheduling tab contains a slider for choosing the 'Run Duration'. This controls how long the Processor should be scheduled to run each time that it is triggered. On the left-hand side of the slider, it is marked 'Lower latency' while the right-hand side is marked 'Higher throughput'. When a Processor finishes running, it must update the repository in order to transfer the FlowFiles to the next Connection. Updating the repository is expensive, so the more work that can be done at once before updating the repository, the more work the Processor can handle (Higher throughput). However, this means that the next Processor cannot start processing those FlowFiles until the previous Process updates this repository. As a result, the latency will be longer (the time required to process the FlowFile from beginning to end will be longer). As a result, the slider provides a spectrum from which the DFM can choose to favor Lower Latency or Higher Throughput.

Properties Tab

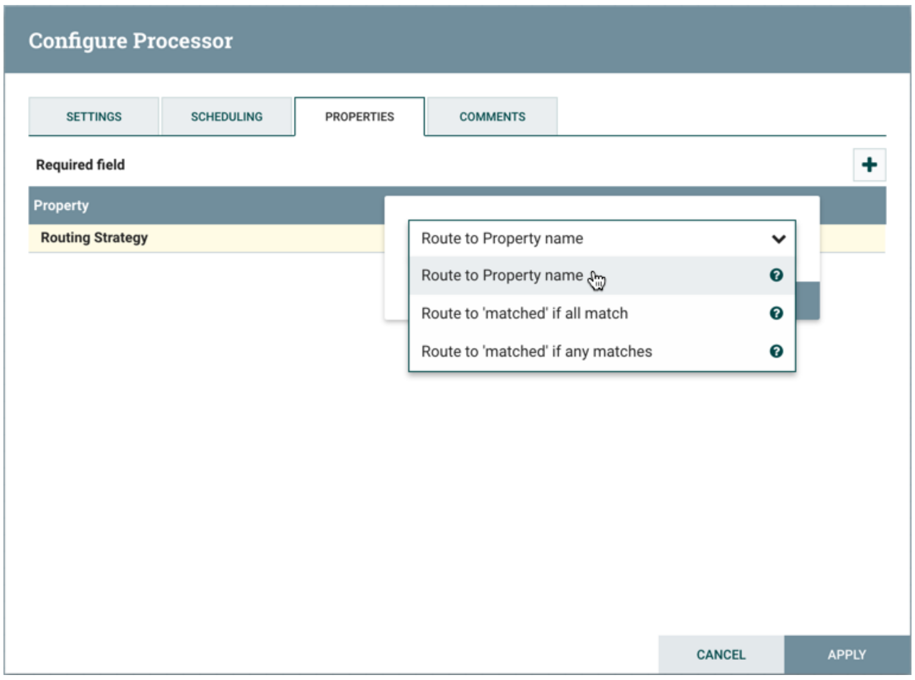
The Properties tab provides a mechanism to configure Processor-specific behavior. There are no default properties. Each type of Processor must define which Properties make sense for its use case. Below, we see the Properties tab for a RouteOnAttribute Processor:



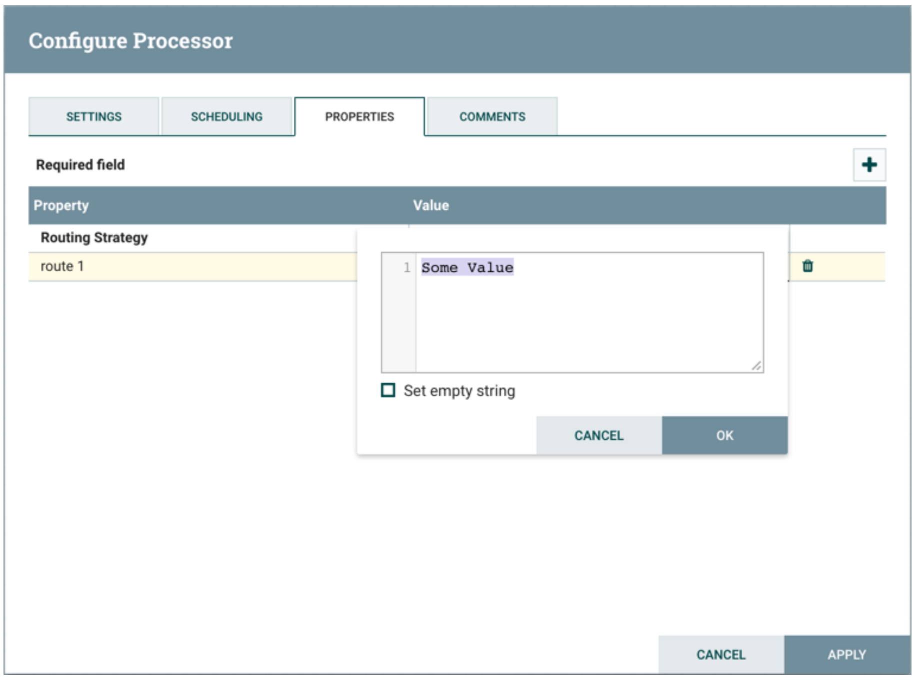
Property	Value
Routing Strategy	Route to Property name

This Processor, by default, has only a single property: 'Routing Strategy'. The default value is 'Route to Property name'. Next to the name of this property is a small question mark symbol (❓). This help symbol is seen in other places throughout the User Interface, and it indicates that more information is available. Hovering over this symbol with the mouse will provide additional details about the property and the default value, as well as historical values that have been set for the Property.

Clicking on the value for the property will allow a DFM to change the value. Depending on the values that are allowed for the property, the user is either provided a drop-down from which to choose a value or is given a text area to type a value:



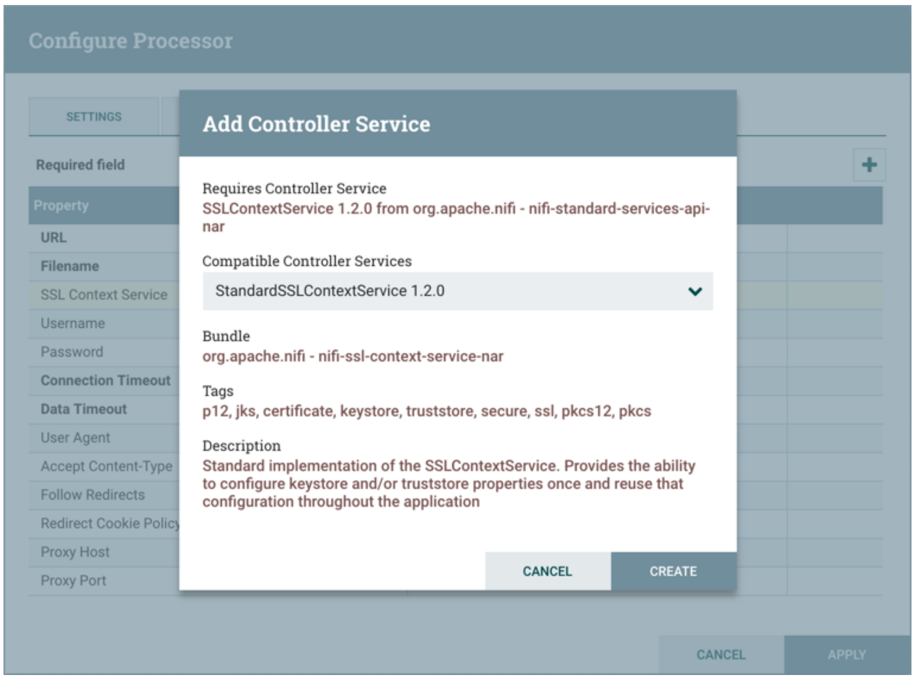
In the top-right corner of the tab is a button for adding a New Property. Clicking this button will provide the DFM with a dialog to enter the name and value of a new property. Not all Processors allow User-Defined properties. In processors that do not allow them, the Processor becomes invalid when User-Defined properties are applied. RouteOnAttribute, however, does allow User-Defined properties. In fact, this Processor will not be valid until the user has added a property.



Note that after a User-Defined property has been added, an icon will appear on the right-hand side of that row (🗑). Clicking it will remove the User-Defined property from the Processor.

Some processors also have an Advanced User Interface (UI) built into them. For example, the UpdateAttribute processor has an Advanced UI. To access the Advanced UI, click the "Advanced" button that appears at the bottom of the Configure Processor window. Only processors that have an Advanced UI will have this button.

Some processors have properties that refer to other components, such as Controller Services, which also need to be configured. For example, the GetHTTP processor has an SSLContextService property, which refers to the StandardSSLContextService controller service. When DFMs want to configure this property but have not yet created and configured the controller service, they have the option to create the service on the spot, as depicted in the image below. For more information about configuring Controller Services, see the Controller Services section.



Add Controller Service

Requires Controller Service
SSLContextService 1.2.0 from org.apache.nifi - nifi-standard-services-api-nar

Compatible Controller Services
StandardSSLContextService 1.2.0

Bundle
org.apache.nifi - nifi-ssl-context-service-nar

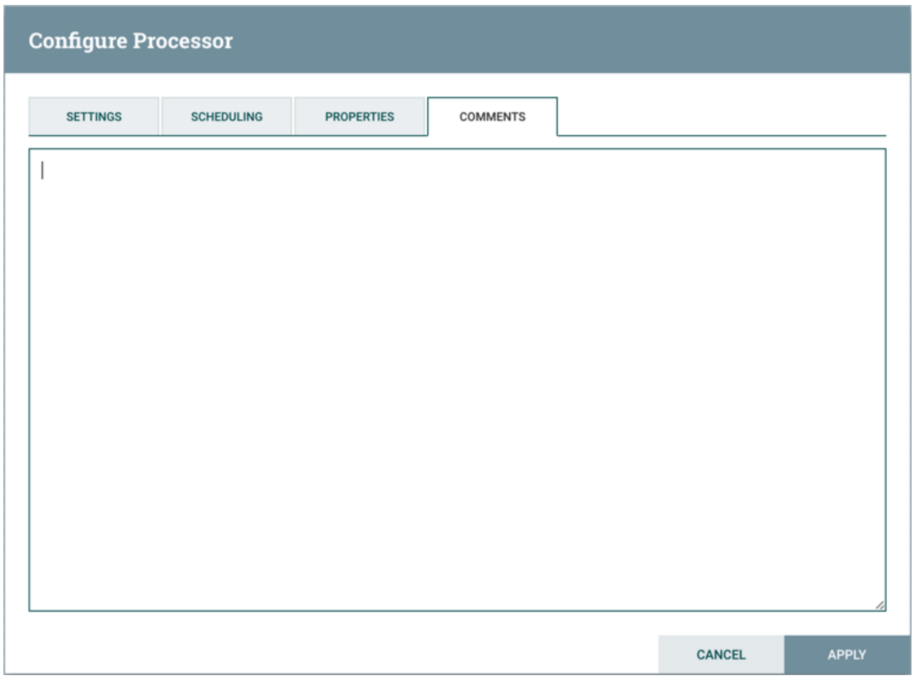
Tags
p12, jks, certificate, keystore, truststore, secure, ssl, pkcs12, pkcs

Description
Standard implementation of the SSLContextService. Provides the ability to configure keystore and/or truststore properties once and reuse that configuration throughout the application

CANCEL CREATE

Comments Tab

The last tab in the Processor configuration dialog is the Comments tab. This tab simply provides an area for users to include whatever comments are appropriate for this component. Use of the Comments tab is optional:

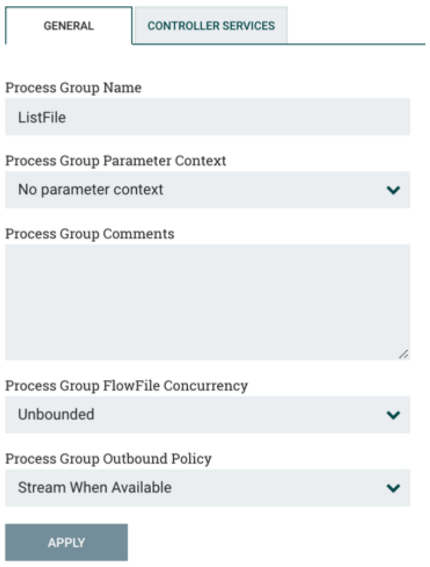


Additional Help

You can access additional documentation about each Processor’s usage by right-clicking on the Processor and selecting 'Usage' from the context menu. Alternatively, select Help from the Global Menu in the top-right corner of the UI to display a Help page with all of the documentation, including usage documentation for all the Processors that are available. Click on the desired Processor to view usage documentation.

Configuring a Process Group

To configure a Process Group, right-click on the Process Group and select the `Configure` option from the context menu. The configuration dialog is opened with two tabs: General and Controller Services.



Process Group Name
ListFile

Process Group Parameter Context
No parameter context

Process Group Comments

Process Group FlowFile Concurrency
Unbounded

Process Group Outbound Policy
Stream When Available

General Tab

This tab contains several different configuration items. First is the Process Group Name. This is the name that is shown at the top of the Process Group on the canvas as well as in the breadcrumbs at the bottom of the UI. For the Root Process Group (i.e., the highest level group), this is also the name that is shown as the title of the browser tab. Note that this information is visible to any other Logbot ETL instance that connects remotely to this instance (using Remote Process Groups, a.k.a., Site-to-Site).

The next configuration element is the Process Group Parameter Context, which is used to provide parameters to components of the flow. From this drop-down, the user is able to choose which Parameter Context should be bound to this Process Group and can optionally create a new one to bind to the Process Group. For more information refer to Parameters and Parameter Contexts.

The third element in the configuration dialog is the Process Group Comments. This provides a mechanism for providing any useful information or context about the Process Group.

The last two elements, Process Group FlowFile Currency and Process Group Outbound Policy, are covered in the following sections.

FlowFile Concurrency

FlowFile Concurrency is used to control how data is brought into the Process Group. There are three options available:

- Unbounded (the default)
- Single FlowFile Per Node
- Single Batch Per Node

When the FlowFile Concurrency is set to "Unbounded", the Input Ports in the Process Group will ingest data as quickly as they are able, provided that backpressure does not prevent them from doing so.

When the FlowFile Concurrency is configured to "Single FlowFile Per Node", the Input Ports will only allow a single FlowFile through at at time. Once that FlowFile enters the Process Group, no additional FlowFiles will be brought in until all FlowFiles have left the Process Group (either by being removed from the system/auto-terminated, or by exiting through an Output Port). This will often result in slower performance, as it reduces the parallelization that Logbot ETL uses to process the data. However, there are several reasons that a user may want to use this approach. A common use case is one in which each incoming FlowFile contains references to several other data items, such as a list of files in a directory. The user may want to process the entire listing before allowing any other data to enter the Process Group.

When the FlowFile Concurrency is configured to "Single Batch Per Node", the Input Ports will behave similarly to the way that they behave in the "Single FlowFile Per Node" mode, but when a FlowFile is ingested, the Input Ports will continue to ingest all data until all of the queues feeding the Input Ports have been emptied. At that point, they will not bring any more data into the Process Group until all data has finished processing and has left the Process Group (see Connecting Batch-Oriented Process Groups).

The FlowFile Concurrency controls only when data will be pulled into the Process Group from an Input Port. It does not prevent a Processor within the Process Group from ingesting data from outside of Logbot ETL.

Outbound Policy

While the FlowFile Concurrency dictates how data should be brought into the Process Group, the Outbound Policy controls the flow of data out of the Process Group. There are two available options available:

- Stream When Available (the default)
- Batch Output

When the Outbound Policy is configured to "Stream When Available", data that arrives at an Output Port is immediately transferred out of the Process Group, assuming that no backpressure is applied.

When the Outbound Policy is configured to "Batch Output", the Output Ports will not transfer data out of the Process Group until all data that is in the Process Group is queued up at an Output Port (i.e., no data leaves the Process Group until all of the data has finished processing). It doesn’t matter whether the data is all queued up for the same Output Port, or if some data is queued up for Output Port A while other data is queued up for Output Port B. These conditions are both considered the same in terms of the completion of the FlowFile Processing.

Using an Outbound Policy of "Batch Output" along with a FlowFile Concurrency of "Single FlowFile Per Node" allows a user to easily ingest a single FlowFile (which in and of itself may represent a batch of data) and then wait until all processing of that FlowFile has completed before continuing on to the next step in the dataflow (i.e., the next component outside of the Process Group). Additionally, when using this mode, each FlowFile that is transferred out of the Process Group will be given a series of attributes named "batch.output.<Port Name>" for each Output Port in the Process Group. The value will be equal to the number of FlowFiles that were routed to that Output Port for this batch of data. For example, consider a case where a single FlowFile is split into 5 FlowFiles, and two FlowFiles go to Output Port A, one goes to Output Port B, and two go to Output Port C, and no

FlowFiles go to Output Port D. In this case, each FlowFile will have attributes `batch.output.A = 2`, `batch.output.B = 1`, `batch.output.C = 2`, `batch.output.D = 0`.

The Outbound Policy of "Batch Output" doesn't provide any benefits when used in conjunction with a FlowFile Concurrency of "Unbounded". As a result, the Outbound Policy is ignored if the FlowFile Concurrency is set to "Unbounded".

Connecting Batch-Oriented Process Groups

A common use case in Logbot ETL is to perform some batch-oriented process and only after that process completes perform another process on that same batch of data.

Logbot ETL makes this possible by encapsulating each of these processes in its own Process Group. The Outbound Policy of the first Process Group should be configured as "Batch Output" while the FlowFile Concurrency should be either "Single FlowFile Per Node" or "Single Batch Per Node". With this configuration, the first Process Group will process an entire batch of data (which will either be a single FlowFile or many FlowFiles depending on the FlowFile Concurrency) as a coherent batch of data. When processing has completed for that batch of data, the data will be held until all FlowFiles are finished processing and ready to leave the Process Group. At that point, the data can be transferred out of the Process Group as a batch. This configuration - when a Process Group is configured with an Outbound Policy of "Batch Output" and an Output Port is connected directly to the Input Port of a Process Group with a FlowFile Concurrency of "Single Batch Per Node" - is treated as a slightly special case. The receiving Process Group will ingest data not only until its input queues are empty but until they are empty AND the source Process Group has transferred all of the data from that batch out of the Process Group. This allows a collection of FlowFiles to be transferred as a single batch of data between Process Groups - even if those FlowFiles are spread across multiple ports.

Caveats

When using a FlowFile Concurrency of "Single FlowFile Per Node", there are a couple of caveats to consider.

First, an Input Port is free to bring data into the Process Group if there is no data queued up in that Process Group on the same node. This means that in a 5-node cluster, for example, there may be up to 5 incoming FlowFiles being processed simultaneously. Additionally, if a connection is configured to use Load Balancing, it may transfer data to another node in the cluster, allowing data to enter the Process Group while that FlowFile is still being processed. As a result, it is not recommended to use Load-Balanced Connections within a Process Group that is not configured for "Unbounded" FlowFile Concurrency.

When using the Outbound Policy of "Batch Output", it is important to consider backpressure. Consider a case where no data will be transferred out of a Process Group until all data is finished processing. Also consider that the connection to Output Port A has a backpressure threshold of 10,000 FlowFiles (the default). If that queue reaches the threshold of 10,000, the upstream Processor will no longer be triggered. As a result, data will not finish processing, and the flow will end in a deadlock, as the Output Port will not run until the processing completes and the Processor will not run until the Output Port runs. To avoid this, if a large number of FlowFiles are expected to be generated from a single input FlowFile, it is recommended that backpressure for Connections ending in an Output Port be configured in such a way to allow for the largest expected number of FlowFiles or backpressure for those Connections be disabled all together (by setting the Backpressure Threshold to 0). See Back Pressure for more information.

Controller Services

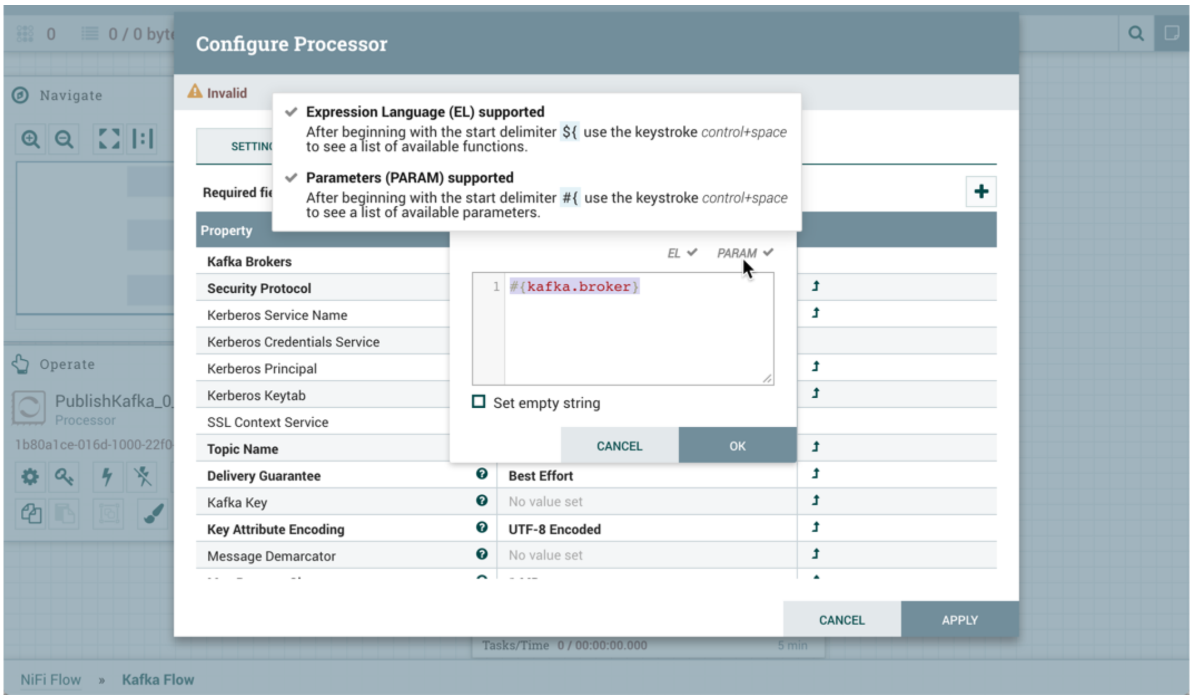
The Controller Services tab in the Process Group configuration dialog is covered in Adding Controller Services for Dataflows.

Parameters

The values of properties in the flow, including sensitive properties, can be parameterized using Parameters. Parameters are created and configured within the Logbot ETL UI. Any property can be configured to reference a Parameter with the following conditions:

- A sensitive property can only reference a Sensitive Parameter
- A non-sensitive property can only reference a Non-Sensitive Parameter
- Properties that reference Controller Services can not use Parameters
- Parameters cannot be referenced in Reporting Tasks or in controller-level Controller Services

The UI indicates whether a Parameter can be used for a property value.



Expression Language (EL) supported
After beginning with the start delimiter ${ use the keystroke *control+space* to see a list of available functions.

Parameters (PARAM) supported
After beginning with the start delimiter #{ use the keystroke *control+space* to see a list of available parameters.

EL ✓ PARAM ✓

```
#{kafka.broker}
```

☐ Set empty string

Property	
Kafka Brokers	
Security Protocol	
Kerberos Service Name	
Kerberos Credentials Service	
Kerberos Principal	
Kerberos Keytab	
SSL Context Service	
Topic Name	
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

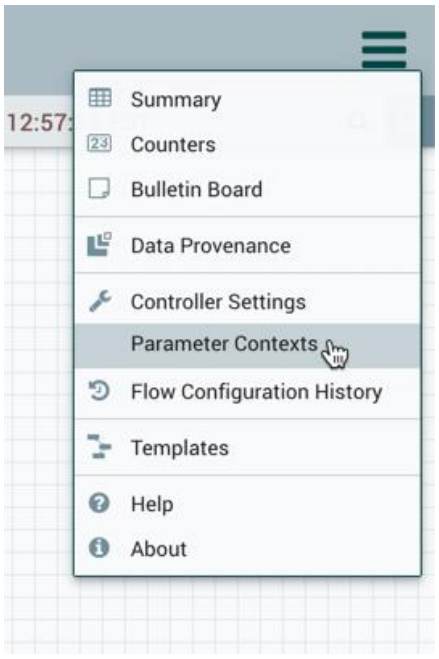
Parameters have numerous advantages over Variables. In addition to sensitive value support, Parameters offer more granular control over access policies. Additionally, properties that reference Parameters are validated against the substituted value, unlike most properties that reference Variables using Expression Language.

Parameter Contexts

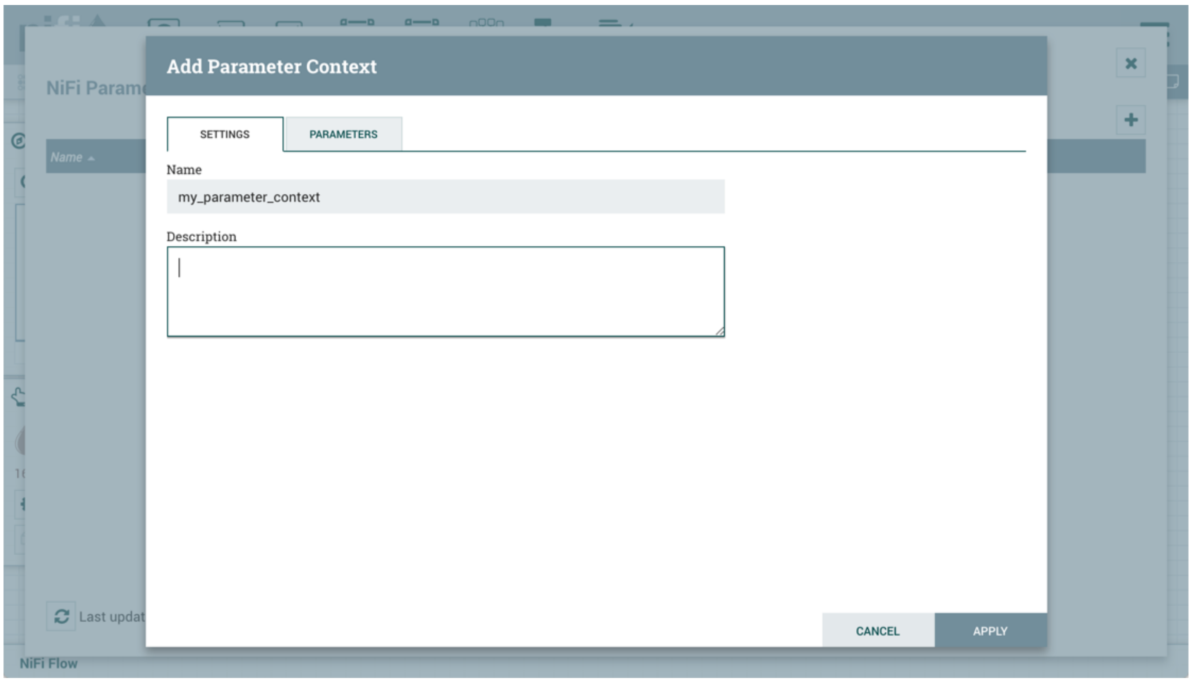
Parameters are created within Parameter Contexts. Parameter Contexts are globally defined/accessible to the Logbot ETL instance. Access policies can be applied to Parameter Contexts to determine which users can create them. Once created, policies to read and write to a specific Parameter Context can also be applied (see Accessing Parameters for more information).

Creating a Parameter Context

To create a Parameter Context, select Parameter Contexts from the Global Menu:



In the Parameter Contexts window, click the + button in the upper-right corner and the Add Parameter Context window opens. The window has two tabs: Settings and Parameters.



Name
my_parameter_context

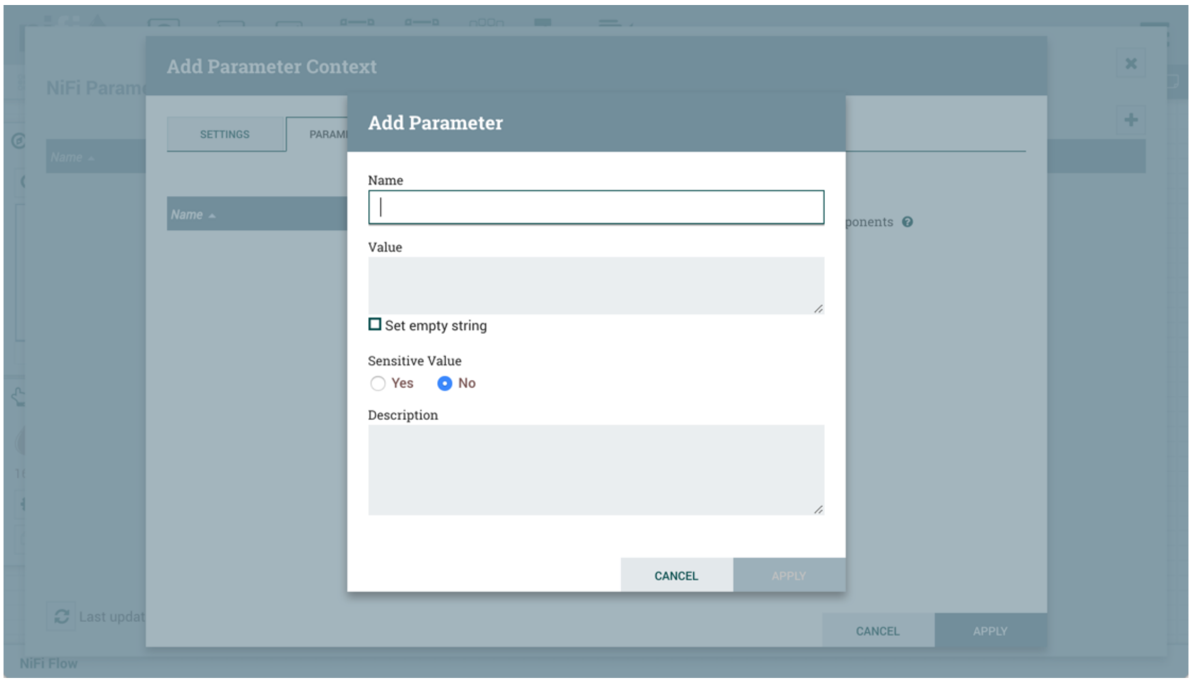
Description

On the "Settings" tab, add a name for the Parameter Context and a description if desired. Select "Apply" to save the Parameter Context or select the "Parameters" tab to add parameters to the context.

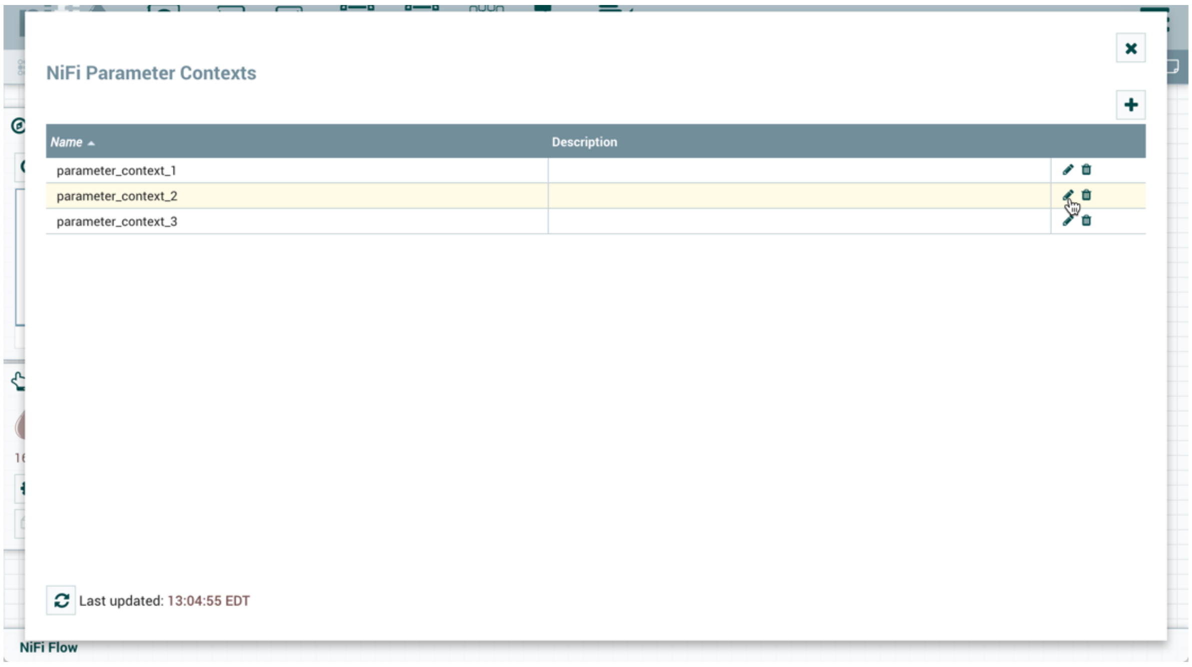
Adding a Parameter to a Parameter Context

Parameters can be added during Parameter Context creation or added to existing Parameter Contexts.

During Parameter Context creation, select the "Parameters" tab. Click the + button to open the Add Parameter window.



To add parameters to an existing Parameter Context, open the Parameter Context window and click the Edit button ( ) in the row of the desired Parameter Context.



Name	Description
parameter_context_1	
parameter_context_2	
parameter_context_3	

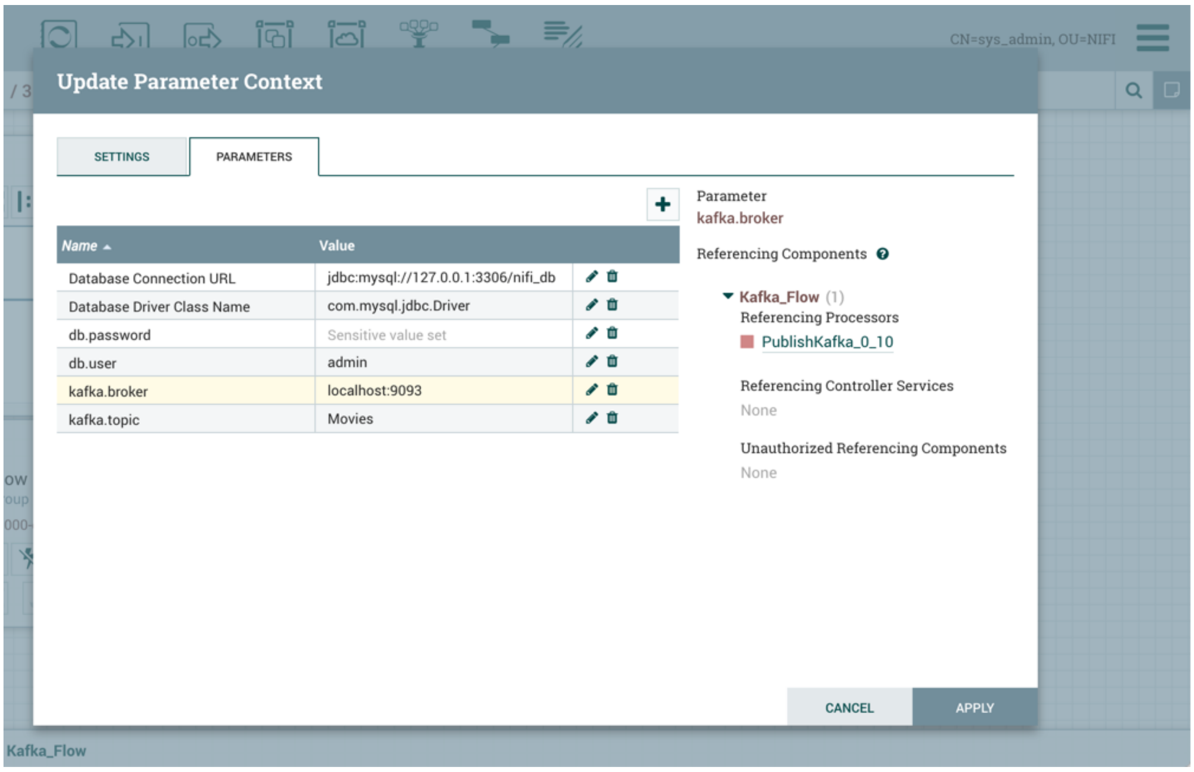
On the "Parameters" tab, click the + button to open the Add Parameter window.

The Add Parameter window has the following settings:

- **Name** - A name that is used to denote the Parameter. Only alpha-numeric characters (a-z, A-Z, 0-9), hyphens (-), underscores (_), periods (.), and spaces are allowed.

- **Value -** The value that will be used when the Parameter is referenced. If a Parameter makes use of the Expression Language, it is important to note that the Expression Language will be evaluated in the context of the component that references the Parameter. Please see the Parameters and Expression Language section below for more information.
- **Set empty string -** Check to explicitly set the value of the Parameter to an empty string. Unchecked by default. (Note: If checked but a value is set, the checkbox is ignored.)
- **Sensitive Value -** Set to "Yes" if the Parameter's Value should be considered sensitive. If sensitive, the value of the Parameter will not be shown in the UI once applied. The default setting is "No". Sensitive Parameters can only be referenced by sensitive properties and Non-Sensitive Parameters by non-sensitive properties. Once a Parameter is created, its sensitivity flag cannot be changed.
- **Description -** A description that explains what the Parameter is, how it is to be used, etc. This field is optional.

Once these settings are configured, select "Apply". The Referencing Components lists the components referenced by the currently selected parameter. Add additional Parameters or edit any existing Parameters.



Name	Value
Database Connection URL	jdbc:mysql://127.0.0.1:3306/nifi_db
Database Driver Class Name	com.mysql.jdbc.Driver
db.password	Sensitive value set
db.user	admin
kafka.broker	localhost:9093
kafka.topic	Movies

Parameter
kafka.broker

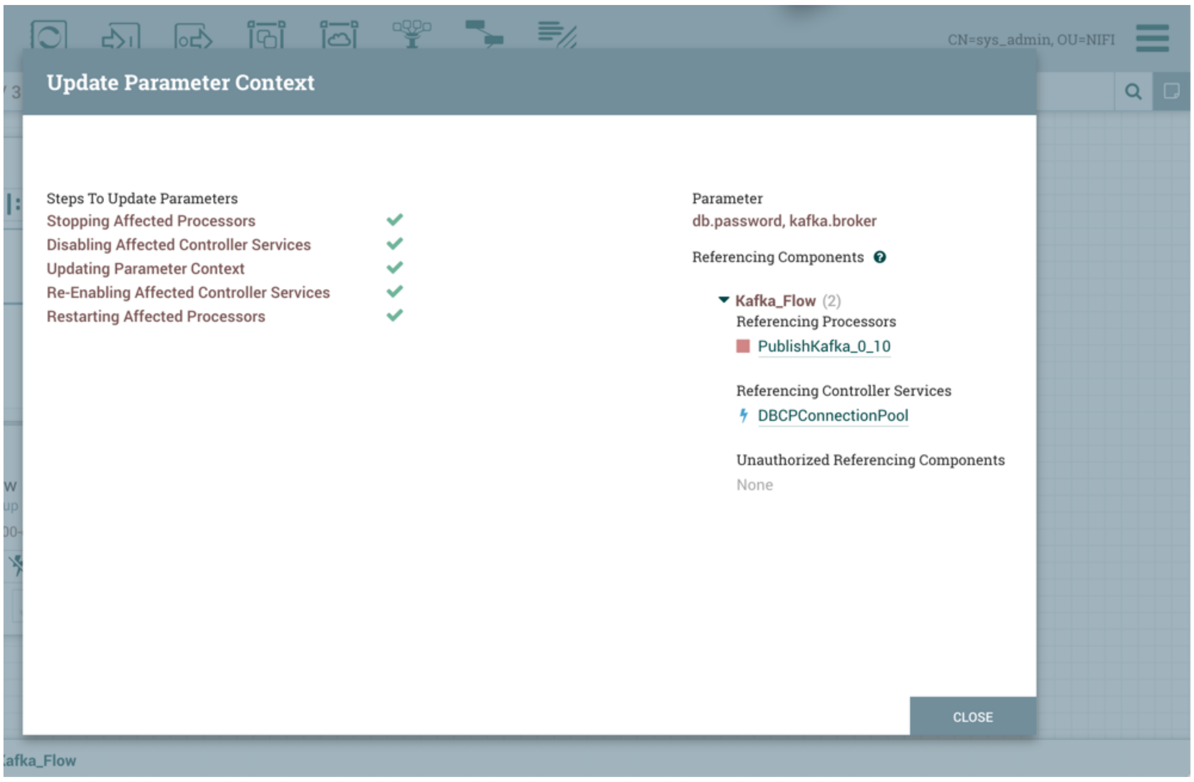
Referencing Components

Kafka_Flow (1)
Referencing Processors
PublishKafka_0_10

Referencing Controller Services
None

Unauthorized Referencing Components
None

To complete the process, select "Apply" from the Parameter Context window. The following operations are performed to validate all components that reference the added or modified parameters: Stopping/Restarting affected Processors, Disabling/Re-enabling affected Controller Services, Updating Parameter Context.



Update Parameter Context

Steps To Update Parameters
Stopping Affected Processors ✓
Disabling Affected Controller Services ✓
Updating Parameter Context ✓
Re-Enabling Affected Controller Services ✓
Restarting Affected Processors ✓

Parameter
db.password, kafka.broker

Referencing Components

Kafka_Flow (2)
Referencing Processors
PublishKafka_0_10

Referencing Controller Services
DBCPConnectionPool

Unauthorized Referencing Components
None

The Referencing Components section now lists an aggregation of all the components referenced by the set of parameters added/edited/deleted, organized by process group.

Parameters and Expression Language

When adding a Parameter that makes use of the Expression Language, it is important to understand the context in which the Expression Language will be evaluated. The expression is always evaluated in the context of the Process or Controller Service that references the Parameter. Take, for example, a scenario where Parameter with the name `Time` is added with a value of `${now()}`. The Expression Language results in a call to determine the system time when it is evaluated. When added as a Parameter, the system time is not evaluated when the Parameter is added, but rather when a Processor or Controller Service evaluates the Expression. That is, if a Processor has a Property whose value is set to `#{Time}` it will function in exactly the same manner as if the Property's value were set to `${now()}`. Each time that the property is referenced, it will produce a different timestamp.

Furthermore, some Properties do not allow for Expression Language, while others allow for Expression Language but do not evaluate expressions against FlowFile attributes. To help understand how this works, consider a Parameter named `File` whose value is `${filename}`. Then consider three different properties, each with a different Expression Language Scope and a FlowFile whose filename is `test.txt`. If each of those Properties is set to `#{File}`, then the follow table illustrates the resultant value.

Configured Property Value	Expression Language Scope	Effective Property Value	Notes
#{File}	FlowFile Attributes	test.txt	The filename is resolved by looking at the `filename` attribute.
#{File}	Variable Registry Only	*Empty String*	FlowFile attributes are not in scope, and we assume there is no Variable in the Variable Registry named "filename"

Configured Property Value	Expression Language Scope	Effective Property Value	Notes
#{File}	None	${filename}	The literal text "${filename}" will be unevaluated.

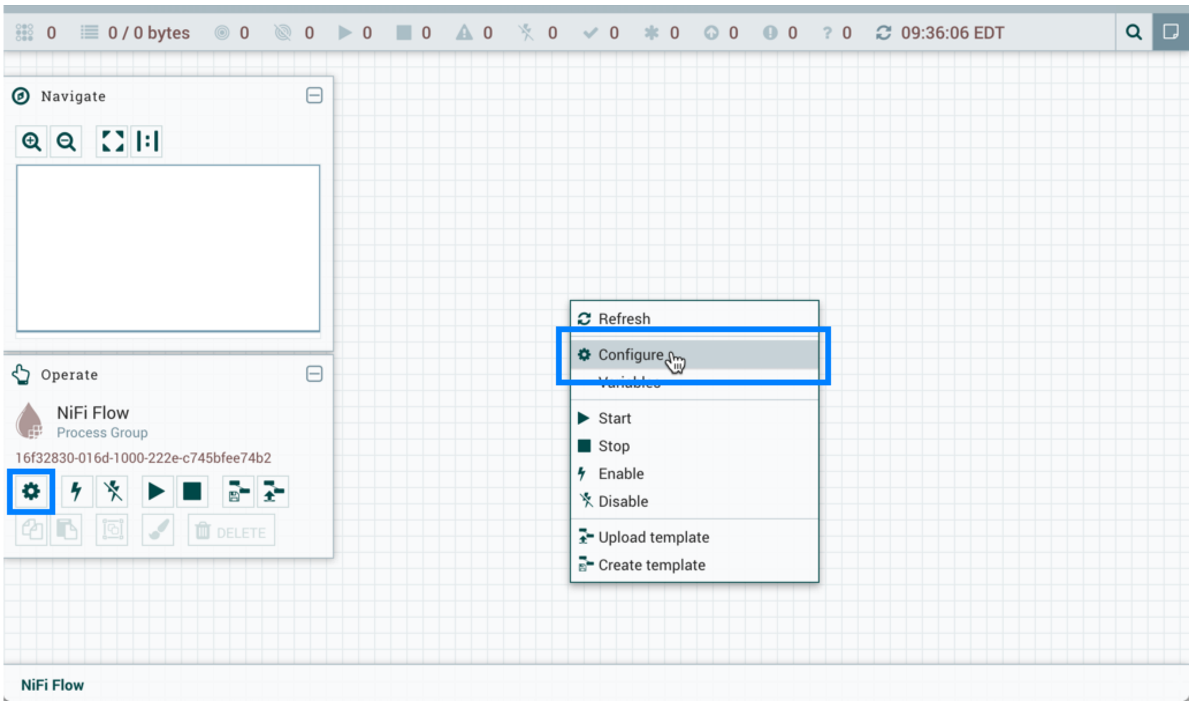
Assigning a Parameter Context to a Process Group

For a component to reference a Parameter, its Process Group must first be assigned a Parameter Context. Once assigned, processors and controller services within that Process Group may only reference Parameters within that Parameter Context.

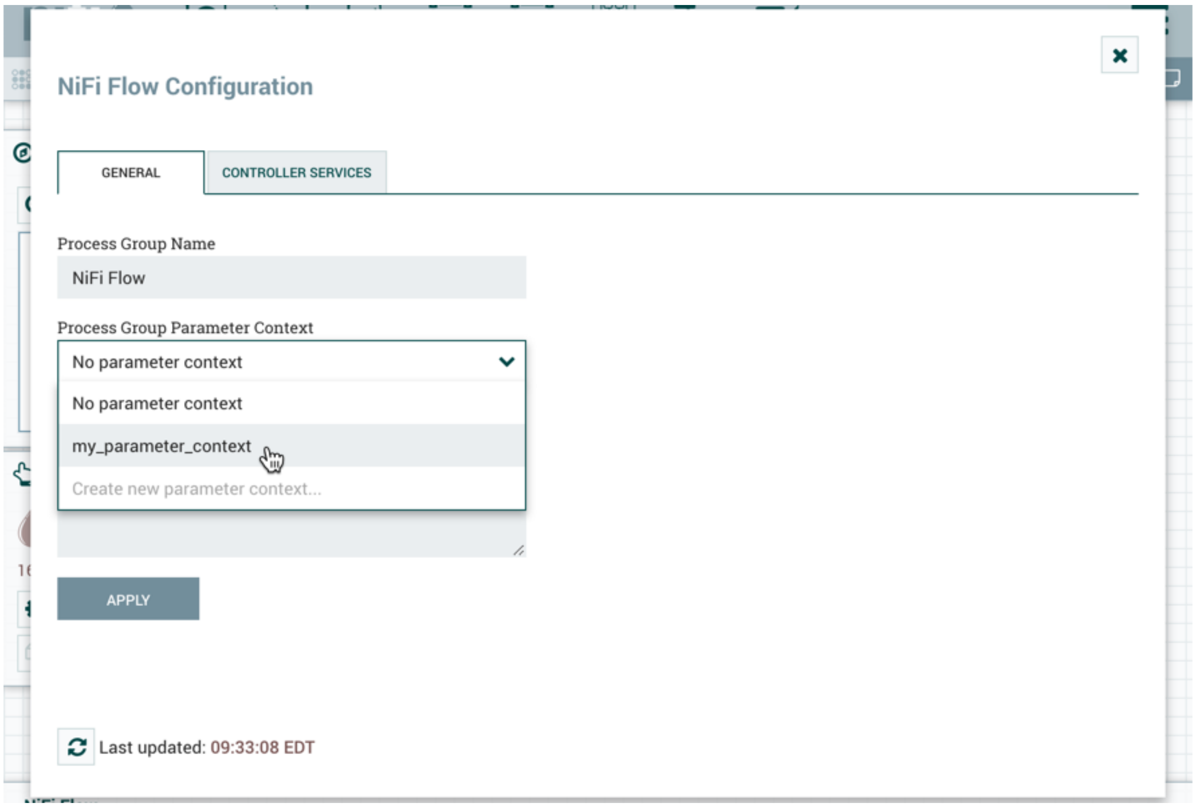
A Process Group can only be assigned one Parameter Context, while a given Parameter Context can be assigned to multiple Process Groups.

A user can only set the Parameter Context of a Process Group to one of the Parameter Contexts that the user has the view policy for. Additionally, in order to set the Parameter Context, the user must have the modify policy for the Process Group. See Accessing Parameters for more information.

To assign a Parameter Context to a Process Group, click Configure, either from the Operate Palette or from the Process Group context menu.



In the Flow Configuration window, select the "General" tab. From the Process Group Parameter Context drop-down menu, select an existing Parameter Context or create a new one.


NiFi Flow Configuration

Process Group Name

NiFi Flow

Process Group Parameter Context

No parameter context

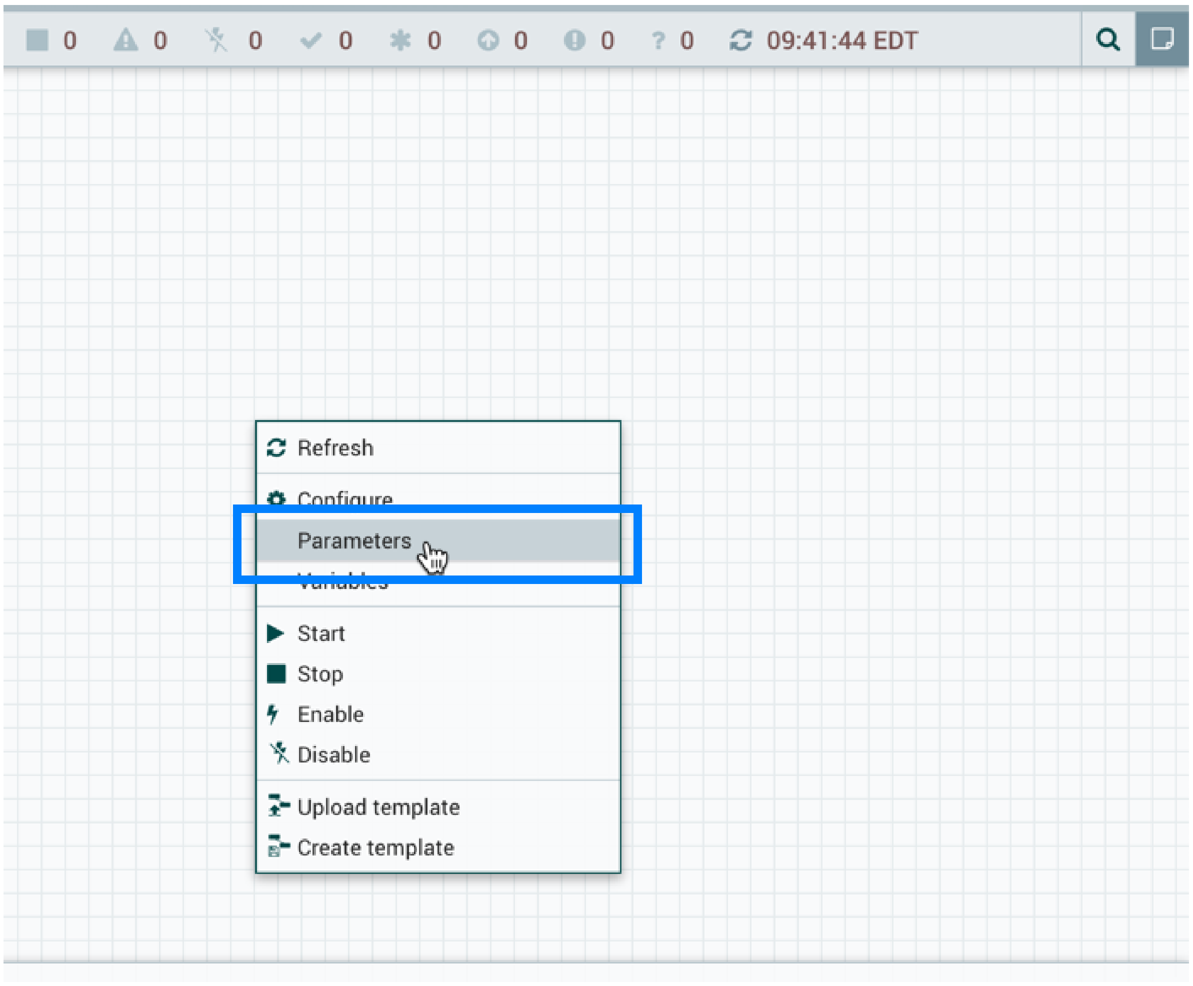
No parameter context

my_parameter_context

Create new parameter context...

Last updated: 09:33:08 EDT

Select "Apply" to save the configuration changes. The Process Group context menu now includes a "Parameters" option which allows quick access to the Update Parameter Context window for the assigned Parameter Context.



If the Parameter Context for a Process Group is changed, all components that reference any Parameters in that Process Group will be stopped, validated, and restarted assuming the components were previously running and are still valid.

If a Parameter Context is unset from a Process Group, it does **NOT** inherit the Parameter Context from the parent Process Group. Instead, no Parameters can be referenced. Any component that does already reference a Parameter will become invalid.

Referencing Parameters

Parameter Reference Syntax

To configure an eligible property to reference a Parameter, use the `#` symbol as the start, with the Parameter's name enclosed in curly braces:

```
#{Parameter.Name}
```

This can be escaped using an additional `#` character at the beginning. To illustrate this, assume that the Parameter `abc` has a value of `xxx` and Parameter `def` has a value of `yyy`. Then, the following user-defined property values will evaluate to these effective values:

User-Entered Literal Property Value	Effective Property Value	Explanation

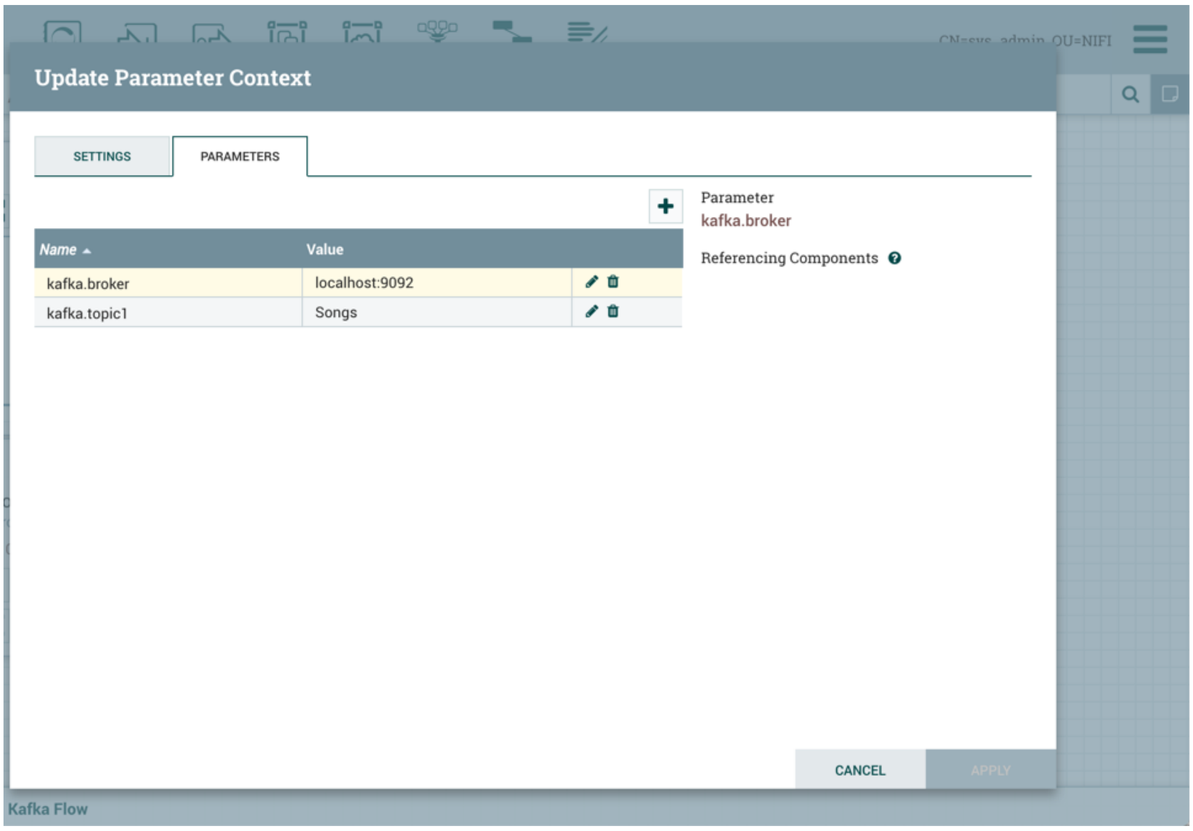
`#{abc}`	`xxx`	Simple substitution
`#{abc}/data`	`xxx/data`	Simple substitution with additional literal data
`#{abc}/#{def}`	`xxx/yyy`	Multiple substitution with additional literal data
`#{abc`	`#{abc`	No { } for parameter replacement
`#abc`	`#abc`	No { } for parameter replacement
`##{abc}`	`#{abc}`	Escaped # for literal interpretation
`###{abc}`	`#xxx`	Escaped # for literal interpretation, followed by simple substitution
`####{abc}`	`##{abc}`	Escaped # for literal interpretation, twice
`#####{abc}`	`##xxx`	Escaped # for literal interpretation, twice, followed by simple substitution
`#{abc/data}`	Exception thrown on property set operation	`/` not a valid parameter name character

When referencing a Parameter from within Expression Language, the Parameter reference is evaluated first. As an example, to replace `xxx` with `zzz` for the `abc` Parameter:

```
${ #{abc}:replace('xxx', 'zzz') }
```

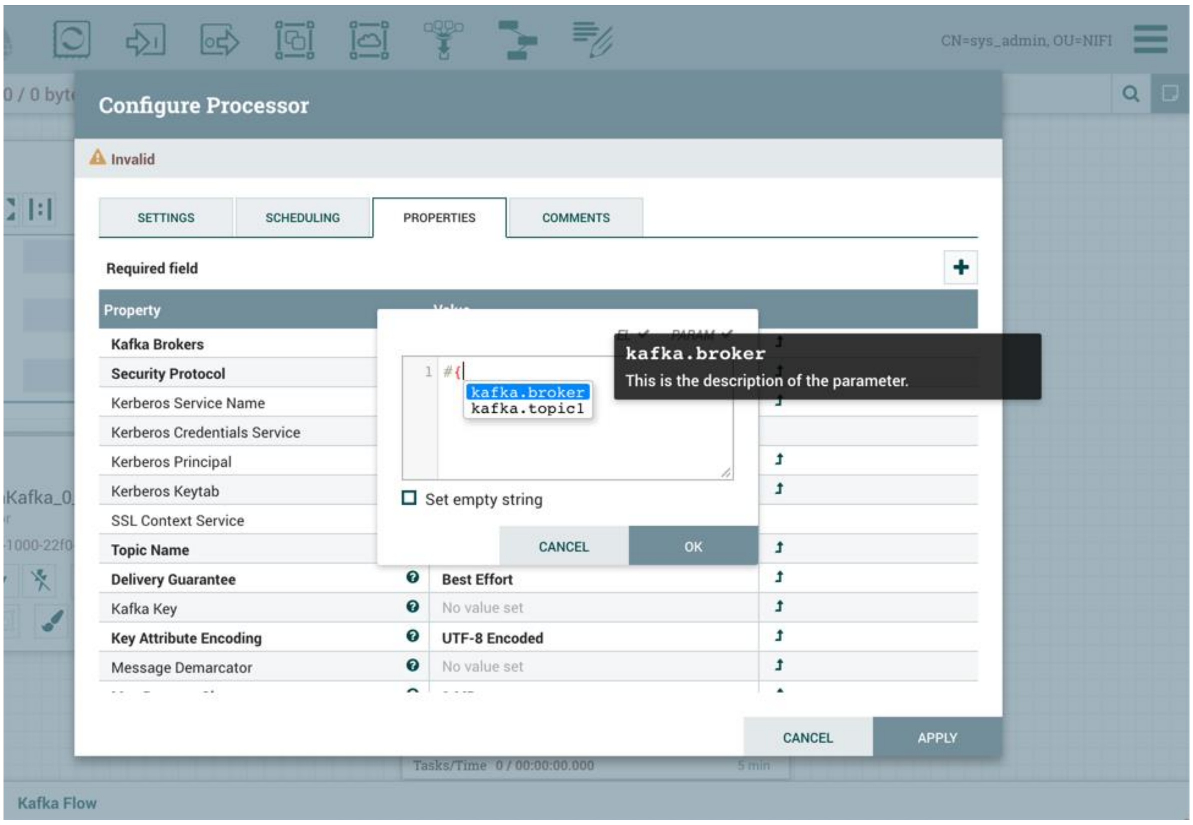
Referencing and Creating Parameters During Component Configuration

Parameters can be easily referenced or created as you configure the components in your flow. For example, assume a process group has the Parameter Context "Kafka Settings" assigned to it. "Kafka Settings" contains the parameters `kafka.broker` and `kafka.topic1`.



Name	Value
kafka.broker	localhost:9092
kafka.topic1	Songs

To reference `kafka.broker` as the value for the "Kafka Brokers" property in the PublishKafka processor, clear the default value and begin a new entry with the start delimiter `#{`. Next use the keystroke `control+space` to show the list of available parameters:


Configure Processor

Invalid

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Kafka Brokers	
Security Protocol	
Kerberos Service Name	
Kerberos Credentials Service	
Kerberos Principal	
Kerberos Keytab	
SSL Context Service	
Topic Name	
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

```
#{
kafka.broker
kafka.topic1
```

kafka.broker
This is the description of the parameter.

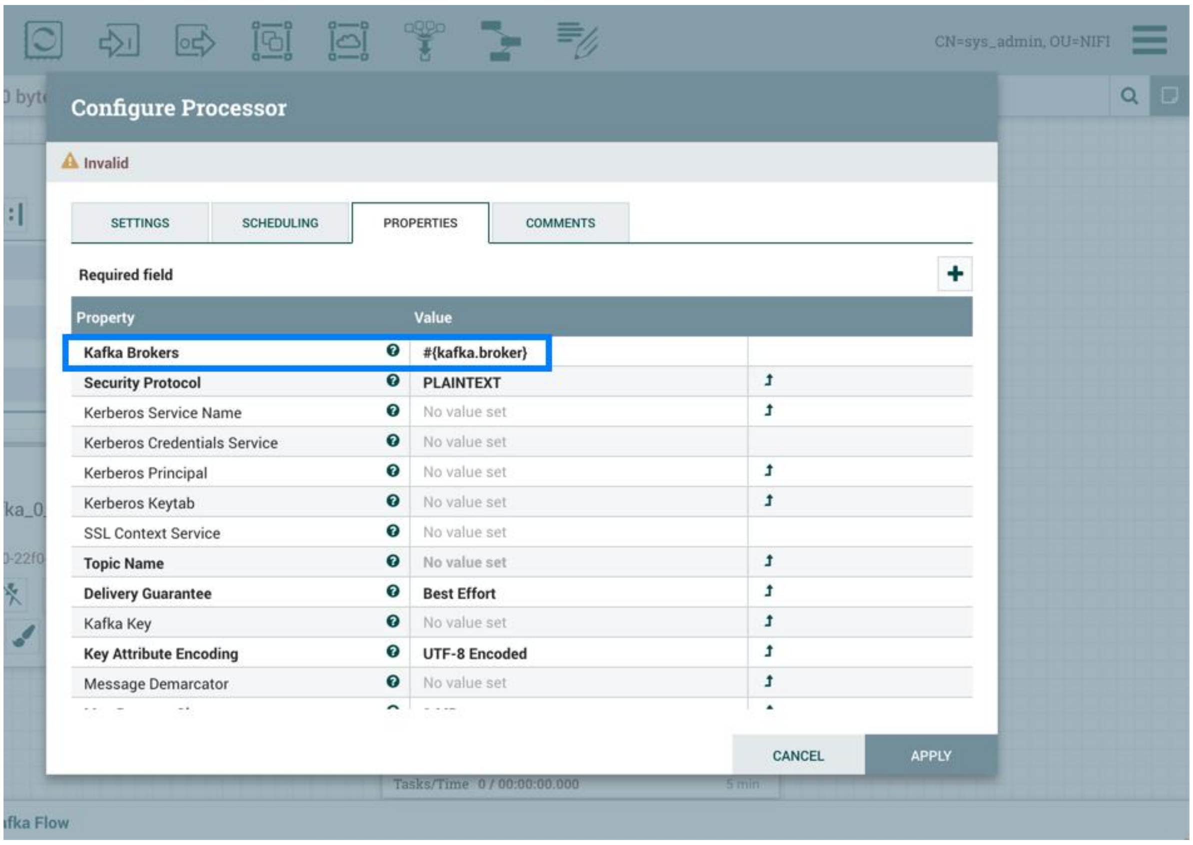
Set empty string

CANCEL OK

CANCEL APPLY

Kafka Flow

Select `kafka.broker` and complete the entry with a closing curly brace `}`.



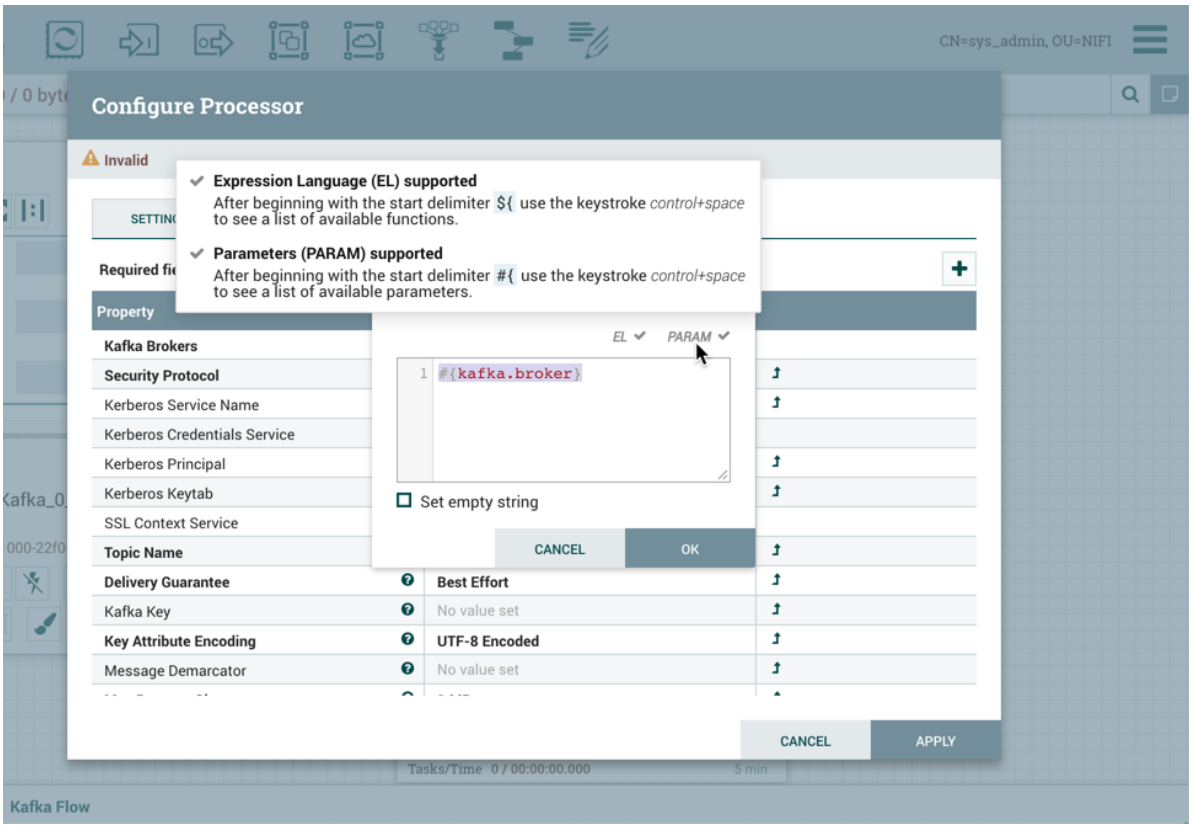
Configure Processor

Invalid

Required field

Property	Value
Kafka Brokers	#{kafka.broker}
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name	No value set
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

Help text describing this process is displayed when hovering over the Expression Language and Parameters eligibility indicators.



Configure Processor

Invalid

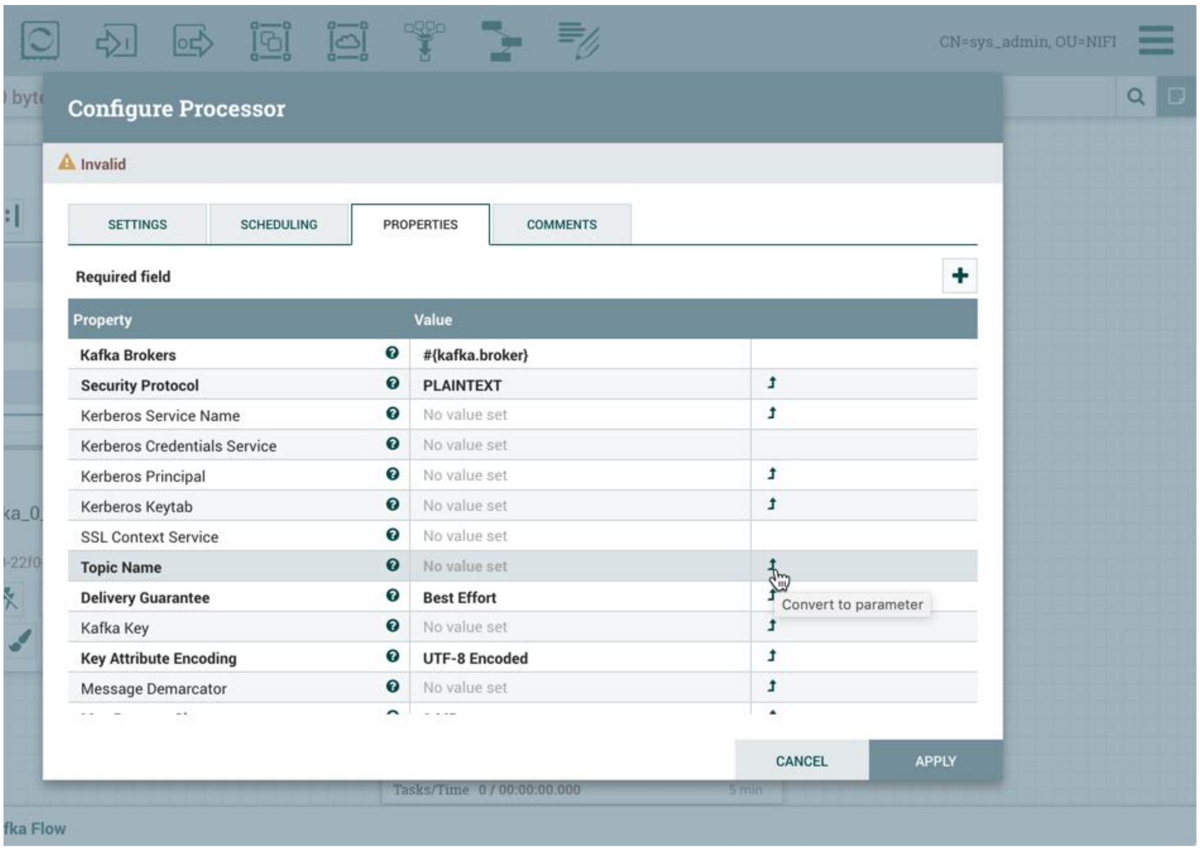
✓ **Expression Language (EL) supported**
After beginning with the start delimiter `${` use the keystroke *control+space* to see a list of available functions.

✓ **Parameters (PARAM) supported**
After beginning with the start delimiter `#{` use the keystroke *control+space* to see a list of available parameters.

```
#{kafka.broker}
```

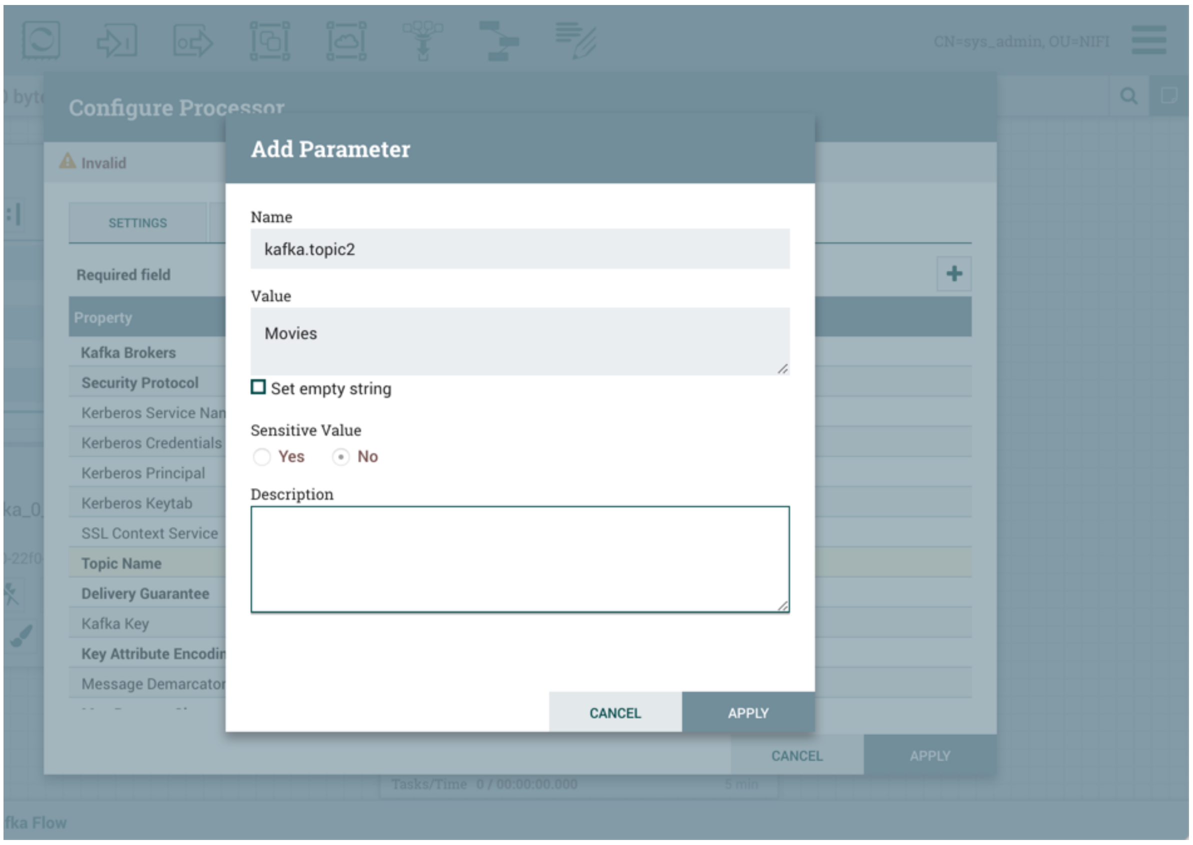
Property	Value
Kafka Brokers	
Security Protocol	
Kerberos Service Name	
Kerberos Credentials Service	
Kerberos Principal	
Kerberos Keytab	
SSL Context Service	
Topic Name	
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

Parameters can also be created on the fly. For example, to create a parameter for the "Topic Name" property, select the "Convert to Parameter" icon (↥) in that property's row. This icon will only be available if the user has appropriate permissions to modify the Parameter Context (see Accessing Parameters for more information).



Property	Value
Kafka Brokers	#{kafka.broker}
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name	No value set
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

The Add Parameter dialog will open. Configure the new parameter as desired.



Add Parameter

Name: kafka.topic2

Value: Movies

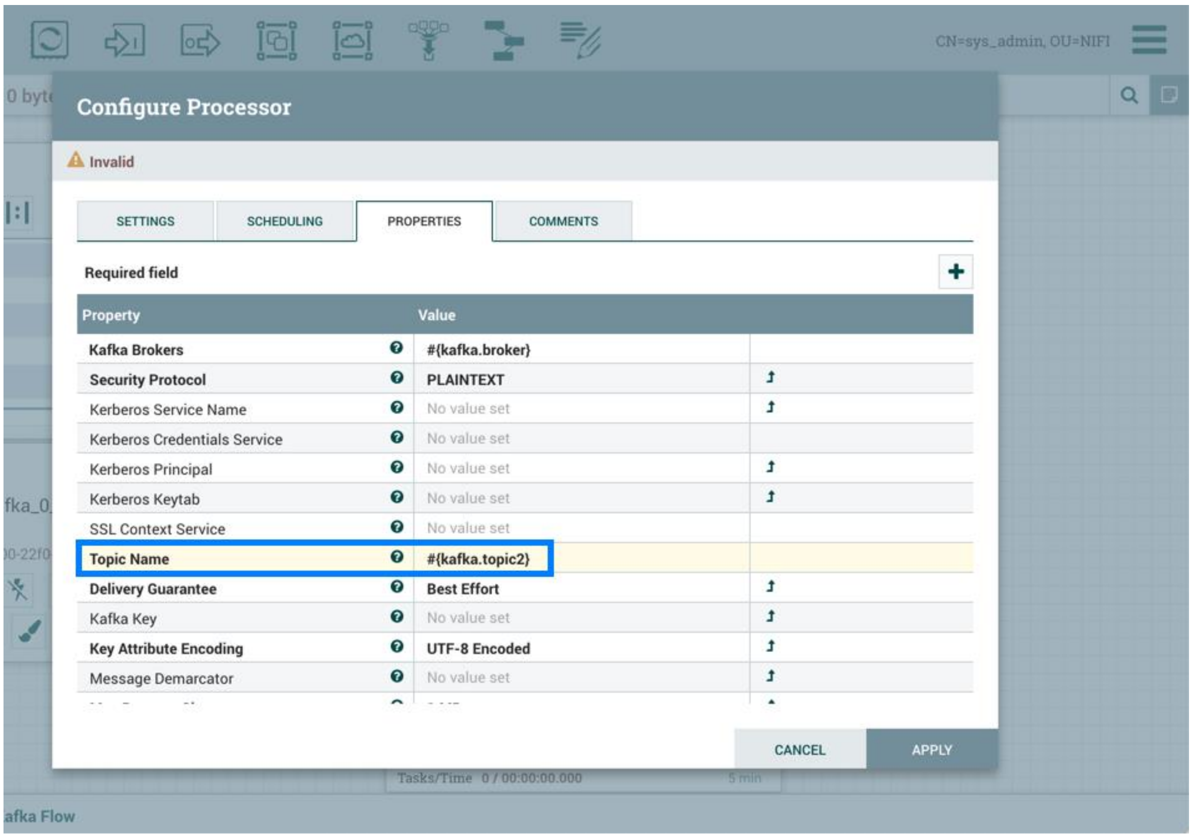
☐ Set empty string

Sensitive Value: Yes [✓ No]

Description:

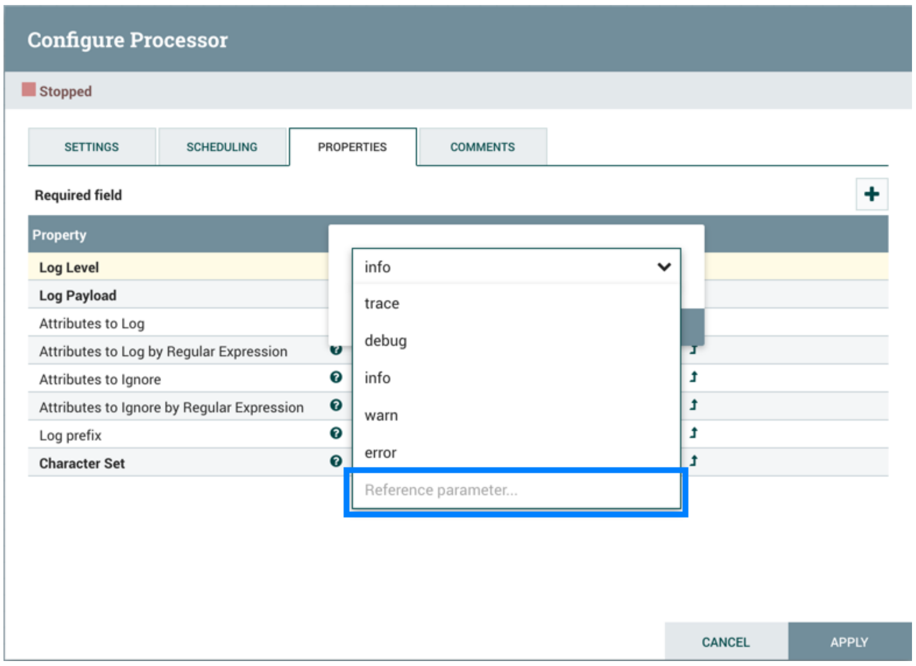
CANCEL APPLY

Select "Apply". The process group’s Parameter Context will be updated and the new parameter will be referenced by the property with the proper syntax applied automatically.

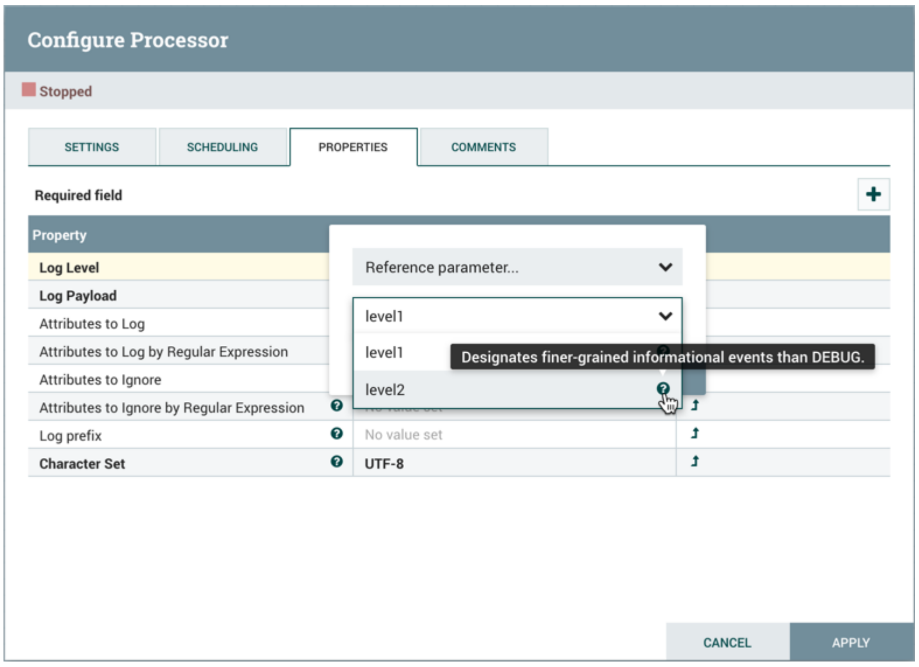


Property	Value
Kafka Brokers	#{kafka.broker}
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name	#{kafka.topic2}
Delivery Guarantee	Best Effort
Kafka Key	No value set
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

Properties values that are selectable can also reference parameters. In addition to applying the "Convert to Parameter" method described earlier, the option "Reference parameter.." is available in the value drop-down menu.



Selecting "Reference parameter…" will display a drop-down list of available parameters, determined by the parameter context assigned to the component's process group and the user's access policies.



Designates finer-grained informational events than DEBUG.

Hovering over the question mark icon (❓) displays the parameter's description.

Using Parameters with Sensitive Properties

Sensitive properties may only reference sensitive Parameters. This is important for versioned flows. The value of the sensitive Parameter itself will NOT be sent to the flow registry, only the fact that the property references the sensitive Parameter. For more information see Parameters in Versioned Flows.

The value of a sensitive property must be set to a single Parameter reference. For example, values of `#{password}123` and `#{password}#{suffix}` are not allowed. Sending `#{password}123` would lead to exposing part of the sensitive property's value. This is in contrast to a non-sensitive property, where a value such as `#{path}/child/file.txt` is valid.

Accessing Parameters

User privileges to Parameters are managed via access policies on the following levels:

- Parameter Context
- Process Group
- Component

For additional information on how to configure access policies, see the Access Policies section in the System Administrator's Guide.

Using Custom Properties with Expression Language

You can use Logbot ETL Expression Language to reference FlowFile attributes, compare them to other values, and manipulate their values when you are creating and configuring dataflows. For more information on Expression Language, see the Expression Language Guide.

In addition to using FlowFile attributes, system properties, and environment properties within Expression Language, you can also define custom properties for Expression Language use. Defining custom properties gives you more flexibility in handling and processing dataflows. You can also create custom properties for connection, server, and service properties, for easier dataflow configuration.

Logbot ETL properties have resolution precedence of which you should be aware when creating custom properties:

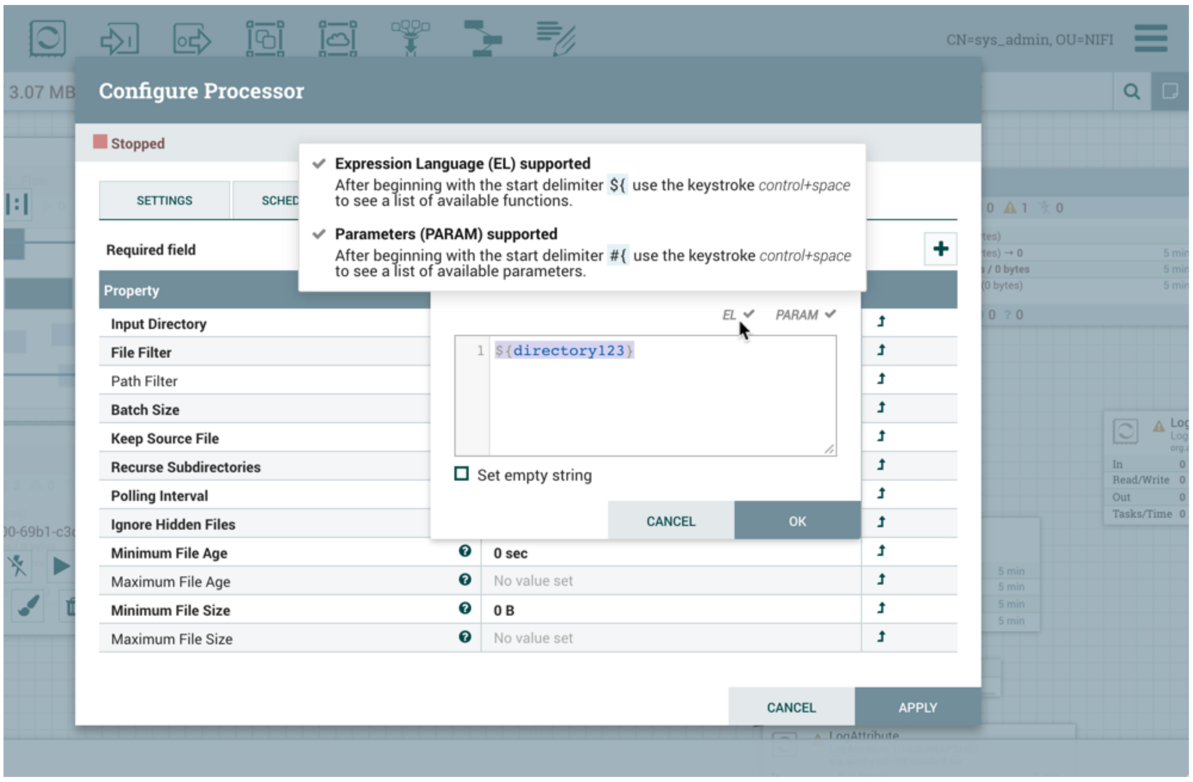
- Processor-specific attributes
- FlowFile properties
- FlowFile attributes
- From Variable Registry
    - User defined properties (custom properties)
    - System properties
    - Operating System environment variables

When you are creating custom properties, ensure that each custom property contains a distinct property value, so that it is not overridden by existing environment properties, system properties, or FlowFile attributes.

To use and manage custom properties with Expression Language you can use **Variables**:

Variables are created and configured within the Logbot ETL UI. They can be used in any field that supports Expression Language. Variables cannot be used for sensitive properties. Logbot ETL automatically picks up new or modified variables. Variables are defined at the Process Group level, as a result, the access policies for viewing and changing variables are derived from the access policies of the Process Group. See Variables for more information.

Expression Language support for a property is indicated in the UI.



✓ **Expression Language (EL) supported**
After beginning with the start delimiter `${` use the keystroke *control+space* to see a list of available functions.

✓ **Parameters (PARAM) supported**
After beginning with the start delimiter `#{` use the keystroke *control+space* to see a list of available parameters.

EL ✓ PARAM ✓

```
${directory123}
```

☐ Set empty string

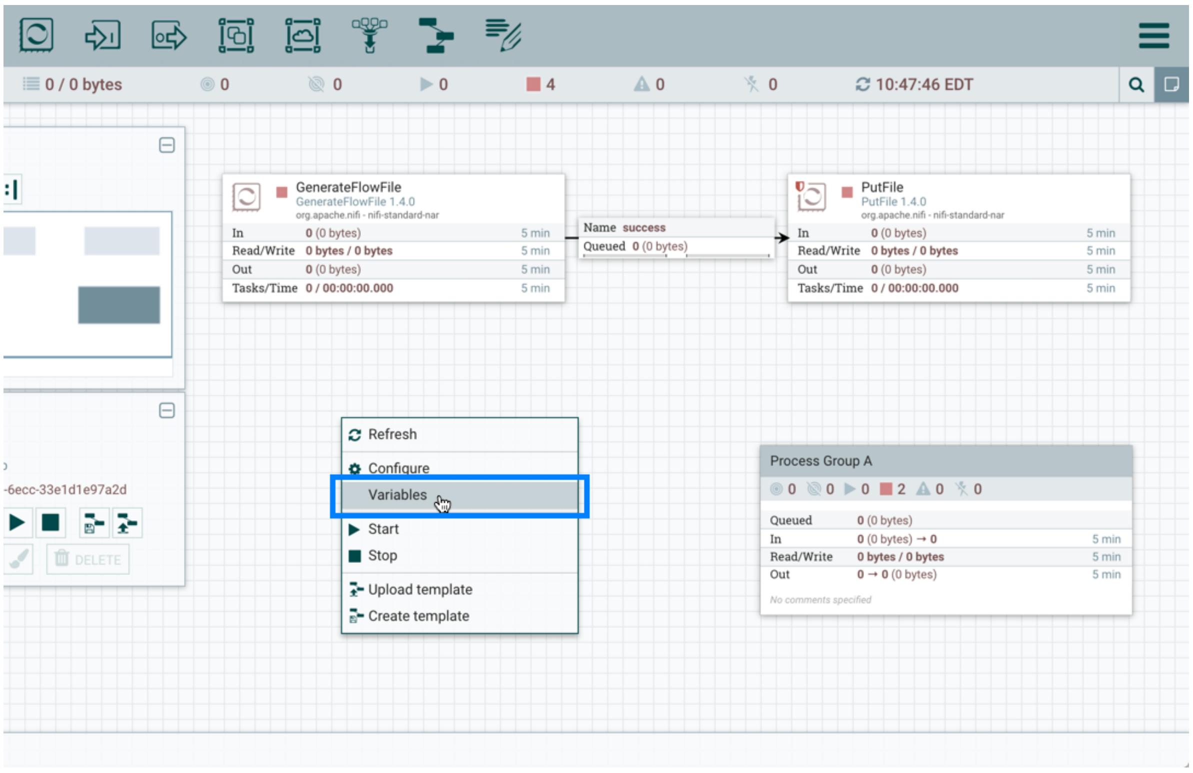
Property	Value
Input Directory	
File Filter	
Path Filter	
Batch Size	
Keep Source File	
Recurse Subdirectories	
Polling Interval	
Ignore Hidden Files	
Minimum File Age	0 sec
Maximum File Age	No value set
Minimum File Size	0 B
Maximum File Size	No value set

Variables

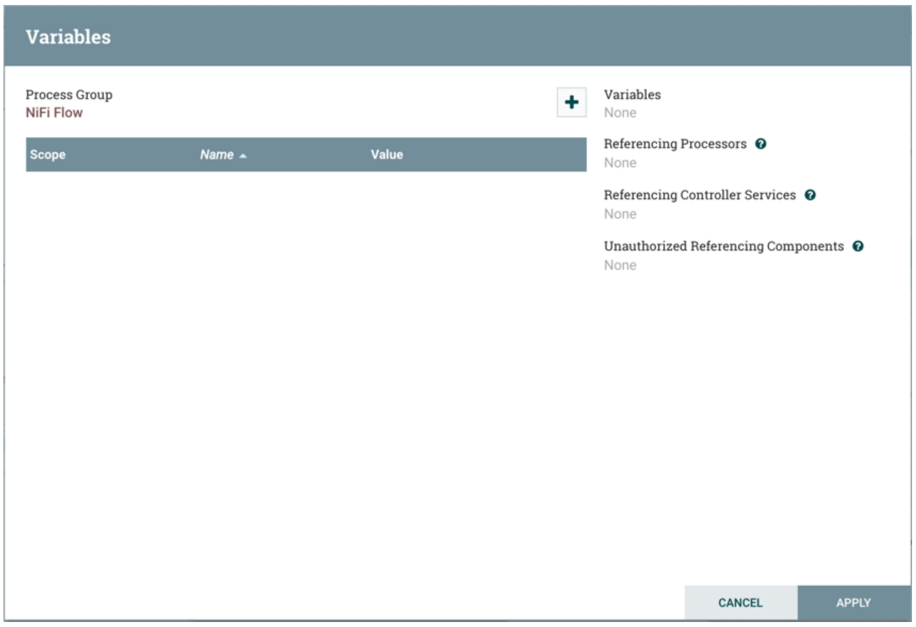
Variables are created and configured within the Logbot ETL UI. They can be used in any field that supports Expression Language. Variables cannot be used for sensitive properties. Variables are defined at the Process Group level, as a result, the access policies for viewing and changing variables are derived from the access policies of the Process Group. Variable values cannot reference other variables or make use of Expression Language.

Variables Window

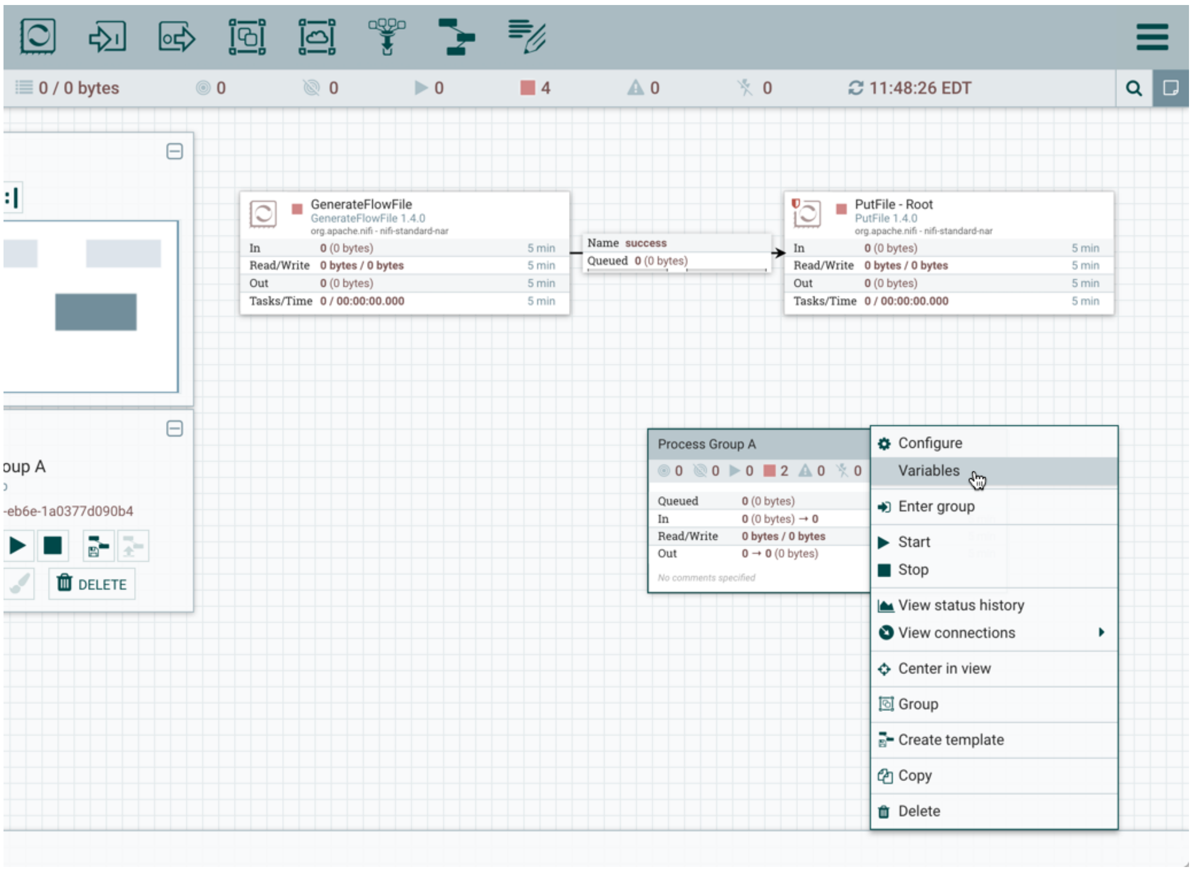
To access the Variables window, right-click on the canvas with nothing selected:



Select "Variables" from the Context Menu:

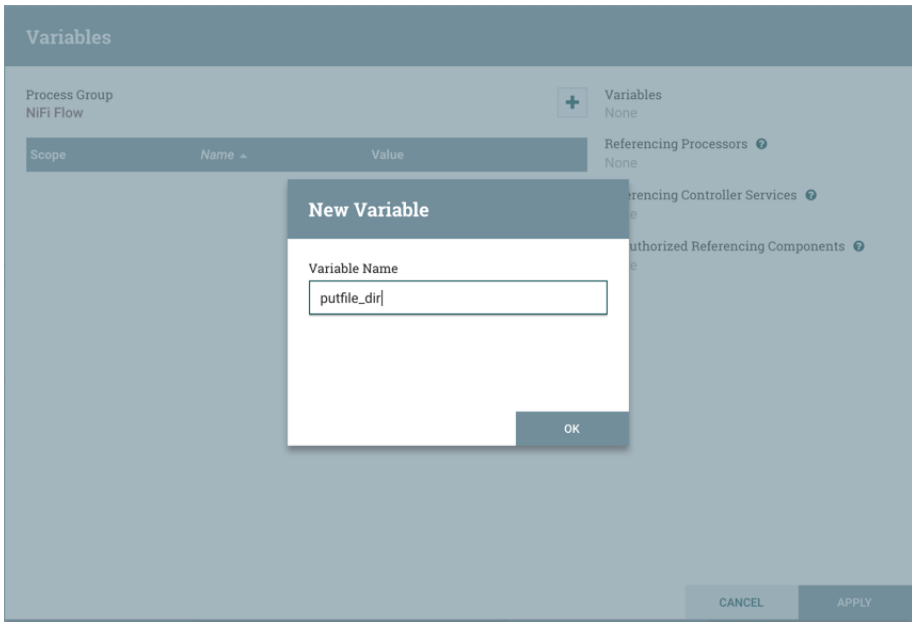


"Variables" is also available in the right-click Context Menu when a process group is selected:

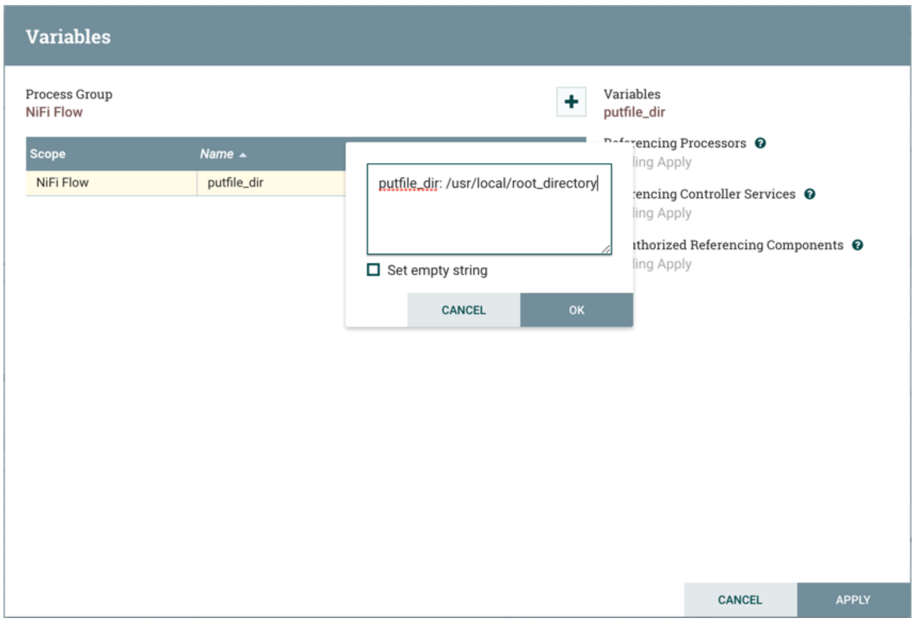


Creating a Variable

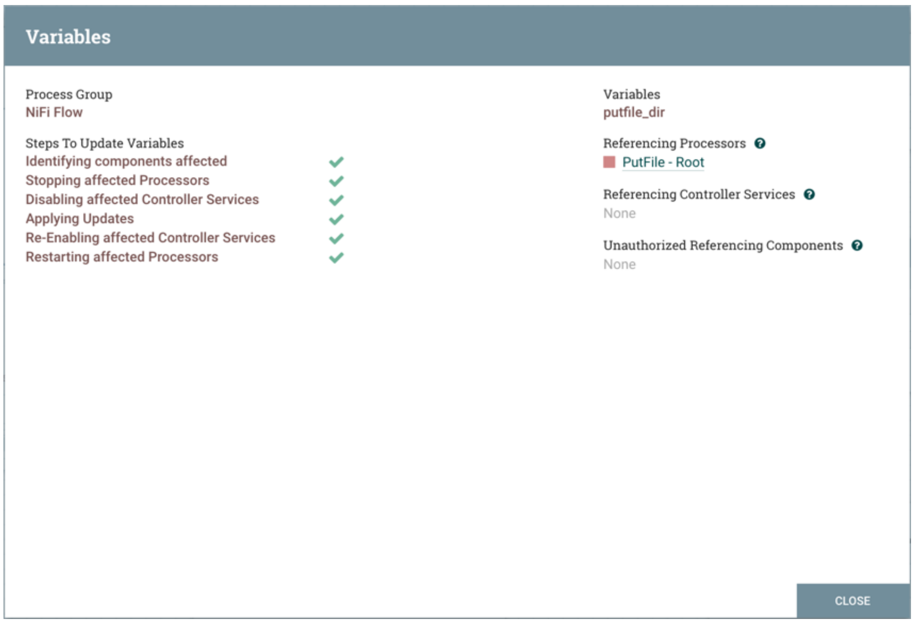
In the Variables window, click the + button to create a new variable. Add a name:



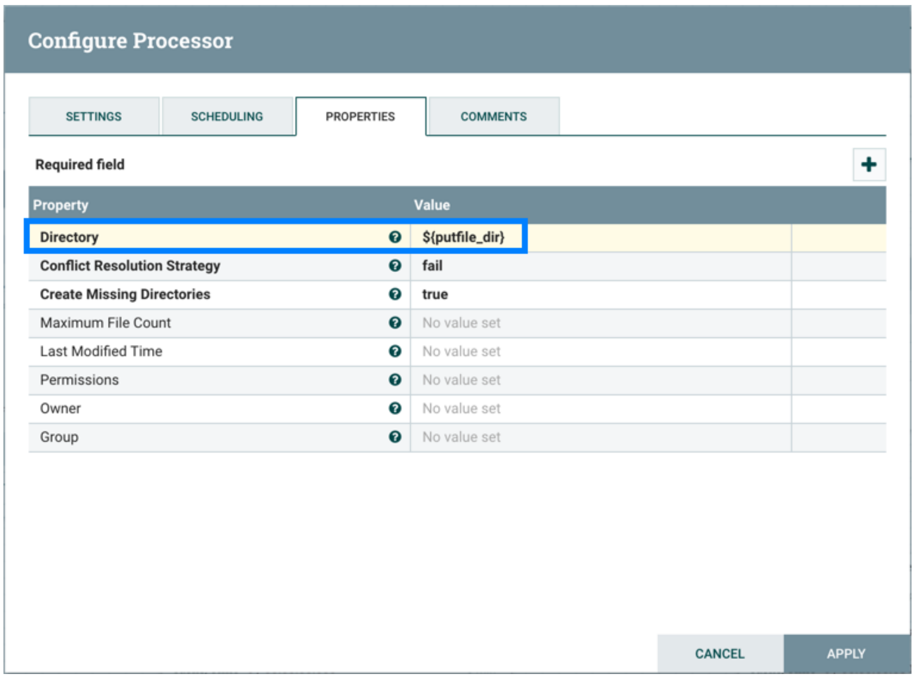
and a value:



Select "Apply":


Variables
Process Group
NiFi Flow
Steps To Update Variables
Identifying components affected ✓
Stopping affected Processors ✓
Disabling affected Controller Services ✓
Applying Updates ✓
Re-Enabling affected Controller Services ✓
Restarting affected Processors ✓
Variables
putfile_dir
Referencing Processors
PutFile - Root
Referencing Controller Services
None
Unauthorized Referencing Components
None
CLOSE

Steps to update the variable are performed (Identifying components affected, Stopping affected Processors, etc.). For example, the Referencing Processors section now lists the "PutFile-Root" processor. Selecting the name of the processor in the list will navigate to that processor on the canvas. Looking at the properties of the processor, `${putfile_dir}` is referenced by the Directory property:


Configure Processor

Required field

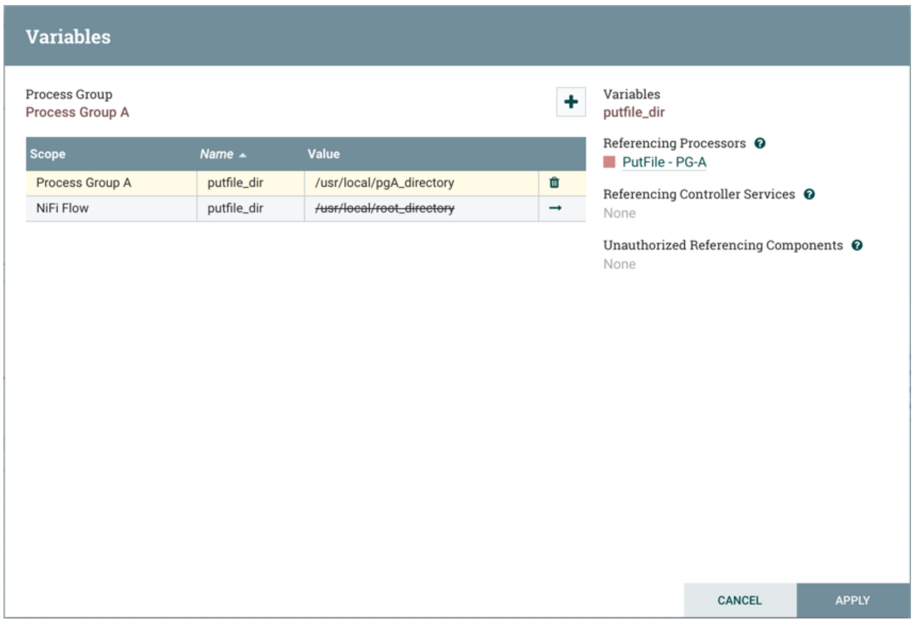
Property	Value
Directory	${putfile_dir}
Conflict Resolution Strategy	fail
Create Missing Directories	true
Maximum File Count	No value set
Last Modified Time	No value set
Permissions	No value set
Owner	No value set
Group	No value set

Variable Scope

Variables are scoped by the Process Group they are defined in and are available to any Processor defined at that level and below (i.e. any descendant Processors).

Variables in a descendant group override the value in a parent group. More specifically, if a variable `x` is declared at the root group and also declared inside a process group, components inside the process group will use the value of `x` defined in the process group.

For example, in addition to the `putfile_dir` variable that exists at the root process group, assume another `putfile_dir` variable was created within Process Group A. If one of the components within Process Group A references `putfile_dir`, both variables will be listed, but the `putfile_dir` from the root group will have a strikethrough indicating that is is being overridden:


Variables

Process Group
Process Group A

Scope	Name	Value
Process Group A	putfile_dir	/usr/local/pgA_directory
NiFi Flow	putfile_dir	~~/usr/local/root_directory~~

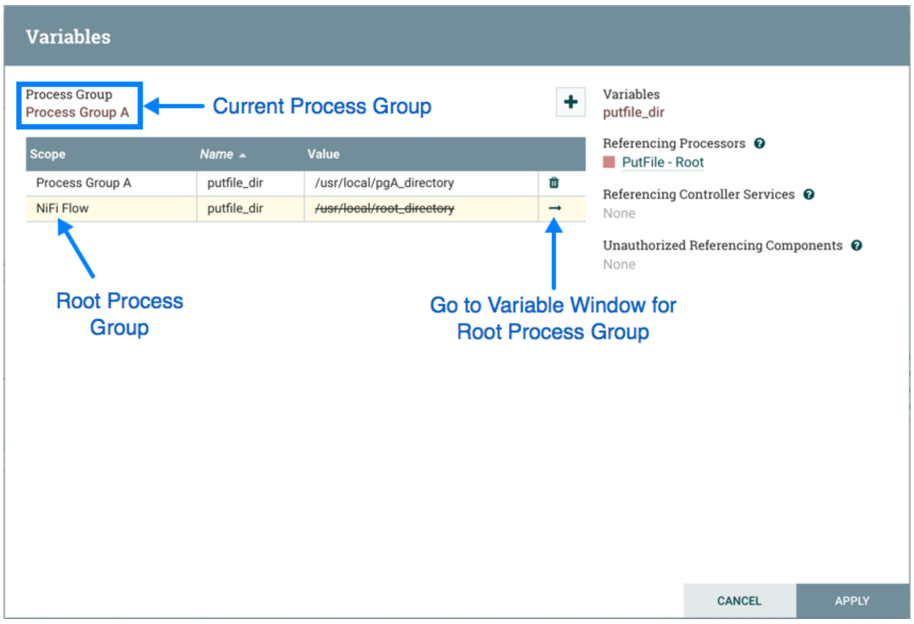
Variables
putfile_dir

Referencing Processors
🟥 PutFile - PG-A

Referencing Controller Services
None

Unauthorized Referencing Components
None

A variable can only be modified for the process group it was created in, which is listed at the top of the Variables window. To modify a variable defined in a different process group, select the "arrow" icon in that variable's row:



Scope	Name	Value
Process Group A	putfile_dir	/usr/local/pgA_directory
NiFi Flow	putfile_dir	~~/usr/local/root_directory~~

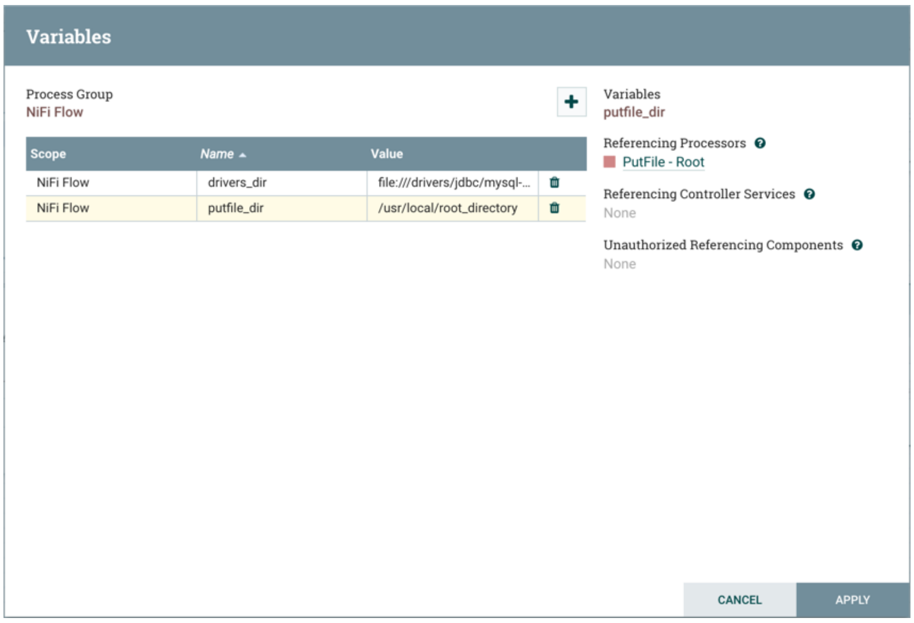
Variables
putfile_dir

Referencing Processors
🟥 PutFile - Root

Referencing Controller Services
None

Unauthorized Referencing Components
None

which will navigate to the Variables window for that process group:



Scope	Name	Value
NiFi Flow	drivers_dir	file:///drivers/jdbc/mysql-...
NiFi Flow	putfile_dir	/usr/local/root_directory

Variables
putfile_dir

Referencing Processors
🟥 PutFile - Root

Referencing Controller Services
None

Unauthorized Referencing Components
None

Controller Services

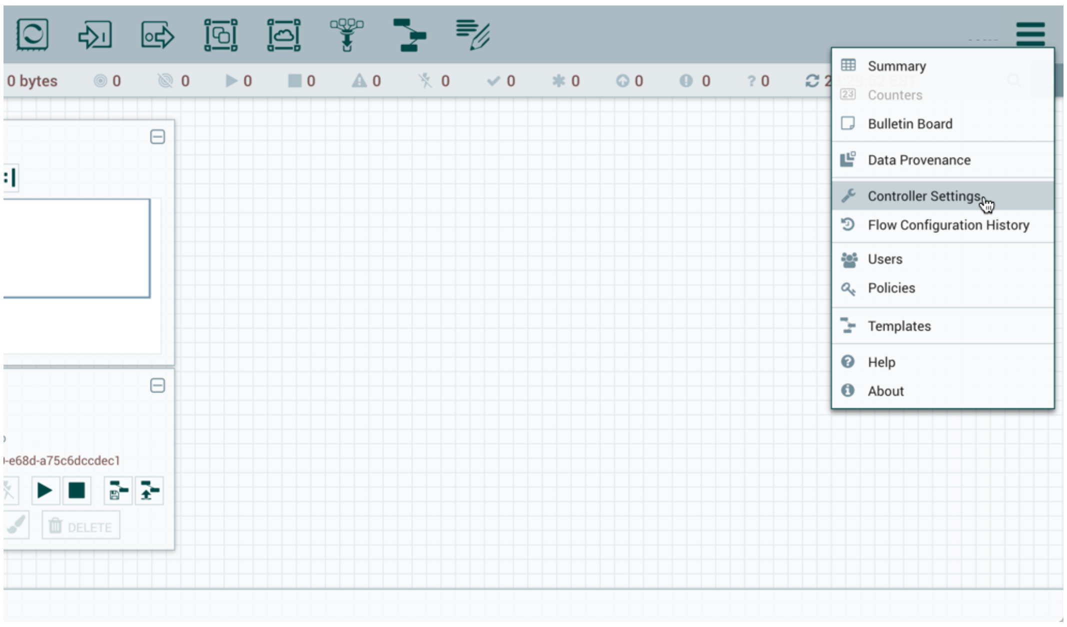
Controller Services are shared services that can be used by reporting tasks, processors, and other services to utilize for configuration or task execution.

Controller Services defined on the controller level are limited to reporting tasks and other services defined there. Controller Services for use by processors in your dataflow must be defined in the configuration of the root process group or sub-process group(s) where they will be used.

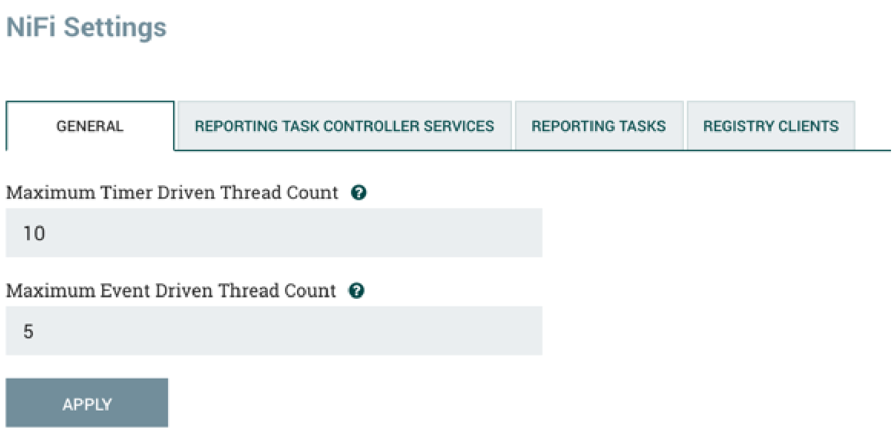
If your Logbot ETL instance is secured, your ability to view and add Controller Services is dependent on the privileges assigned to you. If you do not have access to one or more Controller Services, you are not able to see or access it in the UI. Access privileges can be assigned on a global or Controller Service-specific basis (see Accessing the UI with Multi-Tenant Authorization for more information).

Adding Controller Services for Reporting Tasks

To add a Controller Service for a reporting task, select Controller Settings from the Global Menu.



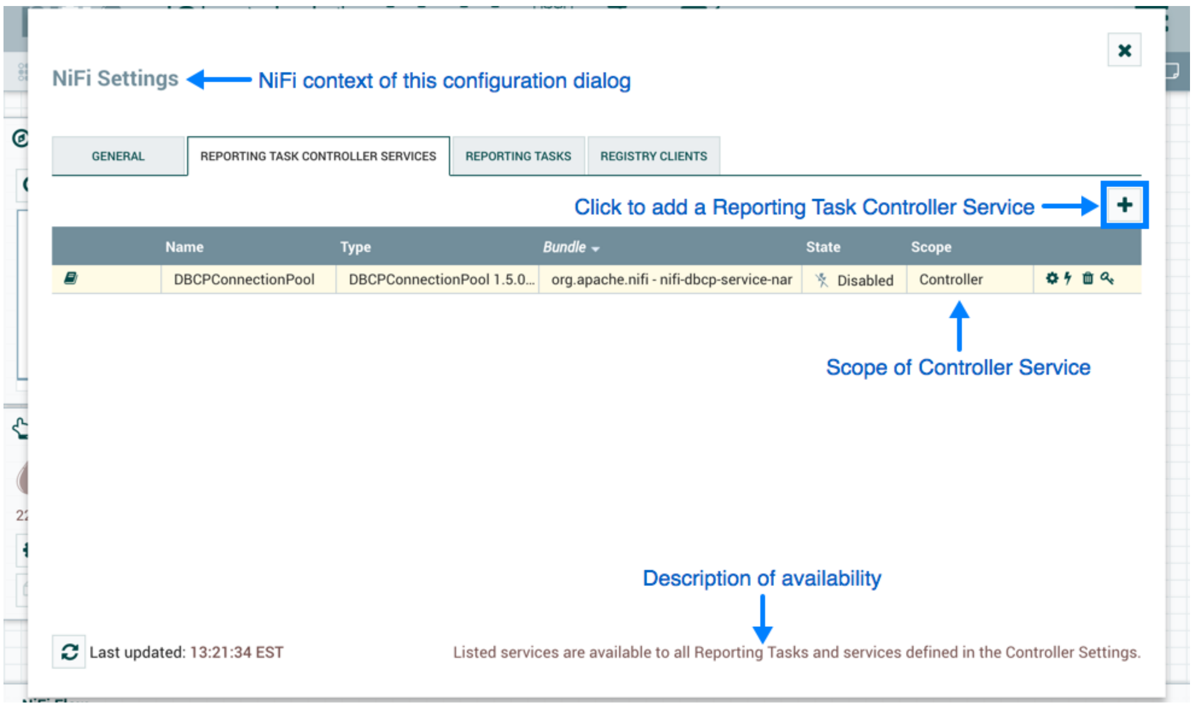
This displays the Logbot ETL Settings window. The window has four tabs: General, Reporting Task Controller Services, Reporting Tasks and Registry Clients. The General tab provides settings for the overall maximum thread counts of the instance.


NiFi Settings

Maximum Timer Driven Thread Count: 10

Maximum Event Driven Thread Count: 5

To the right of the General tab is the Reporting Task Controller Services tab. From this tab, the DFM may click the + button in the upper-right corner to create a new Controller Service.

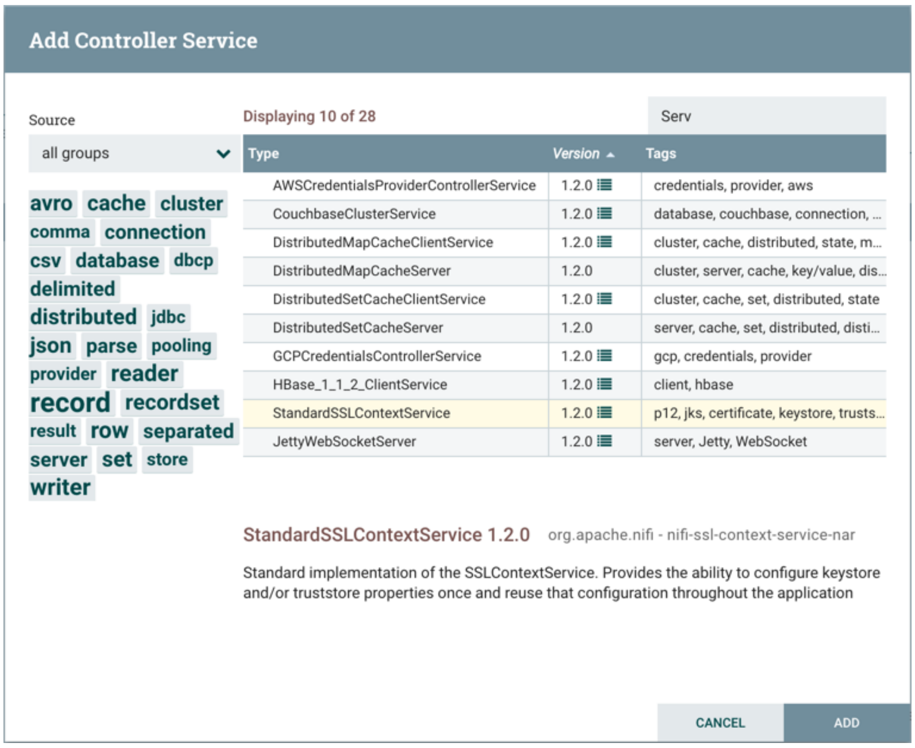

NiFi Settings
NiFi context of this configuration dialog
GENERAL
REPORTING TASK CONTROLLER SERVICES
REPORTING TASKS
REGISTRY CLIENTS
Click to add a Reporting Task Controller Service
Scope of Controller Service
Description of availability

Name	Type	Bundle	State	Scope
DBCPConnectionPool	DBCPConnectionPool 1.5.0...	org.apache.nifi - nifi-dbcp-service-nar	Disabled	Controller

Last updated: 13:21:34 EST

Listed services are available to all Reporting Tasks and services defined in the Controller Settings.

The Add Controller Service window opens. This window is similar to the Add Processor window. It provides a list of the available Controller Services on the right and a tag cloud, showing the most common category tags used for Controller Services, on the left. The DFM may click any tag in the tag cloud in order to narrow down the list of Controller Services to those that fit the categories desired. The DFM may also use the Filter field at the top-right of the window to search for the desired Controller Service or use the Source drop-down at the top-left to filter the list by the group who created them. Upon selecting a Controller Service from the list, the DFM can see a description of the service below. Select the desired controller service and click Add, or simply double-click the name of the service to add it.


Add Controller Service

Source

all groups

avro cache cluster comma connection csv database dbcp delimited distributed jdbc json parse pooling provider reader record recordset result row separated server set store writer

Displaying 10 of 28

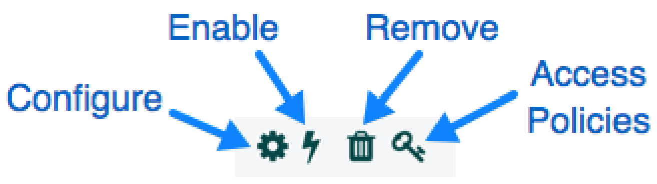
Serv

Type	Version	Tags
AWSCredentialsProviderControllerService	1.2.0	credentials, provider, aws
CouchbaseClusterService	1.2.0	database, couchbase, connection, ...
DistributedMapCacheClientService	1.2.0	cluster, cache, distributed, state, m...
DistributedMapCacheServer	1.2.0	cluster, server, cache, key/value, dis...
DistributedSetCacheClientService	1.2.0	cluster, cache, set, distributed, state
DistributedSetCacheServer	1.2.0	server, cache, set, distributed, disti...
GCPCredentialsControllerService	1.2.0	gcp, credentials, provider
HBase_1_1_2_ClientService	1.2.0	client, hbase
StandardSSLContextService	1.2.0	p12, jks, certificate, keystore, trusts...
JettyWebSocketServer	1.2.0	server, Jetty, WebSocket

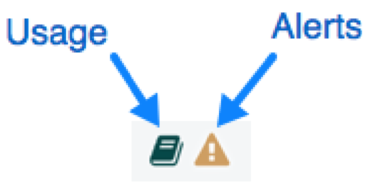
StandardSSLContextService 1.2.0 org.apache.nifi - nifi-ssl-context-service-nar

Standard implementation of the SSLContextService. Provides the ability to configure keystore and/or truststore properties once and reuse that configuration throughout the application

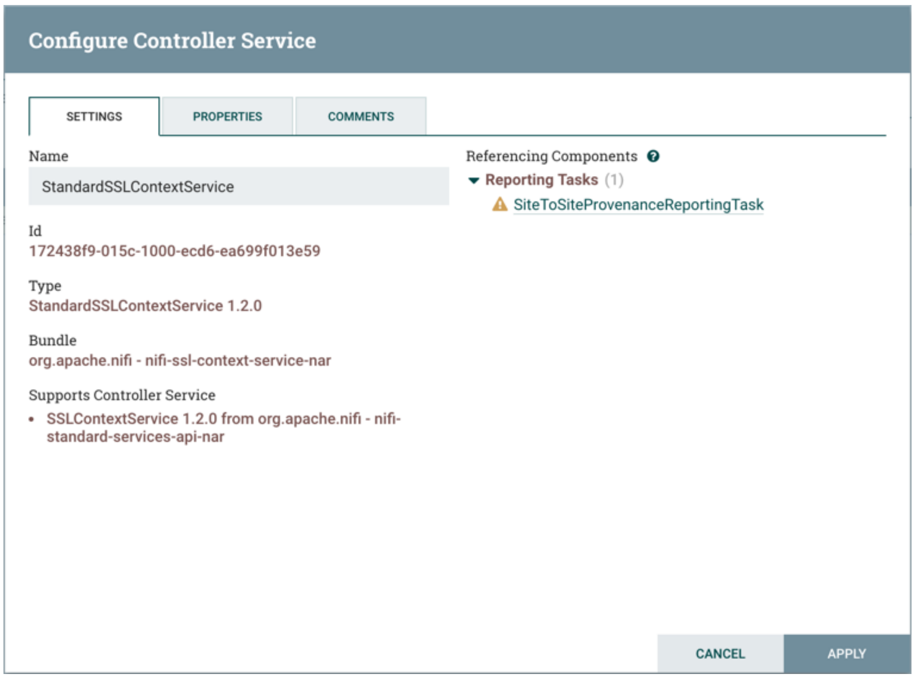
Once you have added a Controller Service, you can configure it by clicking the "Configure" button in the far-right column. Other buttons in this column include "Enable", "Remove" and "Access Policies".



You can obtain information about Controller Services by clicking the "Usage" and "Alerts" buttons in the left-hand column.



When the DFM clicks the "Configure" button, a Configure Controller Service window opens. It has three tabs: Settings, Properties,and Comments. This window is similar to the Configure Processor window. The Settings tab provides a place for the DFM to give the Controller Service a unique name (if desired). It also lists the UUID, Type, Bundle and Support information for the service and provides a list of other components (reporting tasks or other controller services) that reference the service.



Configure Controller Service

SETTINGS | PROPERTIES | COMMENTS

Name
StandardSSLContextService

Id
172438f9-015c-1000-ecd6-ea699f013e59

Type
StandardSSLContextService 1.2.0

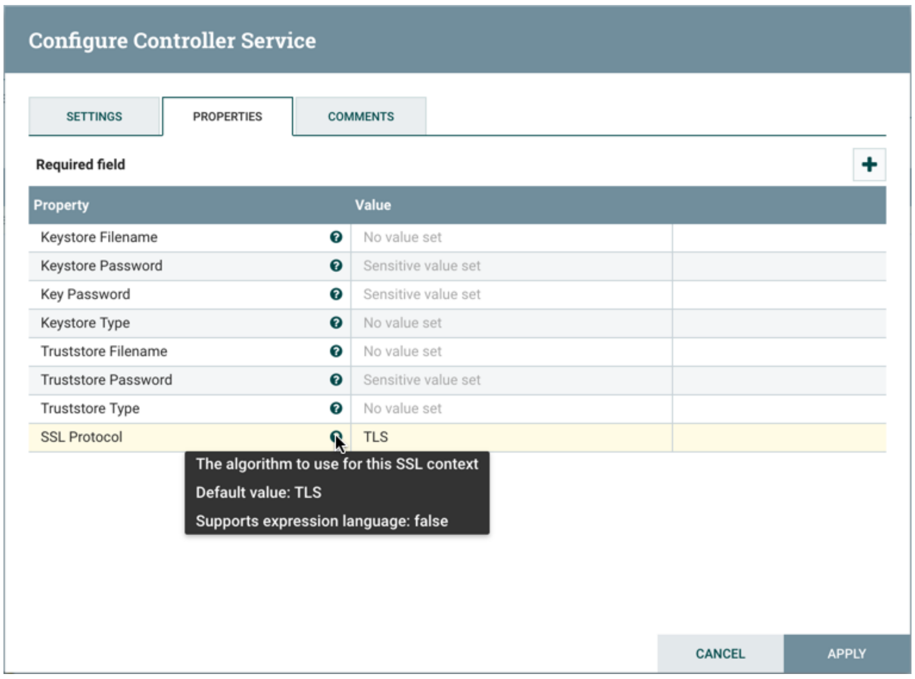
Bundle
org.apache.nifi - nifi-ssl-context-service-nar

Supports Controller Service
- SSLContextService 1.2.0 from org.apache.nifi - nifi-standard-services-api-nar

Referencing Components
Reporting Tasks (1)
SiteToSiteProvenanceReportingTask

CANCEL | APPLY

The Properties tab lists the various properties that apply to the particular controller service. As with configuring processors, the DFM may hover over the question mark icons to see more information about each property.



Configure Controller Service

SETTINGS | PROPERTIES | COMMENTS

Required field

Property	Value
Keystore Filename	No value set
Keystore Password	Sensitive value set
Key Password	Sensitive value set
Keystore Type	No value set
Truststore Filename	No value set
Truststore Password	Sensitive value set
Truststore Type	No value set
SSL Protocol	TLS

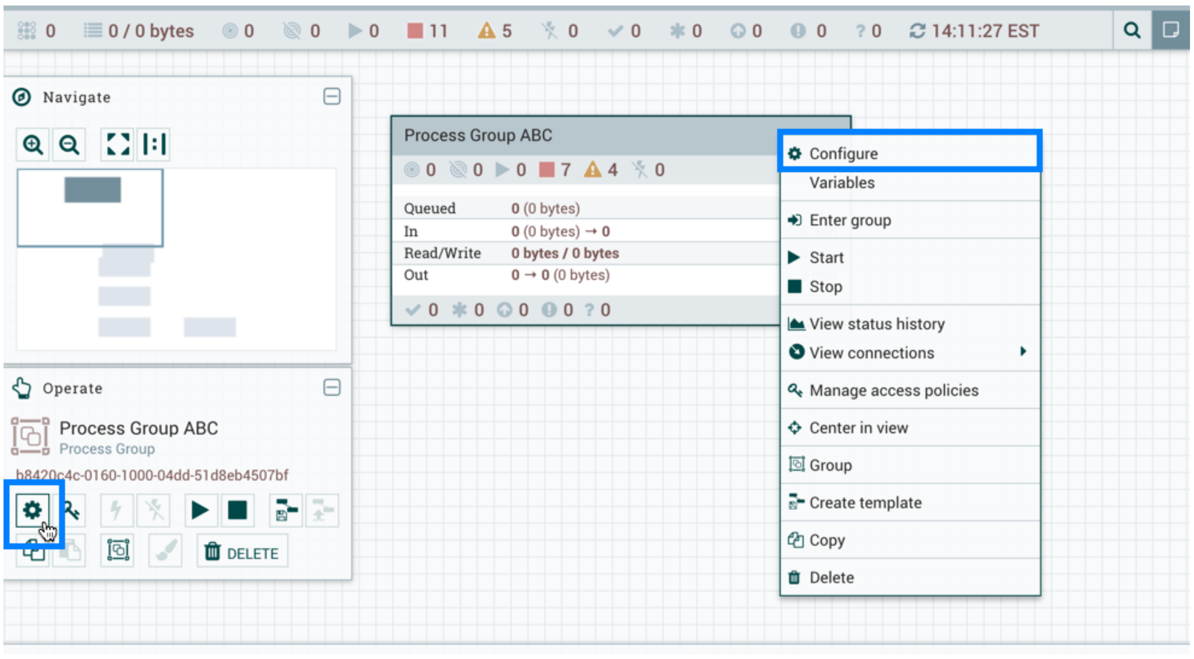
The algorithm to use for this SSL context
Default value: TLS
Supports expression language: false

CANCEL | APPLY

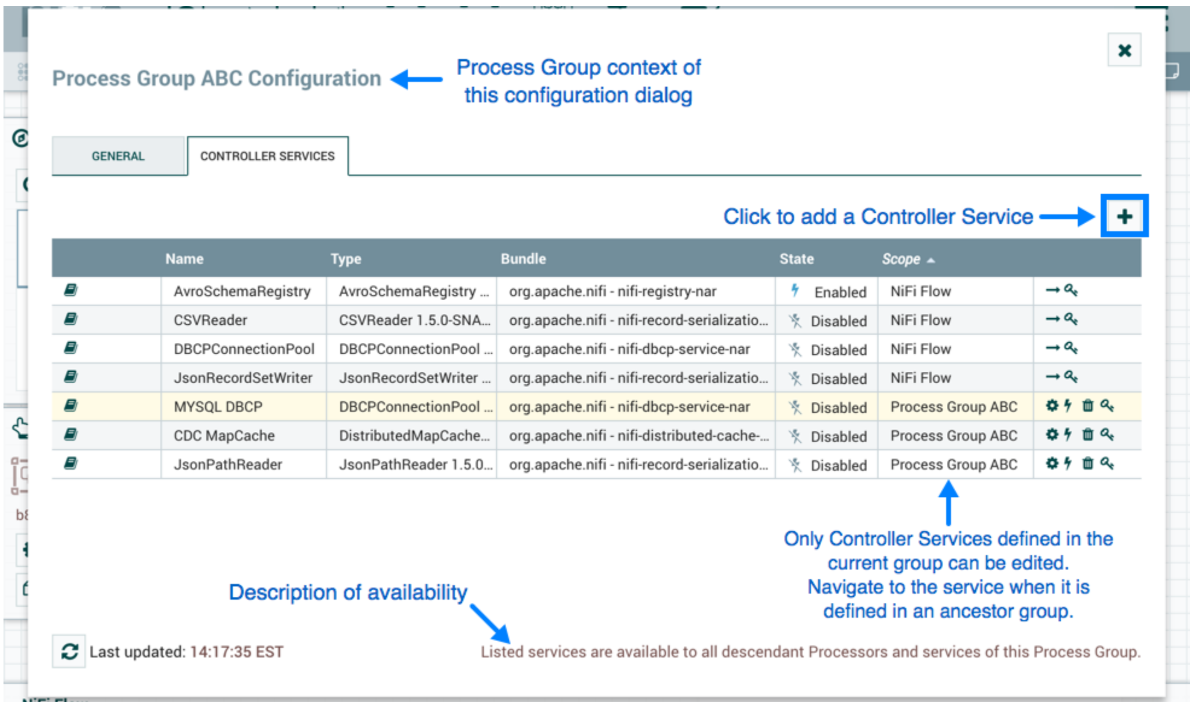
The Comments tab is just an open-text field, where the DFM may include comments about the service. After configuring a Controller Service, click "Apply" to save the configuration and close the window, or click "Cancel" to discard the changes and close the window.

Adding Controller Services for Dataflows

To add a Controller Service for a dataflow, you can either right click a Process Group and select Configure, or click Configure from the Operate Palette.



When you click Configure from the Operate Palette with nothing selected on your canvas, you add a Controller Service for your Root Process Group. That Controller Service is then available to all nested Process Groups in your dataflow. When you select a Process Group on the canvas and then click Configure from either the Operate Palette or the Process Group context menu, the service will be available to all Processors and Controller Services defined in that Process Group and below.

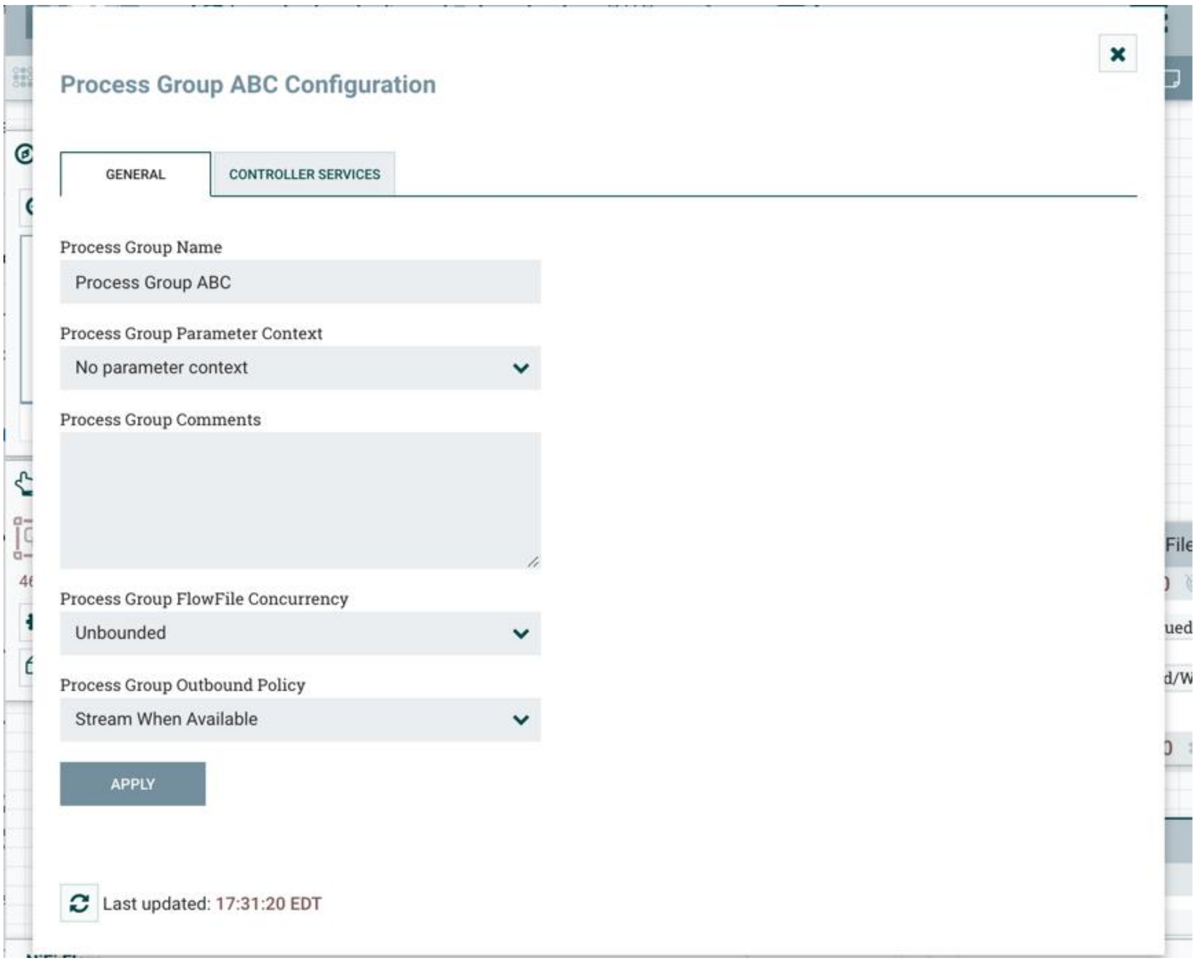

Process Group ABC Configuration
Process Group context of this configuration dialog
GENERAL
CONTROLLER SERVICES
Click to add a Controller Service
Only Controller Services defined in the current group can be edited. Navigate to the service when it is defined in an ancestor group.
Description of availability
Last updated: 14:17:35 EST

	Name	Type	Bundle	State	Scope	
	AvroSchemaRegistry	AvroSchemaRegistry ...	org.apache.nifi - nifi-registry-nar	Enabled	NiFi Flow	
	CSVReader	CSVReader 1.5.0-SNA...	org.apache.nifi - nifi-record-serializatio...	Disabled	NiFi Flow	
	DBCPConnectionPool	DBCPConnectionPool ...	org.apache.nifi - nifi-dbcp-service-nar	Disabled	NiFi Flow	
	JsonRecordSetWriter	JsonRecordSetWriter ...	org.apache.nifi - nifi-record-serializatio...	Disabled	NiFi Flow	
	MYSQL DBCP	DBCPConnectionPool ...	org.apache.nifi - nifi-dbcp-service-nar	Disabled	Process Group ABC	
	CDC MapCache	DistributedMapCache...	org.apache.nifi - nifi-distributed-cache-...	Disabled	Process Group ABC	
	JsonPathReader	JsonPathReader 1.5.0...	org.apache.nifi - nifi-record-serializatio...	Disabled	Process Group ABC	

Listed services are available to all descendant Processors and services of this Process Group.

Use the following steps to add a Controller Service:

1. Click Configure, either from the Operate Palette, or from the Process Group context menu. This displays the process group Configuration window. The window has two tabs: General and Controller Services. The General Tab is for settings that pertain to general information about the process group.


Process Group ABC Configuration

GENERAL | CONTROLLER SERVICES

Process Group Name: Process Group ABC

Process Group Parameter Context: No parameter context

Process Group Comments:

Process Group FlowFile Concurrency: Unbounded

Process Group Outbound Policy: Stream When Available

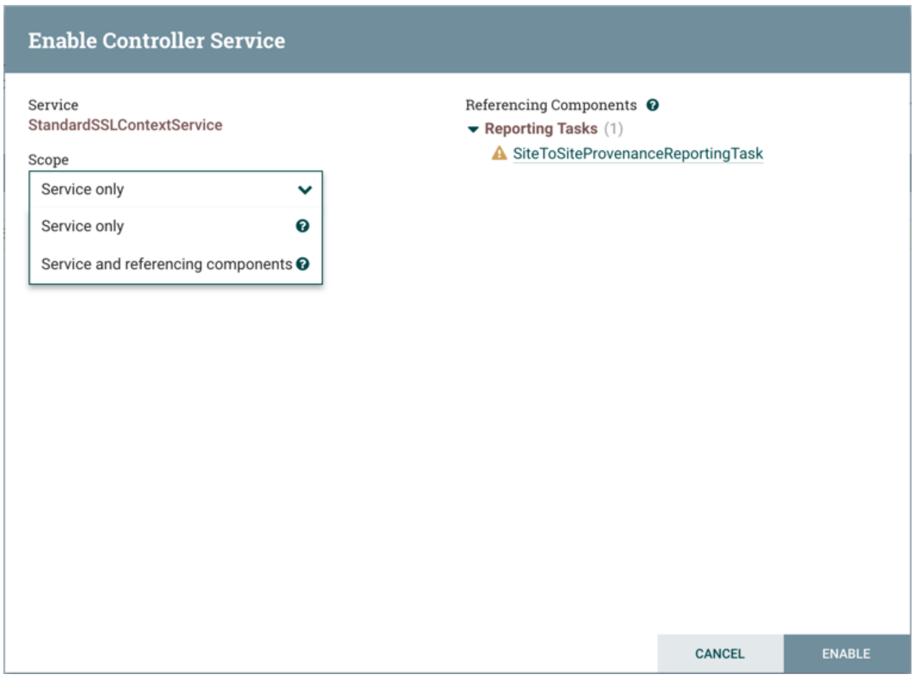
APPLY

Last updated: 17:31:20 EDT

2. From the Process Group Configuration page, select the Controller Services tab.
3. Click the + button to display the Add Controller Service dialog.
4. Select the Controller Service desired, and click Add.
5. Perform any necessary Controller Service configuration tasks by clicking the Configure icon ( ) in the right-hand column.

Enabling/Disabling Controller Services

After a Controller Service has been configured, it must be enabled in order to run. Do this using the "Enable" button ( ) in the far-right column of the Controller Services tab. In order to modify an existing/running controller service, the DFM needs to stop/disable it (as well as all referencing reporting tasks and controller services). Do this using the "Disable" button ( ). Rather than having to hunt down each component that is referenced by that controller service, the DFM has the ability to stop/disable them when disabling the controller service in question. When enabling a controller service, the DFM has the option to either start/enable the controller service and all referencing components or start/enable only the controller service itself.


Enable Controller Service

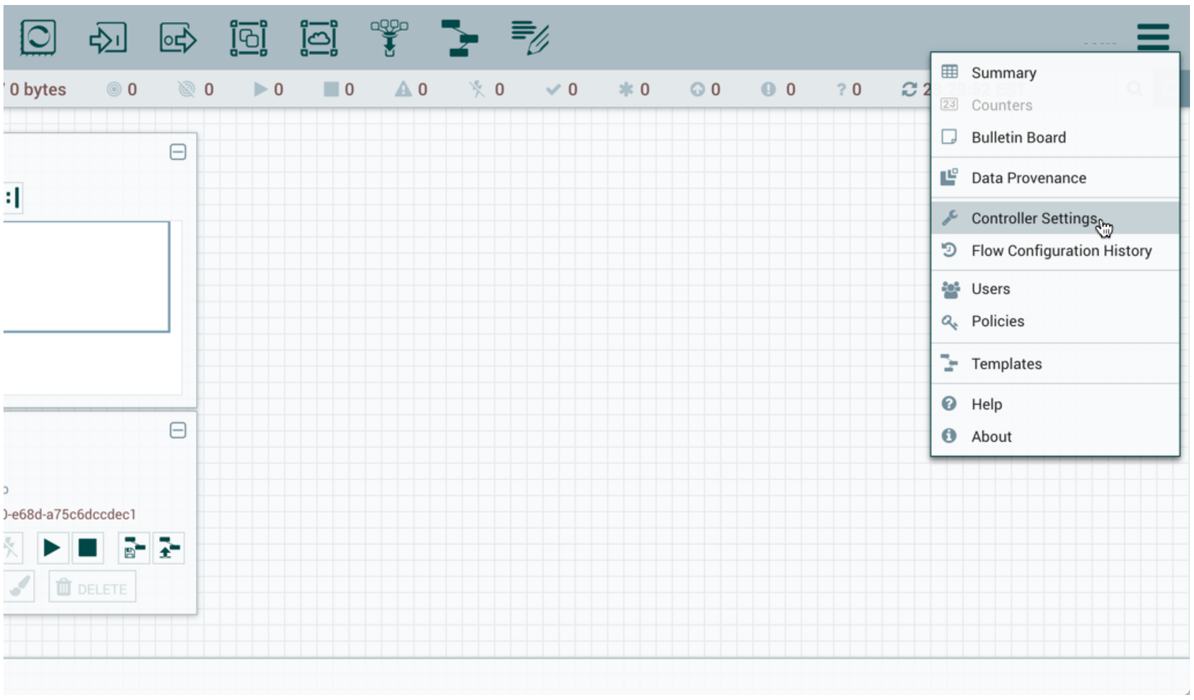
Service
StandardSSLContextService

Scope
Service only
Service only
Service and referencing components

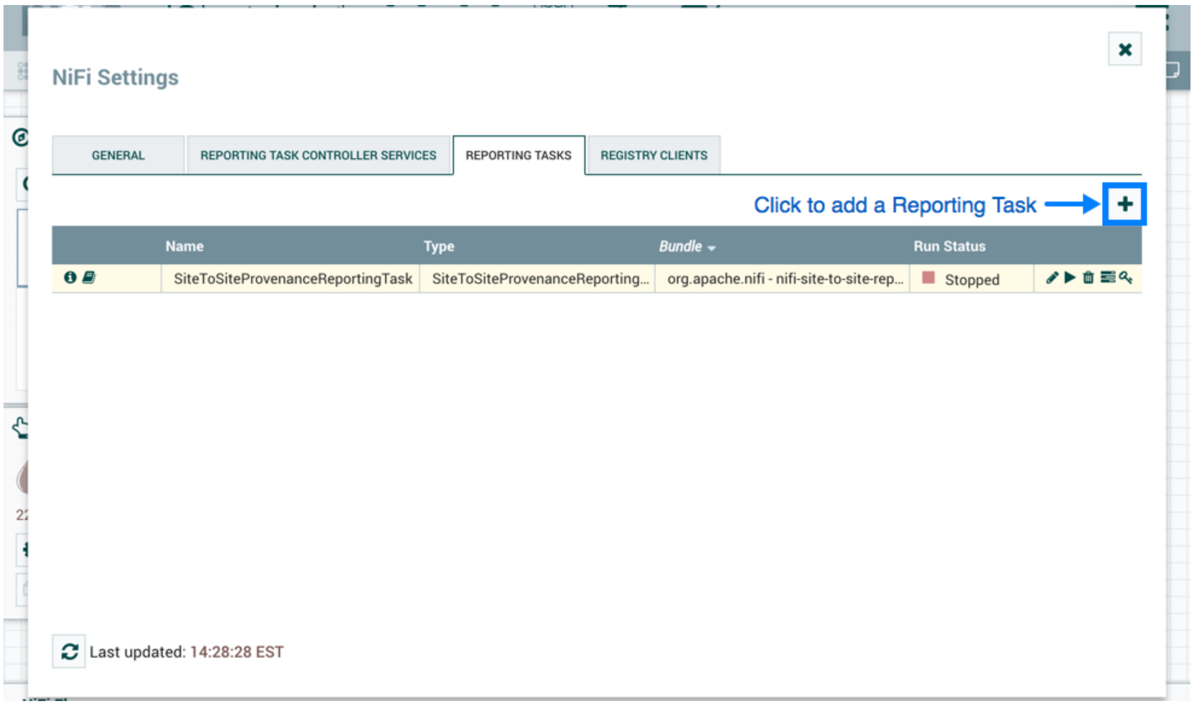
Referencing Components
Reporting Tasks (1)
SiteToSiteProvenanceReportingTask

Reporting Tasks

Reporting Tasks run in the background to provide statistical reports about what is happening in the Logbot ETL instance. The DFM adds and configures Reporting Tasks similar to the process for Controller Services. To add a Reporting Task, select Controller Settings from the Global Menu.



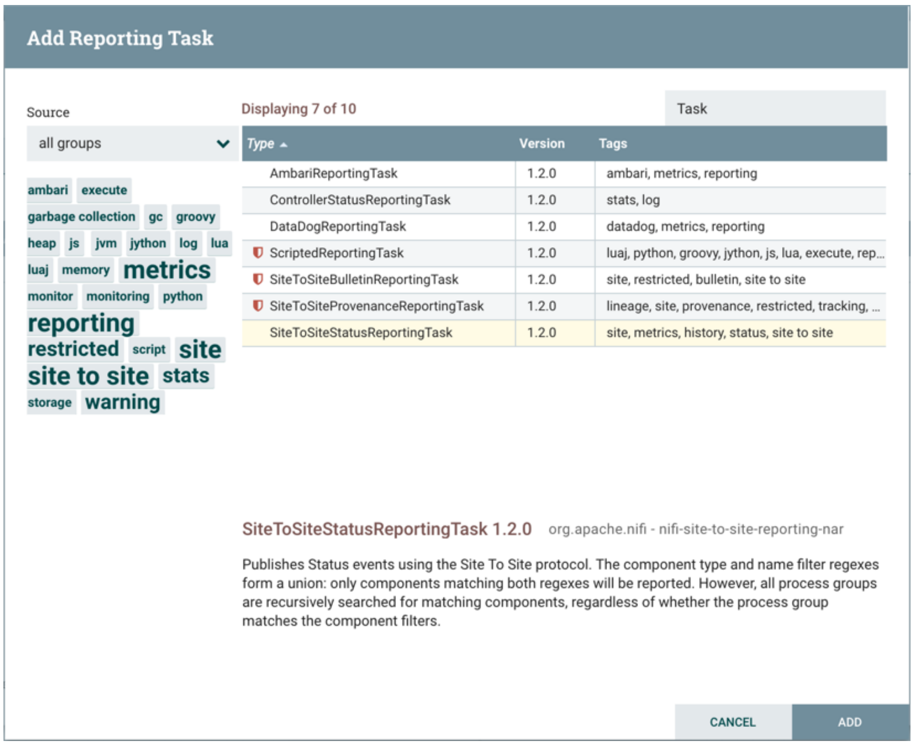
This displays the Logbot ETL Settings window. Select the Reporting Tasks tab and click the + button in the upper-right corner to create a new Reporting Task.


NiFi Settings
GENERAL
REPORTING TASK CONTROLLER SERVICES
REPORTING TASKS
REGISTRY CLIENTS
Click to add a Reporting Task

Name	Type	Bundle	Run Status
SiteToSiteProvenanceReportingTask	SiteToSiteProvenanceReporting...	org.apache.nifi - nifi-site-to-site-rep...	Stopped

Last updated: 14:28:28 EST

The Add Reporting Task window opens. This window is similar to the Add Processor window. It provides a list of the available Reporting Tasks on the right and a tag cloud, showing the most common category tags used for Reporting Tasks, on the left. The DFM may click any tag in the tag cloud in order to narrow down the list of Reporting Tasks to those that fit the categories desired. The DFM may also use the Filter field at the top-right of the window to search for the desired Reporting Task or use the Source drop-down at the top-left to filter the list by the group who created them. Upon selecting a Reporting Task from the list, the DFM can see a description of the task below. Select the desired reporting task and click Add, or simply double-click the name of the service to add it.



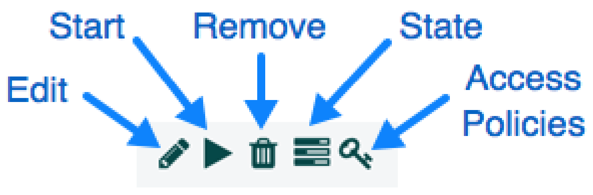
Displaying 7 of 10

Type	Version	Tags
AmbariReportingTask	1.2.0	ambari, metrics, reporting
ControllerStatusReportingTask	1.2.0	stats, log
DataDogReportingTask	1.2.0	datadog, metrics, reporting
ScriptedReportingTask	1.2.0	luaj, python, groovy, jython, js, lua, execute, rep...
SiteToSiteBulletinReportingTask	1.2.0	site, restricted, bulletin, site to site
SiteToSiteProvenanceReportingTask	1.2.0	lineage, site, provenance, restricted, tracking, ...
SiteToSiteStatusReportingTask	1.2.0	site, metrics, history, status, site to site

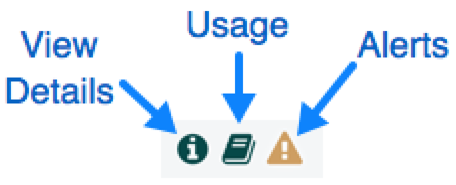
SiteToSiteStatusReportingTask 1.2.0 org.apache.nifi - nifi-site-to-site-reporting-nar

Publishes Status events using the Site To Site protocol. The component type and name filter regexes form a union: only components matching both regexes will be reported. However, all process groups are recursively searched for matching components, regardless of whether the process group matches the component filters.

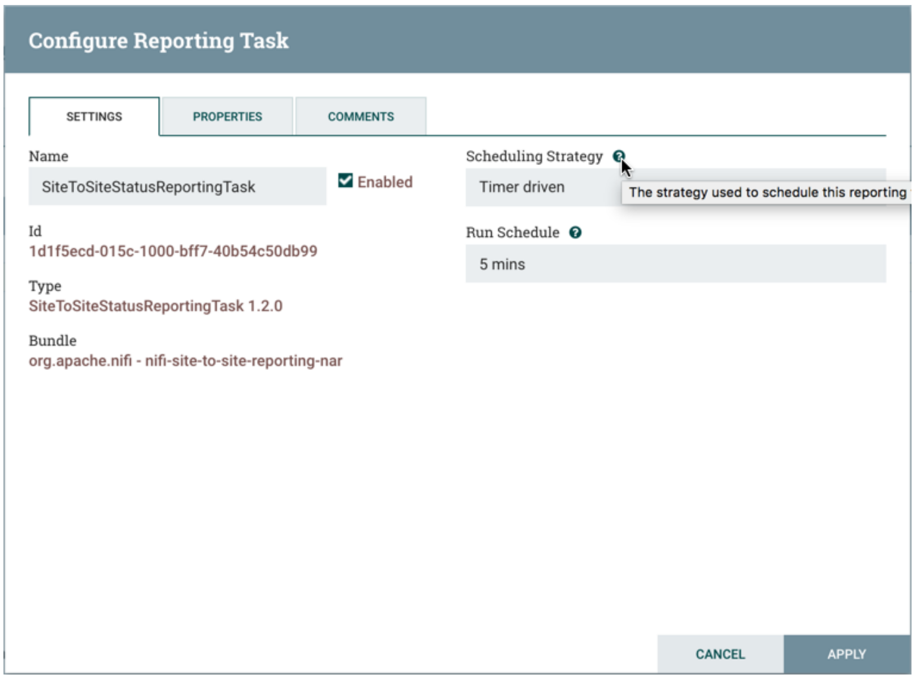
Once a Reporting Task has been added, the DFM may configure it by clicking the "Edit" button in the far-right column. Other buttons in this column include "Start", "Remove", "State" and "Access Policies".



You can obtain information about Reporting Tasks by clicking the "View Details", "Usage", and "Alerts" buttons in the left-hand column.



When the DFM clicks the "Edit" button, a Configure Reporting Task window opens. It has three tabs: Settings, Properties, and Comments. This window is similar to the Configure Processor window. The Settings tab provides a place for the DFM to give the Reporting Task a unique name (if desired). It also lists the UUID, Type, and Bundle information for the task and provides settings for the task's Scheduling Strategy and Run Schedule (similar to the same settings in a processor). The DFM may hover the mouse over the question mark icons to see more information about each setting.


Configure Reporting Task

SETTINGS | PROPERTIES | COMMENTS

Name
SiteToSiteStatusReportingTask
☑ Enabled

Id
1d1f5ecd-015c-1000-bff7-40b54c50db99

Type
SiteToSiteStatusReportingTask 1.2.0

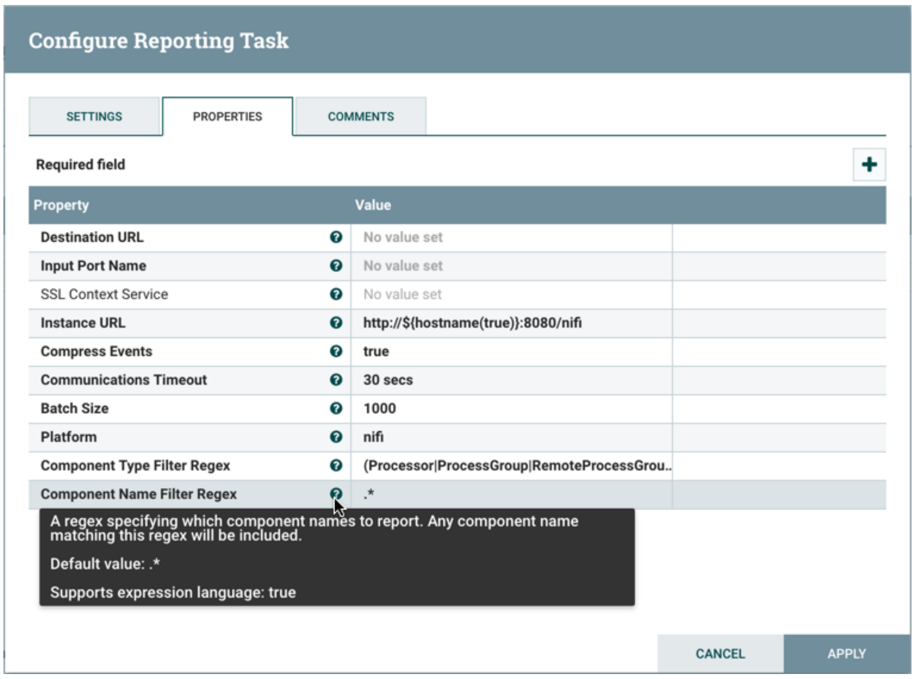
Bundle
org.apache.nifi - nifi-site-to-site-reporting-nar

Scheduling Strategy
Timer driven
The strategy used to schedule this reporting

Run Schedule
5 mins

CANCEL | APPLY

The Properties tab lists the various properties that may be configured for the task. The DFM may hover the mouse over the question mark icons to see more information about each property.



Property	Value
Destination URL	No value set
Input Port Name	No value set
SSL Context Service	No value set
Instance URL	http://${hostname(true)}:8080/nifi
Compress Events	true
Communications Timeout	30 secs
Batch Size	1000
Platform	nifi
Component Type Filter Regex	(Processor|ProcessGroup|RemoteProcessGrou..
Component Name Filter Regex	.*

A regex specifying which component names to report. Any component name matching this regex will be included.

Default value: .*

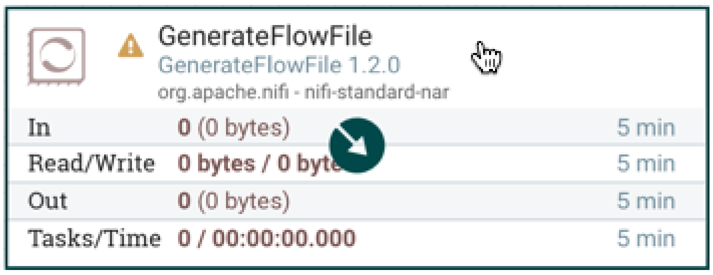
Supports expression language: true

The Comments tab is just an open-text field, where the DFM may include comments about the task. After configuring the Reporting Task, click "Apply" to save the configuration and close the window, or click "Cancel" to discard the changes and close the window.

When you want to run the Reporting Task, click the "Start" button (▶).

Connecting Components

Once processors and other components have been added to the canvas and configured, the next step is to connect them to one another so that Logbot ETL knows what to do with each FlowFile after it has been processed. This is accomplished by creating a Connection between each component. When the user hovers the mouse over the center of a component, a new Connection icon ( ) appears:

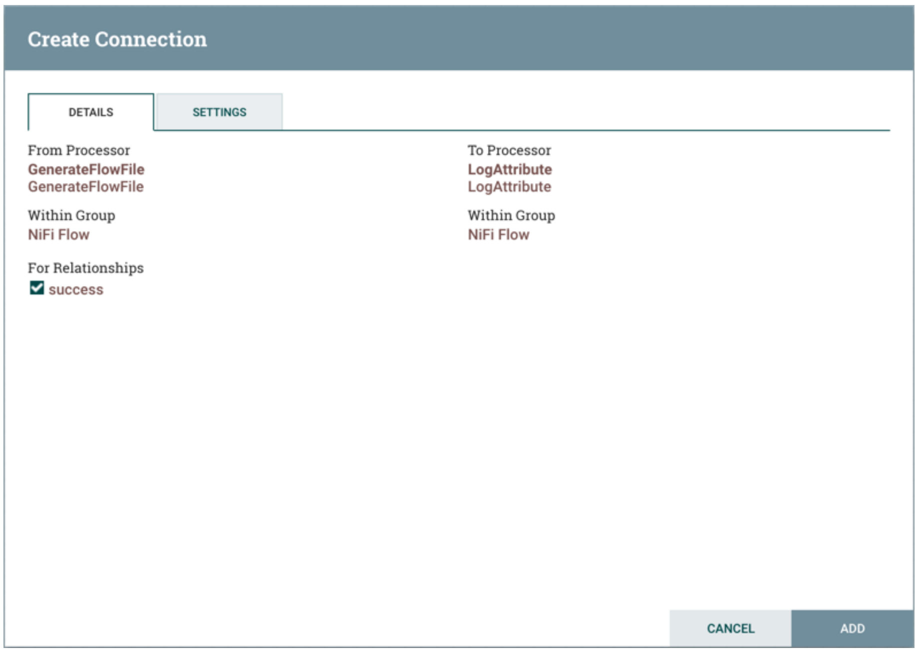

GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 byt	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

The user drags the Connection bubble from one component to another until the second component is highlighted. When the user releases the mouse, a 'Create Connection' dialog appears. This dialog consists of two tabs: 'Details' and 'Settings'. They are discussed in detail below. Note that it is possible to draw a connection so that it loops back on the same processor. This can be useful if the DFM wants the processor to try to re-process FlowFiles if they go down a failure Relationship. To create this type of looping connection, simply drag the connection bubble away and then back to the same processor until it is highlighted. Then release the mouse and the same 'Create Connection' dialog appears.

Details Tab

The Details tab of the 'Create Connection' dialog provides information about the source and destination components, including the component name, the component type, and the Process Group in which the component lives:



From Processor
GenerateFlowFile
GenerateFlowFile

Within Group
NiFi Flow

For Relationships
✓ success

To Processor
LogAttribute
LogAttribute

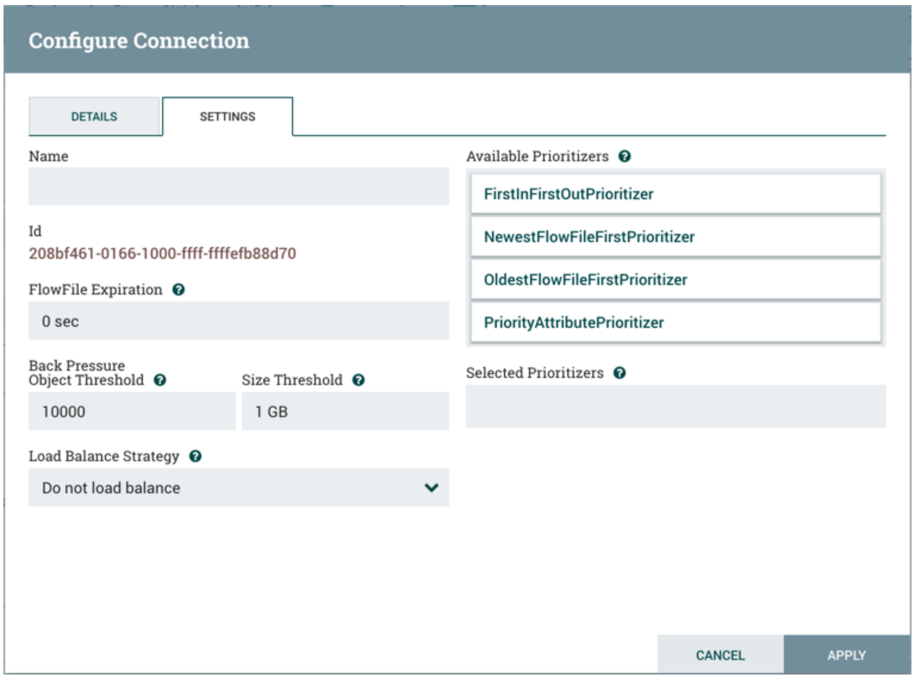
Within Group
NiFi Flow

Additionally, this tab provides the ability to choose which Relationships should be included in this Connection. At least one Relationship must be selected. If only one Relationship is available, it is automatically selected.

If multiple Connections are added with the same Relationship, any FlowFile that is routed to that Relationship will automatically be 'cloned', and a copy will be sent to each of those Connections.

Settings

The Settings tab provides the ability to configure the Connection's Name, FlowFile Expiration, Back Pressure Thresholds, Load Balance Strategy and Prioritization:


Configure Connection

DETAILS SETTINGS

Name

Id
208bf461-0166-1000-ffff-ffffefb88d70

FlowFile Expiration: 0 sec

Back Pressure Object Threshold: 10000

Size Threshold: 1 GB

Load Balance Strategy: Do not load balance

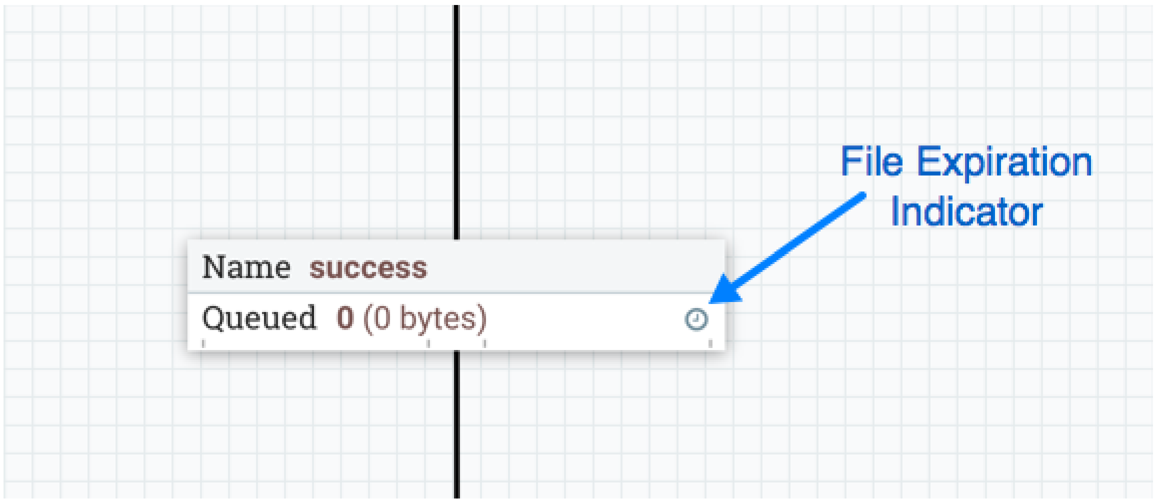
Available Prioritizers
- FirstInFirstOutPrioritizer
- NewestFlowFileFirstPrioritizer
- OldestFlowFileFirstPrioritizer
- PriorityAttributePrioritizer

Selected Prioritizers

The Connection name is optional. If not specified, the name shown for the Connection will be names of the Relationships that are active for the Connection.

FlowFile Expiration

FlowFile expiration is a concept by which data that cannot be processed in a timely fashion can be automatically removed from the flow. This is useful, for example, when the volume of data is expected to exceed the volume that can be sent to a remote site. In this case, the expiration can be used in conjunction with Prioritizers to ensure that the highest priority data is processed first and then anything that cannot be processed within a certain time period (one hour, for example) can be dropped. The expiration period is based on the time that the data entered the Logbot ETL instance. In other words, if the file expiration on a given connection is set to '1 hour', and a file that has been in the Logbot ETL instance for one hour reaches that connection, it will expire. The default value of `0 sec` indicates that the data will never expire. When a file expiration other than '0 sec' is set, a small clock icon appears on the connection label, so the DFM can see it at-a-glance when looking at a flow on the canvas.

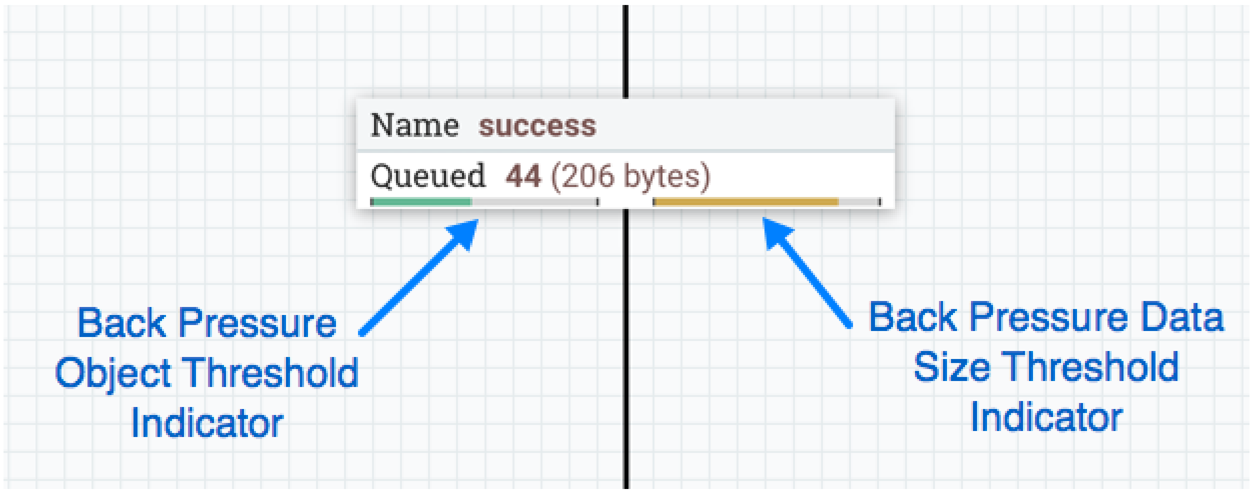


Back Pressure

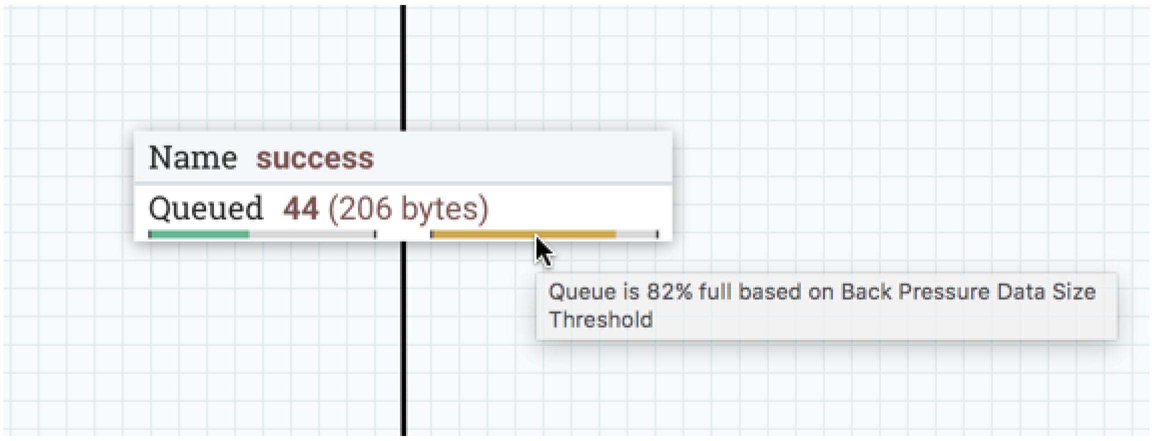
Logbot ETL provides two configuration elements for Back Pressure. These thresholds indicate how much data should be allowed to exist in the queue before the component that is the source of the Connection is no longer scheduled to run. This allows the system to avoid being overrun with data. The first option provided is the "Back pressure object threshold." This is the number of FlowFiles that can be in the queue before back pressure is applied. The second configuration option is the "Back pressure data size threshold." This specifies the maximum amount of data (in size) that should be queued up before applying back pressure. This value is configured by entering a number followed by a data size (`B` for bytes, `KB` for kilobytes, `MB` for megabytes, `GB` for gigabytes, or `TB` for terabytes).

By default each new connection added will have a default Back Pressure Object Threshold of `10,000 objects` and Back Pressure Data Size Threshold of `1 GB`. These defaults can be changed by modifying the appropriate properties in the *Logbot ETL.properties* file.

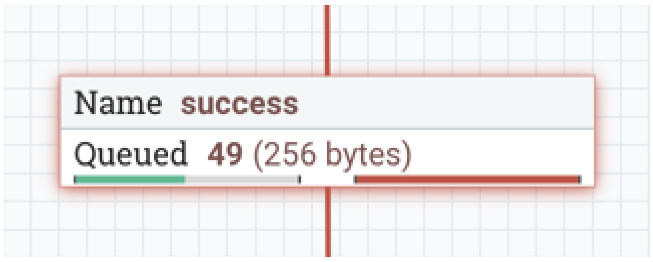
When back pressure is enabled, small progress bars appear on the connection label, so the DFM can see it at-a-glance when looking at a flow on the canvas. The progress bars change color based on the queue percentage: Green (0-60%), Yellow (61-85%) and Red (86-100%).


Name success
Queued 44 (206 bytes)
Back Pressure Object Threshold Indicator
Back Pressure Data Size Threshold Indicator

Hovering your mouse over a bar displays the exact percentage.


Name success
Queued 44 (206 bytes)
Queue is 82% full based on Back Pressure Data Size Threshold

When the queue is completely full, the Connection is highlighted in red.



Load Balancing

Load Balance Strategy

To distribute the data in a flow across the nodes in the cluster, Logbot ETL offers the following load balance strategies:

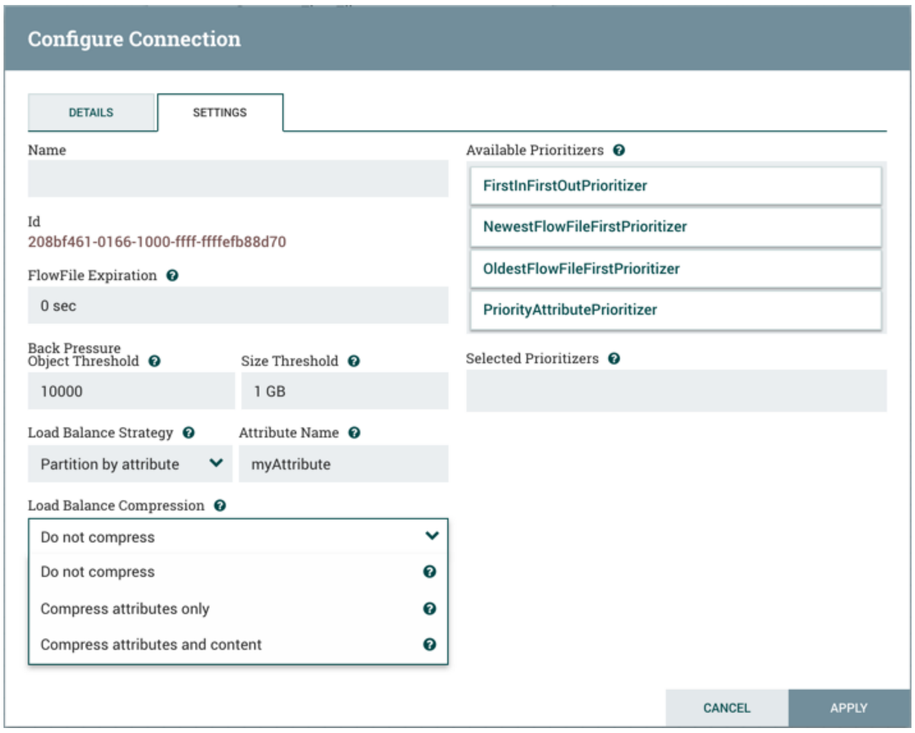
- **Do not load balance**: Do not load balance FlowFiles between nodes in the cluster. This is the default.
- **Partition by attribute**: Determines which node to send a given FlowFile to based on the value of a user-specified FlowFile Attribute. All FlowFiles that have the same value for the Attribute will be sent to the same node in the cluster. If the destination node is disconnected from the cluster or if unable to communicate, the data does not fail over to another node. The data will queue, waiting for the node to be available again. Additionally, if a node joins or leaves the cluster necessitating a rebalance of the data, consistent hashing is applied to avoid having to redistribute all of the data.
- **Round robin**: FlowFiles will be distributed to nodes in the cluster in a round-robin fashion. If a node is disconnected from the cluster or if unable to communicate with a node, the data that is queued for that node will be automatically redistributed to another node(s). If a node is not able to receive the data as fast other nodes in the cluster, the node may also be skipped for one or more iterations in order to maximize throughput of data distribution across the cluster.
- **Single node**: All FlowFiles will be sent to a single node in the cluster. Which node they are sent to is not configurable. If the node is disconnected from the cluster or if unable to communicate with the node, the data that is queued for that node will remain queued until the node is available again.

In addition to the UI settings, there are Cluster Node Properties related to load balancing that must also be configured in *Logbot ETL.properties*.

Logbot ETL persists the nodes that are in a cluster across restarts. This prevents the redistribution of data until all of the nodes have connected. If the cluster is shutdown and a node is not intended to be brought back up, the user is responsible for removing the node from the cluster via the "Cluster" dialog in the UI (see Managing Nodes for more information).

Load Balance Compression

After selecting the load balance strategy, the user can configure whether or not data should be compressed when being transferred between nodes in the cluster.



Name

Id
208bf461-0166-1000-ffff-ffffefb88d70

FlowFile Expiration: 0 sec

Back Pressure Object Threshold: 10000

Size Threshold: 1 GB

Load Balance Strategy: Partition by attribute

Attribute Name: myAttribute

Load Balance Compression: Do not compress
- Do not compress
- Compress attributes only
- Compress attributes and content

Available Prioritizers
- FirstInFirstOutPrioritizer
- NewestFlowFileFirstPrioritizer
- OldestFlowFileFirstPrioritizer
- PriorityAttributePrioritizer

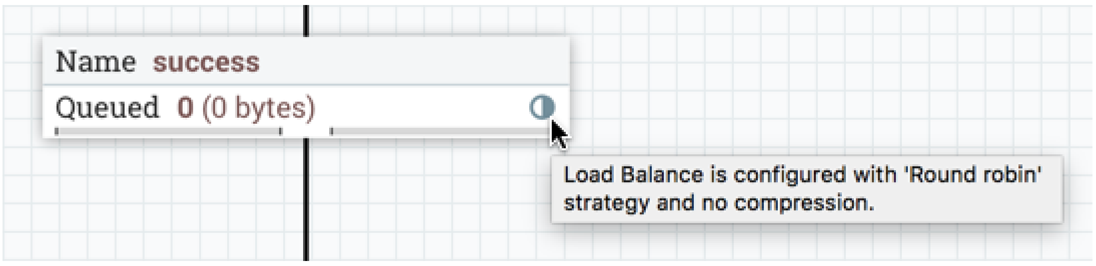
Selected Prioritizers

The following compression options are available:

- **Do not compress**: FlowFiles will not be compressed. This is the default.
- **Compress attributes only**: FlowFile attributes will be compressed, but FlowFile contents will not.
- **Compress attributes and content**: FlowFile attributes and contents will be compressed.

Load Balance Indicator

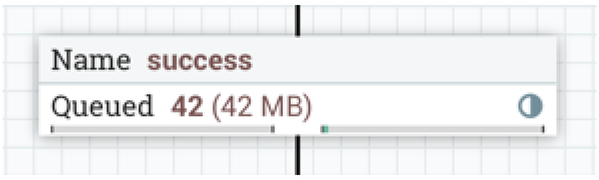
When a load balance strategy has been implemented for a connection, a load balance indicator (◑) will appear on the connection:



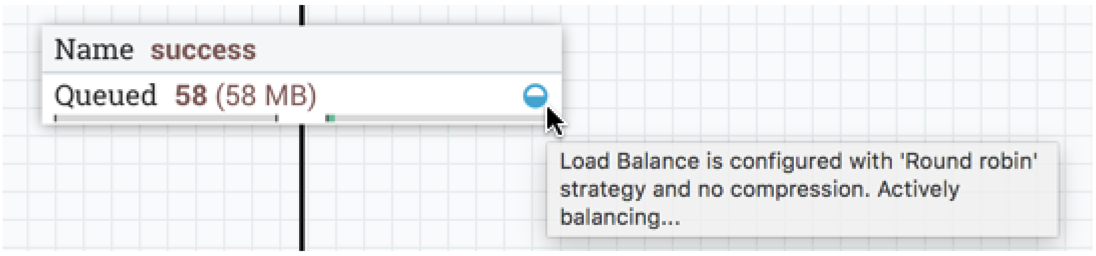
Name success
Queued 0 (0 bytes)

Load Balance is configured with 'Round robin' strategy and no compression.

Hovering over the icon will display the connection's load balance strategy and compression configuration. The icon in this state also indicates that all data in the connection has been distributed across the cluster.

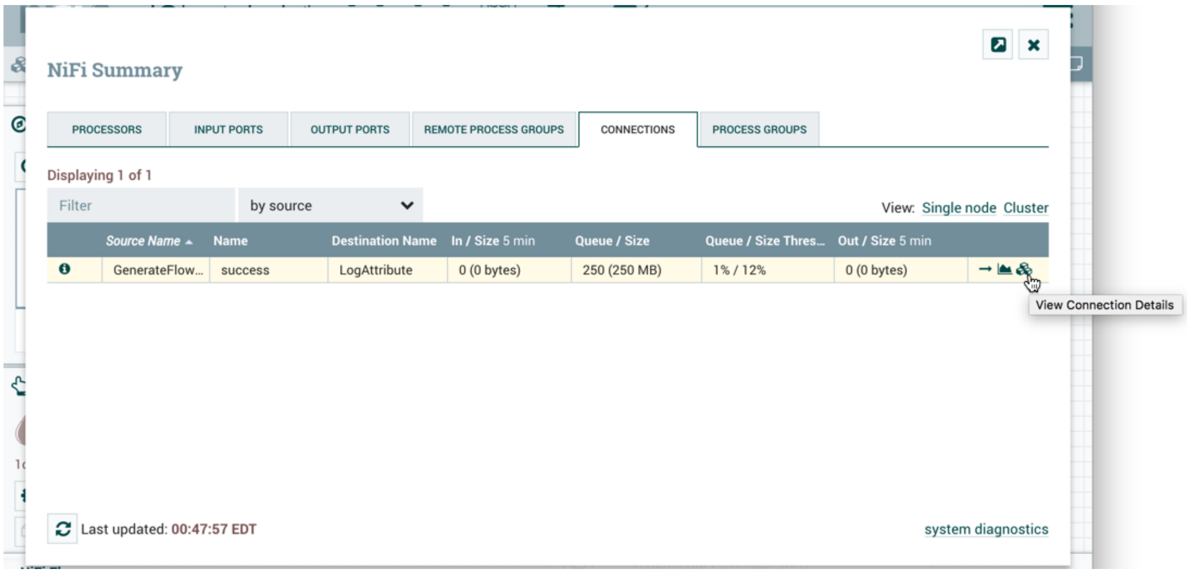

Name success
Queued 42 (42 MB)

When data is actively being transferred between the nodes in the cluster, the load balance indicator will change orientation and color:


Name success
Queued 58 (58 MB)
Load Balance is configured with 'Round robin' strategy and no compression. Actively balancing...

Cluster Connection Summary

To see where data has been distributed among the cluster nodes, select Summary from the Global Menu. Then select the "Connections" tab and the "View Connection Details" icon for a source:

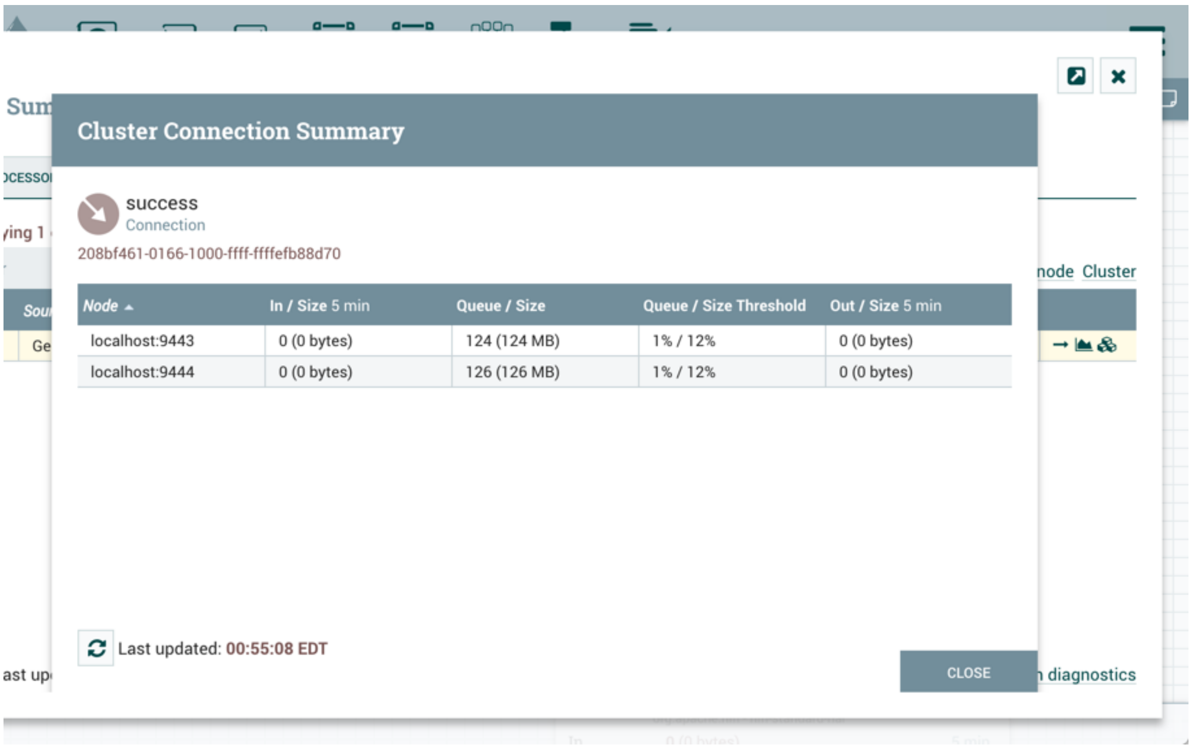


Displaying 1 of 1

	Source Name	Name	Destination Name	In / Size 5 min	Queue / Size	Queue / Size Thres...	Out / Size 5 min
●	GenerateFlow...	success	LogAttribute	0 (0 bytes)	250 (250 MB)	1% / 12%	0 (0 bytes)

Last updated: 00:47:57 EDT

This will open the Cluster Connection Summary dialog, which shows the data on each node in the cluster:


Cluster Connection Summary

success
Connection
208bf461-0166-1000-ffff-ffffefb88d70

Node	In / Size 5 min	Queue / Size	Queue / Size Threshold	Out / Size 5 min
localhost:9443	0 (0 bytes)	124 (124 MB)	1% / 12%	0 (0 bytes)
localhost:9444	0 (0 bytes)	126 (126 MB)	1% / 12%	0 (0 bytes)

Last updated: 00:55:08 EDT

Prioritization

The right-hand side of the tab provides the ability to prioritize the data in the queue so that higher priority data is processed first. Prioritizers can be dragged from the top ('Available prioritizers') to the bottom ('Selected prioritizers'). Multiple prioritizers can be selected. The prioritizer that is at the top of the 'Selected prioritizers' list is the highest priority. If two FlowFiles have the same value according to this prioritizer, the second prioritizer will determine which FlowFile to process first, and so on. If a prioritizer is no longer desired, it can then be dragged from the 'Selected prioritizers' list to the 'Available prioritizers' list.

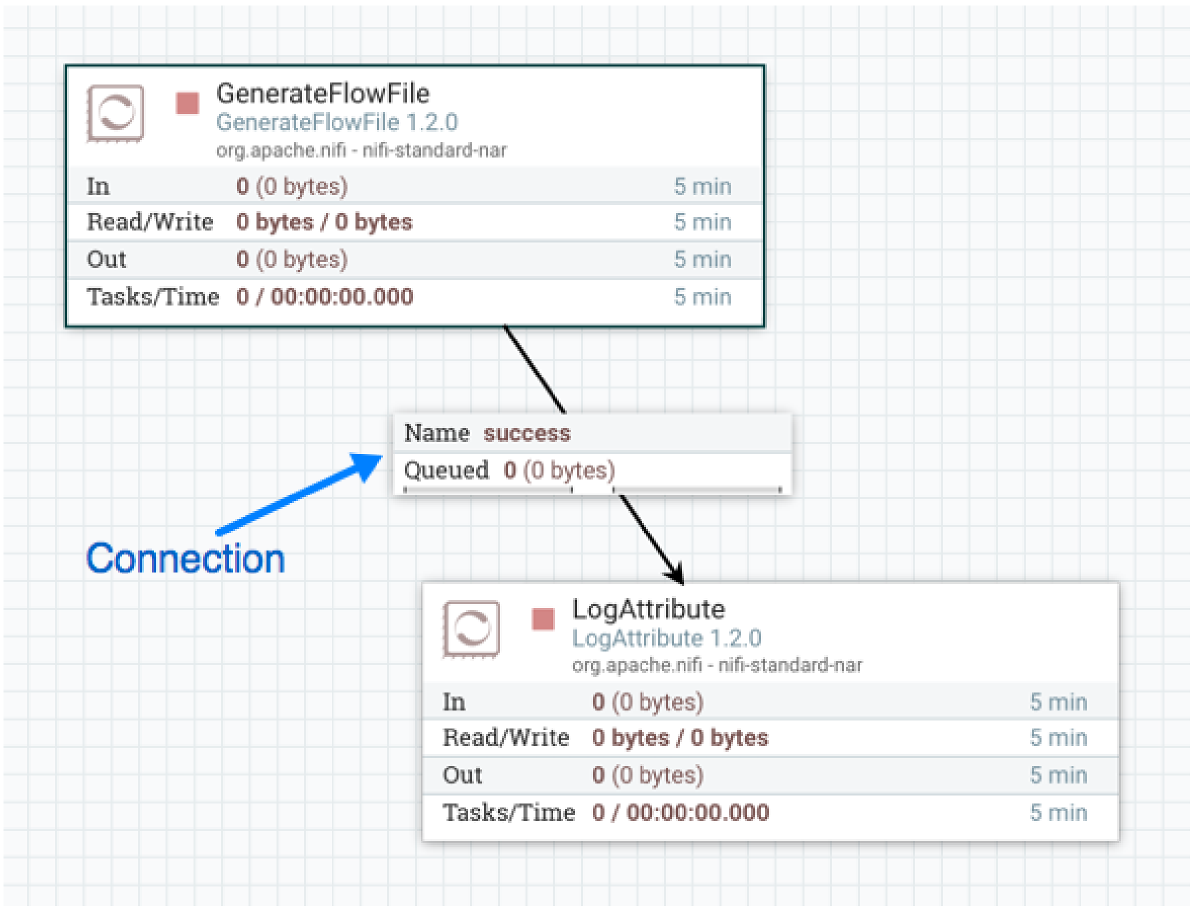
The following prioritizers are available:

- **FirstInFirstOutPrioritizer**: Given two FlowFiles, the one that reached the connection first will be processed first.
- **NewestFlowFileFirstPrioritizer**: Given two FlowFiles, the one that is newest in the dataflow will be processed first.
- **OldestFlowFileFirstPrioritizer**: Given two FlowFiles, the one that is oldest in the dataflow will be processed first. 'This is the default scheme that is used if no prioritizers are selected'.
- **PriorityAttributePrioritizer**: Given two FlowFiles, an attribute called “priority” will be extracted. The one that has the lowest priority value will be processed first.
    - Note that an UpdateAttribute processor should be used to add the "priority" attribute to the FlowFiles before they reach a connection that has this prioritizer set.
    - If only one has that attribute it will go first.
    - Values for the "priority" attribute can be alphanumeric, where "a" will come before "z" and "1" before "9"
    - If "priority" attribute cannot be parsed as a long, unicode string ordering will be used. For example: "99" and "100" will be ordered so the flowfile with "99" comes first, but "A-99" and "A-100" will sort so the flowfile with "A-100" comes first.

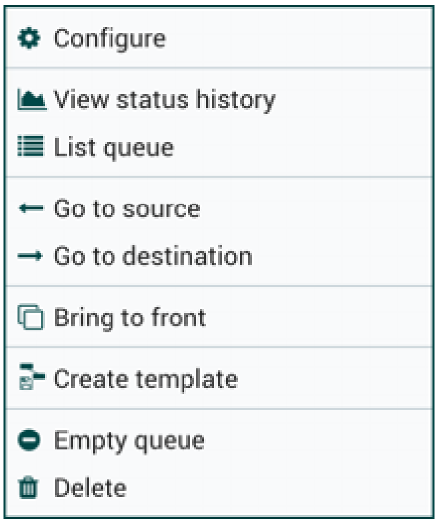
With a Load Balance Strategy configured, the connection has a queue per node in addition to the local queue. The prioritizer will sort the data in each queue independently.

Changing Configuration and Context Menu Options

After a connection has been drawn between two components, the connection's configuration may be changed, and the connection may be moved to a new destination; however, the processors on either side of the connection must be stopped before a configuration or destination change may be made.


GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Connection
LogAttribute
LogAttribute 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

To change a connection's configuration or interact with the connection in other ways, right-click on the connection to open the connection context menu.

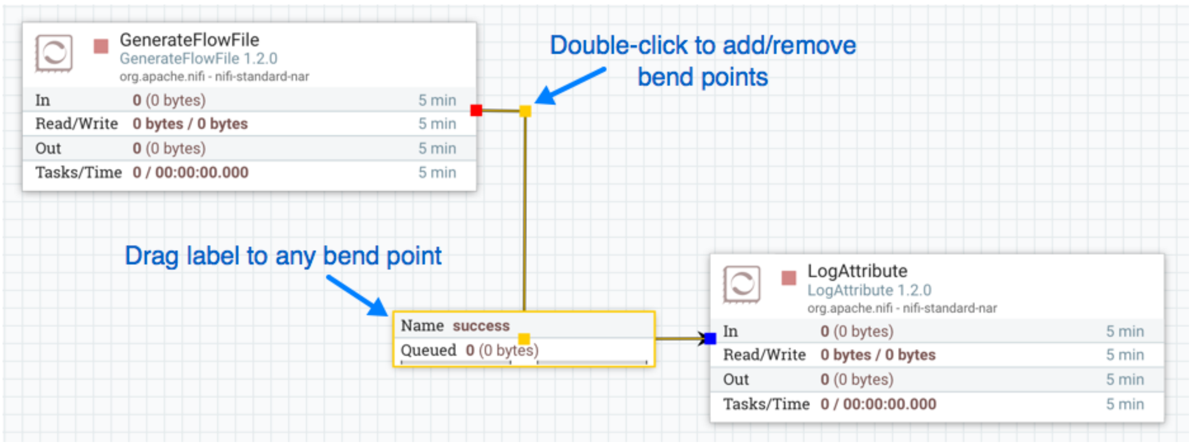


The following options are available:

- **Configure**: This option allows the user to change the configuration of the connection.
- **View status history**: This option opens a graphical representation of the connection’s statistical information over time.
- **List queue**: This option lists the queue of FlowFiles that may be waiting to be processed.
- **Go to source**: This option can be useful if there is a long distance between the connection’s source and destination components on the canvas. By clicking this option, the view of the canvas will jump to the source of the connection.
- **Go to destination**: Similar to the "Go to source" option, this option changes the view to the destination component on the canvas and can be useful if there is a long distance between two connected components.
- **Bring to front**: This option brings the connection to the front of the canvas if something else (such as another connection) is overlapping it.
- **Empty queue**: This option allows the DFM to clear the queue of FlowFiles that may be waiting to be processed. This option can be especially useful during testing, when the DFM is not concerned about deleting data from the queue. When this option is selected, users must confirm that they want to delete the data in the queue.
- **Delete**: This option allows the DFM to delete a connection between two components. Note that the components on both sides of the connection must be stopped and the connection must be empty before it can be deleted.

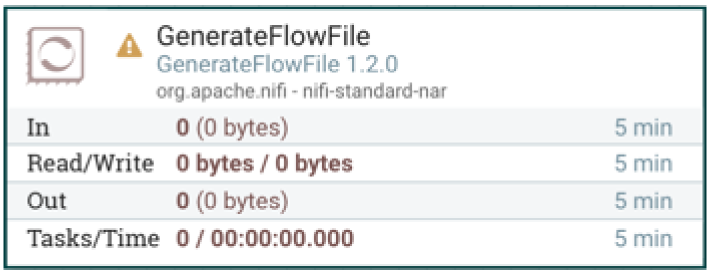
Bending Connections

To add a bend point (or elbow) to an existing connection, simply double-click on the connection in the spot where you want the bend point to be. Then, you can use the mouse to grab the bend point and drag it so that the connection is bent in the desired way. You can add as many bend points as you want. You can also use the mouse to drag and move the label on the connection to any existing bend point. To remove a bend point, simply double-click it again.


GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Double-click to add/remove bend points
Drag label to any bend point
Name success
Queued 0 (0 bytes)
LogAttribute
LogAttribute 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Processor Validation

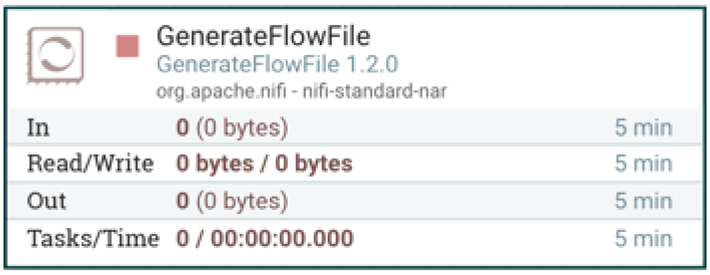
Before trying to start a Processor, it’s important to make sure that the Processor’s configuration is valid. A status indicator is shown in the top-left of the Processor. If the Processor is invalid, the indicator will show a yellow Warning indicator with an exclamation mark indicating that there is a problem:



GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

In this case, hovering over the indicator icon with the mouse will provide a tooltip showing all of the validation errors for the Processor. Once all of the validation errors have been addressed, the status indicator will change to a Stop icon, indicating that the Processor is valid and ready to be started but currently is not running:



GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Site-to-Site

When sending data from one instance of Logbot ETL to another, there are many different protocols that can be used. The preferred protocol, though, is the Logbot ETL Site-to-Site Protocol. Site-to-Site makes it easy to securely and efficiently transfer data to/from nodes in one Logbot ETL instance or data producing application to nodes in another Logbot ETL instance or other consuming application.

Using Site-to-Site provides the following benefits:

- Easy to configure
    - After entering the URL(s) of the remote Logbot ETL instance/cluster, the available ports (endpoints) are automatically discovered and provided in a drop-down list.
- Secure
    - Site-to-Site optionally makes use of Certificates in order to encrypt data and provide authentication and authorization. Each port can be configured to allow only specific users, and only those users will be able to see that the port even exists. For information on configuring the Certificates, see the Security Configuration section of the System Administrator’s Guide.
- Scalable
    - As nodes in the remote cluster change, those changes are automatically detected and data is scaled out across all nodes in the cluster.
- Efficient
    - Site-to-Site allows batches of FlowFiles to be sent at once in order to avoid the overhead of establishing connections and making multiple round-trip requests between peers.
- Reliable
    - Checksums are automatically produced by both the sender and receiver and compared after the data has been transmitted, in order to ensure that no corruption has occurred. If the checksums don’t match, the transaction will simply be canceled and tried again.
- Automatically load balanced
    - As nodes come online or drop out of the remote cluster, or a node’s load becomes heavier or lighter, the amount of data that is directed to that node will automatically be adjusted.
- FlowFiles maintain attributes
    - When a FlowFile is transferred over this protocol, all of the FlowFile’s attributes are automatically transferred with it. This can be very advantageous in many situations, as all of the context and enrichment that has been determined by one instance of Logbot ETL travels with the data, making for easy routing of the data and allowing users to easily inspect the data.
- Adaptable
    - As new technologies and ideas emerge, the protocol for handling Site-to-Site communications are able to change with them. When a connection is made to a remote Logbot ETL instance, a handshake is performed in order to negotiate which protocol and which version of the protocol will be used. This allows new capabilities to be added while still maintaining backward compatibility with all older instances. Additionally, if a vulnerability or deficiency is ever discovered in a protocol, it allows a newer version of Logbot ETL to forbid communication over the compromised versions of the protocol.

Site-to-Site is a protocol transferring data between two Logbot ETL instances. Both end can be a standalone Logbot ETL or a Logbot ETL cluster. In this section, the Logbot ETL instance initiates the communications is called *Site-to-Site client Logbot ETL instance* and the other end as *Site-to-Site server Logbot ETL instance* to clarify what configuration needed on each Logbot ETL instances.

A Logbot ETL instance can be both client and server for Site-to-Site protocol, however, it can only be a client or server within a specific Site-to-Site communication. For example, if there are three Logbot ETL instances A, B and C. A pushes data to B, and B pulls data from C. $A \text{ — } push \rightarrow B \leftarrow pull \text{ — } C$. Then B is not only a *server* in the communication between A and B, but also a *client* in B and C.

It is important to understand which Logbot ETL instance will be the client or server in order to design your data flow, and configure each instance accordingly. Here is a summary of what components run on which side based on data flow direction:

- Push: a client *sends* data to a Remote Process Group, the server *receives* it with an Input Port
- Pull: a client *receives* data from a Remote Process Group, the server *sends* data through an Output Port

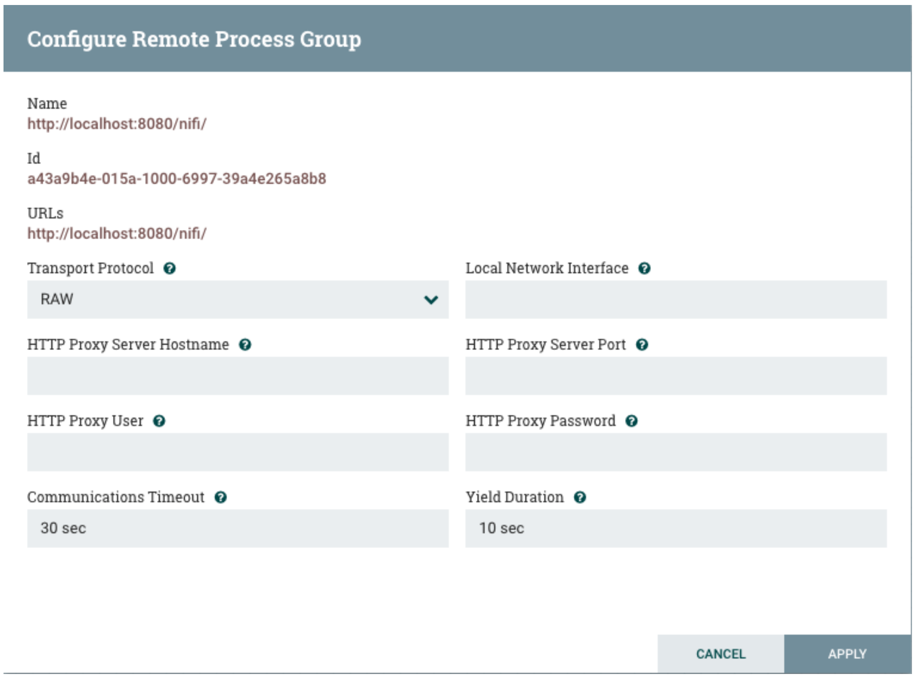
Configure Site-to-Site client Logbot ETL instance

Remote Process Group: In order to communicate with a remote Logbot ETL instance via Site-to-Site, simply drag a Remote Process Group onto the canvas and enter the URL(s) of the remote Logbot ETL instance (for more information on the components of a Remote Process Group, see the Remote Process Group Transmission section of this guide.) The URL is the same URL you would use to go to that instance's User Interface or in the case of a cluster, the URLs of the cluster nodes. At this point, you can drag a connection to or from the Remote Process Group in the same way you would drag a connection to or from a Processor or a local Process Group. When you drag the connection, you will have a chance to choose which Port to connect to. Note that it may take up to one minute for the Remote Process Group to determine which ports are available.

If the connection is dragged starting from the Remote Process Group, the ports shown will be the Output Ports of the remote group, as this indicates that you will be pulling data from the remote instance. If the connection instead ends on the Remote Process Group, the ports shown will be the Input Ports of the remote group, as this implies that you will be pushing data to the remote instance.

If the remote instance is configured to use secure data transmission, you will see only ports that you are authorized to communicate with. For information on configuring Logbot ETL to run securely, see the System Administrator's Guide.

Transport Protocol: On a Remote Process Group creation or configuration dialog, you can choose Transport Protocol to use for Site-to-Site communication as shown in the following image:


Configure Remote Process Group

Name
http://localhost:8080/nifi/

Id
a43a9b4e-015a-1000-6997-39a4e265a8b8

URLs
http://localhost:8080/nifi/

Transport Protocol: RAW	Local Network Interface:
HTTP Proxy Server Hostname:	HTTP Proxy Server Port:
HTTP Proxy User:	HTTP Proxy Password:
Communications Timeout: 30 sec	Yield Duration: 10 sec

By default, it is set to *RAW* which uses raw socket communication using a dedicated port. *HTTP* transport protocol is especially useful if the remote Logbot ETL instance is in a restricted network that only allow access through HTTP(S) protocol or only accessible from a specific HTTP Proxy server. For accessing through a HTTP Proxy Server, BASIC and DIGEST authentication are supported.

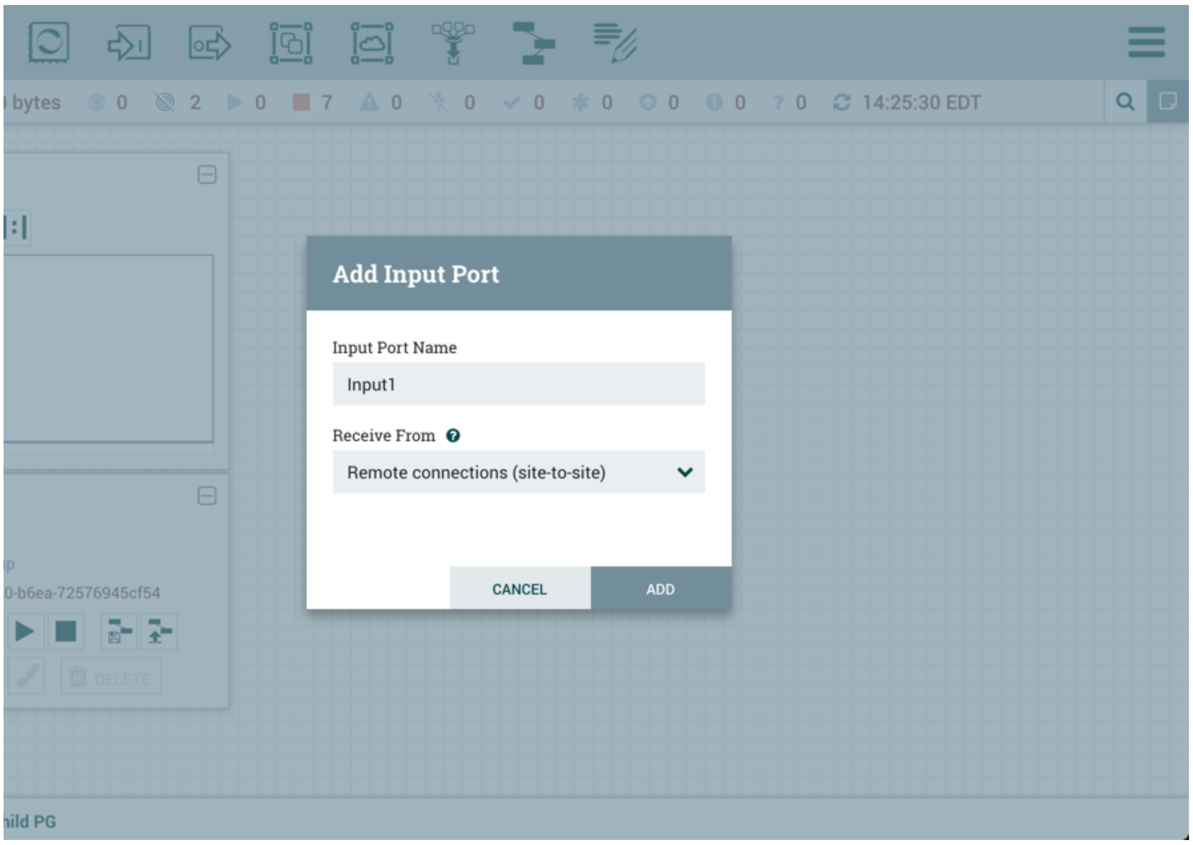
Local Network Interface: In some cases, it may be desirable to prefer one network interface over another. For example, if a wired interface and a wireless interface both exist, the wired interface may be preferred. This can be configured by specifying the name of the network interface to use in this box. If the value entered is not valid, the Remote Process Group will not be valid and will not communicate with other Logbot ETL instances until this is resolved.

Configure Site-to-Site Server Logbot ETL Instance

Retrieve Site-to-Site Details: If your Logbot ETL is running securely, in order for another Logbot ETL instance to retrieve information from your instance, it needs to be added to the Global Access "retrieve site-to-site details" policy. This will allow the other instance to query your instance for details such as name, description, available peers (nodes when clustered), statistics, OS port information and available Input and Output ports. Utilizing Input and Output ports in a secured instance requires additional policy configuration as described below.

Input Port: In order to allow another Logbot ETL instance to push data to your local instance, you can simply drag an Input Port onto the Root Process Group of your canvas. After entering a name for the port, it will be added to your flow. You can now right-click on the Input Port and choose Configure in order to adjust the name and the number of concurrent tasks that are used for the port.

To create an Input Port for Site-to-Site in a child Process Group, enter the name for the port and select "Remote connections (site-to-site)" from the Receive From drop-down menu.



Add Input Port

Input Port Name: Input1

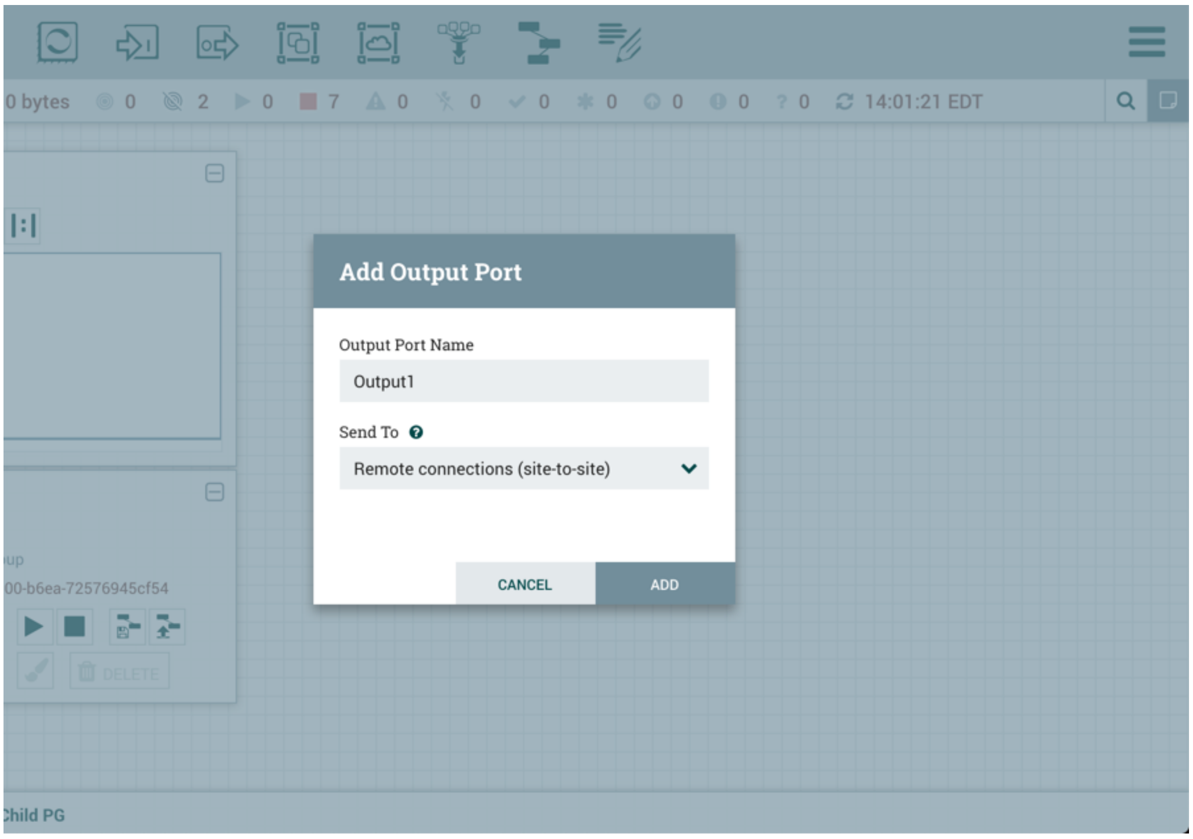
Receive From: Remote connections (site-to-site)

CANCEL ADD

If Site-to-Site is configured to run securely, you will need to manage the input port’s "receive data via site-to-site" component access policy. Only those users who have been added to the policy will be able to communicate with the port.

Output Port: Similar to an Input Port, a DataFlow Manager may choose to add an Output Port to the Root Process Group. The Output Port allows an authorized Logbot ETL instance to remotely connect to your instance and pull data from the Output Port. After dragging an Output Port onto the canvas, right-click and choose Configure to adjust the name and how many concurrent tasks are allowed. Manage the output port’s "receive data via site-to-site" component access policy to control which users are authorized to pull data from the instance being configured.

To create an Output Port for Site-to-Site in a child Process Group, enter the name for the port and select "Remote connections (site-to-site)" from the Send To drop-down menu.



In addition to other instances of Logbot ETL, some other applications may use a Site-to-Site client in order to push data to or receive data from a Logbot ETL instance. For example, Logbot ETL provides an Apache Storm spout and an Apache Spark Receiver that are able to pull data from Logbot ETL's Output Ports for Site-to-Site connections.

For information on how to enable and configure Site-to-Site on a Logbot ETL instance, see the Site-to-Site Properties section of the System Administrator's Guide.

For information on how to configure access policies, see the Access Policies section of the System Administrator's Guide.

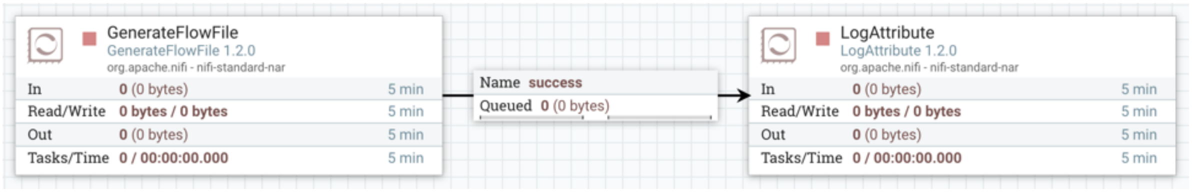
Example Dataflow

This section has described the steps required to build a dataflow. Now, to put it all together. The following example dataflow consists of just two processors: GenerateFlowFile and LogAttribute. These processors are normally used for testing, but they can also be used to build a quick flow for demonstration purposes and see Logbot ETL in action.

After you drag the GenerateFlowFile and LogAttribute processors to the canvas and connect them (using the guidelines provided above), configure them as follows:

- Generate FlowFile
  - On the Scheduling tab, set Run schedule to: 5 sec. Note that the GenerateFlowFile processor can create many FlowFiles very quickly; that's why setting the Run schedule is important so that this flow does not overwhelm the system Logbot ETL is running on.
  - On the Properties tab, set File Size to: 10 KB
- Log Attribute
  - On the Settings tab, under Auto-terminate relationships, select the checkbox next to Success. This will terminate FlowFiles after this processor has successfully processed them.
  - Also on the Settings tab, set the Bulletin level to Info. This way, when the dataflow is running, this processor will display the bulletin icon (see Anatomy of a Processor), and the user may hover over it with the mouse to see the attributes that the processor is logging.

The dataflow should look like the following:



GenerateFlowFile
GenerateFlowFile 1.2.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

LogAttribute
LogAttribute 1.2.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Now see the following section on how to start and stop the dataflow. When the dataflow is running, be sure to note the statistical information that is displayed on the face of each processor (see Anatomy of a Processor).

Command and Control of the DataFlow

When a component is added to the Logbot ETL canvas, it is in the Stopped state. In order to cause the component to be triggered, the component must be started. Once started, the component can be stopped at any time. From a Stopped state, the component can be configured, started, or disabled.

Starting a Component

In order to start a component, the following conditions must be met:

- The component's configuration must be valid.
- All defined Relationships for the component must be connected to another component or auto-terminated.
- The component must be stopped.
- The component must be enabled.
- The component must have no active tasks. For more information about active tasks, see the "Anatomy of …" sections under Monitoring of DataFlow (Anatomy of a Processor, Anatomy of a Process Group, Anatomy of a Remote Process Group).

Components can be started by selecting all of the components to start and then clicking the "Start" button (▶) in the Operate Palette or by right-clicking a single component and choosing Start from the context menu.

If starting a Process Group, all components within that Process Group (including child Process Groups) will be started, with the exception of those components that are invalid or disabled.

Once started, the status indicator of a Processor will change to a Play symbol (▶).

Stopping a Component

A component can be stopped any time that it is running. A component is stopped by right-clicking on the component and clicking Stop from the context menu, or by selecting the component and clicking the "Stop" button (■) in the Operate Palette.

If a Process Group is stopped, all of the components within the Process Group (including child Process Groups) will be stopped.

Once stopped, the status indicator of a component will change to the Stop symbol (🟥).

Stopping a component does not interrupt its currently running tasks. Rather, it stops scheduling new tasks to be performed. The number of active tasks is shown in the top-right corner of the Processor (See Anatomy of a Processor for more information). See Terminating a Component’s tasks for how to terminate the running tasks.

Terminating a Component’s tasks

When a component is stopped, it does not interrupt the currently running tasks. This allows for the current execution to complete while no new tasks are scheduled, which is the desired behaviour in many cases. In some cases, it is desirable to terminate the running tasks, particularly in cases where a task has hung and is no longer responsive, or while developing new flows.

To be able to terminate the running task(s), the component must first be stopped (see Stopping a Component). Once the component is in the Stopped state, the Terminate option will become available only if there are tasks still running (See Anatomy of a Processor). The Terminate option (⌛) can be accessed either via the context menu or the Operations Palette while the component is selected.

The number of tasks that are actively being terminated will be displayed in parentheses next to the number of active tasks e.g. 0 (1). For example, if there is one active task at the time that Terminate is selected, this will display "0 (1)" - meaning 0 active tasks and 1 task being terminated.

A task may not terminate immediately, as different components may respond to the Terminate command differently. However, the components can be reconfigured and started/stopped regardless of whether there are tasks still in the terminating state.

Enabling/Disabling a Component

When a component is enabled, it is able to be started. Users may choose to disable components when they are part of a dataflow that is still being assembled, for example. Typically, if a component is not intended to be run, the component is disabled, rather than being left in the Stopped state. This helps to distinguish between components that are intentionally not running and those that may have been stopped temporarily (for instance, to change the component's configuration) and inadvertently were never restarted.

When it is desirable to re-enable a component, it can be enabled by selecting the component and clicking the "Enable" button ( ) in the Operate Palette. This is available only when the selected component or components are disabled. Alternatively, a component can be enabled by checking the checkbox next to the "Enabled" option in the Settings tab of the Processor configuration dialog or the configuration dialog for a Port.

Once enabled, the component's status indicator will change to either Invalid (⚠) or Stopped (■), depending on whether or not the component is valid.

A component is then disabled by selecting the component and clicking the "Disable" button ( ) in the Operate Palette, or by clearing the checkbox next to the "Enabled" option in the Settings tab of the Processor configuration dialog or the configuration dialog for a Port.

Only Ports and Processors can be enabled and disabled.

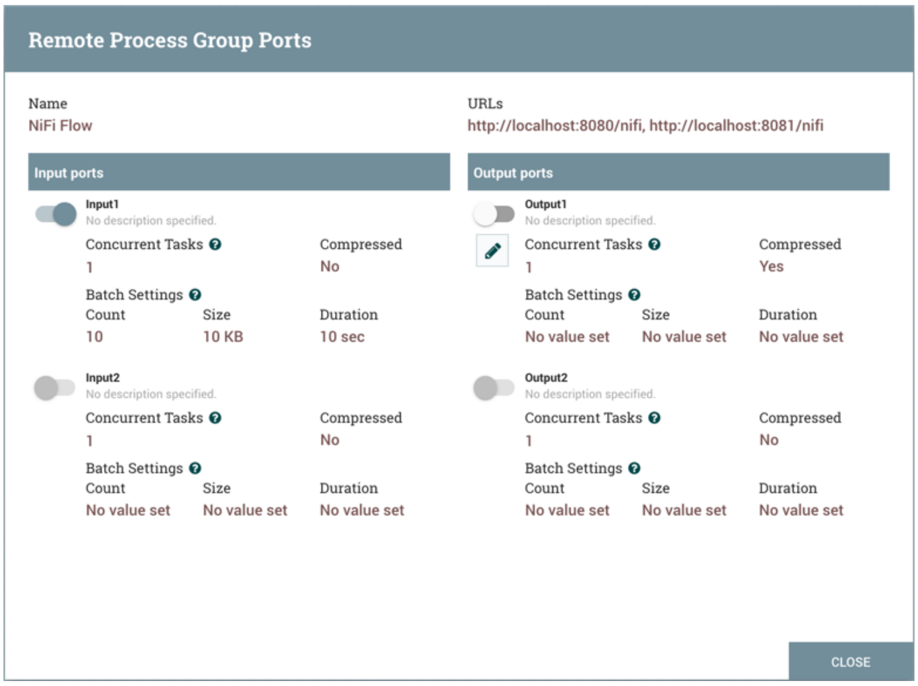
Remote Process Group Transmission

Remote Process Groups provide a mechanism for sending data to or retrieving data from a remote instance of Logbot ETL. When a Remote Process Group (RPG) is added to the canvas, it is added with the Transmission Disabled, as indicated by the icon ( ) in the top-left corner. When Transmission is Disabled, it can be enabled by right-clicking on the RPG and clicking the "Enable transmission" menu item. This will cause all ports for which there is a Connection to begin transmitting data. This will cause the status indicator to then change to the Transmission Enabled icon ( ).

If there are problems communicating with the Remote Process Group, a Warning indicator ( ) may instead be present in the top-left corner. Hovering over this Warning indicator with the mouse will provide more information about the problem.

Individual Port Transmission

There are times when the DFM may want to either enable or disable transmission for only a specific port within the Remote Process Group. This can be accomplished by right-clicking on the Remote Process Group and choosing the "Manage remote ports" menu item. This provides a configuration dialog from which ports can be configured:



Remote Process Group Ports

Name
NiFi Flow

URLs
http://localhost:8080/nifi, http://localhost:8081/nifi

Input ports

Input1
No description specified.

Concurrent Tasks		Compressed
1		No

Batch Settings		
Count	Size	Duration
10	10 KB	10 sec

Input2
No description specified.

Concurrent Tasks		Compressed
1		No

Batch Settings		
Count	Size	Duration
No value set	No value set	No value set

Output ports

Output1
No description specified.

Concurrent Tasks		Compressed
1		Yes

Batch Settings		
Count	Size	Duration
No value set	No value set	No value set

Output2
No description specified.

Concurrent Tasks		Compressed
1		No

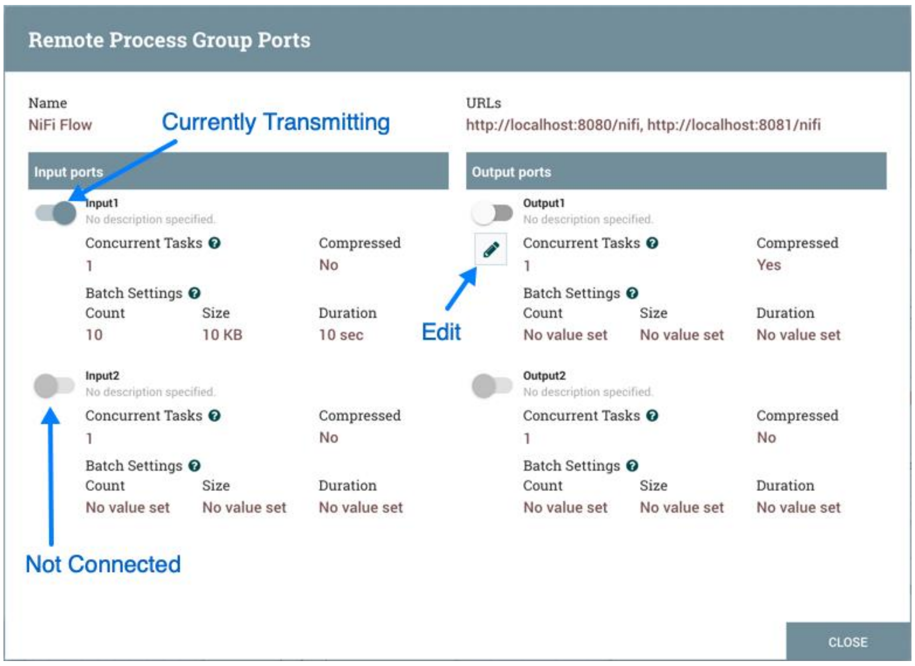
Batch Settings		
Count	Size	Duration
No value set	No value set	No value set

CLOSE

The left-hand side lists all of the Input Ports that the remote instance of Logbot ETL allows data to be sent to. The right-hand side lists all of the Output Ports from which this instance is able to pull data. If the remote instance is using secure communications (the URL of the Logbot ETL instance begins with `https://`, rather than `http://`), any ports that the remote instance has not made available to this instance will not be shown.

If a port that is expected to be shown is not shown in this dialog, ensure that the instance has proper permissions and that the Remote Process Group’s flow is current. This can be checked by closing the Remote Process Group Ports dialog and looking at the bottom-left corner of the Remote Process Group. The date and time when the flow was last refreshed is displayed. If the flow appears to be outdated, it can be updated by right-clicking on the Remote Process Group and selecting "Refresh remote". (See Anatomy of a Remote Process Group for more information).

Each port is shown with its Name, its Description, configured number of Concurrent Tasks, and whether or not data sent to this port will be Compressed. Additionally, the port’s configured Batch Settings (Count, Size and Duration) are displayed. To the left of this information is a toggle switch to turn the port on or off. Ports that have no connections attached to them are grayed out:



Remote Process Group Ports

Name: NiFi Flow
URLs: http://localhost:8080/nifi, http://localhost:8081/nifi

Input ports

Input1
No description specified.
Concurrent Tasks: 1 | Compressed: No
Batch Settings: Count 10 | Size 10 KB | Duration 10 sec

Input2
No description specified.
Concurrent Tasks: 1 | Compressed: No
Batch Settings: Count No value set | Size No value set | Duration No value set

Output ports

Output1
No description specified.
Concurrent Tasks: 1 | Compressed: Yes
Batch Settings: Count No value set | Size No value set | Duration No value set

Output2
No description specified.
Concurrent Tasks: 1 | Compressed: No
Batch Settings: Count No value set | Size No value set | Duration No value set

The on/off toggle switch provides a mechanism to enable and disable transmission for each port in the Remote Process Group independently. Those ports that are connected but are not currently transmitting can be configured by clicking the pencil icon ( ) below the on/off toggle switch. Clicking this icon will allow the DFM to change the number of Concurrent Tasks, whether or not compression should be used when transmitting data to or from this port, and Batch Settings.

For an Input Port, the batch settings control how Logbot ETL sends data to the remote input port in a transaction. Logbot ETL will transfer flow files, as they are queued in incoming relationships, until any of the limits (Count, Size, Duration) is met. If none of the settings are configured, a 500 milliseconds batch duration is used by default.

For an Output Port, the batch settings tells the remote Logbot ETL how Logbot ETL prefers to receive data from the remote output port in a transaction. The remote Logbot ETL will use the specified settings (Count, Size, Duration) to control the transfer of flow files. If none of the settings are configured, a 5 seconds batch duration is used by default.

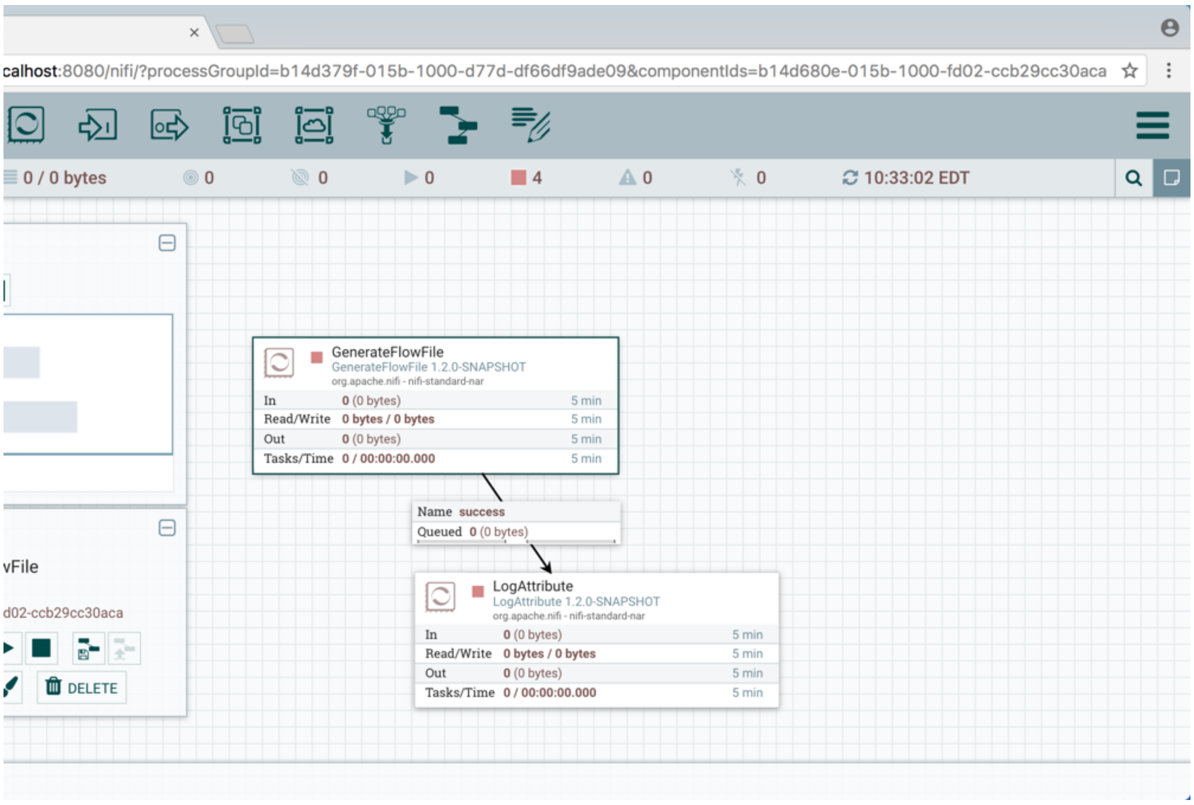
Navigating within a DataFlow

Logbot ETL provides various mechanisms for getting around a dataflow. The Logbot ETL User Interface section describes various ways to navigate around the Logbot ETL canvas; however, once a flow exists on the canvas, there are additional ways to get from one component to another. When multiple Process Groups exist in a flow, breadcrumbs appear at the bottom of the screen, providing a way to navigate between them. In addition, to enter a Process Group that is currently visible on the canvas, simply double-click it, thereby "drilling down" into it. Connections also provide a way to jump from one location to another within the flow. Right-click on a connection and select "Go to source" or "Go to destination" in order to jump to one end of the connection or another. This can be very useful in large, complex dataflows, where the connection lines may be long and span large areas of the canvas. Finally, all components provide the ability to jump forward or backward within the flow. Right-click any component (e.g., a processor, process group, port, etc.) and select either "Upstream connections" or "Downstream connections". A dialog window will open, showing the available upstream or downstream connections that the user may jump to. This can be especially useful when trying to follow a dataflow in a backward direction. It is typically easy to follow the path of a dataflow from start to finish, drilling down into nested process groups; however, it can be more difficult to follow the dataflow in the other direction.

Component Linking

A hyperlink can be used to navigate directly to a component on the Logbot ETL canvas. This is especially useful when Multi-Tenant Authorization is configured. For example, a URL can be given to a user to direct them to the specific process group to which they have privileges.

The default URL for a Logbot ETL instance is `http://<hostname>:8080/Logbot ETL`, which points to the root process group. When a component is selected on the canvas, the URL is updated with the component's process group id and component id in the form `http://<hostname>:8080/Logbot ETL/?processGroupId=<UUID>&componentIds=<UUIDs>;`. In the following screenshot, the GenerateFlowFile processor in the process group PG1 is the selected component:


calhost:8080/nifi/?processGroupId=b14d379f-015b-1000-d77d-df66df9ade09&componentIds=b14d680e-015b-1000-fd02-ccb29cc30aca

GenerateFlowFile
GenerateFlowFile 1.2.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

LogAttribute
LogAttribute 1.2.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

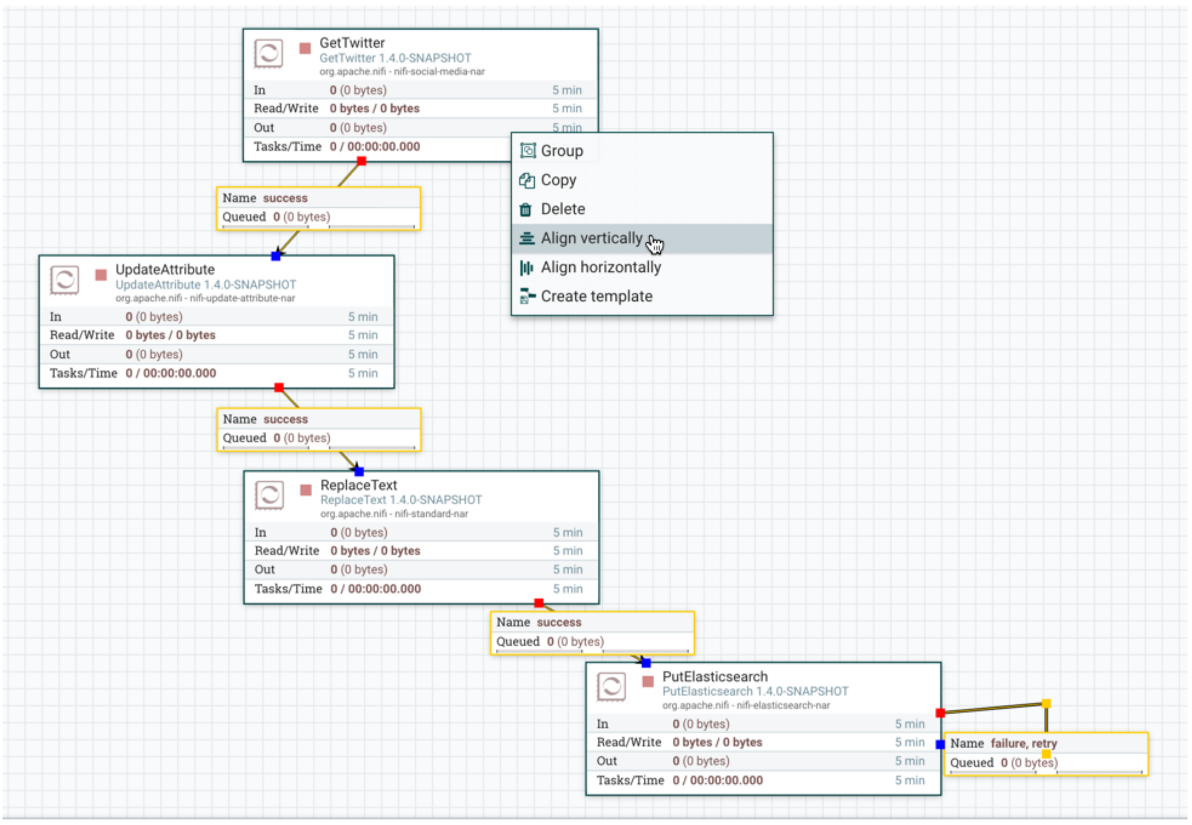
Linking to multiple components on the canvas is supported, with the restriction that the length of the URL cannot exceed a 2000 character limit.

Component Alignment

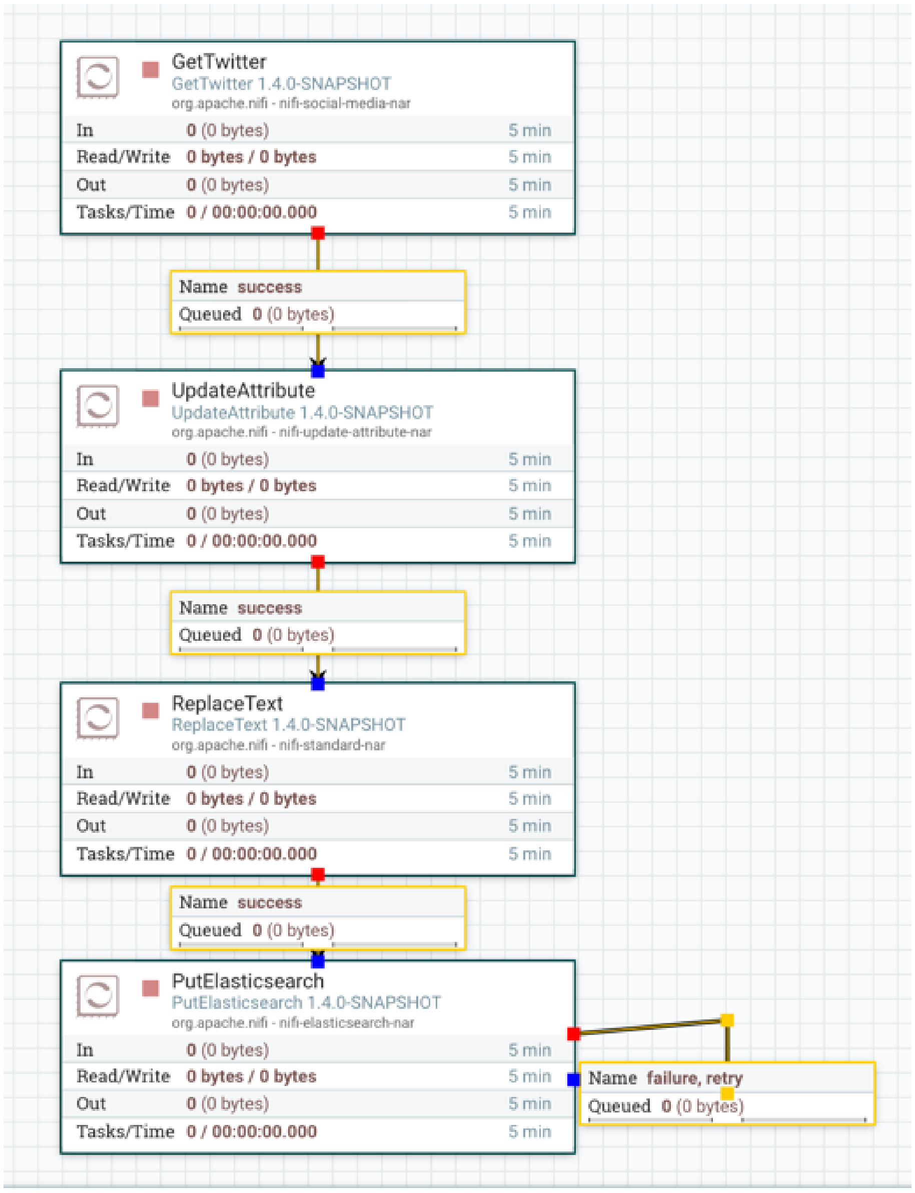
Components on the Logbot ETL canvas can be aligned to more precisely arrange your dataflow. To do this, first select all the components you want to align. Then right-click to see the context menu and select “Align vertically” or “Align horizontally” depending on your desired result.

Align Vertically

Here is an example of aligning components vertically on your canvas. With all components selected/highlighted, right-click:


GetTwitter
GetTwitter 1.4.0-SNAPSHOT
org.apache.nifi - nifi-social-media-nar
UpdateAttribute
UpdateAttribute 1.4.0-SNAPSHOT
org.apache.nifi - nifi-update-attribute-nar
ReplaceText
ReplaceText 1.4.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
PutElasticsearch
PutElasticsearch 1.4.0-SNAPSHOT
org.apache.nifi - nifi-elasticsearch-nar
Name success
Queued 0 (0 bytes)
Name failure, retry
Group
Copy
Delete
Align vertically
Align horizontally
Create template

and select "Align vertically" to achieve these results:



GetTwitter
GetTwitter 1.4.0-SNAPSHOT
org.apache.nifi - nifi-social-media-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

UpdateAttribute
UpdateAttribute 1.4.0-SNAPSHOT
org.apache.nifi - nifi-update-attribute-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

ReplaceText
ReplaceText 1.4.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

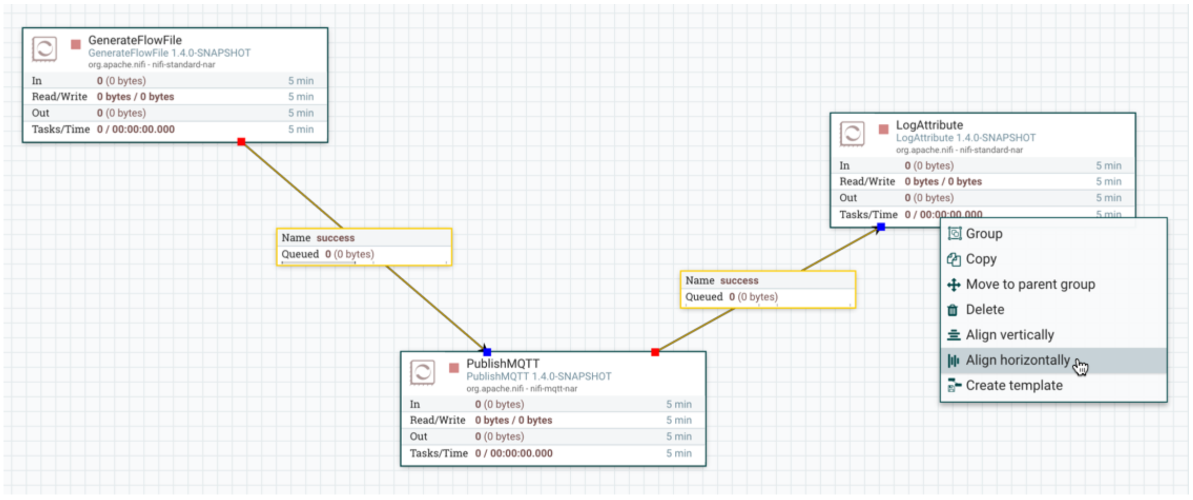
PutElasticsearch
PutElasticsearch 1.4.0-SNAPSHOT
org.apache.nifi - nifi-elasticsearch-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

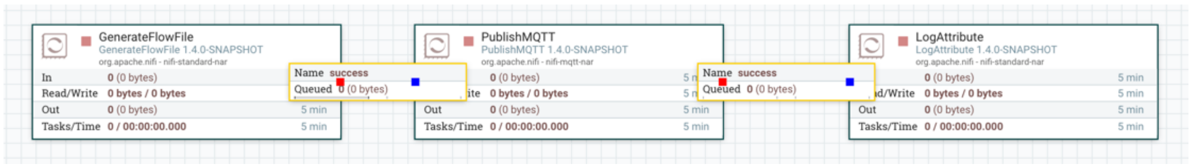
Name failure, retry
Queued 0 (0 bytes)

Align Horizontally

Here is an example of aligning components horizontally on your canvas. With all components selected/highlighted, right-click:


GenerateFlowFile
GenerateFlowFile 1.4.0-SNAPSHOT
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
PublishMQTT
PublishMQTT 1.4.0-SNAPSHOT
org.apache.nifi - nifi-mqtt-nar
LogAttribute
LogAttribute 1.4.0-SNAPSHOT
Group
Copy
Move to parent group
Delete
Align vertically
Align horizontally
Create template

and select "Align horizontally" to achieve these results:



Search Components in DataFlow

Logbot ETL UI provides searching functionality in order to help easily find components on the canvas. You can use search to find components by name, type, identifier, configuration properties, and their values. Search also makes it possible to refine and narrow the search result based on certain conditions using Filters and Keywords.

Example 1: The result will contain components that match for "processor1".

processor1

Filters

Filters can be added to the search box as key-value pairs where the keys are predefined and check certain conditions based on the given value. The syntax is "key:value".

Example 2: The search will be executed under Process Groups (directly or via contained Process Groups) containing the string "myGroup" in their name or id. The result will contain components that match for "processor1".

group:myGroup processor1

Filters can be used together with other search terms and multiple filters can be used. The only constraint is that the search must start with the filters. Unknown filters or known filters with unknown values are ignored. If the same filter key appears multiple times, the first will be used. The order of different filters has no effect on the result.

Example 3: Search will be restricted to the currently active process group (and process groups within that). The result will contain components that match for "import" but property matches will be excluded.

scope:here properties:exclude import

The supported filters are the following:

scope: This filter narrows the scope of the search based on the user’s currently active Process Group. The only valid value is "here". The usage of this filter looks like "scope:here". Any other value is considered as invalid, thus the filter will be ignored during search.

group: This filter narrows the scope of the search based on the provided Process Group name or id. Search will be restricted to groups (and their components - including subgroups and their components) the names or ids of which match the filter value. If no group matches the filter, the result list will be empty.

properties: With this, users can prevent property matches to appear in the search result. Valid values are: "no", "none", "false", "exclude" and "0".

Keywords

Users can use pre-defined (case-insensitive) keywords in the search box that will check certain conditions.

Example 4: "disabled" will be treated both as keyword and regular search term. The result will contain disabled Ports and Processors as all other components that match for "disabled" in any way.

```
disabled
```

Keywords can be used with filters (see below) but not with other search terms (otherwise they won't be treated as keywords) and only one keyword can be used at a time. Note however that keywords will also be treated as general search terms at the same time.

Example 5: Search will be restricted to the currently selected process group (and its sub process groups). "invalid" here (as it is alone after the filter) will be treated both as a keyword and a regular search term. The result will contain invalid Processors and Ports as well as all other components that match for "invalid" in any way.

```
scope:here invalid
```

The supported keywords are the following:

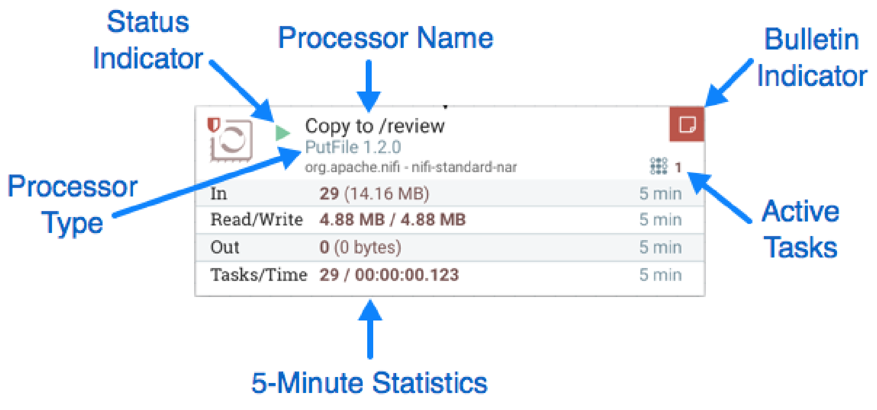
- **Scheduled state**
  - **disabled**: Adds disabled Ports and Processors to the result list.
  - **invalid**: Adds Ports and Processors to the result list where the component is invalid.
  - **running**: Adds running Ports and Processors to the result list.
  - **stopped**: Adds stopped Ports and Processors to the result list.
  - **validating**: Adds Processors to the result list that are validating at the time.
- **Scheduling strategy**
  - **event**: Adds Processors to the result list where the Scheduling Strategy is "Event Driven".
  - **timer**: Adds Processors to the result list where the Scheduling Strategy is "Timer Driven".
- **Execution**
  - **primary:** Adds Processors to the result list that are set to run on the primary node only (whether if the Processor is currently running or not).
- **Back pressure**
  - **back pressure**: Adds Connections to the result list that are applying back pressure at the time.
  - **pressure**: See "back pressure".
- **Expiration**
  - **expiration**: Adds Connections to the result list that contain expired Flow Files.
  - **expires**: See "expiration".
- **Transmission**
  - **not transmitting**: Adds Remote Process Groups to the result list that are not transmitting data at the time.
  - **transmitting**: Adds Remote Process Groups to the result list that are transmitting data at the time.
  - **transmission disabled**: See "not transmitting".
  - **transmitting enabled**: See "transmitting".

Monitoring of DataFlow

Logbot ETL provides a great deal of information about the DataFlow in order to monitor its health and status. The Status bar provides information about the overall system health (see Logbot ETL User Interface). Processors, Process Groups, and Remote Process Groups provide fine-grained details about their operations. Connections and Process Groups provide information about the amount of data in their queues. The Summary Page provides information about all of the components on the canvas in a tabular format and also provides System Diagnostics that include disk usage, CPU utilization, and Java Heap and Garbage Collection information. In a clustered environment, this information is available per-node or as aggregates across the entire cluster. We will explore each of these monitoring artifacts below.

Anatomy of a Processor

Logbot ETL provides a sigLogbot ETLcant amount of information about each Processor on the canvas. The following diagram shows the anatomy of a Processor:



Copy to /review
PutFile 1.2.0
org.apache.nifi - nifi-standard-nar
1

In	29 (14.16 MB)	5 min
Read/Write	4.88 MB / 4.88 MB	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	29 / 00:00:00.123	5 min

The image outlines the following elements:

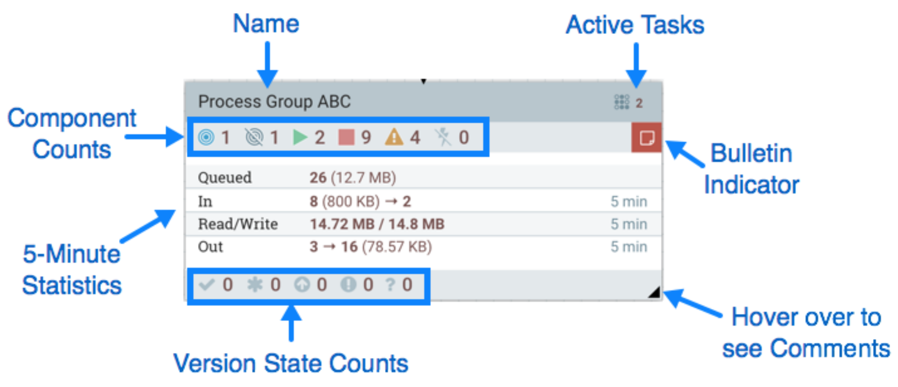
- **Processor Type**: Logbot ETL provides several different types of Processors in order to allow for a wide range of tasks to be performed. Each type of Processor is designed to perform one specific task. The Processor type (PutFile, in this example) describes the task that this Processor performs. In this case, the Processor writes a FlowFile to disk - or "Puts" a FlowFile to a File.
- **Bulletin Indicator**: When a Processor logs that some event has occurred, it generates a Bulletin to notify those who are monitoring Logbot ETL via the User Interface. The DFM is able to configure which bulletins should be displayed in the User Interface by updating the "Bulletin level" field in the "Settings" tab of the Processor configuration dialog. The default value is `WARN`, which means that only warnings and errors will be displayed in the UI. This icon is not present unless a Bulletin exists for this Processor. When it is present, hovering over the icon with the mouse will provide a tooltip explaining the message provided by the Processor as well as the Bulletin level. If the instance of Logbot ETL is clustered, it will also show the Node that emitted the Bulletin. Bulletins automatically expire after five minutes.
- **Status Indicator**: Shows the current Status of the Processor. The following indicators are possible:
    - ▶ **Running**: The Processor is currently running.
    - 🟥 **Stopped**: The Processor is valid and enabled but is not running.
    - ⚠ **Invalid**: The Processor is enabled but is not currently valid and cannot be started. Hovering over this icon will provide a tooltip indicating why the Processor is not valid.
    - **Disabled**: The Processor is not running and cannot be started until it has been enabled. This status does not indicate whether or not the Processor is valid.
- **Processor Name**: This is the user-defined name of the Processor. By default, the name of the Processor is the same as the Processor Type. In the example, this value is "Copy to /review".
- **Active Tasks**: The number of tasks that this Processor is currently executing. This number is constrained by the "Concurrent tasks" setting in the "Scheduling" tab of the Processor configuration dialog. Here, we can see that the Processor is currently performing one task. If the Logbot ETL instance is clustered, this value represents the number of tasks that are currently executing across all nodes in the cluster.
- **5-Minute Statistics**: The Processor shows several different statistics in tabular form. Each of these statistics represents the amount of work that has been performed in the past five

minutes. If the Logbot ETL instance is clustered, these values indicate how much work has been done by all of the Nodes combined in the past five minutes. These metrics are:

- **In**: The amount of data that the Processor has pulled from the queues of its incoming Connections. This value is represented as <count> (<size>) where <count> is the number of FlowFiles that have been pulled from the queues and <size> is the total size of those FlowFiles' content. In this example, the Processor has pulled 29 FlowFiles from the input queues, for a total of 14.16 megabytes (MB).
- **Read/Write**: The total size of the FlowFile content that the Processor has read from disk and written to disk. This provides valuable information about the I/O performance that this Processor requires. Some Processors may only read the data without writing anything while some will not read the data but will only write data. Others will neither read nor write data, and some Processors will both read and write data. In this example, we see that in the past five minutes, this Processor has read 4.88 MB of the FlowFile content and has written 4.88 MB as well. This is what we would expect, since this Processor simply copies the contents of a FlowFile to disk. Note, however, that this is not the same as the amount of data that it pulled from its input queues. This is because some of the files that it pulled from the input queues already exist in the output directory, and the Processor is configured to route FlowFiles to failure when this occurs. Therefore, for those files which already existed in the output directory, data was neither read nor written to disk.
- **Out**: The amount of data that the Processor has transferred to its outbound Connections. This does not include FlowFiles that the Processor removes itself, or FlowFiles that are routed to connections that are auto-terminated. Like the "In" metric above, this value is represented as <count> (<size>) where <count> is the number of FlowFiles that have been transferred to outbound Connections and <size> is the total size of those FlowFiles' content. In this example, all of the Relationships are configured to be auto-terminated, so no FlowFiles are reported as having been transferred Out.
- **Tasks/Time**: The number of times that this Processor has been triggered to run in the past 5 minutes, and the amount of time taken to perform those tasks. The format of the time is <hour>:<minute>:<second>. Note that the amount of time taken can exceed five minutes, because many tasks can be executed in parallel. For instance, if the Processor is scheduled to run with 60 Concurrent tasks, and each of those tasks takes one second to complete, it is possible that all 60 tasks will be completed in a single second. However, in this case we will see the Time metric showing that it took 60 seconds, instead of 1 second. This time can be thought of as "System Time," or said another way, this value is 60 seconds because that’s the amount of time it would have taken to perform the action if only a single concurrent task were used.

Anatomy of a Process Group

The Process Group provides a mechanism for grouping components together into a logical construct in order to organize the DataFlow in a way that makes it more understandable from a higher level. The following image highlights the different elements that make up the anatomy of a Process Group:



Queued	26 (12.7 MB)	
In	8 (800 KB) → 2	5 min
Read/Write	14.72 MB / 14.8 MB	5 min
Out	3 → 16 (78.57 KB)	5 min

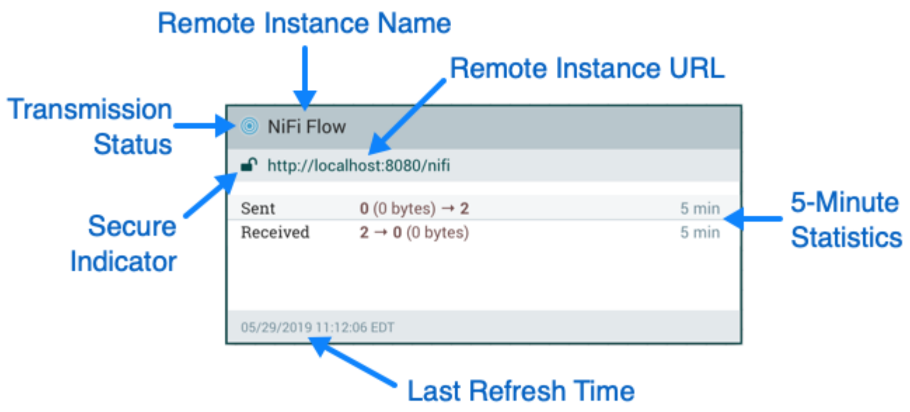
The Process Group consists of the following elements:

- **Name**: This is the user-defined name of the Process Group. This name is set when the Process Group is added to the canvas. The name can later by changed by right-clicking on the Process Group and clicking the "Configure" menu option. In this example, the name of the Process Group is "Process Group ABC."
- **Bulletin Indicator**: When a child component of a Process Group emits a bulletin, that bulletin is propagated to the component’s parent Process Group, as well. When any component has an active Bulletin, this indicator will appear, allowing the user to hover over the icon with the mouse to see the Bulletin.
- **Active Tasks**: The number of tasks that are currently executing by the components within this Process Group. Here, we can see that the Process Group is currently performing two tasks. If the Logbot ETL instance is clustered, this value represents the number of tasks that are currently executing across all nodes in the cluster.
- **Statistics**: Process Groups provide statistics about the amount of data that has been processed by the Process Group in the past 5 minutes as well as the amount of data currently enqueued within the Process Group. The following elements comprise the "Statistics" portion of a Process Group:
    - **Queued**: The number of FlowFiles currently enqueued within the Process Group. This field is represented as <count> (<size>) where <count> is the number of FlowFiles that are currently enqueued in the Process Group and <size> is the total size of those FlowFiles' content. In this example, the Process Group currently has 26 FlowFiles enqueued with a total size of 12.7 megabytes (MB).
    - **In**: The number of FlowFiles that have been transferred into the Process Group through all of its Input Ports over the past 5 minutes. This field is represented as <count> / <size> → <ports> where <count> is the number of FlowFiles that have entered the Process Group in the past 5 minutes, <size> is the total size of those FlowFiles' content and <ports> is the number of Input Ports. In this example, 8 FlowFiles have entered the Process Group with a total size of 800 KB and two Input Ports exist.
    - **Read/Write**: The total size of the FlowFile content that the components within the Process Group have read from disk and written to disk. This provides valuable information about the I/O performance that this Process Group requires. In this

example, we see that in the past five minutes, components within this Process Group have read 14.72 MB of the FlowFile content and have written 14.8 MB.
    - **Out**: The number of FlowFiles that have been transferred out of the Process Group through its Output Ports over the past 5 minutes. This field is represented as <ports> → <count> (<size>) where <ports> is the number of Output Ports, <count> is the number of FlowFiles that have exited the Process Group in the past 5 minutes and <size> is the total size of those FlowFiles' content. In this example, there are three Output Ports, 16 FlowFiles have exited the Process Group and their total size is 78.57 KB.
- **Component Counts**: The Component Counts element provides information about how many components of each type exist within the Process Group. The following provides information about each of these icons and their meanings:
    - **Transmitting Ports**: The number of Remote Process Group Ports that currently are configured to transmit data to remote instances of Logbot ETL or pull data from remote instances of Logbot ETL.
    - **Non-Transmitting Ports**: The number of Remote Process Group Ports that are currently connected to components within this Process Group but currently have their transmission disabled.
    - ▶ **Running Components**: The number of Processors, Input Ports, and Output Ports that are currently running within this Process Group.
    - ■ **Stopped Components**: The number of Processors, Input Ports, and Output Ports that are currently not running but are valid and enabled. These components are ready to be started.
    - ⚠ **Invalid Components**: The number of Processors, Input Ports, and Output Ports that are enabled but are currently not in a valid state. This may be due to misconfigured properties or missing Relationships.
    - **Disabled Components**: The number of Processors, Input Ports, and Output Ports that are currently disabled. These components may or may not be valid. If the Process Group is started, these components will not cause any errors but will not be started.
- **Version State Counts**: The Version State Counts element provides information about how many versioned process groups are within the Process Group. See Version States for more information.
- **Comments**: When the Process Group is added to the canvas, the user is given the option of specifying Comments in order to provide information about the Process Group. The comments can later be changed by right-clicking on the Process Group and clicking the "Configure" menu option.

Anatomy of a Remote Process Group

When creating a DataFlow, it is often necessary to transfer data from one instance of Logbot ETL to another. In this case, the remote instance of Logbot ETL can be thought of as a Process Group. For this reason, Logbot ETL provides the concept of a Remote Process Group. From the User Interface, the Remote Process Group looks similar to the Process Group. However, rather than showing information about the inner workings and state of a Remote Process Group, such as queue sizes, the information rendered about a Remote Process Group is related to the interaction that occurs between this instance of Logbot ETL and the remote instance.



The image above shows the different elements that make up a Remote Process Group. Here, we provide an explanation of the icons and details about the information provided.

- **Transmission Status**: The Transmission Status indicates whether or not data Transmission between this instance of Logbot ETL and the remote instance is currently enabled. The icon shown will be the Transmission Enabled icon (◎) if any of the Input Ports or Output Ports is currently configured to transmit or the Transmission Disabled icon (⊘) if all of the Input Ports and Output Ports that are currently connected are stopped.
- **Remote Instance Name**: This is the name of the Logbot ETL instance that was reported by the remote instance. When the Remote Process Group is first created, before this information has been obtained, the URL(s) of the remote instance will be shown here instead.
- **Remote Instance URL**: This is the URL of the remote instance that the Remote Process Group points to. This URL is entered when the Remote Process Group is added to the canvas and it cannot be changed.
- **Secure Indicator**: This icon indicates whether or not communications with the remote Logbot ETL instance are secure. If communications with the remote instance are secure, this will be indicated by the "Locked" icon (🔒). If the communications are not secure, this will be indicated by the "Unlocked" icon (🔓). If the communications are secure, this instance of Logbot ETL will not be able to communicate with the remote instance until an administrator for the remote instance grants access. Whenever the Remote Process Group is added to the canvas, this will automatically initiate a request to have a user for this instance of Logbot ETL created on the remote instance. This instance will be unable to communicate with the remote instance until an administrator on the remote instance adds the user to the system and adds the "Logbot ETL" role to the user. In the event that communications are not secure, the Remote Process Group is able to receive data from anyone, and the data is not encrypted while it is transferred between instances of Logbot ETL.
- **5-Minute Statistics**: Two statistics are shown for Remote Process Groups: **Sent** and **Received**. Both of these are in the format <count> (<size>) where <count> is the number of

FlowFiles that have been sent or received in the previous five minutes and <size> is the total size of those FlowFiles' content.

- **Last Refresh Time**: The information that is pulled from a remote instance and rendered on the Remote Process Group in the User Interface is periodically refreshed in the background. This element indicates the time at which that refresh last happened, or if the information has not been refreshed for a sigLogbot ETLcant amount of time, the value will change to indicate *Remote flow not current*. Logbot ETL can be triggered to initiate a refresh of this information by right-clicking on the Remote Process Group and choosing the "Refresh remote" menu item.

Queue Interaction

The FlowFiles enqueued in a Connection can be viewed when necessary. The Queue listing is opened via `List queue` in a Connection's context menu. The listing will return the top 100 FlowFiles in the active queue according to the configured priority. The listing can be performed even if the source and destination are actively running.

Additionally, details for a Flowfile in the listing can be viewed by clicking the "Details" button (ℹ) in the left most column. From here, the FlowFile details and attributes are available as well as buttons for downloading or viewing the content. Viewing the content is only available if the `Logbot ETL.content.viewer.url` has been configured. If the source or destination of the Connection are actively running, there is a chance that the desired FlowFile will no longer be in the active queue.

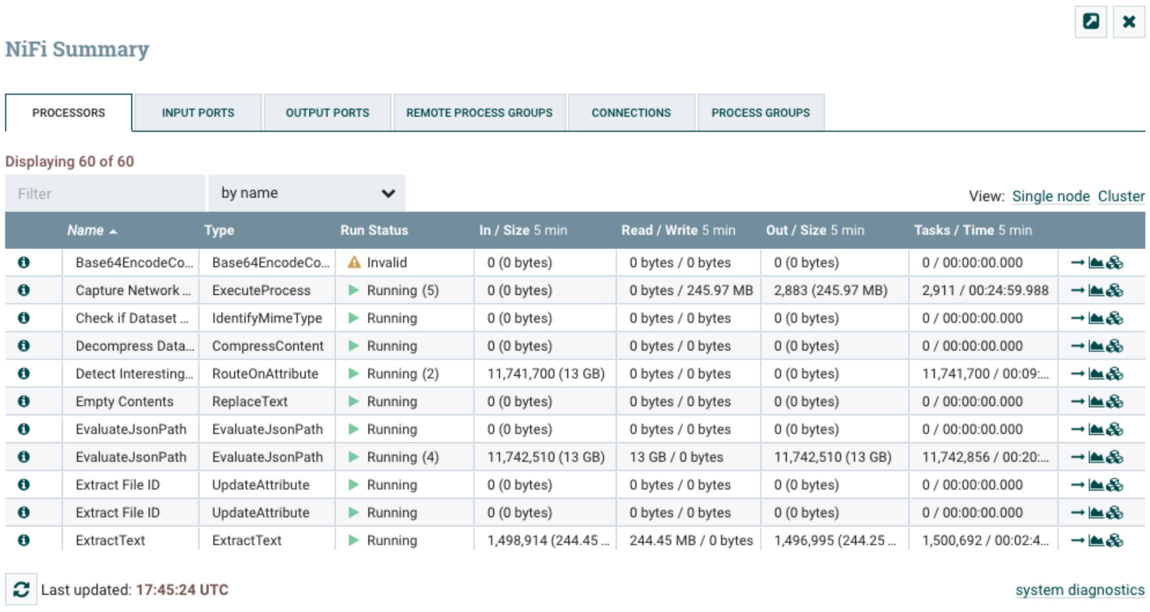
The FlowFiles enqueued in a Connection can also be deleted when necessary. The removal of the FlowFiles is initiated via `Empty queue` in the Connection's context menu. This action can also be performed if the source and destination are actively running.

If the analytics prediction feature is enabled, hovering over the queue will also reveal predicted statistics on when the queue may encounter back pressure, either due to the object count or content size meeting the current threshold settings. Predictions will only be available when Logbot ETL has enough data in its internal repository and if its model is accurate enough to broadcast a prediction. For more information, see the Analytics Framework section in the System Administrator's Guide.

Summary Page

While the Logbot ETL canvas is useful for understanding how the configured DataFlow is laid out, this view is not always optimal when trying to discern the status of the system. In order to help the user understand how the DataFlow is functioning at a higher level, Logbot ETL provides a Summary page. This page is available in the Global Menu in the top-right corner of the User Interface. See the Logbot ETL User Interface section for more information about the location of this toolbar.

The Summary Page is opened by selecting Summary from the Global Menu. This opens the Summary table dialog:



NiFi Summary

PROCESSORS | INPUT PORTS | OUTPUT PORTS | REMOTE PROCESS GROUPS | CONNECTIONS | PROCESS GROUPS

Displaying 60 of 60

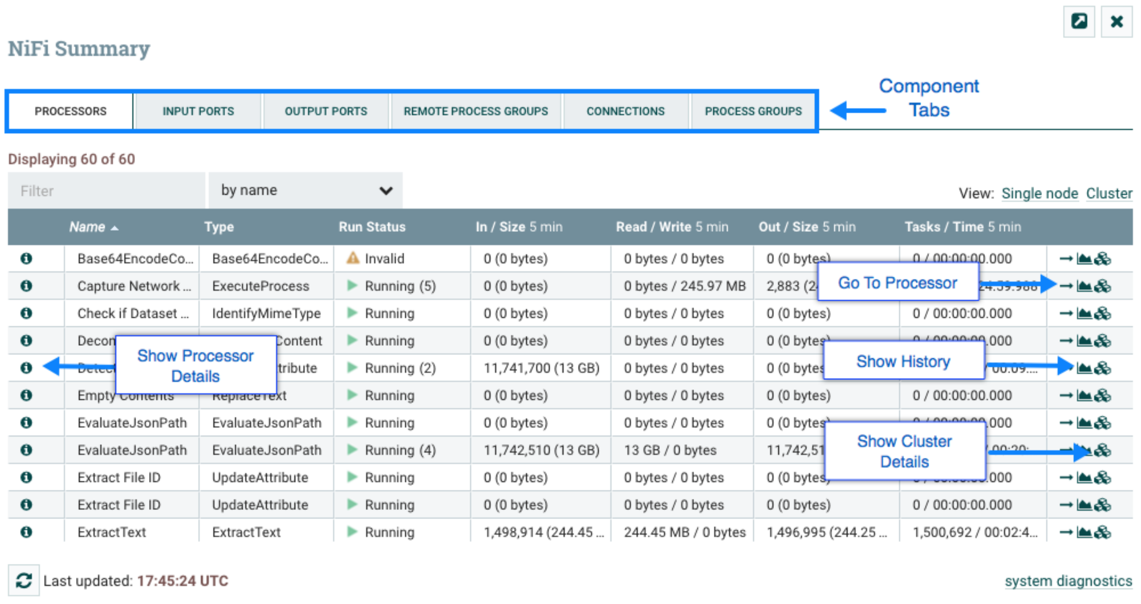
Filter | by name | View: Single node Cluster

	Name ▲	Type	Run Status	In / Size 5 min	Read / Write 5 min	Out / Size 5 min	Tasks / Time 5 min	
ℹ	Base64EncodeCo...	Base64EncodeCo...	⚠ Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Capture Network ...	ExecuteProcess	▶ Running (5)	0 (0 bytes)	0 bytes / 245.97 MB	2,883 (245.97 MB)	2,911 / 00:24:59.988	→
ℹ	Check if Dataset ...	IdentifyMimeType	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Decompress Data...	CompressContent	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Detect Interesting...	RouteOnAttribute	▶ Running (2)	11,741,700 (13 GB)	0 bytes / 0 bytes	0 (0 bytes)	11,741,700 / 00:09:...	→
ℹ	Empty Contents	ReplaceText	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	EvaluateJsonPath	EvaluateJsonPath	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	EvaluateJsonPath	EvaluateJsonPath	▶ Running (4)	11,742,510 (13 GB)	13 GB / 0 bytes	11,742,510 (13 GB)	11,742,856 / 00:20:...	→
ℹ	Extract File ID	UpdateAttribute	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Extract File ID	UpdateAttribute	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	ExtractText	ExtractText	▶ Running	1,498,914 (244.45 ...	244.45 MB / 0 bytes	1,496,995 (244.25 ...	1,500,692 / 00:02:4...	→

Last updated: 17:45:24 UTC

system diagnostics

This dialog provides a great deal of information about each of the components on the canvas. Below, we have annotated the different elements within the dialog in order to make the discussion of the dialog easier.



NiFi Summary

PROCESSORS | INPUT PORTS | OUTPUT PORTS | REMOTE PROCESS GROUPS | CONNECTIONS | PROCESS GROUPS

Displaying 60 of 60

Filter | by name | View: Single node Cluster

	Name ▲	Type	Run Status	In / Size 5 min	Read / Write 5 min	Out / Size 5 min	Tasks / Time 5 min	
ℹ	Base64EncodeCo...	Base64EncodeCo...	⚠ Invalid	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Capture Network ...	ExecuteProcess	▶ Running (5)	0 (0 bytes)	0 bytes / 245.97 MB	2,883 (2	24:59.988	→
ℹ	Check if Dataset ...	IdentifyMimeType	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	Decon	Content	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes	.000	→
ℹ	Detec	tribute	▶ Running (2)	11,741,700 (13 GB)	0 bytes / 0 bytes	0 (0 bytes	00:09...	→
ℹ	Empty Contents	ReplaceText	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	EvaluateJsonPath	EvaluateJsonPath	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes	.000	→
ℹ	EvaluateJsonPath	EvaluateJsonPath	▶ Running (4)	11,742,510 (13 GB)	13 GB / 0 bytes	11,742,51	00:20:	→
ℹ	Extract File ID	UpdateAttribute	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes	.000	→
ℹ	Extract File ID	UpdateAttribute	▶ Running	0 (0 bytes)	0 bytes / 0 bytes	0 (0 bytes)	0 / 00:00:00.000	→
ℹ	ExtractText	ExtractText	▶ Running	1,498,914 (244.45 ...	244.45 MB / 0 bytes	1,496,995 (244.25 ...	1,500,692 / 00:02:4...	→

Last updated: 17:45:24 UTC

system diagnostics

The Summary page is primarily comprised of a table that provides information about each of the components on the canvas. Above this table is a set of five tabs that can be used to view the different types of components. The information provided in the table is the same information that is provided for each component on the canvas. Each of the columns in the table may be sorted by clicking on the heading of the column. For more on the types of information displayed, see the sections Anatomy of a Processor, Anatomy of a Process Group, and Anatomy of a Remote Process Group above.

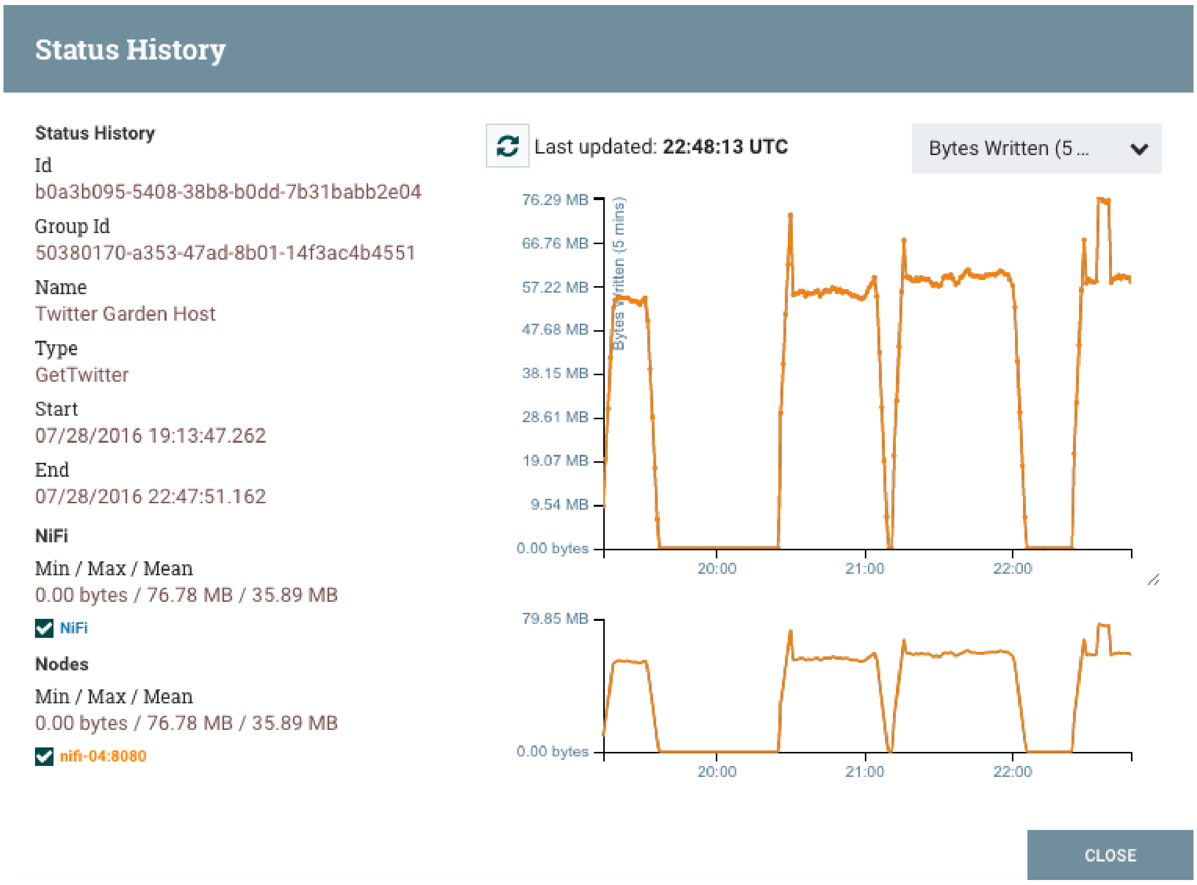
The Summary page also includes the following elements:

- **Bulletin Indicator**: As in other places throughout the User Interface, when this icon is present, hovering over the icon will provide information about the Bulletin that was generated, including the message, the severity level, the time at which the Bulletin was generated, and (in a clustered environment) the node that generated the Bulletin. Like all the columns in the Summary table, this column where bulletins are shown may be sorted by clicking on the heading so that all the currently existing bulletins are shown at the top of the list.
- **Details**: Clicking the Details icon will provide the user with the details of the component. This dialog is the same as the dialog provided when the user right-clicks on the component and chooses the "View Configuration" menu item.
- **Go To**: Clicking this button will close the Summary page and take the user directly to the component on the Logbot ETL canvas. This may change the Process Group that the user is currently in. This icon is not available if the Summary page has been opened in a new browser tab or window (by clicking the "Pop Out" button, as described below).
- **Status History**: Clicking the Status History icon will open a new dialog that shows a historical view of the statistics that are rendered for this component. See the section Historical Statistics of a Component for more information.
- **Refresh**: The "Refresh" button allows the user to refresh the information displayed without closing the dialog and opening it again. The time at which the information was last refreshed is shown just to the right of the "Refresh" button. The information on the page is not automatically refreshed.
- **Filter**: The Filter element allows users to filter the contents of the Summary table by typing in all or part of some criteria, such as a Processor Type or Processor Name. The types of filters available differ according to the selected tab. For instance, if viewing the Processor tab, the user is able to filter by name or by type. When viewing the Connections tab, the user is able to filter by source, by name, or by destination name. The filter is automatically applied when the contents of the text box are changed. Below the text box is an indicator of how many entries in the table match the filter and how many entries exist in the table.
- **Pop-Out**: When monitoring a flow, it is helpful to be able to open the Summary table in a separate browser tab or window. The "Pop Out" button, next to the "Close" button, will cause the entire Summary dialog to be opened in a new browser tab or window (depending on the configuration of the browser). Once the page is "popped out", the dialog is closed in the original browser tab/window. In the new tab/window, the "Pop Out" button and the "Go To" button will no longer be available.
- **System Diagnostics**: The System Diagnostics window provides information about how the system is performing with respect to system resource utilization. While this is intended mostly for administrators, it is provided in this view because it does provide a summary of the system. This dialog shows information such as CPU utilization, how full the disks are, and Java-specific metrics, such as memory size and utilization, as well as Garbage Collection information.

Historical Statistics of a Component

While the Summary table and the canvas show numeric statistics pertaining to the performance of a component over the past five minutes, it is often useful to have a view of historical statistics as well. This information is available by right-clicking on a component and choosing the "Status History" menu option or by clicking on the Status History in the Summary page (see Summary Page for more information).

The amount of historical information that is stored is configurable in the Logbot ETL properties but defaults to `24 hours`. For specific configuration information reference the Component Status Repository of the System Administrator’s Guide. When the Status History dialog is opened, it provides a graph of historical statistics:



Status History
Id
b0a3b095-5408-38b8-b0dd-7b31babb2e04
Group Id
50380170-a353-47ad-8b01-14f3ac4b4551
Name
Twitter Garden Host
Type
GetTwitter
Start
07/28/2016 19:13:47.262
End
07/28/2016 22:47:51.162
NiFi
Min / Max / Mean
0.00 bytes / 76.78 MB / 35.89 MB
✓ NiFi
Nodes
Min / Max / Mean
0.00 bytes / 76.78 MB / 35.89 MB
✓ nifi-04:8080

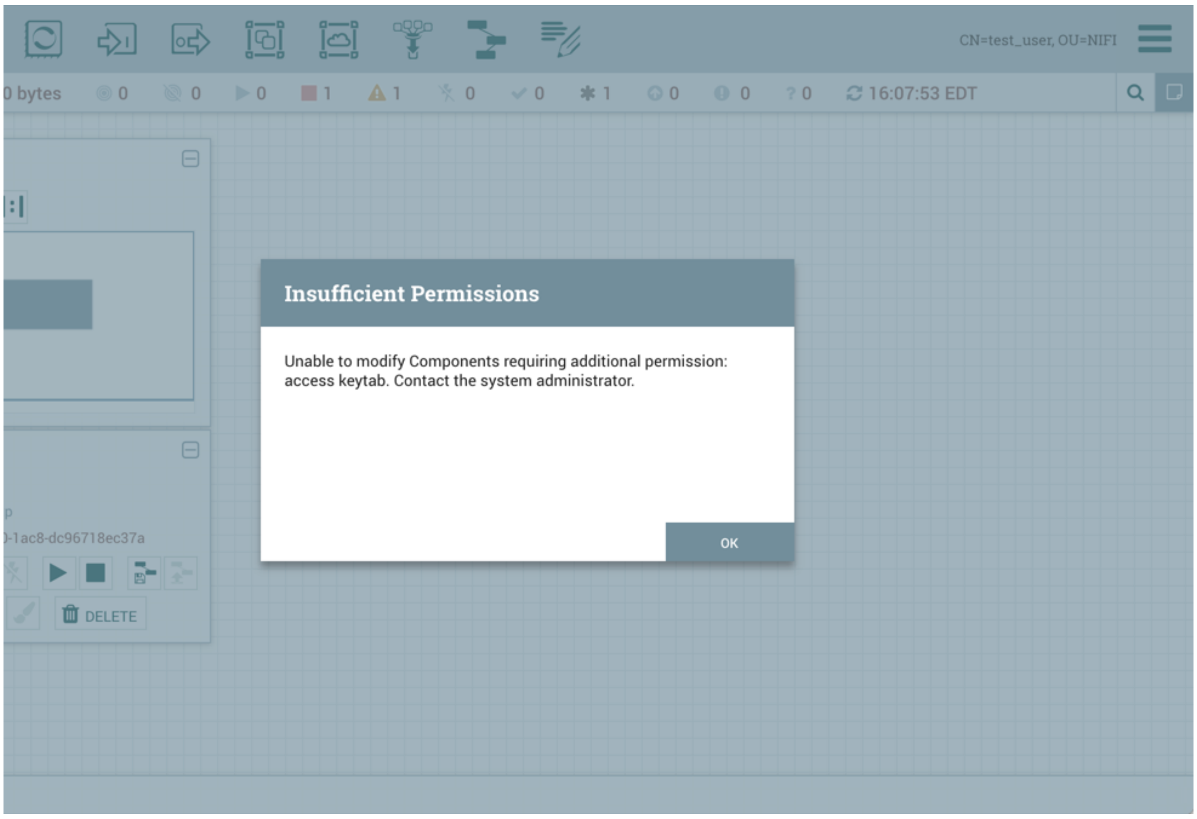
The left-hand side of the dialog provides information about the component that the stats are for, as well as a textual representation of the statistics being graphed. The following information is provided on the left-hand side:

- **Id**: The ID of the component for which the stats are being shown.
- **Group Id**: The ID of the Process Group in which the component resides.
- **Name**: The Name of the Component for which the stats are being shown.
- **Component-Specific Entries**: Information is shown for each different type of component. For example, for a Processor, the type of Processor is displayed. For a Connection, the source and destination names and IDs are shown.
- **Start**: The earliest time shown on the graph.
- **End**: The latest time shown on the graph.
- **Min/Max/Mean**: The minimum, maximum, and mean (arithmetic mean, or average) values are shown. These values are based only on the range of time selected, if any time range is

selected. If this instance of Logbot ETL is clustered, these values are shown for the cluster as a whole, as well as each individual node. In a clustered environment, each node is shown in a different color. This also serves as the graph's legend, showing the color of each node that is shown in the graph. Hovering the mouse over the Cluster or one of the nodes in the legend will also make the corresponding node bold in the graph.

The right-hand side of the dialog provides a drop-down list of the different types of metrics to render in the graphs below. The top graph is larger so as to provide an easier-to-read rendering of the information. In the bottom-right corner of this graph is a small handle ( ) that can be dragged to resize the graph. The blank areas of the dialog can also be dragged around to move the entire dialog.

The bottom graph is much shorter and provides the ability to select a time range. Selecting a time range here will cause the top graph to show only the time range selected, but in a more detailed manner. Additionally, this will cause the Min/Max/Mean values on the left-hand side to be recalculated. Once a selection has been created by dragging a rectangle over the graph, double-clicking on the selected portion will cause the selection to fully expand in the vertical direction (i.e., it will select all values in this time range). Clicking on the bottom graph without dragging will remove the selection.


Insufficient Permissions

Unable to modify Components requiring additional permission: access keytab. Contact the system administrator.

Templates

DFMs have the ability to build very large and complex DataFlows using Logbot ETL. This is achieved by using the basic components: Processor, Funnel, Input/Output Port, Process Group, and Remote Process Group. These can be thought of as the most basic building blocks for constructing a DataFlow. At times, though, using these small building blocks can become tedious if the same logic needs to be repeated several times.

To solve this issue, Logbot ETL provides the concept of a Template. A Template is a way of combining these basic building blocks into larger building blocks. Once a DataFlow has been created, parts of it can be formed into a Template. This Template can then be dragged onto the canvas, or can be exported as an XML file and shared with others. Templates received from others can then be imported into an instance of Logbot ETL and dragged onto the canvas.

Creating a Template

To create a Template, select the components that are to be a part of the template, and then click the "Create Template" ( ) button in the Operate Palette (See Logbot ETL User Interface for more information on the Operate Palette).

Clicking this button without selecting anything will create a Template that contains all of the contents of the current Process Group. This means that creating a Template with nothing selected while on the Root Process Group will create a single Template that contains the entire flow.

After clicking this button, the user is prompted to provide a name and an optional description for the template. Each template must have a unique name. After entering the name and optional description, clicking the "Create" button will generate the template and notify the user that the template was successfully created, or provide an appropriate error message if unable to create the template for some reason.

It is important to note that if any Processor that is Templated has a sensitive property (such as a password), the value of that sensitive property is not included in the Template. As a result, when dragging the Template onto the canvas, newly created Processors may not be valid if they are missing values for their sensitive properties. Additionally, any Connection that was selected when making the Template is not included in the Template if either the source or the destination of the Connection is not also included in the Template.

Importing a Template

After receiving a Template that has been exported from another Logbot ETL, the first step needed to use the template is to import the template into this instance of Logbot ETL. You may import templates into any Process Group where you have the appropriate authorization.

From the Operate Palette, click the "Upload Template" () button (see Logbot ETL User Interface for more information on the Operate Palette). This will display the Upload Template dialog. Click the find icon and use the File Selection dialog to choose which template file to upload. Select the file and click Open. Clicking the "Upload" button will attempt to import the Template into this instance of Logbot ETL. The Upload Template dialog will update to show "Success" or an error message if there was a problem importing the template.

Instantiating a Template

Once a Template has been created (see Creating a Template) or imported (see Importing a Template), it is ready to be instantiated, or added to the canvas. This is accomplished by dragging the Template icon ( ) from the Components Toolbar (see Logbot ETL User Interface) onto the canvas.

This will present a dialog to choose which Template to add to the canvas. After choosing the Template to add, simply click the "Add" button. The Template will be added to the canvas with the upper-left-hand side of the Template being placed wherever the user dropped the Template icon.

This leaves the contents of the newly instantiated Template selected. If there was a mistake, and this Template is no longer wanted, it may be deleted.

Managing Templates

One of the most powerful features of Logbot ETL Templates is the ability to easily export a Template to an XML file and to import a Template that has already been exported. This provides a very simple mechanism for sharing parts of a DataFlow with others. You can select Templates from the Global Menu (see Logbot ETL User Interface) to open a dialog that displays all of the Templates that are currently available, filter the templates to see only those of interest, export, and delete Templates.

Exporting a Template

Once a Template has been created, it can be shared with others in the Template Management page. To export a Template, locate the Template in the table. The Filter in the top-right corner can be used to help find the appropriate Template if several are available. Then click the "Download" button ( ). This will download the template as an XML file to your computer. This XML file can then be sent to others and imported into other instances of Logbot ETL (see Importing a Template).

Removing a Template

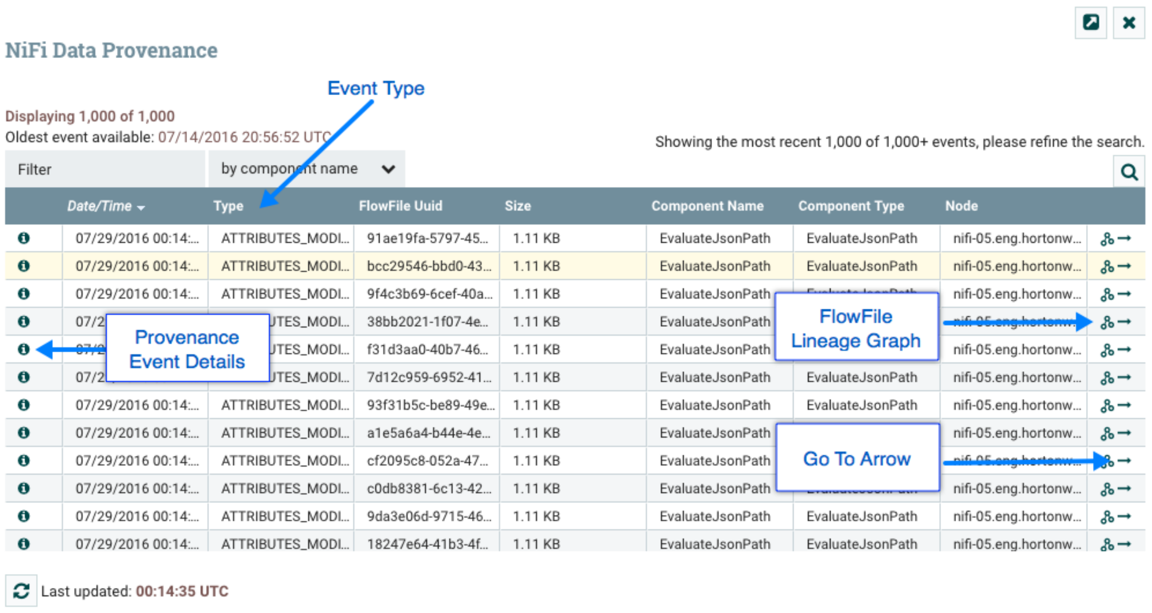
Once it is decided that a Template is no longer needed, it can be easily removed from the Template Management page. To delete a Template, locate it in the table (the Filter in the top-right corner may be used to find the appropriate Template if several are available) and click the "Delete" button ( ). This will prompt for confirmation. After confirming the deletion, the Template will be removed from this table and will no longer be available to add to the canvas.

Data Provenance

While monitoring a dataflow, users often need a way to determine what happened to a particular data object (FlowFile). Logbot ETL's Data Provenance page provides that information. Because Logbot ETL records and indexes data provenance details as objects flow through the system, users may perform searches, conduct troubleshooting and evaluate things like dataflow compliance and optimization in real time. By default, Logbot ETL updates this information every five minutes, but that is configurable.

To access the Data Provenance page, select "Data Provenance" from the Global Menu. This opens a dialog window that allows the user to see the most recent Data Provenance information available, search the information for specific items, and filter the search results. It is also possible to open additional dialog windows to see event details, replay data at any point within the dataflow, and see a graphical representation of the data's lineage, or path through the flow. (These features are described in depth below.)

When authorization is enabled, accessessing Data Provenance information requires the 'query provenance' Global Policy as well as the 'view provenance' Component Policy for the component which generated the event. In addition, access to event details which include FlowFile attributes and content require the 'view the data' Component Policy for the component which generated the event.



NiFi Data Provenance

Displaying 1,000 of 1,000
Oldest event available: 07/14/2016 20:56:52 UTC

Showing the most recent 1,000 of 1,000+ events, please refine the search.

Filter | by component name

Date/Time	Type	FlowFile Uuid	Size	Component Name	Component Type	Node
07/29/2016 00:14:...	ATTRIBUTES_MODI...	91ae19fa-5797-45...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	bcc29546-bbd0-43...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	9f4c3b69-6cef-40a...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
07/2	UTES_MODI...	38bb2021-1f07-4e...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
[illegible]	UTES_MODI...	f31d3aa0-40b7-46...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
07/2	UTES_MODI...	7d12c959-6952-41...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	93f31b5c-be89-49e...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	a1e5a6a4-b44e-4e...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	cf2095c8-052a-47...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	c0db8381-6c13-42...	1.11 KB	EvaluateJsonPath	[illegible]	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	9da3e06d-9715-46...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...
07/29/2016 00:14:...	ATTRIBUTES_MODI...	18247e64-41b3-4f...	1.11 KB	EvaluateJsonPath	EvaluateJsonPath	nifi-05.eng.hortonw...

Last updated: 00:14:35 UTC

Provenance Events

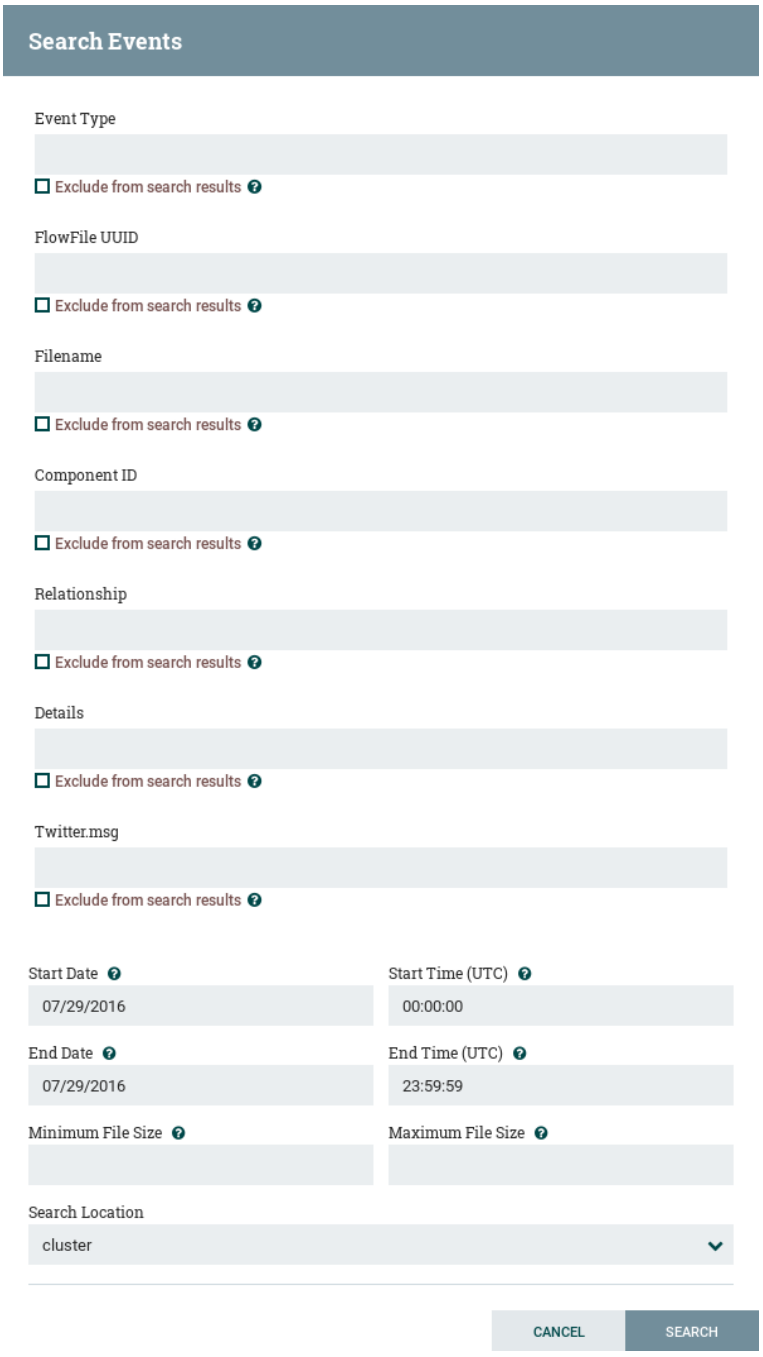
Each point in a dataflow where a FlowFile is processed in some way is considered a 'provenance event'. Various types of provenance events occur, depending on the dataflow design. For example, when data is brought into the flow, a RECEIVE event occurs, and when data is sent out of the flow, a SEND event occurs. Other types of processing events may occur, such as if the data is cloned (CLONE event), routed (ROUTE event), modified (CONTENT_MODIFIED or ATTRIBUTES_MODIFIED event), split (FORK event), combined with other data objects (JOIN event), and ultimately removed from the flow (DROP event).

The provenance event types are:

Provenance Event	Description
ADDINFO	Indicates a provenance event when additional information such as a new linkage to a new URI or UUID is added
ATTRIBUTES_MODIFIED	Indicates that a FlowFile's attributes were modified in some way
CLONE	Indicates that a FlowFile is an exact duplicate of its parent FlowFile
CONTENT_MODIFIED	Indicates that a FlowFile's content was modified in some way
CREATE	Indicates that a FlowFile was generated from data that was not received from a remote system or external process
DOWNLOAD	Indicates that the contents of a FlowFile were downloaded by a user or external entity
DROP	Indicates a provenance event for the conclusion of an object's life for some reason other than object expiration
EXPIRE	Indicates a provenance event for the conclusion of an object's life due to the object not being processed in a timely manner
FETCH	Indicates that the contents of a FlowFile were overwritten using the contents of some external resource
FORK	Indicates that one or more FlowFiles were derived from a parent FlowFile
JOIN	Indicates that a single FlowFile is derived from joining together multiple parent FlowFiles
RECEIVE	Indicates a provenance event for receiving data from an external process
REMOTE_INVOCATION	Indicates that a remote invocation was requested to an external endpoint (e.g. deleting a remote resource)
REPLAY	Indicates a provenance event for replaying a FlowFile
ROUTE	Indicates that a FlowFile was routed to a specified relationship and provides information about why the FlowFile was routed to this relationship
SEND	Indicates a provenance event for sending data to an external process
UNKNOWN	Indicates that the type of provenance event is unknown because the user who is attempting to access the event is not authorized to know the type

Searching for Events

One of the most common tasks performed in the Data Provenance page is a search for a given FlowFile to determine what happened to it. To do this, click the "Search" button in the upper-right corner of the Data Provenance page. This opens a dialog window with parameters that the user can define for the search. The parameters include the processing event of interest, distinguishing characteristics about the FlowFile or the component that produced the event, the timeframe within which to search, and the size of the FlowFile.



Search Events

Event Type

☐ Exclude from search results

FlowFile UUID

☐ Exclude from search results

Filename

☐ Exclude from search results

Component ID

☐ Exclude from search results

Relationship

☐ Exclude from search results

Details

☐ Exclude from search results

Twitter.msg

☐ Exclude from search results

Start Date: 07/29/2016

Start Time (UTC): 00:00:00

End Date: 07/29/2016

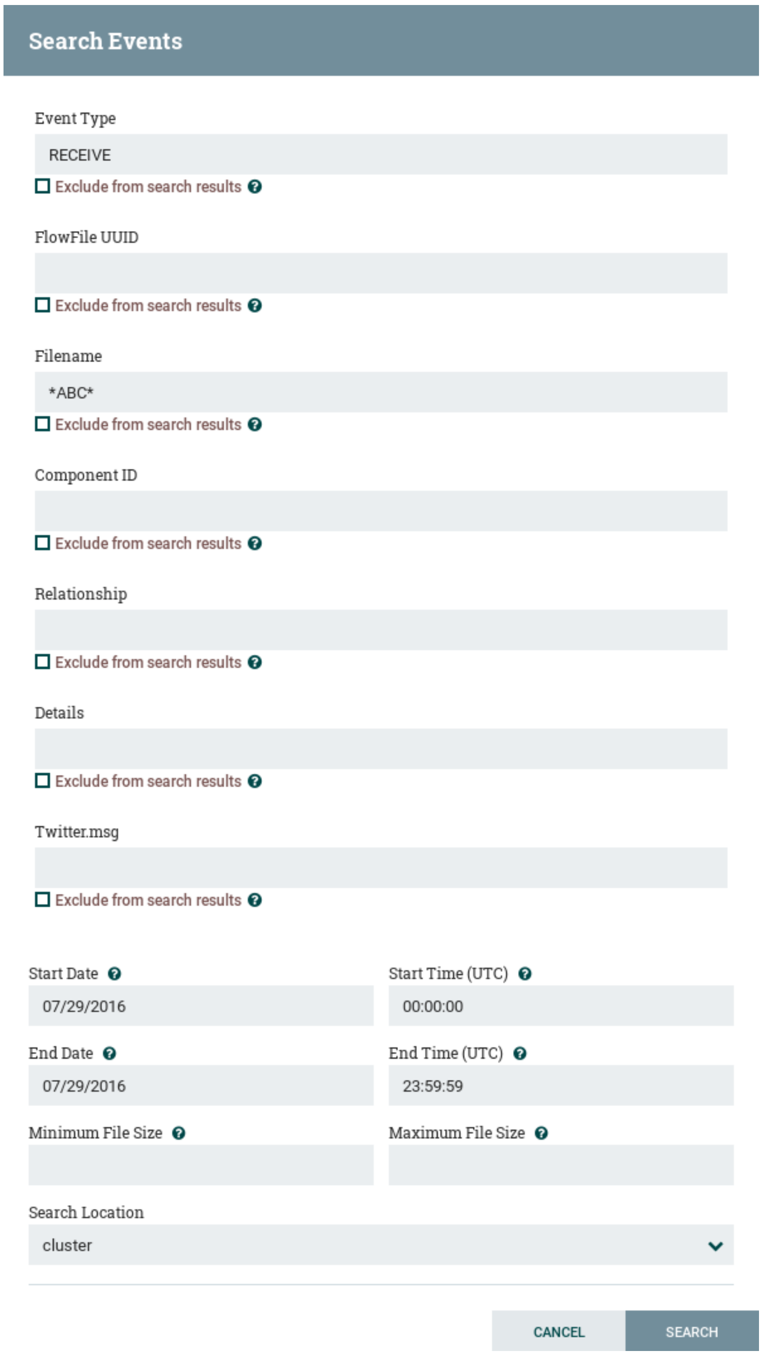
End Time (UTC): 23:59:59

Minimum File Size

Maximum File Size

Search Location: cluster

For example, to determine if a particular FlowFile was received, search for an Event Type of "RECEIVE" and include an identifier for the FlowFile, such as its uuid or filename. The asterisk (*) may be used as a wildcard for any number of characters. So, to determine whether a FlowFile with "ABC" anywhere in its filename was received at any time on July 29, 2016, the search shown in the following image could be performed:



Search Events

Event Type

RECEIVE

☐ Exclude from search results

FlowFile UUID

☐ Exclude from search results

Filename

ABC

☐ Exclude from search results

Component ID

☐ Exclude from search results

Relationship

☐ Exclude from search results

Details

☐ Exclude from search results

Twitter.msg

☐ Exclude from search results

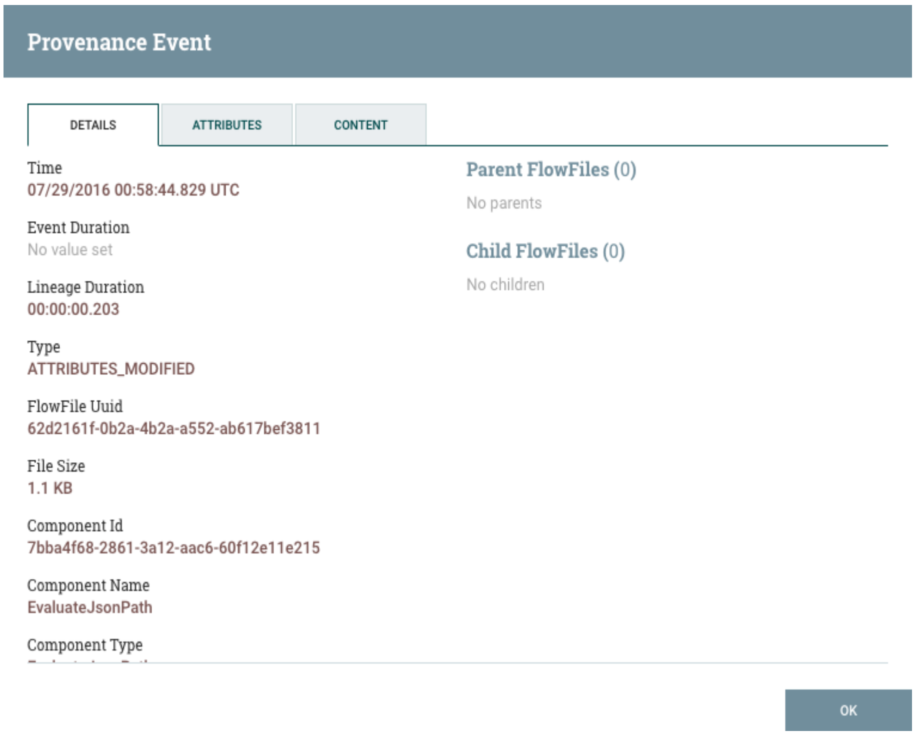
Start Date: 07/29/2016	Start Time (UTC): 00:00:00
End Date: 07/29/2016	End Time (UTC): 23:59:59
Minimum File Size:	Maximum File Size:

Search Location: cluster

If all filenames that do not have "ABC" anywhere in the filename is desired, then click the checkbox with the label "Exclude from search results" beneath this entry before performing the search.

Details of an Event

In the far-left column of the Data Provenance page, there is a "View Details" icon for each event ( ). Clicking this button opens a dialog window with three tabs: Details, Attributes, and Content.


Provenance Event

DETAILS | ATTRIBUTES | CONTENT

Time
07/29/2016 00:58:44.829 UTC

Event Duration
No value set

Lineage Duration
00:00:00.203

Type
ATTRIBUTES_MODIFIED

FlowFile Uuid
62d2161f-0b2a-4b2a-a552-ab617bef3811

File Size
1.1 KB

Component Id
7bba4f68-2861-3a12-aac6-60f12e11e215

Component Name
EvaluateJsonPath

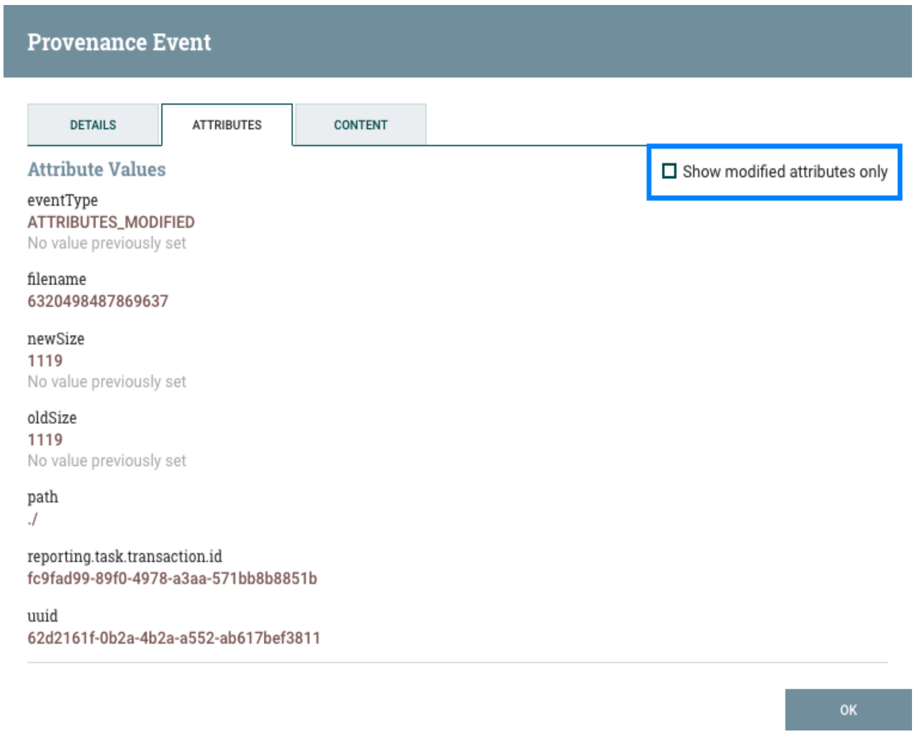
Component Type

Parent FlowFiles (0)
No parents

Child FlowFiles (0)
No children

The Details tab shows various details about the event, such as when it occurred, what type of event it was, and the component that produced the event. The information that is displayed will vary according to the event type. This tab also shows information about the FlowFile that was processed. In addition to the FlowFile’s UUID, which is displayed on the left side of the Details tab, the UUIDs of any parent or children FlowFiles that are related to that FlowFile are displayed on the right side of the Details tab.

The Attributes tab shows the attributes that exist on the FlowFile as of that point in the flow. In order to see only the attributes that were modified as a result of the processing event, the user may select the checkbox next to "Only show modified" in the upper-right corner of the Attributes tab.


Provenance Event

Attribute Values

eventType
ATTRIBUTES_MODIFIED
No value previously set

filename
6320498487869637

newSize
1119
No value previously set

oldSize
1119
No value previously set

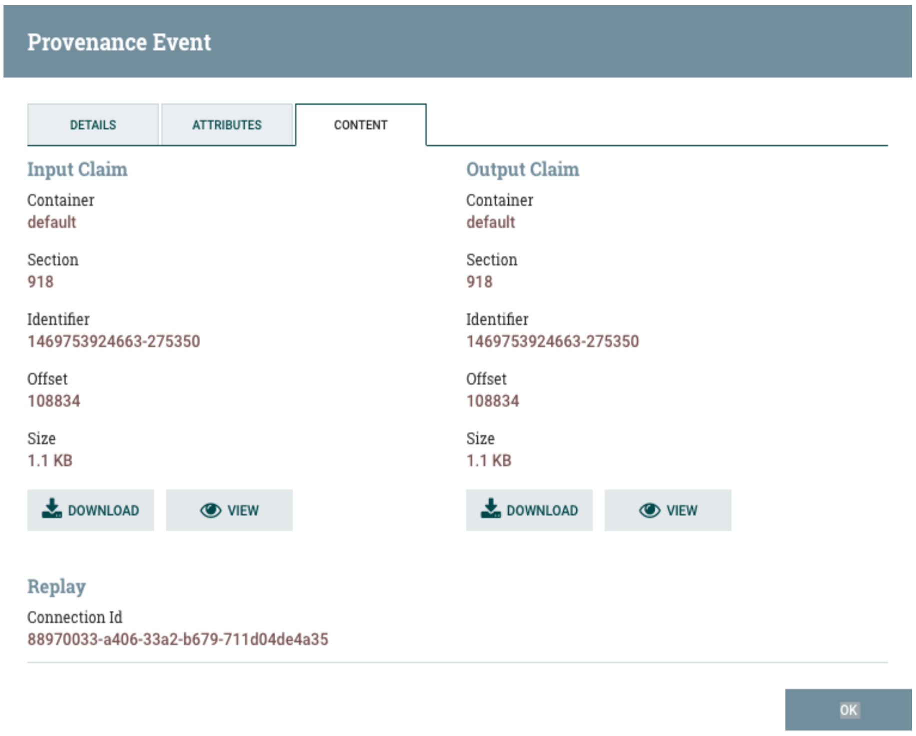
path
./

reporting.task.transaction.id
fc9fad99-89f0-4978-a3aa-571bb8b8851b

uuid
62d2161f-0b2a-4b2a-a552-ab617bef3811

Replaying a FlowFile

A DFM may need to inspect a FlowFile’s content at some point in the dataflow to ensure that it is being processed as expected. And if it is not being processed properly, the DFM may need to make adjustments to the dataflow and replay the FlowFile again. The Content tab of the View Details dialog window is where the DFM can do these things. The Content tab shows information about the FlowFile’s content, such as its location in the Content Repository and its size. In addition, it is here that the user may click the "Download" button to download a copy of the FlowFile’s content as it existed at this point in the flow. The user may also click the "Submit" button to replay the FlowFile at this point in the flow. Upon clicking "Submit", the FlowFile is sent to the connection feeding the component that produced this processing event.



Input Claim

Container
default

Section
918

Identifier
1469753924663-275350

Offset
108834

Size
1.1 KB

Output Claim

Container
default

Section
918

Identifier
1469753924663-275350

Offset
108834

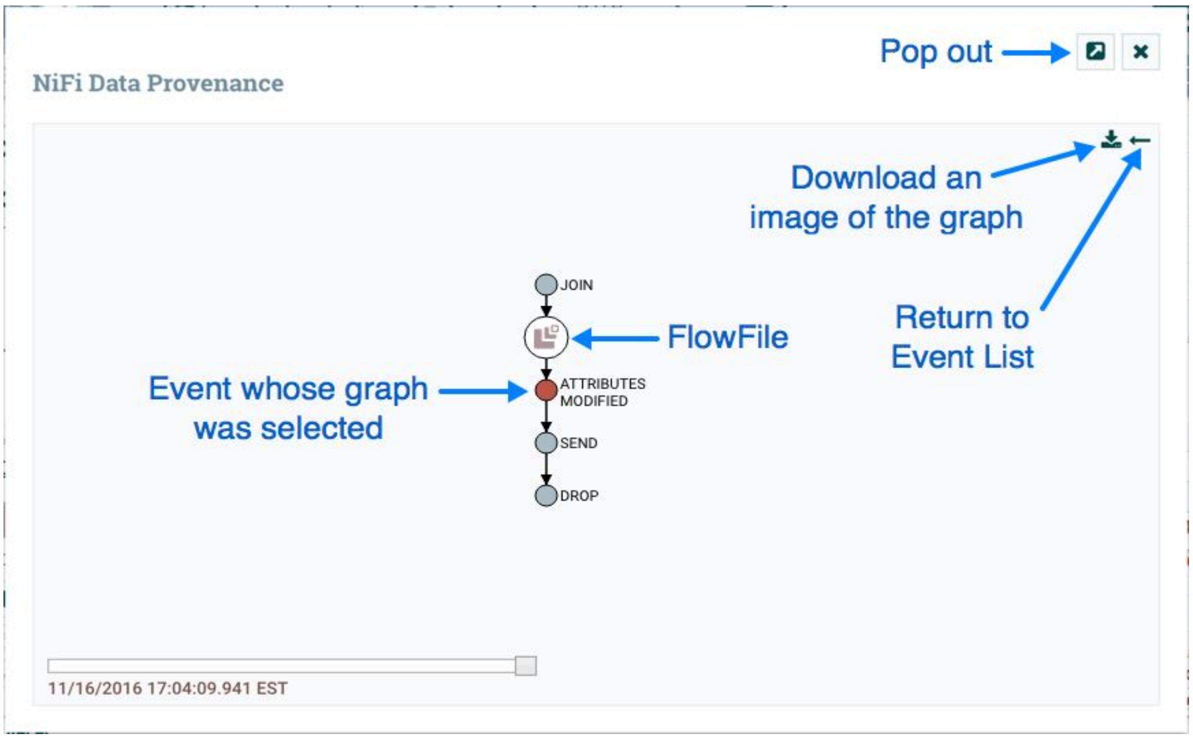
Size
1.1 KB

Replay

Connection Id
88970033-a406-33a2-b679-711d04de4a35

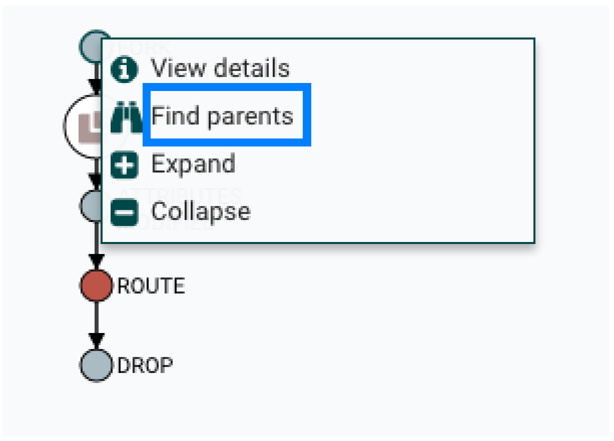
Viewing FlowFile Lineage

It is often useful to see a graphical representation of the lineage or path a FlowFile took within the dataflow. To see a FlowFile’s lineage, click on the "Show Lineage" icon ( ) in the far-right column of the Data Provenance table. This opens a graph displaying the FlowFile ( ) and the various processing events that have occurred. The selected event will be highlighted in red. It is possible to right-click or double-click on any event to see that event’s details (see Details of an Event). To see how the lineage evolved over time, click the slider at the bottom-left of the window and move it to the left to see the state of the lineage at earlier stages in the dataflow.

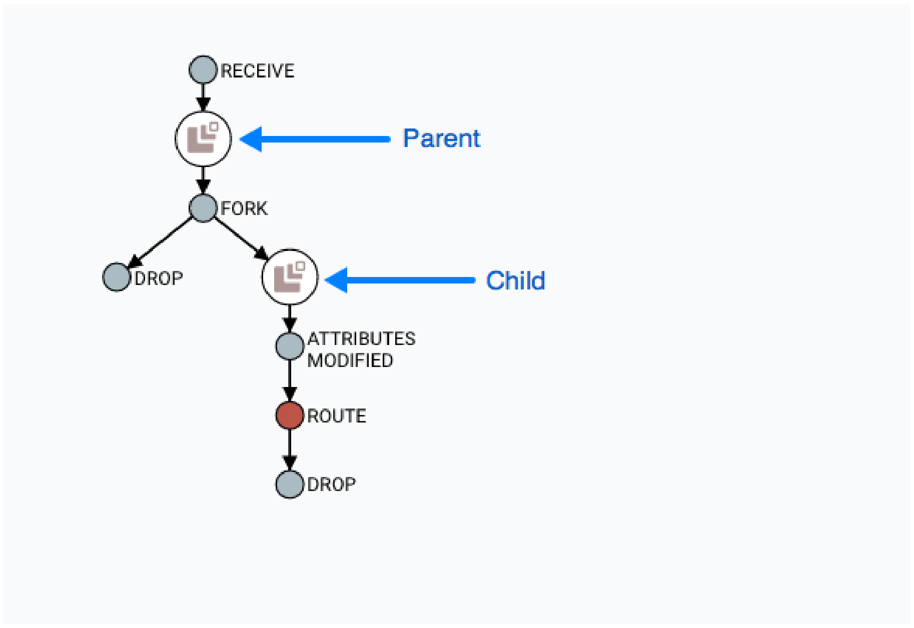

NiFi Data Provenance
Pop out
Download an image of the graph
Return to Event List
FlowFile
Event whose graph was selected
JOIN
ATTRIBUTES MODIFIED
SEND
DROP
11/16/2016 17:04:09.941 EST

Find Parents

Sometimes, a user may need to track down the original FlowFile that another FlowFile was spawned from. For example, when a FORK or CLONE event occurs, Logbot ETL keeps track of the parent FlowFile that produced other FlowFiles, and it is possible to find that parent FlowFile in the Lineage. Right-click on the event in the lineage graph and select "Find parents" from the context menu.

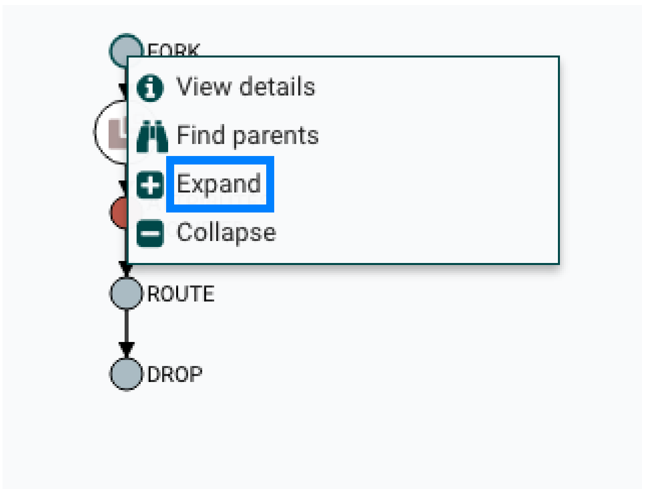


Once "Find parents" is selected, the graph is re-drawn to show the parent FlowFile and its lineage as well as the child and its lineage.

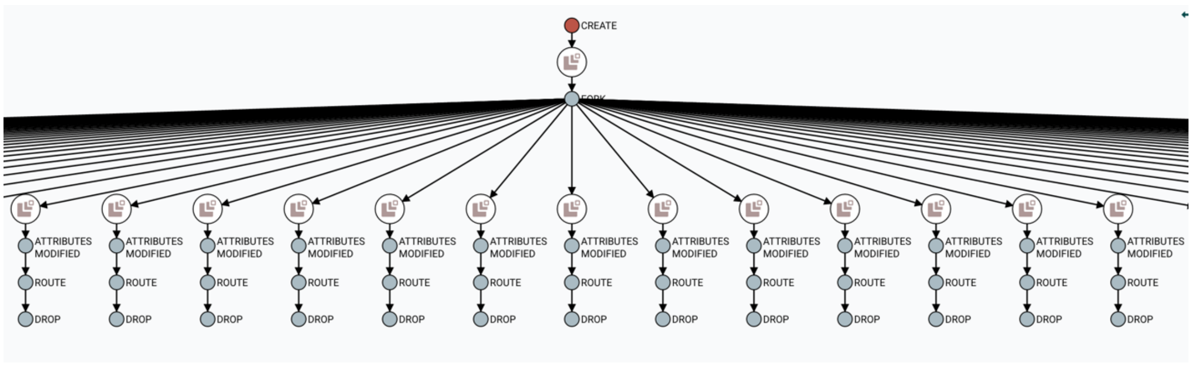


Expanding an Event

In the same way that it is useful to find a parent FlowFile, the user may also want to determine what children were spawned from a given FlowFile. To do this, right-click on the event in the lineage graph and select "Expand" from the context menu.



Once "Expand" is selected, the graph is re-drawn to show the children and their lineage.


CREATE
FORK
ATTRIBUTES MODIFIED
ROUTE
DROP

Experimental Warning

While all Apache licensed code is provided "on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied" (see Apache License, Version 2.0), some features of Logbot ETL may be marked **experimental**. Experimental features may:

- have undergone less extensive testing than is normal for standard Logbot ETL features
- interact with unstable external dependencies
- be subject to change (any exposed APIs should **not** be considered covered under the minor release backward compatibility guidelines of Semantic Versioning)
- potentially cause data loss
- not be directly supported by the community in the event issues arise

Every attempt is made to provide more detailed and specific information around the nature of the experimental warning on a per-feature basis. Questions around specific experimental features should be directed to the Logbot ETL Developer Mailing List.

Other Management Features

In addition to the Summary Page, Data Provenance Page, Template Management Page, and Bulletin Board Page, there are other tools in the Global Menu (see Logbot ETL User Interface) that are useful to the DFM. Select Flow Configuration History to view all the changes that have been made to the dataflow. The history can aid in troubleshooting, such as if a recent change to the dataflow has caused a problem and needs to be fixed. The DFM can see what changes have been made and adjust the flow as needed to fix the problem. While Logbot ETL does not have an "undo" feature, the DFM can make new changes to the dataflow that will fix the problem.

Select Node Status History to view instance specific metrics from the last 24 hours or if the instance runs for less time, then since it has been started. The status history can help the DFM in troubleshooting performance issues and provides a general view on the health of the node. The status history includes information about the memory usage and disk usage among other things.

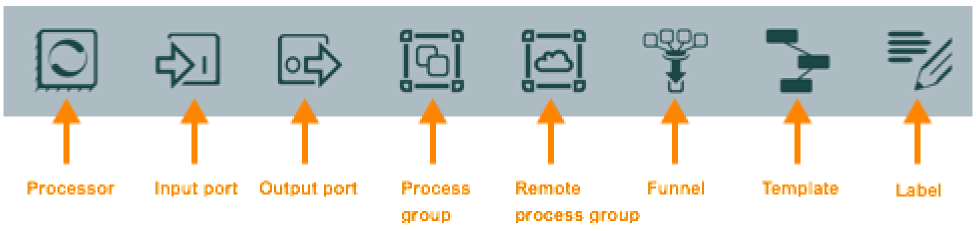
Two other tools in the Global Menu are Controller Settings and Users. The Controller Settings page provides the ability to change the name of the Logbot ETL instance, add comments describing the Logbot ETL instance, and set the maximum number of threads that are available to the application. It also provides tabs where DFMs may add and configure Controller Services and Reporting Tasks. The Users page is used to manage user access, which is described in the System Administrator’s Guide.

Build your first Dataflow

You can use Lobot ETL to create data flows for any protocols or data sources. Start your flow with a listener component, which gets the data from a source system. On the other end of the flow, add a transmitter component that writes the data to a target system. Additionally, you can add other components in between that process the data.

Logbot ETL components

Logbot ETL components are the building blocks of data flows. The Logbot ETL Component toolbar contains several data flow components that you can use when creating a new flow.



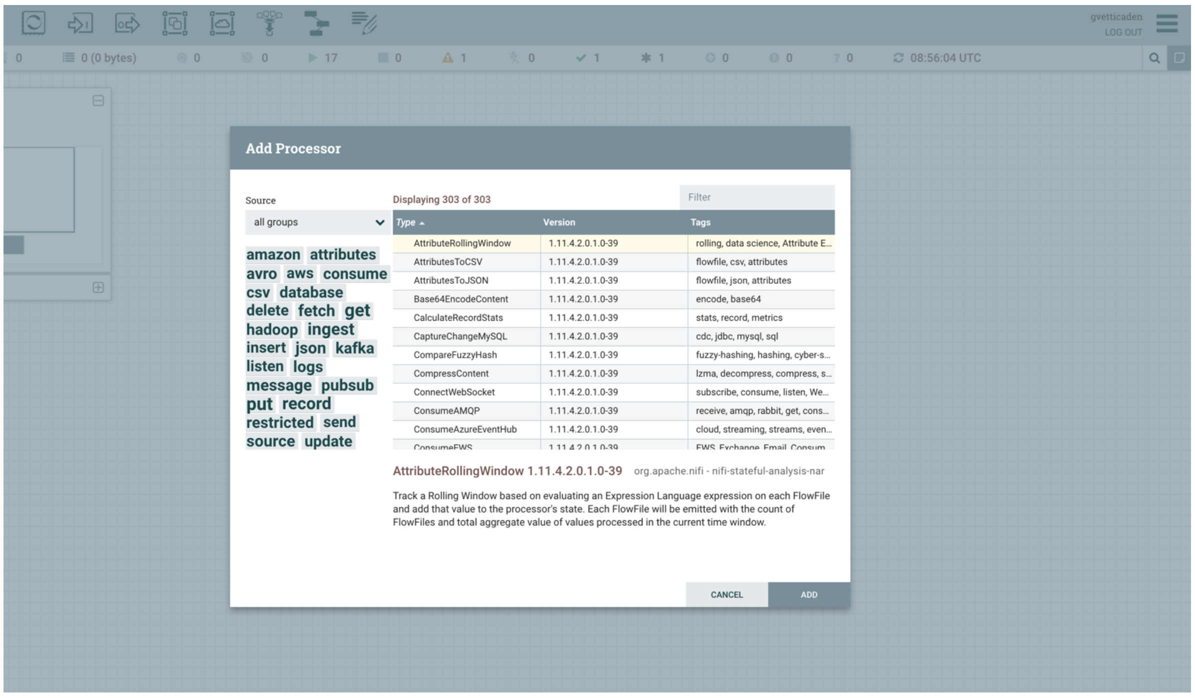
Create a data flow

This section walks you through the process of creating Logbot ETL data flows from scratch using basic flow options.

For information about building advanced data flows (using parameters, custom properties, controller services, or reporting tasks), see the *Logbot ETL User Guide*.

1. Add data flow components.

   You can add processors and other components to build your data flow as you need. To add a processor to your flow, drag the processor icons to the canvas from the component toolbar in the top-left of the screen. For more information, see *Adding Components to the Canvas* in the *Logbot ETL User Guide*.


gvetticaden
LOG OUT
0
0 (0 bytes)
0
0
17
0
1
0
1
1
0
0
0
08:56:04 UTC
Add Processor
Source
all groups
amazon attributes avro aws consume csv database delete fetch get hadoop ingest insert json kafka listen logs message pubsub put record restricted send source update
Filter
CANCEL
ADD

Displaying 303 of 303

Type	Version	Tags
AttributeRollingWindow	1.11.4.2.0.1.0-39	rolling, data science, Attribute E...
AttributesToCSV	1.11.4.2.0.1.0-39	flowfile, csv, attributes
AttributesToJSON	1.11.4.2.0.1.0-39	flowfile, json, attributes
Base64EncodeContent	1.11.4.2.0.1.0-39	encode, base64
CalculateRecordStats	1.11.4.2.0.1.0-39	stats, record, metrics
CaptureChangeMySQL	1.11.4.2.0.1.0-39	cdc, jdbc, mysql, sql
CompareFuzzyHash	1.11.4.2.0.1.0-39	fuzzy-hashing, hashing, cyber-s...
CompressContent	1.11.4.2.0.1.0-39	lzma, decompress, compress, s...
ConnectWebSocket	1.11.4.2.0.1.0-39	subscribe, consume, listen, We...
ConsumeAMQP	1.11.4.2.0.1.0-39	receive, amqp, rabbit, get, cons...
ConsumeAzureEventHub	1.11.4.2.0.1.0-39	cloud, streaming, streams, even...
ConsumeEWS	1.11.4.2.0.1.0-39	EWS, Exchange, Email, Consum

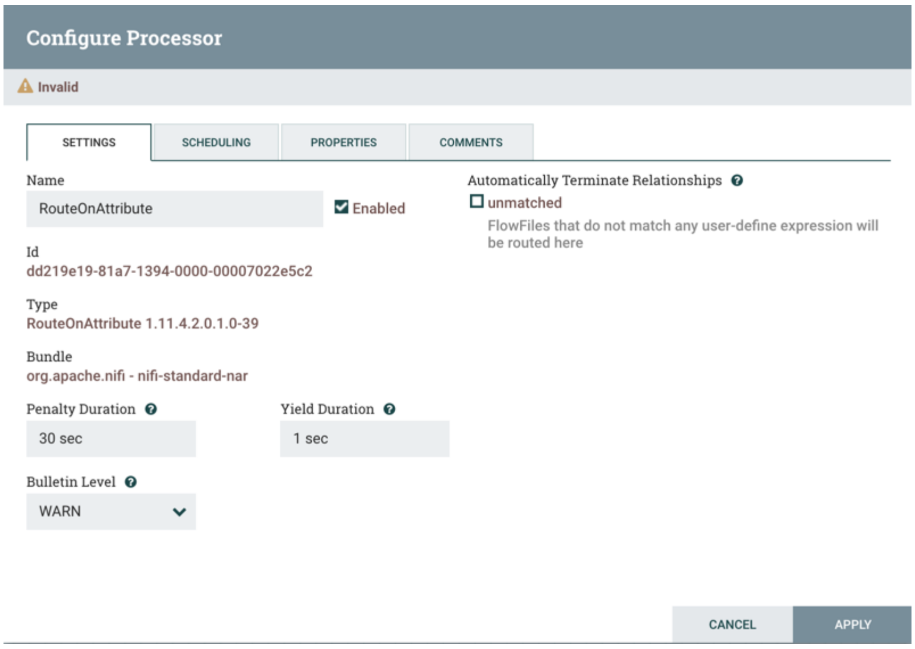
AttributeRollingWindow 1.11.4.2.0.1.0-39 org.apache.nifi - nifi-stateful-analysis-nar

Track a Rolling Window based on evaluating an Expression Language expression on each FlowFile and add that value to the processor's state. Each FlowFile will be emitted with the count of FlowFiles and total aggregate value of values processed in the current time window.

2. Configure the processors.
   a. Right-click the processor on the canvas and choose Configure in the context menu. Alternatively, double-click on the processor.

      A configuration dialog with the following tabs is displayed: Settings, Scheduling, Properties, and Comments.

   b. Configure the processor according to the behavior you desire in your data flow.
   c. When you have finished configuring the options you need, click Apply to save the changes or cancel all changes by clicking Cancel.


Configure Processor

Invalid

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Name: RouteOnAttribute

☑ Enabled

Id: dd219e19-81a7-1394-0000-00007022e5c2

Type: RouteOnAttribute 1.11.4.2.0.1.0-39

Bundle: org.apache.nifi - nifi-standard-nar

Penalty Duration: 30 sec

Yield Duration: 1 sec

Bulletin Level: WARN

Automatically Terminate Relationships

☐ unmatched

FlowFiles that do not match any user-define expression will be routed here

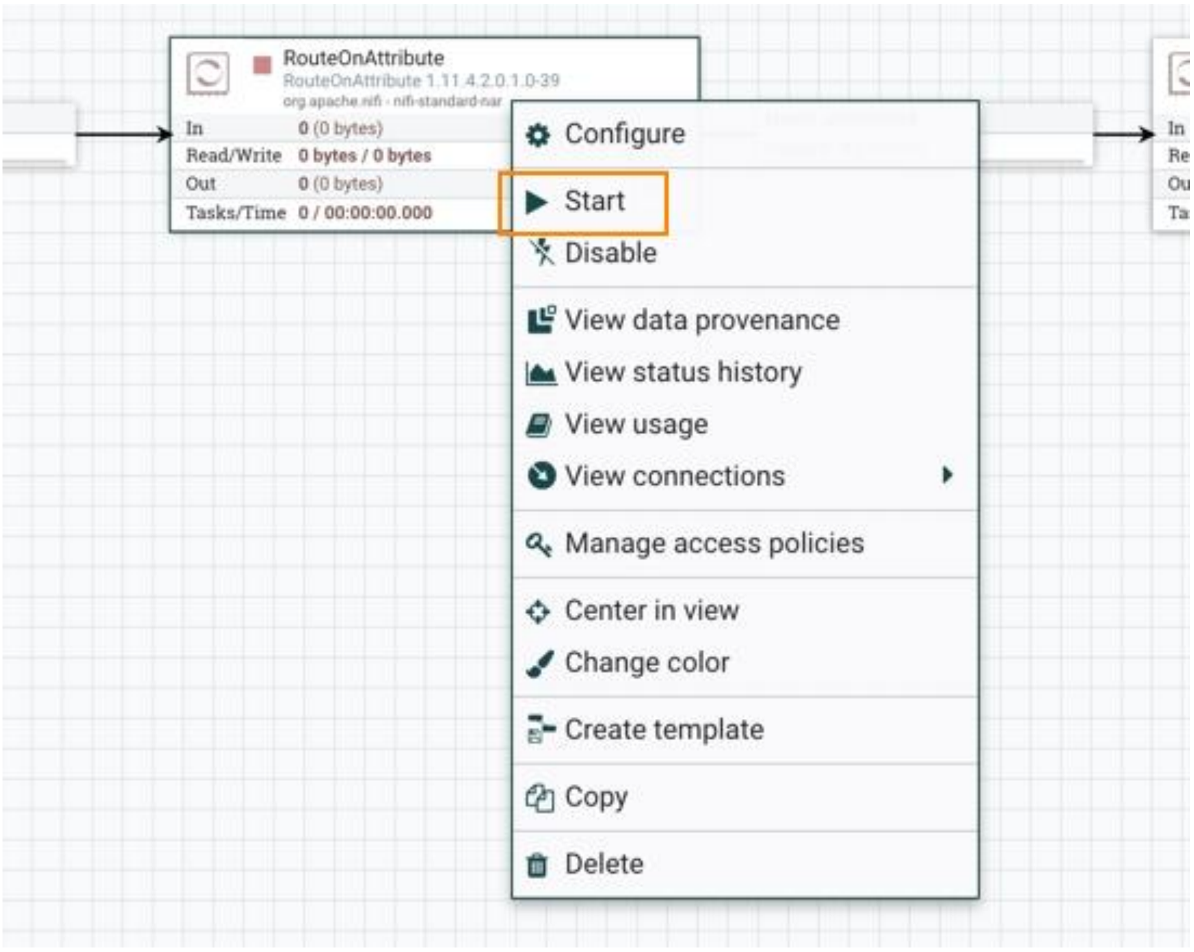
CANCEL | APPLY

3. Connect data flow components.

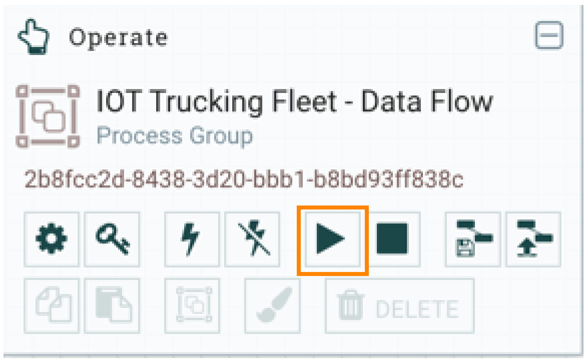
   You can connect the data flow components by hovering over a component and dragging the connection icon to another component. You can then choose which relationships you want to include for this connection.

4. Start your data flow.

   When all components are connected, the flow is still not working. To start the processors, click each one individually and then right-click and choose the Start menu item.



Alternatively, you can select all processors and then click the Start icon on the Operate palette.


Operate
IOT Trucking Fleet - Data Flow
Process Group
2b8fcc2d-8438-3d20-bbb1-b8bd93ff838c
DELETE

After you start the data flow, the icon in the top-left corner of the processors changes from a stopped icon to a running icon. You can stop the processors by using the Stop icon on the Operate palette or the Stop context menu item.

By following the above steps, you can choose any processor and other Logbot ETL components to create a flow suitable for your purpose.

Learn about Logbot ETL processors

Reviewing the list of available Logbot ETL processors can help you better understand the capabilities of the out-of-the-box processors.

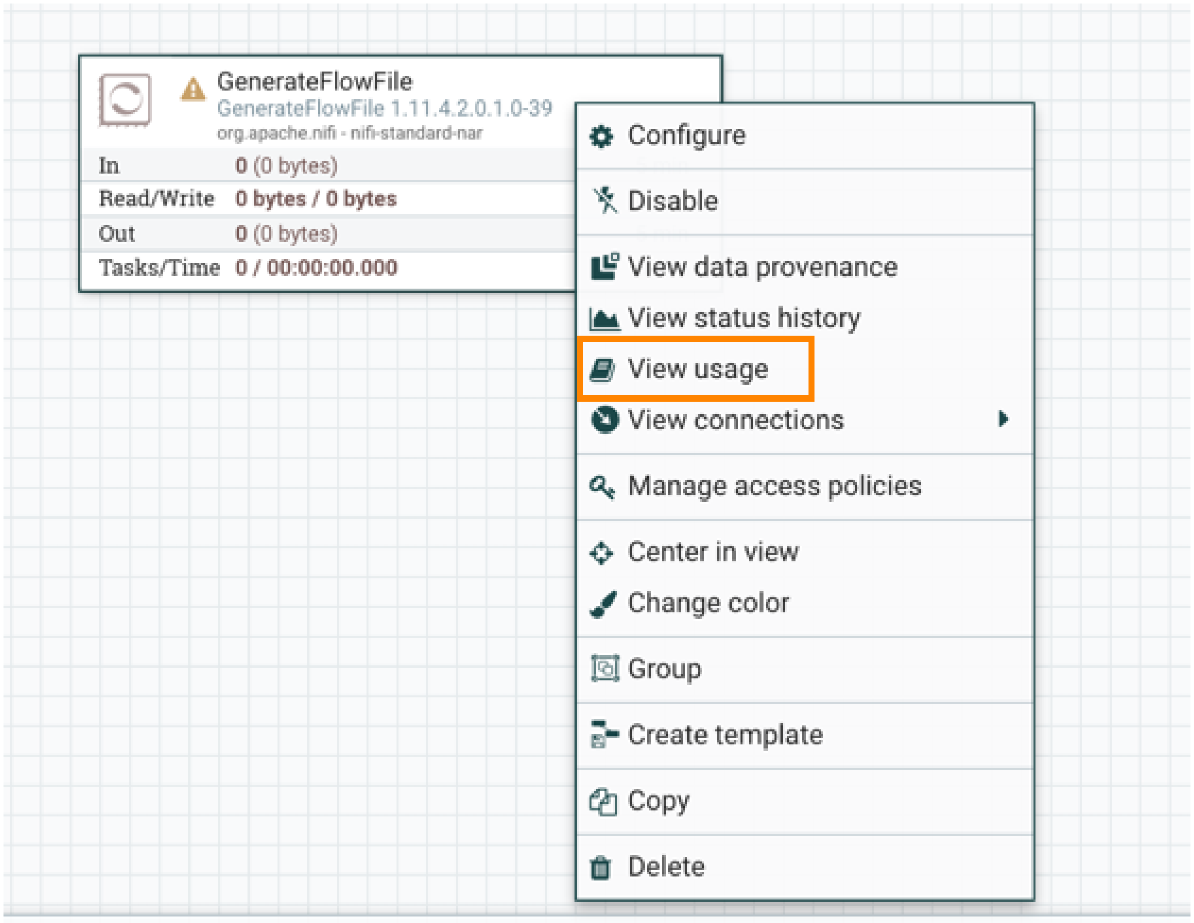
See what processors are available

To create an effective data flow, you must understand what types of processors are available and what they are suited for. Logbot ETL contains many processors that are available out of the box. These provide capabilities to ingest data from different systems, route, transform, process, split, and aggregate data and also distribute data to many systems. The number of processors increases in almost every Logbot ETL release.

For the complete list of Logbot ETL processors available in this release, see the list of *Supported Logbot ETL Processors*.

Get more information for a processor

Each processor has multiple properties and relationships, and it can be challenging to remember how all of the different pieces work for the processor. You can review a processor's usage information by right-clicking the processor and choosing the View usage menu item.



Work with attributes

Each FlowFile is created with several attributes, and these attributes change over the life of the FlowFile. The concept of a FlowFile provides three primary benefits:

- It enables you to make routing decisions in the flow so that FlowFiles that meet certain criteria can be handled differently than other FlowFiles. This is done using the RouteOnAttribute and similar processors.
- Attributes are used to configure processors in such a way that the configuration of the processor is dependent on the data itself. For instance, the PutFile processor uses the attributes to know where to store each FlowFile, while the directory and filename attributes may be different for each FlowFile.
- Attributes provide valuable context about the data. This is useful when reviewing the provenance data for a FlowFile. You can search for provenance data that match specific criteria, and it also enables you to view this context when inspecting the details of a Provenance Event. By doing this, you gain valuable insight as to why the data was processed one way or another, by glancing at this context that is carried along with the content.

Each FlowFile has a minimum set of attributes called Common Attributes:

`filename`
: A filename that can be used to store the data to a local or remote file system.

`path`
: The name of a directory that can be used to store the data to a local or remote file system.

`uuid`
: A Universally Unique Identifier that distinguishes the FlowFile from other FlowFiles in the system.

`entryDate`
: The date and time at which the FlowFile entered the system (that is, was created). The value of this attribute is a number that represents the number of milliseconds since midnight, Jan. 1, 1970 (UTC).

`lineageStartDate`
: Any time that a FlowFile is cloned, merged, or split, this results in a 'child' FlowFile being created. As those children are then cloned, merged, or split, a chain of ancestors is built. This value represents the date and time at which the oldest ancestor entered the system. Another way to think about this is that this attribute represents the latency of the FlowFile through the system. The value is a number that represents the number of milliseconds since midnight, Jan. 1, 1970 (UTC).

`fileSize`
: This attribute represents the number of bytes taken up by the FlowFile’s content.

Note
The `uuid`, `entryDate`, `lineageStartDate`, and `fileSize` attributes are system-generated and cannot be changed.

To extract attributes, add user-defined attributes, route FlowFiles based on their attributes, or perform other complex tasks with the help of attributes, see *Working With Attributes* in the *Logbot ETL Getting Started Guide* for details.

Work with data flow templates

Templates make it easier to reuse and distribute Logbot ETL flows or specific parts of data flows.

Besides reducing development time, reuse also supports the adoption of standards and patterns in data flow design. Data flow templates can help you to build a library of reusable elements that you can use as pre-built components in your new flows. Templates can be made from any subset of a flow: a single processor, a flattened subflow, or a process group holding a subflow. Alternatively, templates can be made from full flows as well. For more in-depth information about using templates, see the *Logbot ETL User Guide*.

Create a template

Learn how to create data flow templates that you can use as building blocks in another data flow.

Follow these steps to create a template from a reusable subflow:

1. Select the components/flow elements you want to include in the template.
2. Click the Create Template icon on the Operate palette.
3. Provide a name and optionally a description for the template.
4. Click Create.

Download a template

You can export templates from Logbot ETL and use them as a starting point for developing other data flows. You can retain or change all configurations of the downloaded template.

Templates can be exported from Logbot ETL and used as a starting point for developing a data flow. All configurations of the downloaded template are retained by default and can also be changed.
To download a template from Logbot ETL:

1. Go to the Templates option in the Global Menu at the top-right hand corner of the Logbot ETL UI.
2. Click the download icon of the template you want to download.

The template XML file is downloaded.

Upload a template

You can import a data flow template to Logbot ETL and use it as a starting point for developing a data flow. When using the template, you can retain or change all of its configurations.

Templates can be imported to Logbot ETL and used as a starting point for developing a data flow. All configurations of the downloaded template are retained and can also be changed.
To upload a template into Logbot ETL:

1. Click the Upload Template icon from the Operate palette.
2. Click the Search icon in the Upload template window and navigate to the XML file you want to upload.
3. Select the file.
4. Click Upload.

The template is uploaded and displayed in the templates table in the Logbot ETL UI. You can now use it to create data flows.

Use a template

Learn how you can build data flows with the help of templates in Cloudera Flow Management (CFM).

When you have created or imported templates, you can use these as building blocks in various data flows, just as you use a any data flow component.
To use a template in a data flow:

1. Click and drag the Template icon from the Component toolbar to the canvas.

   The Add template window containing a list of all available templates is displayed.

2. Choose the template that you would like to add to the canvas and click Add.

   There are a certain important points to remember when working with templates:

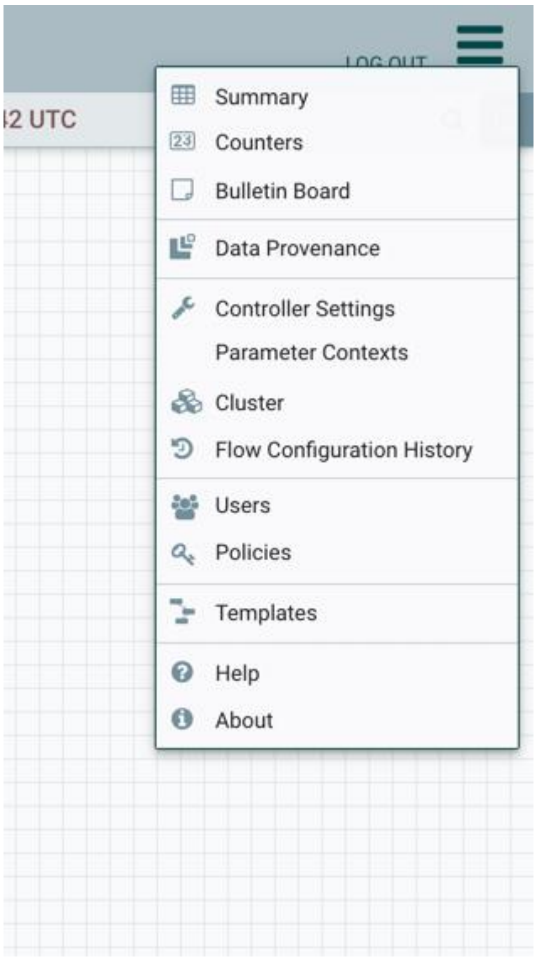
   - Sensitive properties (for example, a password configured in a processor) is not added to the template. These sensitive properties have to be populated each time that the template is used.
   - If a component that is included in the template references a controller service, the controller service is also added to the template. This means that each time that the template is used in a data flow, it creates a copy of the controller service.

Manage templates

You can access and work with your templates using the Logbot ETL Templates dialog.

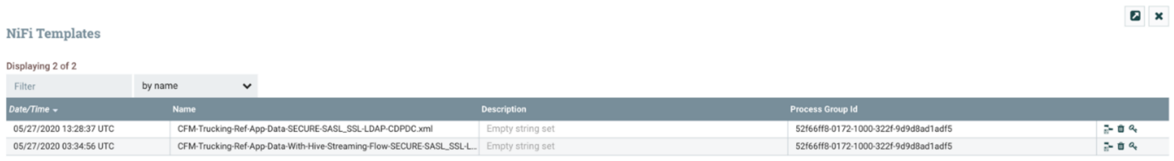
To open the dialog that displays all available templates with the ability to filter, export, and delete them:

1. Select Templates from the Global Menu.



The Logbot ETL Templates window is displayed.

2. Check the list of available templates. You can filter the list by name or by description to find the template of your interest.
3. On the right side of the table you can find the Download icon that you can use to download a template as an XML file.



NiFi Templates

Displaying 2 of 2

Filter | by name

Date/Time	Name	Description	Process Group Id	
05/27/2020 13:28:37 UTC	CFM-Trucking-Ref-App-Data-SECURE-SASL_SSL-LDAP-CDPDC.xml	Empty string set	52f66ff8-0172-1000-322f-9d9d8ad1adf5	
05/27/2020 03:34:56 UTC	CFM-Trucking-Ref-App-Data-With-Hive-Streaming-Flow-SECURE-SASL_SSL-L...	Empty string set	52f66ff8-0172-1000-322f-9d9d8ad1adf5	

Examples

In this chapter different types of processors will be described along with usage examples.

ExecuteScript Processor

This section describes various "recipes" on how to accomplish certain tasks with the Logbot ETL processor ExecuteScript, with examples given in Javascript. Recipes in this article series include:
Part 1 - Introduction to the Logbot ETL API and FlowFiles

- Getting a flow file from an incoming queue
- Creating new flow files
- Working with flow file attributes
- Transferring a flow file
- Logging

Part 2 - *FlowFile I/O and Error Handling*

- Reading from a flow file
- Writing to a flow file
- Reading and writing to/from a flow file
- Error Handling

Part 3 - *Advanced Features*

- Using Dynamic Properties
- Adding Modules
- State Management
- Accessing Controller Services

Introduction

ExecuteScript is a versatile processor that allows the user to code custom logic in a programming language that will be executed each time the ExecuteScript processor is triggered. The following variable bindings are provided to the script to enable access to Logbot ETL components:
session: This is a reference to the ProcessSession assigned to the processor. The session allows you to perform operations on flow files such as create(), putAttribute(), and transfer(), as well as read() and write().
context: This is a reference to the ProcessContext for the processor. It can be used to retrieve processor properties, relationships, Controller Services, and the StateManager.
log: This is a reference to the ComponentLog for the processor. Use it to log messages to Logbot ETL, such as log.info('Hello world!')
REL_SUCCESS: This is a reference to the "success" relationship defined for the processor. It could also be inherited by referencing the static member of the parent class (ExecuteScript), but some engines such as Lua do not allow for referencing static members, so this is a convenience variable. It also saves having to use the fully-qualified name for the relationship.
REL_FAILURE: This is a reference to the "failure" relationship defined for the processor. As with REL_SUCCESS, it could also be inherited by referencing the static member of the parent class (ExecuteScript), but some engines such as Lua do not allow for referencing static

members, so this is a convenience variable. It also saves having to use the fully-qualified name for the relationship.
Dynamic Properties: Any dynamic properties defined in ExecuteScript are passed to the script engine as variables set to the PropertyValue object corresponding to the dynamic property. This allows you to get the String value of the property, but also to evaluate the property with respect to Logbot ETL Expression Language, cast the value as an appropriate data type (such as Boolean, e.g.), etc. Because the dynamic property name becomes the variable name for the script, you must be aware of the variable naming properties for the chosen script engine. For example, Groovy does not allow periods (.) in variable names, so an error will occur if "my.property" was a dynamic property name.
Interaction with these variables is done via the Logbot ETL Java API, each recipe below will discuss the relevant API calls as they are introduced. The recipes in the following section perform various functions on flow files, such as reading/writing attributes, transferring to a relationship, logging, etc. Please note that the examples are snippets and do not run as-is. For example, if a flow file has been retrieved from the queue with session.get(), it must be transferred to a relationship or removed, or else an error will occur. The snippets are meant to be plain and clear to illustrate only the concept(s) presented, without the addition of boilerplate code to make them working examples. In a later article I will put them all together to show full working scripts that perform useful tasks.

Recipes

Recipe: Get an incoming flow file from the session
Use Case: You have incoming connection(s) to ExecuteScript and want to retrieve one flow file from the queue(s) for processing.
Approach: Use the get() method from the session object. This method returns the FlowFile that is next highest priority FlowFile to process. If there is no FlowFile to process, the method will return null. Note that it is possible to have null returned even if there is a steady flow of FlowFiles into the processor. This can happen if there are multiple concurrent tasks for the processor, and the other task(s) have already retrieved the FlowFiles. If the script requires a FlowFile to continue processing, then it should immediately return if null is returned from session.get()
Examples:
Javascript

```
var flowFile = session.get();

if (flowFile != null) {

   // All processing code goes here

}
```

Recipe: Get multiple incoming flow files from the session
Use Case: You have incoming connection(s) to ExecuteScript and want to retrieve multiple flow files from the queue(s) for processing.
Approach: Use the get(*maxResults*) method from the session object. This method returns up to *maxResults* FlowFiles from the work queue. If no FlowFiles are available, an empty list is

returned (the method does not return null). NOTE: If multiple incoming queues are present, the behavior is unspecified in terms of whether all queues or only a single queue will be polled in a single call.
Examples:
Javascript

```
flowFileList = session.get(100)

if(!flowFileList.isEmpty()) {

  for each (var flowFile in flowFileList) {

       // Process each FlowFile here

  }

}
```

Recipe: Create a new FlowFile
Use Case: You want to generate a new FlowFile to send to the next processor
Approach: Use the create() method from the session object. This method returns a new FlowFile object, which you can perform further processing on
Examples:
Javascript

```
var flowFile = session.create();

// Additional processing here
```

Recipe: Create a new FlowFile from a parent FlowFile
Use Case: You want to generate new FlowFile(s) based on an incoming FlowFile
Approach: Use the create(*parentFlowFile*) method from the session object. This method takes a parent FlowFile reference and returns a new child FlowFile object. The newly created FlowFile will inherit all of the parent's attributes except for the UUID. This method will automatically generate a Provenance FORK event or a Provenance JOIN event, depending on whether or not other FlowFiles are generated from the same parent before the ProcessSession is committed.
Examples:
Javascript

```
var flowFile = session.get();

if (flowFile != null) {
  var newFlowFile = session.create(flowFile);

  // Additional processing here

}
```

Recipe: Add an attribute to a flow file
Use Case: You have a flow file to which you'd like to add a custom attribute.
Approach: Use the putAttribute(*flowFile*, *attributeKey*, *attributeValue*) method from the session object. This method updates the given FlowFile's attributes with the given key/value pair. NOTE: The "uuid" attribute is fixed for a FlowFile and cannot be modified; if the key is named "uuid", it will be ignored.
Also this is a good point to mention that FlowFile objects are immutable; this means that if you update a FlowFile's attributes (or otherwise alter it) via the API, you will get a new reference to the new version of the FlowFile. This is very important when it comes to transferring FlowFiles to relationships. You must keep a reference to the latest version of a FlowFile, and you must transfer or remove the latest version of all FlowFiles retrieved from or created by the session, otherwise you will get an error when executing. Most often, the variable used to store a FlowFile reference will be overwritten with the latest version returned from a method that alters the FlowFile (intermediate FlowFile references will be automatically discarded). In these examples you will see this technique of reusing a flowFile reference when adding attributes. Note that the current reference to the FlowFile is passed into the putAttribute() method. The resulting FlowFile has an attribute named 'myAttr' with a value of 'myValue'. Also note that the method takes a String for the value; if you have an Object you will have to serialize it to a String. Finally, please note that if you are adding multiple attributes, it is better to create a Map and use putAllAttributes() instead (see next recipe for details).
Examples:
Javascript

```
var flowFile = session.get();

if (flowFile != null) {

   flowFile = session.putAttribute(flowFile, 'myAttr', 'myValue')

}
```

Recipe: Add multiple attributes to a flow file
Use Case: You have a flow file to which you'd like to add custom attributes.
Approach: Use the putAllAttributes(*flowFile*, *attributeMap*) method from the session object. This method updates the given FlowFile's attributes with the key/value pairs from the given Map. NOTE: The "uuid" attribute is fixed for a FlowFile and cannot be modified; if the key is named "uuid", it will be ignored.
The technique here is to create a Map (aka dictionary in Jython, hash in JRuby) of the attribute key/value pairs you'd like to update, then call putAllAttributes() on it. This is much

more efficient than calling putAttribute() for each key/value pair, as the latter case will cause the framework to create a temporary version of the FlowFile for each attribute added (see above recipe for discussion on FlowFile immutability). The examples show a map of two entries myAttr1 and myAttr2, set to '1' and the language-specific coercion of the number 2 as a String (to adhere to the method signature of requiring String values for both key and value). Note that a session.transfer() is not specified here (so the code snippets below do not work as-is), see the following recipe for that.
Examples:
Javascript

```
var number2 = 2;

var attrMap = {'myAttr1':'1', 'myAttr2': number2.toString()}

var flowFile = session.get()

if (flowFile != null) {

    flowFile = session.putAllAttributes(flowFile, attrMap)

}
```

Recipe: Get an attribute from a flow file
Use Case: You have a flow file from which you'd like to inspect an attribute.
Approach: Use the getAttribute(*attributeKey*) method from the FlowFile object. This method returns the String value for the given attributeKey, or null if the attributeKey is not found. The examples show the retrieval of the value for the "filename" attribute.
Examples:
Javascript

```
var flowFile = session.get()

if (flowFile != null) {

    var myAttr = flowFile.getAttribute('filename')

}
```

Recipe: Get all attributes from a flow file
Use Case: You have a flow file from which you'd like to retrieve its attributes.
Approach: Use the getAttributes() method from the FlowFile object. This method returns a Map with String keys and String values, representing the key/value pairs of attributes for the flow file. The examples show an iteration over the Map of all attributes for a FlowFile.
Examples:
Javascript

```
var flowFile = session.get()

if (flowFile != null) {

    var attrs = flowFile.getAttributes();

    for each (var attrKey in attrs.keySet()) {

       // Do something with attrKey (the key) and/or attrs[attrKey]
(the value)

  }

}
```

Recipe: Transfer a flow file to a relationship
Use Case: After processing a flow file (new or incoming), you want to transfer the flow file to a relationship ("success" or "failure"). In this simple case let us assume there is a variable called "errorOccurred" that indicates which relationship to which the FlowFile should be transferred. Additional error handling techniques will be discussed in part 2 of this series.
Approach: Use the transfer(*flowFile*, *relationship*) method from the session object. From the documentation: this method transfers the given FlowFile to the appropriate destination processor work queue(s) based on the given relationship. If the relationship leads to more than one destination the state of the FlowFile is replicated such that each destination receives an exact copy of the FlowFile though each will have its own unique identity.
NOTE: ExecuteScript will perform a session.commit() at the end of each execution to ensure the operations have been committed. You do not need to (and should not) perform a session.commit() within the script.
Examples:
Javascript

```
var flowFile = session.get();

if (flowFile != null) {

   // All processing code goes here

   if(errorOccurred) {

     session.transfer(flowFile, REL_FAILURE)

   }

   else {
     session.transfer(flowFile, REL_SUCCESS)

   }

}
```

Recipe: Send a message to the log at a specified logging level
Use Case: You want to report some event that has occurred during processing to the logging framework.
Approach: Use the log variable with the warn(), trace(), debug(), info(), or error() methods. These methods can take a single String, or a String followed by an array of Objects, or a String followed by an array of Objects followed by a Throwable. The first one is used for simple messages. The second is used when you have some dynamic objects/values that you want to log. To refer to these in the message string use "{}" in the message. These are evaluated against the Object array in order of appearance, so if the message reads "Found these things: {} {} {}" and the Object array is ['Hello',1,true], then the logged message will be "Found these things: Hello 1 true". The third form of these logging methods also takes a Throwable parameter, and is useful when an exception is caught and you want to log it.
Examples:
Javascript

```
var ObjectArrayType = Java.type("java.lang.Object[]");

var objArray = new ObjectArrayType(3);

objArray[0] = 'Hello';

objArray[1] = 1;

objArray[2] = true;

log.info('Found these things: {} {} {}', objArray)
```

Introduction to FlowFile I/O

Flow files in Logbot ETL are made of two major components, attributes and content. Attributes are metadata about the content / flow file, and we saw how to manipulate them using ExecuteScript in Part 1 of this series. The content of a flow file is, at its heart, simply a collection of bytes and has no inherent structure, schema, format, etc. Various Logbot ETL processors assume the incoming flow files have a particular schema/format (or determine it from attributes such as "mime.type" or infer it in other ways). These processors can then act upon the content based on the assumption that the files really do have that format (and will often transfer to a "failure" relationship if they do not). Also processors may output flow files in a specified format.

Input and Output (I/O) for the contents of flow files is provided via the ProcessSession API and thus the "session" variable for ExecuteScript (see Part 1 for more information). One mechanism for this is to pass a callback object into a call to session.read() or session.write(). An InputStream and/or OutputStream will be created for the FlowFile object, and the callback object will be invoked using the corresponding callback interface, with the InputStream and/or OutputStream references passed in for use by the callback. There are three main callback interfaces, each with its own use case:

InputStreamCallback

This interface is used by the session.read(*flowFile*, *inputStreamCallback*) method to provide an InputStream from which to read the contents of the flow file. The interface has a single method:

```
void process(InputStream in) throws IOException
```

This interface provides a managed input stream for use. The input stream is automatically opened and closed though it is ok to close the stream manually. This is the form you would use if you are only reading from a particular flow file, and not writing back out to it.

An example is when you want to process an incoming flow file, but create many output flow files, such as the SplitText processor does.

OutputStreamCallback

This interface is used by the session.write(*flowFile*, *outputStreamCallback*) method to provide an OutputStream to which to write the contents of the flow file. The interface has a single method:

```
void process(OutputStream out) throws IOException
```

This interface provides a managed output stream for use. The output stream is automatically opened and closed though it is ok to close the stream manually - and quite important if any streams wrapping these streams open resources which should be cleared.

An example is when ExecuteScript will be generating data, either from within or from an external file, but not a flow file. Then you would use session.create() to create a new FlowFile, then session.write(*flowFile*, *outputStreamCallback*) to insert content.

StreamCallback

This interface is used by the session.write(*flowFile*, *streamCallback*) method to provide an InputStream and OutputStream, from which to read from and/or write to the contents of the flow file. The interface has a single method:

```
void process(InputStream in, OutputStream out) throws
IOException
```

This interface provides managed input and output streams for use. The input stream is automatically opened and closed though it is ok to close the streams manually - and quite important if any streams wrapping these streams open resources which should be cleared.
An example is when you want to process an incoming flow file and overwrite its contents with something new, such as the EncryptContent processor does.
Since these callbacks are Java objects, the script will have to create one and pass it into the session method(s), the recipes will illustrate this for the various scripting languages. Also there are other methods of reading from and writing to flow files, which include:

- Using session.read(*flowFile*) to return an InputStream. This alleviates the need for an InputStreamCallback, instead it returns an InputStream that you can read from. In exchange you must manage (close, e.g.) the InputStream manually.

- Using session.importFrom(*inputStream*, *flowFile*) to write from an InputStream to a FlowFile. This replaces the need for a session.write() with an OutputStreamCallback passed in.

- Now, on to the recipes :)

Recipes

Recipe: Read the contents of an incoming flow file using a callback
Use Case: You have incoming connection(s) to ExecuteScript and want to retrieve the contents of a flow file from the queue(s) for processing.
Approach: Use the read(*flowFile*, *inputStreamCallback*) method from the session object. An InputStreamCallback object is needed to pass into the read() method. Note that because InputStreamCallback is an object, the contents are only visible to that object by default. If you need to use the data outside the read() method, use a more globally-scoped variable. The examples will store the full contents of the incoming flow file into a String (using Apache Commons' IOUtils class). NOTE: For large flow files, this is not the best technique; rather you should read in only as much data as you need, and process that as appropriate. For something like SplitText, you could read in a line at a time and process it within the InputStreamCallback, or use the session.read(flowFile) approach mentioned earlier to get an InputStream reference to use outside of a callback.
Examples:
Javascript

```
var InputStreamCallback =
Java.type("org.apache.nifi.processor.io.InputStreamCallback")

var IOUtils = Java.type("org.apache.commons.io.IOUtils")

var StandardCharsets =
Java.type("java.nio.charset.StandardCharsets")



var flowFile = session.get();

if(flowFile != null) {

  // Create a new InputStreamCallback, passing in a function
to define the interface method

  session.read(flowFile,

    new InputStreamCallback(function(inputStream) {

        var text = IOUtils.toString(inputStream,
StandardCharsets.UTF_8);

        // Do something with text here

    }));

}
```

Recipe: Write content to an outgoing flow file using a callback
Use Case: You want to generate content for an outgoing flow file.
Approach: Use the write(*flowFile*, *outputStreamCallback*) method from the session object. An OutputStreamCallback object is needed to pass into the write() method. Note that because OutputStreamCallback is an object, the contents are only visible to that object by default. If you need to use the data outside the write() method, use a more globally-scoped variable. The examples will write a sample String to a flowFile.
Examples:
Javascript

```
var OutputStreamCallback =
Java.type("org.apache.nifi.processor.io.OutputStreamCallback")
;
```

```
var IOUtils = Java.type("org.apache.commons.io.IOUtils");

var StandardCharsets =
Java.type("java.nio.charset.StandardCharsets");



var flowFile = session.get();

if(flowFile != null) {

  // Create a new OutputStreamCallback, passing in a function
to define the interface method

  flowFile = session.write(flowFile,

    new OutputStreamCallback(function(outputStream) {

        outputStream.write("Hello
World!".getBytes(StandardCharsets.UTF_8))

    }));

}
```

Recipe: Overwrite an incoming flow file with updated content using a callback
Use Case: You want to reuse the incoming flow file but want to modify its content for the outgoing flow file.
Approach: Use the write(*flowFile, streamCallback*) method from the session object. An StreamCallback object is needed to pass into the write() method. StreamCallback provides both an InputStream (from the incoming flow file) and an outputStream (for the next version of that flow file), so you can use the InputStream to get the current contents of the flow file, then modify them and write them back out to the flow file. This overwrites the contents of the flow file, so for append you'd have to handle that by appending to the read-in contents, or use a different approach (with session.append() rather than session.write()). Note that because StreamCallback is an object, the contents are only visible to that object by default. If you need to use the data outside the write() method, use a more globally-scoped variable. The examples will reverse the contents of the incoming flowFile (assumed to be a String) and write out the reversed string to a new version of the flowFile.

Examples:
Javascript

```
var StreamCallback =
Java.type("org.apache.nifi.processor.io.StreamCallback");

var IOUtils = Java.type("org.apache.commons.io.IOUtils");

var StandardCharsets =
Java.type("java.nio.charset.StandardCharsets");



var flowFile = session.get();

if(flowFile != null) {

  // Create a new StreamCallback, passing in a function to
define the interface method

  flowFile = session.write(flowFile,

    new StreamCallback(function(inputStream, outputStream) {

        var text = IOUtils.toString(inputStream,
StandardCharsets.UTF_8)


outputStream.write(text.split("").reverse().join("").getBytes(
StandardCharsets.UTF_8))

    }));

}
```

Recipe: Handle errors during script processing
Use Case: An error occurs in the script (either by data validation or a thrown exception), and you want the script to handle it gracefully.
Approach: For exceptions, use the exception-handling mechanism for the scripting language (often they are try/catch block(s)). For data validation, you can use a similar approach, but define a boolean variable like "valid" and an if/else clause rather than a try/catch clause. ExecuteScript defines "success" and "failure" relationships; often your processing will transfer "good" flow files to success and "bad" flow files to failure (logging an error in the latter case). It is possible

Examples:
Javascript

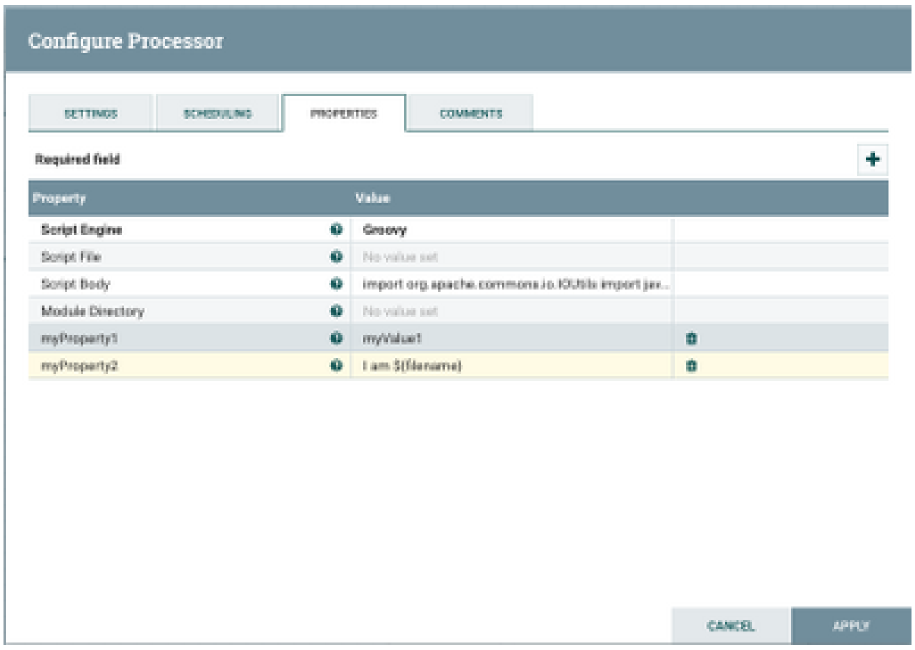
```
var flowFile = session.get();

if(flowFile != null) {

  try {

    // Something that might throw an exception here



    // Last operation is transfer to success (failures handled
in the catch block)

    session.transfer(flowFile, REL_SUCCESS)

} catch(e) {

  log.error('Something went wrong', e)

  session.transfer(flowFile, REL_FAILURE)

}

}
```

Advanced Features

Dynamic Properties

One such capability is the concept of dynamic properties, also called User-Defined properties. These are properties for a processor for which a user can set both the property name and value. Not all processors support/use dynamic properties, but ExecuteScript will pass dynamic properties as variables which reference a PropertyValue object corresponding to the property's value. There are two important things to note here:

1. Because the property name is bound as-is to a variable name, the naming convention for dynamic properties must be supported for the specified programming language. For example, Groovy does not support period (.) as a valid variable character, so a dynamic property such as "my.value" will cause the processor to fail. A valid alternative in this case is "myValue".
2. The PropertyValue object is used (rather than a String representation of the value) to allow the script to perform various operations on the property's value before evaluating it to a String. If the property is known to contain a literal value, you can call the getValue() method on the variable to get its String representation. If instead the value could contain Expression Language or you want to cast the value to something other than String (such as the value 'true' to a Boolean object), there are methods for these operations too. These examples are illustrated in the recipes below, assuming we have two properties 'myProperty1' and 'myProperty2' defined as such:



Property	Value
Script Engine	Groovy
Script File	No value set
Script Body	import org.apache.commons.io.IOUtils import jav...
Module Directory	No value set
myProperty1	myValue1
myProperty2	I am ${filename}

Recipe: Get the value of a dynamic property
Use Case: A user has entered a dynamic property for use in the script (configuration parameter, e.g.).
Approach: Use the getValue() method from the variable's PropertyValue object. This method returns a String representation of the value of the dynamic property. Note that if Expression Language is present in the value, getValue() will not evaluate it (see the following recipe for that functionality).
Examples:

Javascript

```
var myValue1 = myProperty1.getValue()
```

Recipe: Get the value of a dynamic property after evaluating Expression Language constructs
Use Case: A user has entered a dynamic property for use in the script (configuration parameter, e.g.), and it refers to attribute(s) in an incoming flow file.
Approach: Use the evaluateAttributeExpressions(*flowFile*) method from the variable's PropertyValue object. This method, followed by getValue(), returns a String representation of the value of the dynamic property after any Expression Language constructs have been evaluated. If a flow file is not available, but variables have been defined in the environment or Variable Registry, you can use evaluateAttributeExpressions() without a parameter
Examples:
Javascript

```
var myValue1 = myProperty1.getValue()var myValue2 =
myProperty2.evaluateAttributeExpressions(flowFile).getValue()
```

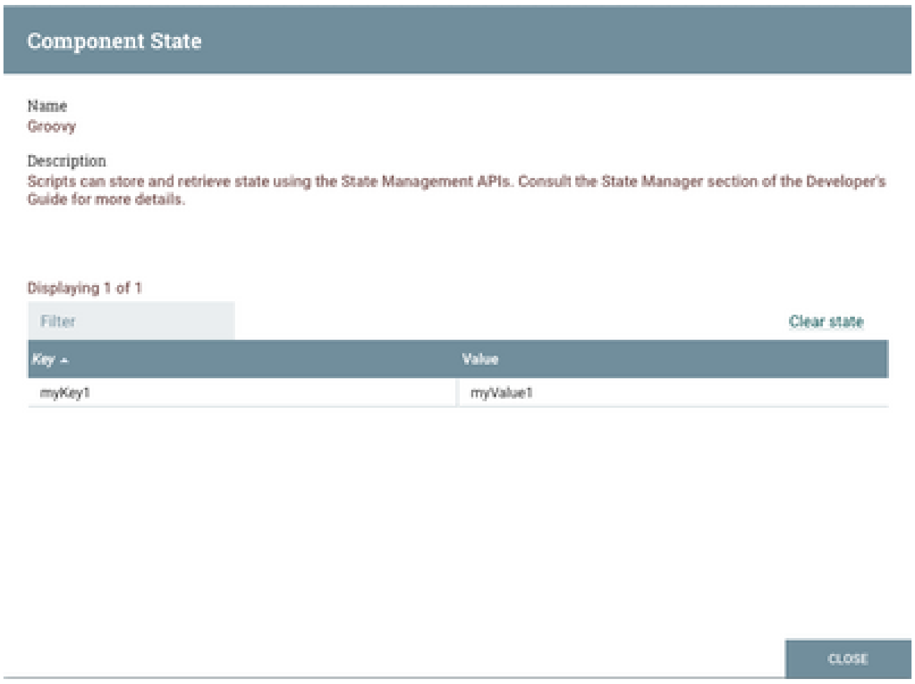
State Management

Logbot ETL offers the capability for Processors and other Logbot ETL components to persist some information to enable some stateful functionality around the component. For example, the QueryDatabaseTable processor keeps track of the largest values it has seen for the specified columns, such that the next time it runs, it will only fetch rows whose values are larger than those that have been seen so far (i.e. stored in the State Manager).
An important concept in terms of state management is Scope. Logbot ETL components can choose to store their state at the cluster level or the local level. Note that in a standalone Logbot ETL instance, "cluster scope" is identical to "local scope". The choice of scope is usually about whether in a flow, the same processor on each node can share the state data. If the instances across the cluster do not need to share state, then use local scope. In Java these options are provided as an enum called Scope, so when I refer to Scope.CLUSTER and Scope.LOCAL, I mean cluster and local scope respectively.
To employ state management features in ExecuteScript (language-specific examples are below), you get a reference to the StateManager by calling ProcessContext's getStateManager() method (recall that each engine gets a variable named "context" with an instance of ProcessContext). You can then call the following methods on the StateManager object:
void setState(Map<String, String> *state*, Scope *scope*) - Updates the value of the component's state at the given scope, setting it to given value. Notice that the value is a Map; the concept of "component state" is the Map of all key/value pairs that each constitute a lower-level of state. The Map is updated all at once to provide atomicity.
StateMap getState(Scope *scope*) - Returns the current state for the component at the given scope. This method never returns null; rather it is a StateMap object and if the state has not yet been set, the StateMap's version will be -1, and the map of values will be empty. Often a new Map<String,String> will be created to store the updated values, then setState() or replace() will be called.

boolean replace(StateMap *oldValue*, Map<String, String> *newValue*, Scope *scope*) - Updates the value of the component's state (at the given scope) to the new value if and only if the value currently is the same as the given oldValue. If the state was updated to the new value, true is returned; otherwise false is returned if the state's value was not equal to oldValue.
void clear(Scope *scope*) - Clears all keys and values from the component's state at the given scope.
Recipe: Get the current map of key/value pairs
Use Case: The script needs to get the current key/value pairs from the state manager for use in the script (update, e.g.).
Approach: Use the getStateManager() method from ProcessContext, then getStateMap() from the StateManager, then toMap() to convert to a Map<String,String> of key/value pairs. Note that StateMap also has a get(*key*) method for simply retrieving a value, but this is not often used as the Map is usually updated and must be set for the StateManager once this is done.
Examples:
Javascript

```
var Scope = Java.type('org.apache.nifi.components.state.Scope');

var oldMap = context.stateManager.getState(Scope.LOCAL).toMap();
```

NOTE: Only the Scope class was referenced explicitly in the script, so it is the only one imported. If you refer to StateManager, StateMap, etc. you will need to import those classes as well.
Recipe: Update the map of key/value pairs
Use Case: The script wants to update the state map with a new Map of key/value pairs.
Approach: To get the current StateMap object, again use the getStateManager() method from ProcessContext, then getStateMap() from the StateManager. The examples assume a new Map, but using the above recipe (with the toMap() method), you could create a new Map using the existing values, then just update the desired entries. Note that replace() will not work if there is no current map (i.e. the StateMap.getVersion() returns -1), so the examples check and call setState() or replace() accordingly. When running from a new instance of ExecuteScript, the StateMap version will be -1, so after a single execution, if you right-click on the ExecuteScript processor and choose View State, you should see something like this:


Component State

Name
Groovy

Description
Scripts can store and retrieve state using the State Management APIs. Consult the State Manager section of the Developer's Guide for more details.

Displaying 1 of 1

Key	Value
myKey1	myValue1

Examples:
Javascript

```
var Scope = Java.type('org.apache.nifi.components.state.Scope');

var stateManager = context.stateManager;

var stateMap = stateManager.getState(Scope.CLUSTER);

var newMap = {'myKey1': 'myValue1'};

if (stateMap.version == -1) {

  stateManager.setState(newMap, Scope.CLUSTER);

} else {

  stateManager.replace(stateMap, newMap, Scope.CLUSTER);

}
```

Recipe: Clear the state map
Use Case: The script wants to clear all the key/value pairs for the state map.
Approach: Use the getStateManager() method from ProcessContext, then call StateManager's clear(scope) method.
Examples:
Javascript

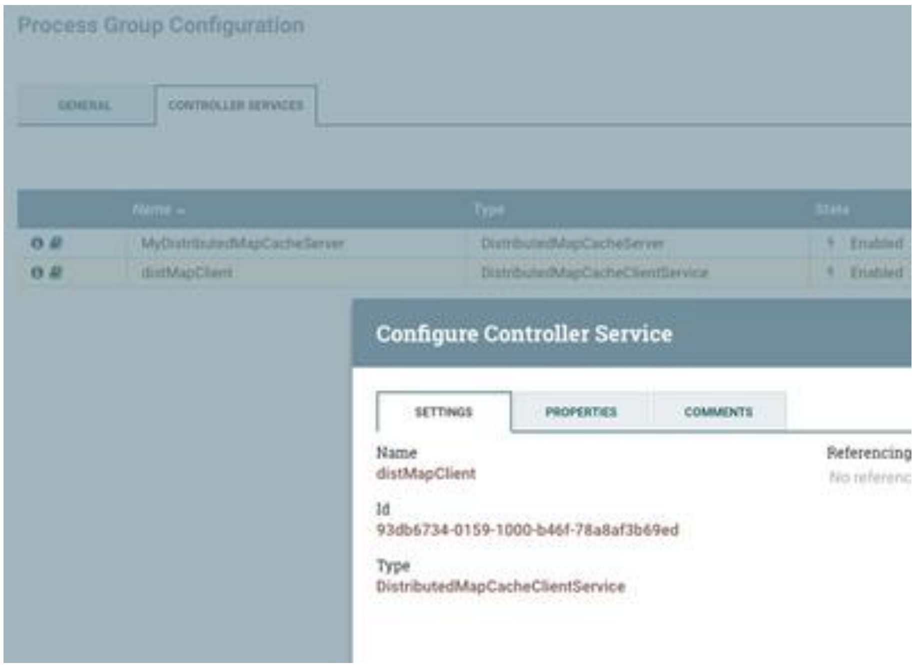
```
var Scope = Java.type('org.apache.nifi.components.state.Scope');

context.stateManager.clear(Scope.LOCAL);
```

Accessing Controller Services

In the Java Archive (NAR) structure, Controller Services are exposed as interfaces, usually in an API JAR. For example, the DistributedCacheClient is an interface extending from ControllerService, and it resides in the nifi-distributed-cache-client-service-api JAR, which is located in the nifi-standard-services-api-nar NAR. Other NARs that wish to reference interfaces (to create a new type of client implementation, e.g.) must specify the nifi-standard-services-api-nar as its parent NAR, and it then refers to provided instances of the API JARs in the processor submodule.
That's a little more low-level detail than you need to be able to leverage Controller Services, but I mention it for two reasons:

the scripting processors do have access to *some* of the Controller Service interfaces (and associated classes) in the nifi-standard-services-api-nar. These include DBCPService, DistributedMapCacheClient, DistributedSetCacheClient, HttpContextMap and SSLContextService. However I don't believe the other APIs included in the nifi-standard-services-api-nar will be available, and no custom ControllerService interfaces will be recognized. Having said that, you may be able to use the approach from the blog post in #1 to do so.

Processors that always intend on using a Controller Service instance create a property (i.e. PropertyDescriptor object) and call identifiesControllerService(*class*) on it. When the UI component is rendered, it will find all the Controller Services that implement the desired interface, and under-the-hood the component's ID is used, but its friendly name is displayed to the user.
For ExecuteScript, we can let the user choose a Controller Service instance by letting them specify the name or ID. If we allow the user to specify the name, then the script will have to perform a lookup trying to match the name to a (single!) element in the list of Controller Service instances of that type. That technique is described in the blog post above, so I won't cover it here. If the user enters the ID of the instance, then it is much easier to get access to the object as we will see below. The examples will use a DistributedMapCacheClientService instance called "distMapClient", connecting to a DistributedMapCacheServer instance (with standard defaults, localhost:4557), where the client instance has an ID of 93db6734-0159-1000-b46f-78a8af3b69ed:



Name	Type	State
MyDistributedMapCacheServer	DistributedMapCacheServer	Enabled
distMapClient	DistributedMapCacheClientService	Enabled

Configure Controller Service

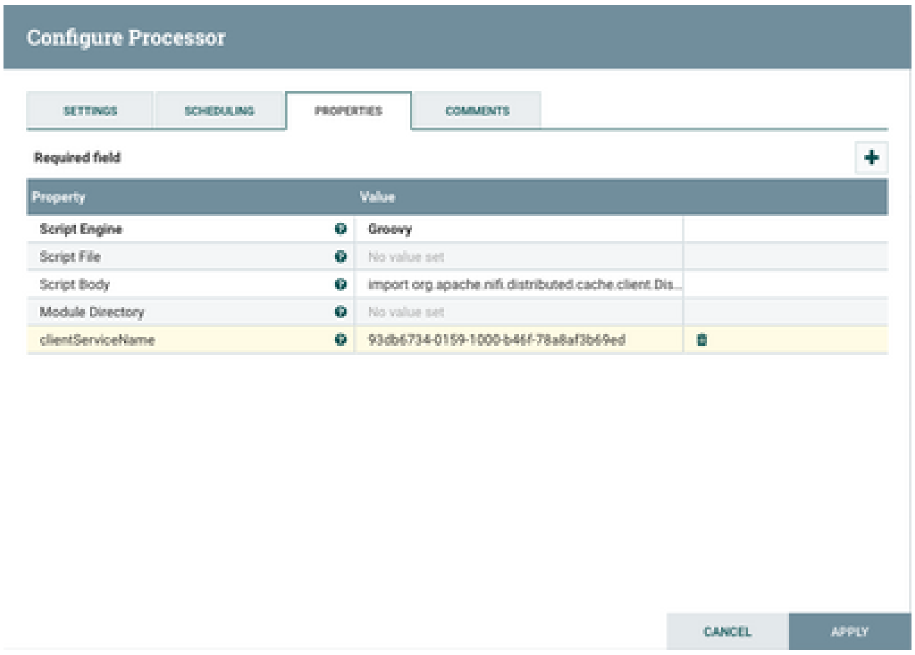
Name
distMapClient

Id
93db6734-0159-1000-b46f-78a8af3b69ed

Type
DistributedMapCacheClientService

Referencing
No referenc

In the ExecuteScript config, a dynamic property is created, called "clientServiceId" and set to 93db6734-0159-1000-b46f-78a8af3b69ed:


Configure Processor

Property	Value
Script Engine	Groovy
Script File	No value set
Script Body	import org.apache.nifi.distributed.cache.client.Dis...
Module Directory	No value set
clientServiceName	93db6734-0159-1000-b46f-78a8af3b69ed

We can then use clientServiceId.asControllerService(*DistributedMapCacheClient*), where the parameter is a reference to the Class object for DistributedMapCacheClient. For the examples, I have pre-populated the cache with a String key 'a' set to the String value 'hello'. For a handy Groovy script to work with a DistributedMapCacheServer, check out my article here.

Once we have a DistributedMapCacheClient instance, then to retrieve a value we can call its get(*key*, *serializer*, *deserializer*) method. In our case since the keys and values are Strings, we just need an instance of Serializer<String> and Deserializer<String> to pass into the get() method. The approach for all the languages is similar to the creation of the StreamCallback instances described in Part 2 of this article series. The examples will get the value for key 'a' from the pre-populated server and log the result ("Result = hello")

Recipe: Get the value of a property stored in a DistributedMapCacheServer

Use Case: A user has populated values into a DistributedMapCacheServer (configuration data, e.g.) and the script wants access to them.
Approach: Use the approach described above, creating a StringSerializer and StringDeserializer object, then getting the DistributedMapCacheClientService instance by ID, then calling get() on the service. Logging of the result is included here for convenience.
Examples:
Javascript

```
var DistributedMapCacheClient =
Java.type('org.apache.nifi.distributed.cache.client.DistributedMapCacheClient');

var Serializer =
Java.type('org.apache.nifi.distributed.cache.client.Serializer');

var Deserializer =
Java.type('org.apache.nifi.distributed.cache.client.Deserializer');

var StandardCharsets =
Java.type('java.nio.charset.StandardCharsets');



var StringSerializer = new Serializer(function(value, out) {

  out.write(value.getBytes(StandardCharsets.UTF_8));

})

var StringDeserializer = new Deserializer(function(arr) {

  // For some reason I had to build a string from the character
codes in the "arr" array

  var s = "";

  for(var i = 0; i < arr.length; i++) {

    s = s + String.fromCharCode(arr[i]);

  }

  return s;
})



var myDistClient =
clientServiceId.asControllerService(DistributedMapCacheClient.class)
;

var result = myDistClient.get('a', StringSerializer,
StringDeserializer);

log.info("Result = "+ result);
```

Running SQL on FlowFiles using QueryRecord Processor

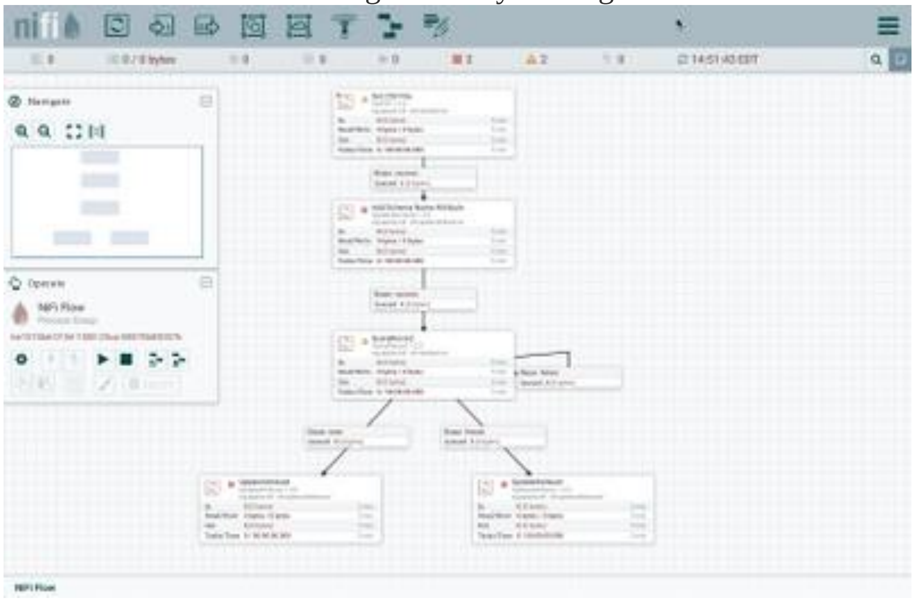
This tutorial walks you through a NiFI flow that utilizes the QueryRecord processor and Record Reader/Writer controller services to convert a CVS file into JSON format and then query the data using SQL.

QueryRecord Flow Demo Configuration

Support Files

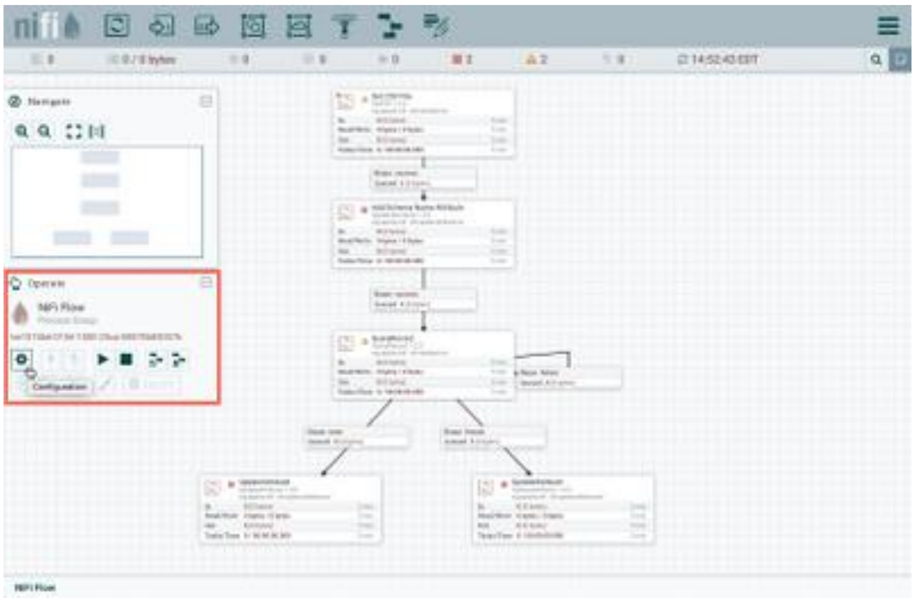
Here is a template of the flow discussed in this tutorial: queryrecord-csvtojson.xml
Here is the CSV file used in the flow: users.txt (Change the extension from .txt to .csv after downloading)
Note: The CSV data originated from the site, RandomUser. This useful site provides a free API to pull down randomly generated user data. For example: https://randomuser.me/api/0.6/?results=10&format=SQL.

Import Template

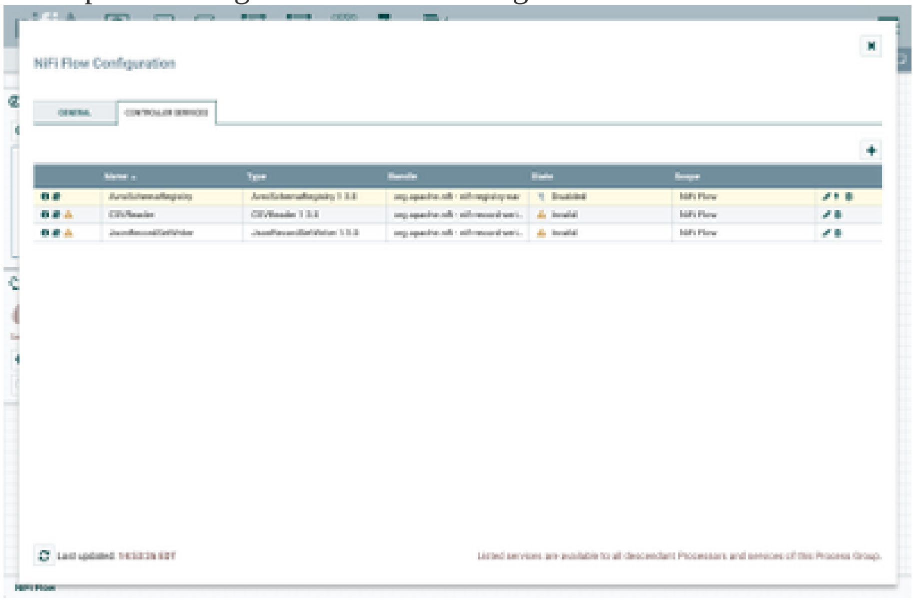
Start Logbot ETL. Import the provided template and add it to the canvas.
You should see the following flow on your Logbot ETL canvas:


Enable Controller Services

Select the gear icon from the Operate Palette:



This opens the Logbot ETL Flow Configuration window. Select the Controller Services tab:

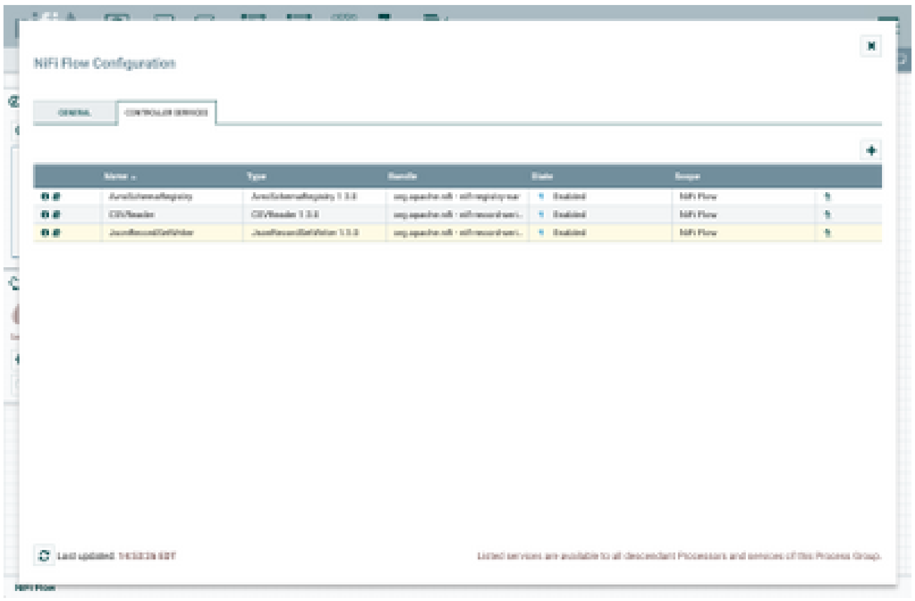


Name	Type	Bundle	State	Scope
AvroSchemaRegistry	AvroSchemaRegistry 1.13.2	org.apache.nifi - nifi-registry-nar	Disabled	NiFi Flow
CSVReader	CSVReader 1.13.2	org.apache.nifi - nifi-record-seri…	Invalid	NiFi Flow
JsonRecordSetWriter	JsonRecordSetWriter 1.13.2	org.apache.nifi - nifi-record-seri…	Invalid	NiFi Flow

Last updated: 14:53:26 EDT

Listed services are available to all descendant Processors and services of this Process Group.

Enable AvroSchemaRegistry by selecting the lightning bolt icon/button. This will then allow you to enable the CSVReader and JSONRecordSetWriter controller services. Select the lightning bolt icons for both of these services. All the controller services should be enabled at this point:



NiFi Flow Configuration

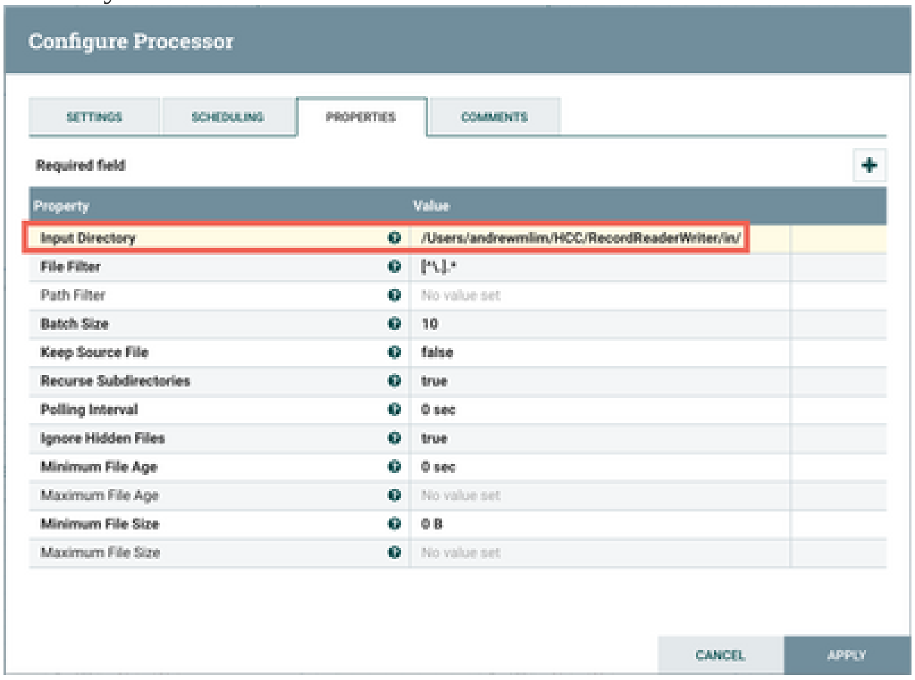
Name	Type	Bundle	State	Scope
AvroSchemaRegistry	AvroSchemaRegistry 1.3.0	org.apache.nifi - nifi-registry-nar	Enabled	NiFi Flow
CSVReader	CSVReader 1.3.0	org.apache.nifi - nifi-record-seri...	Enabled	NiFi Flow
JsonRecordSetWriter	JsonRecordSetWriter 1.3.0	org.apache.nifi - nifi-record-seri...	Enabled	NiFi Flow

Last updated: 14:53:39 EDT

Listed services are available to all descendant Processors and services of this Process Group.

Update Directory Path in GetFile Processor

Change the Input Directory path in the GetFile processor to point to your local input directory:



Configure Processor

Required field

Property	Value
Input Directory	/Users/andrewmlim/HCC/RecordReaderWriter/in/
File Filter	[^\.].*
Path Filter	No value set
Batch Size	10
Keep Source File	false
Recurse Subdirectories	true
Polling Interval	0 sec
Ignore Hidden Files	true
Minimum File Age	0 sec
Maximum File Age	No value set
Minimum File Size	0 B
Maximum File Size	No value set

The flow is now ready to run.

Flow Overview

Here is a quick overview of the flow:
1. GetFile ingests a CSV file of user data from a local directory
2. UpdateAttribute adds Schema Name "users" as an attribute to the flowfile
3. QueryRecord converts the flowfile contents from CSV to JSON by:

- Using a CSVReader controller service that references a schema in an AvroSchemaRegistry controller service

- The AvroSchemaRegistry contains a "users" schema which defines information about each record (field names, field ids, field types)
- Using a JsonRecordSetWriter controller service that references the same AvroSchemaRegistry schema

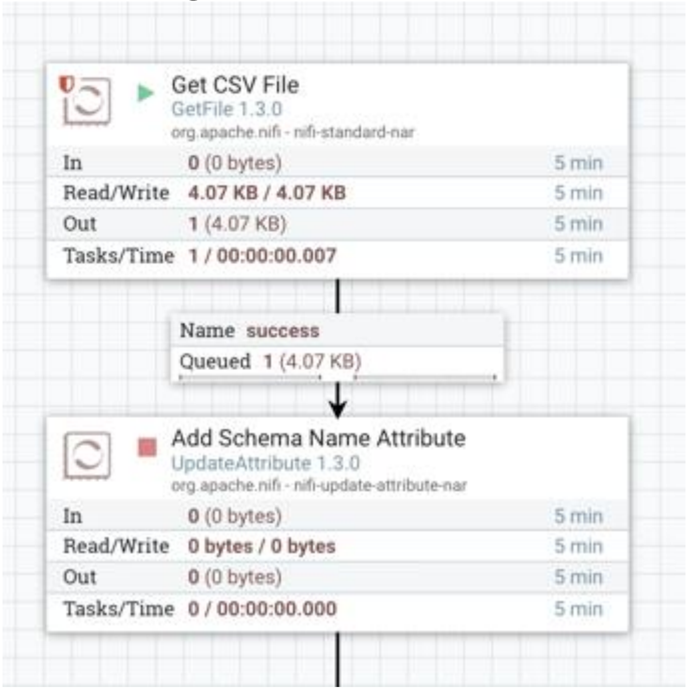
4. QueryRecord queries the flowfile using SQL:

- The "female" property uses SQL to select rows from the data where gender = 'F'
- The "male" property uses SQL to selects rows from the data where gender = 'M'

5. The female and male user data are sent to UpdateAttribute processors to provide a simple place to hold the data.

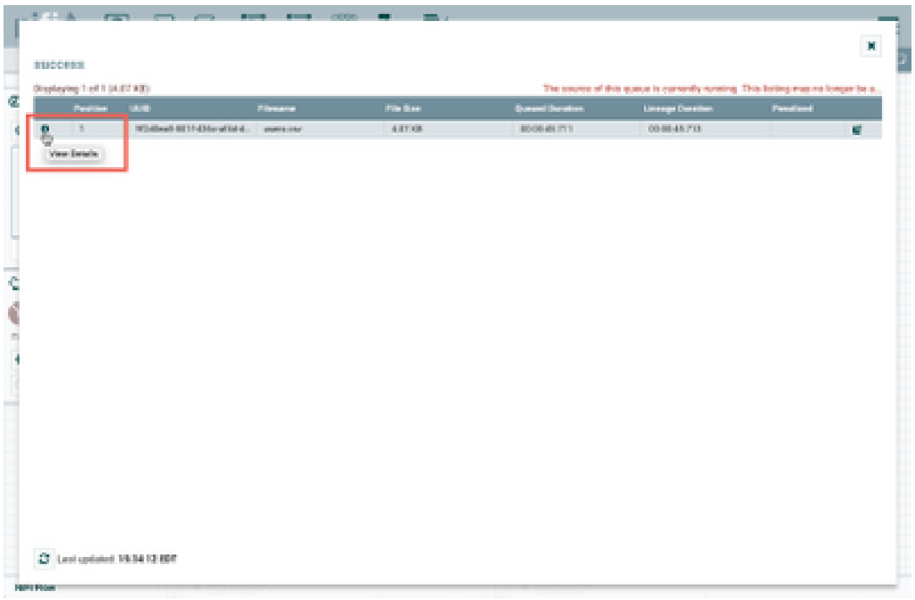
Flow Details

Let's look at each of the processors in the flow in detail:
Get CSV File (GetFile Processor)
FlowFiles are generated from the users.csv file in the local directory. Start the processor:



```
Get CSV File
GetFile 1.3.0
org.apache.nifi - nifi-standard-nar
In          0 (0 bytes)              5 min
Read/Write  4.07 KB / 4.07 KB        5 min
Out         1 (4.07 KB)              5 min
Tasks/Time  1 / 00:00:00.007         5 min

Name    success
Queued  1 (4.07 KB)

Add Schema Name Attribute
UpdateAttribute 1.3.0
org.apache.nifi - nifi-update-attribute-nar
In          0 (0 bytes)              5 min
Read/Write  0 bytes / 0 bytes        5 min
Out         0 (0 bytes)              5 min
Tasks/Time  0 / 00:00:00.000         5 min
```

One flowfile is generated with the CSV data as the contents. Right click on the connection between the GetFile Processor and the UpdateAttribute Processor. In the context menu, select "List Queue" and click the View Details button ("i" icon):



SUCCESS

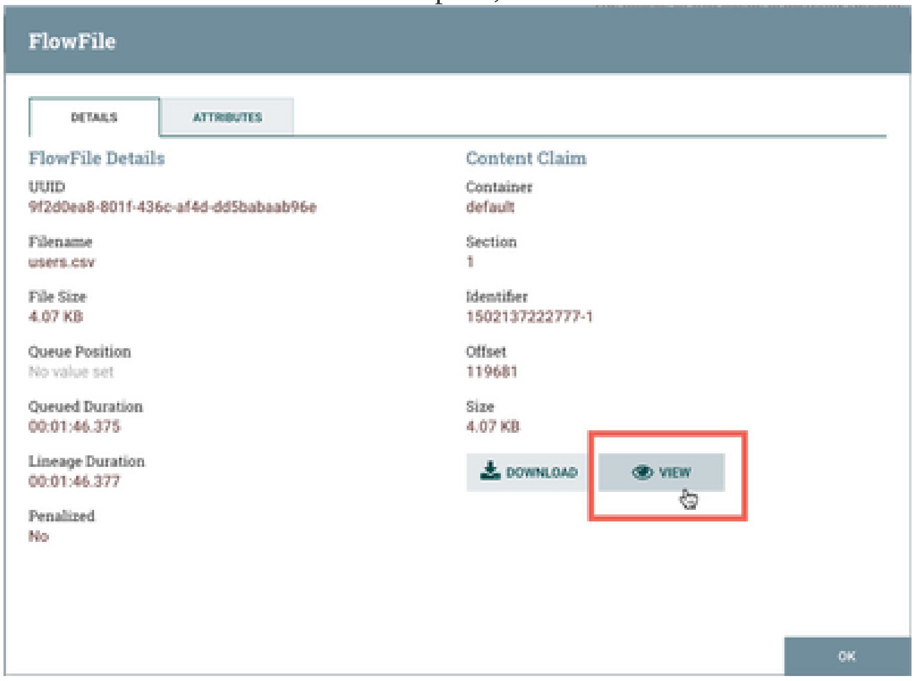
Displaying 1 of 1 (4.07 KB)

The source of this queue is currently running. This listing may no longer be a...

	Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized
	1	9f2d0ea8-801f-436c-af4d-d...	users.csv	4.07 KB	00:00:45.711	00:00:45.713	

Last updated: 19:34:12 EDT

From the FlowFile window that opens, select the "View" button from the Details tab:



FlowFile

FlowFile Details

UUID
9f2d0ea8-801f-436c-af4d-dd5babaab96e

Filename
users.csv

File Size
4.07 KB

Queue Position
No value set

Queued Duration
00:01:46.375

Lineage Duration
00:01:46.377

Penalized
No

Content Claim

Container
default

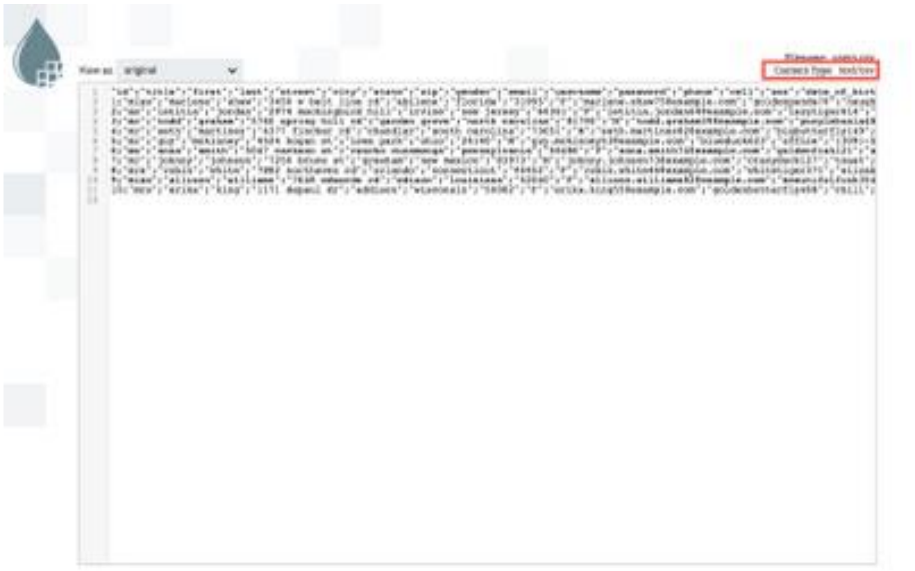
Section
1

Identifier
1502137222777-1

Offset
119681

Size
4.07 KB

to view the CSV contents of the flowfile:



View as: original

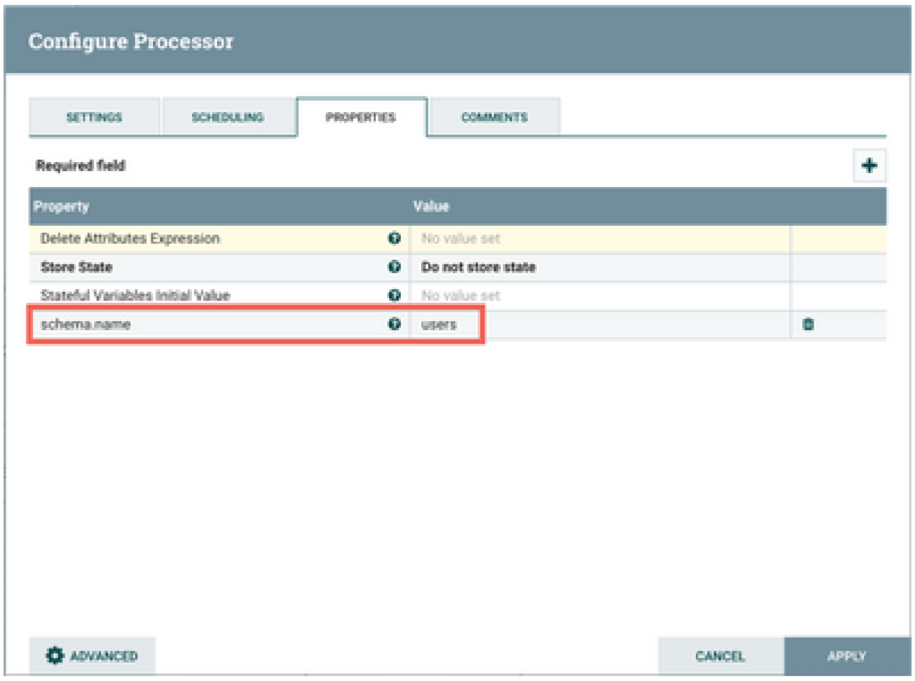
filename: users.csv
Content Type: text/csv

```
"id","title","first","last","street","city","state","zip","gender","email","username","password","phone","cell","ssn","date_of_birt
1,"miss","marlene","shaw","3450 w belt line rd","abilene","florida","31995","F","marlene.shaw75@example.com","goldenpanda70",[illegible]
2,"ms","letitia","jordan","2974 mockingbird hill","irvine","new jersey","64361","F","letitia.jordan64@example.com","lazytiger614",[illegible]
3,"mr","todd","graham","5760 spring hill rd","garden grove","north carolina","81790","M","todd.graham39@example.com","purplekoala48",[illegible]
4,"mr","seth","martinez","4377 fincher rd","chandler","south carolina","73651","M","seth.martinez82@example.com","bigbutterfly149",[illegible]
5,"mr","guy","mckinney","4524 hogan st","iowa park","ohio","24140","M","guy.mckinney53@example.com","blueduck623",[illegible]
6,"ms","anna","smith","5047 cackson st","rancho cucamonga","pennsylvania","56486","F","anna.smith74@example.com","goldenfish121",[illegible]
7,"mr","johnny","johnson","7250 bruce st","gresham","new mexico","83973","M","johnny.johnson73@example.com","crazyduck127",[illegible]
8,"mrs","robin","white","7882 northaven rd","orlando","connecticut","40452","F","robin.white46@example.com","whitetiger371",[illegible]
9,"miss","allison","williams","7648 edwards rd","edison","louisiana","52040","F","allison.williams82@example.com","beautifulfish354",[illegible]
10,"mrs","erika","king","1171 depaul dr","addison","wisconsin","50082","F","erika.king55@example.com","goldenbutterfly498",[illegible]
```

Note that there are 10 total users.

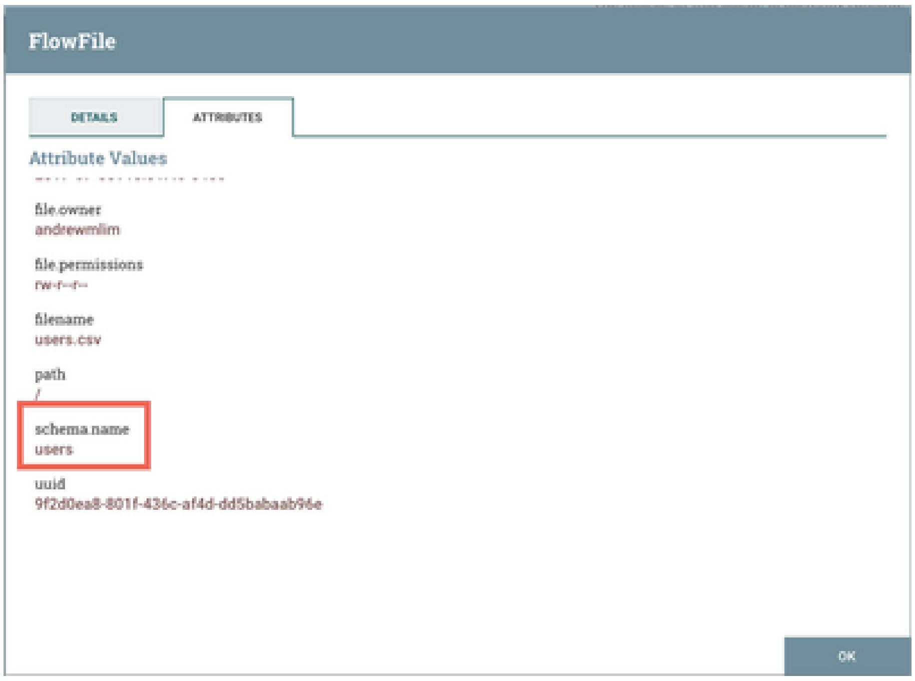
Add Schema Name Attribute (UpdateAttribute Processor)

The next step in the flow is an UpdateAttribute processor which adds the schema.name attribute with the value of "users" to the flowfile:



Property	Value
Delete Attributes Expression	No value set
Store State	Do not store state
Stateful Variables Initial Value	No value set
schema.name	users

Start the processor, and view the attributes of the flowfile to confirm this:



FlowFile

DETAILS | ATTRIBUTES

Attribute Values

file.owner
andrewmlim

file.permissions
rw-r--r--

filename
users.csv

path
/

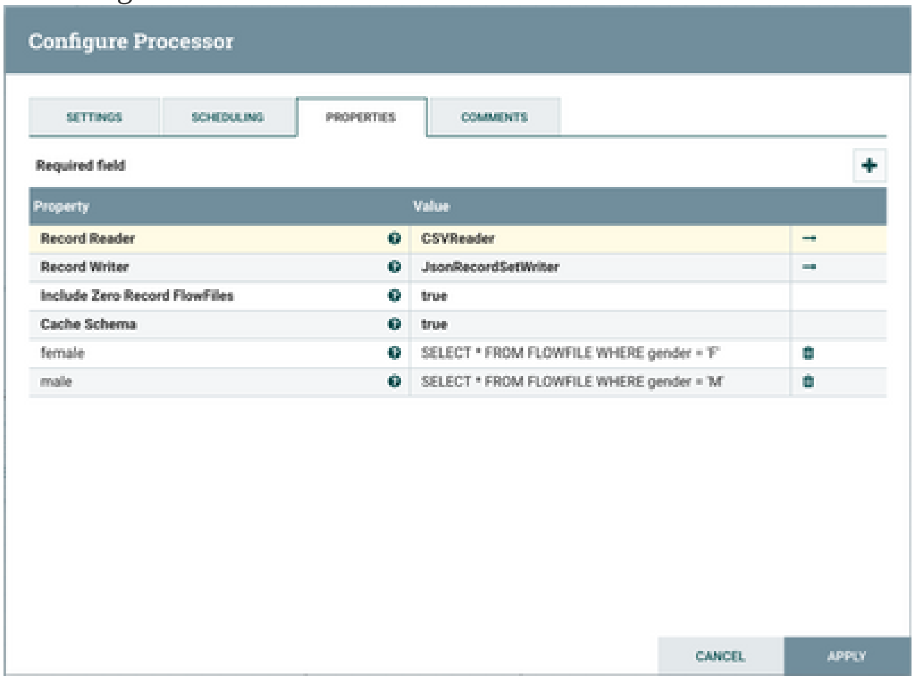
schema.name
users

uuid
9f2d0ea8-801f-436c-af4d-dd5babaab96e

OK

QueryRecord Processor

The next processor, QueryRecord, allows users to write SQL SELECT statements to run over their data as it streams through the system. Each FlowFile is treated as if it were a database table named FLOWFILE. Multiple SQL queries can be added to the processor. Looking at the configuration:



Configure Processor

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Record Reader	CSVReader
Record Writer	JsonRecordSetWriter
Include Zero Record FlowFiles	true
Cache Schema	true
female	SELECT * FROM FLOWFILE WHERE gender = 'F'
male	SELECT * FROM FLOWFILE WHERE gender = 'M'

CANCEL | APPLY

Record Reader is set to "CSVReader" and Record Writer is set to "JsonRecordSetWriter". The "CSVReader" controller service parses the incoming CSV data and determines the data's schema. The "JsonRecordSetWriter" controller service determines the data's schema and writes that data into JSON. More details about these controller services can be found below. In order to distinguish the results of each query and route the data appropriately, the name of the property is the name of the Relationship that data matching the query should be routed to.

The first added property is named "female" and will include user data where the gender of the user is "F":

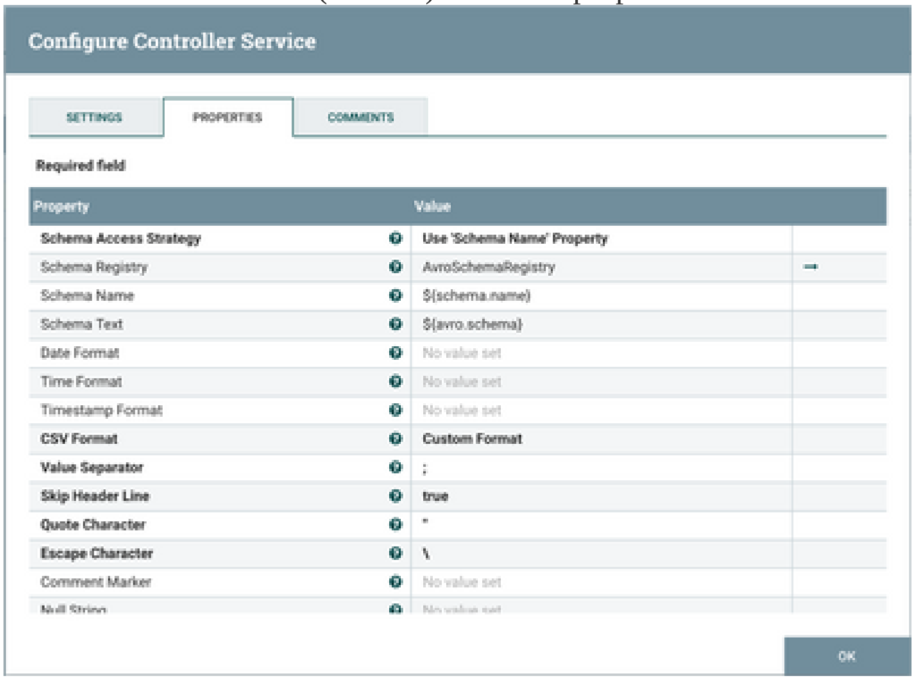
```
SELECT *

FROM FLOWFILE

WHERE gender = 'F'
```

(NOTE: When entering a value for a property in Logbot ETL, you can use Shift + Enter to insert a newline in your value.)
The second property is named "male" and will include user data where the gender of the user is "M":

```
SELECT *

FROM FLOWFILE

WHERE gender = 'M'
```

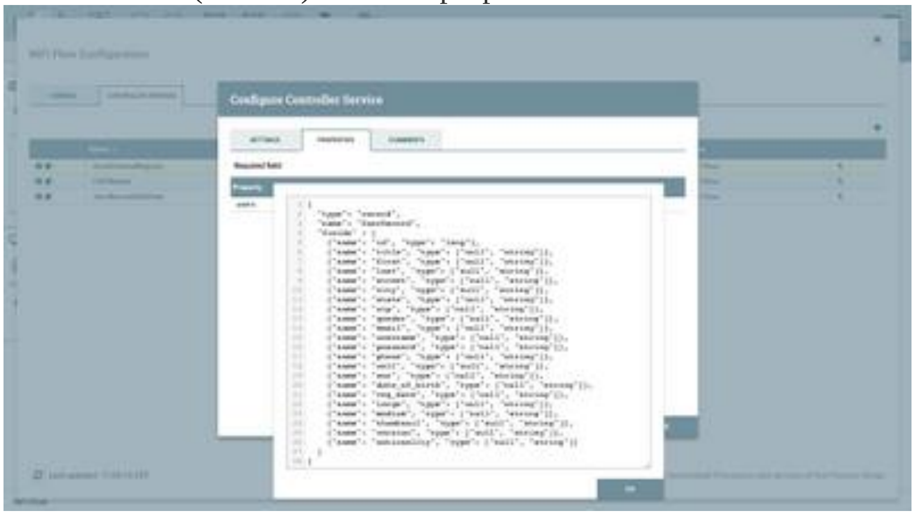
CSVReader Controller Service

Select the arrow icon next to the "CSV Reader" which opens the Controller Services list in the Logbot ETL Flow Configuration. "CSVReader" should be highlighted in the list. Select the View Details button ("i" icon) to see the properties:



Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	AvroSchemaRegistry
Schema Name	${schema.name}
Schema Text	${avro.schema}
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set
CSV Format	Custom Format
Value Separator	;
Skip Header Line	true
Quote Character	"
Escape Character	\
Comment Marker	No value set
Null String	No value set

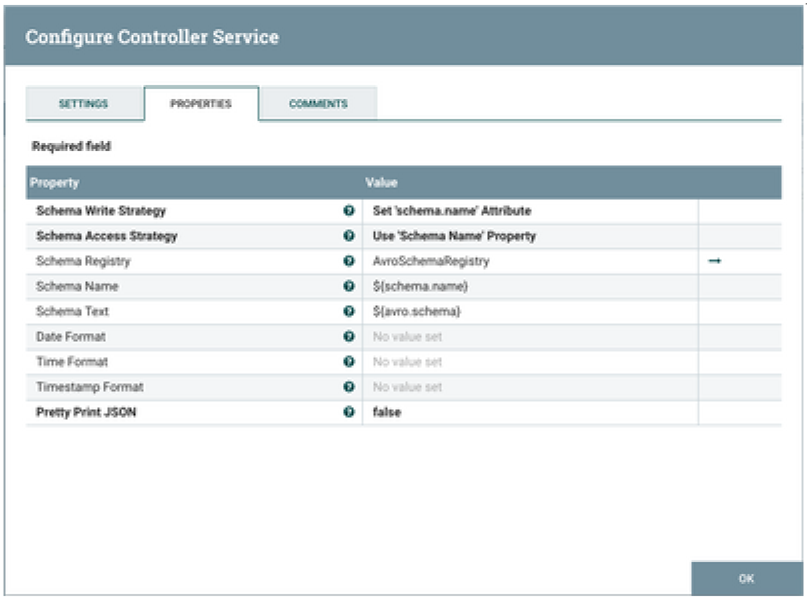
With Schema Access Strategy property set to "Use 'Schema Name' Property", the reader specifies the schema expected in an attribute, which in this flow is schema.name. The Schema Registry property is set to the AvroSchemaRegistry Controller Service which defines the "users" schema. Select the arrow icon next to "AvroSchemaRegistry" and select the View Details button ("i" icon) to see its properties:



```
{
  "type": "record",
  "name": "UserRecord",
  "fields" : [
    {"name": "id", "type": "long"},
    {"name": "title", "type": ["null", "string"]},
    {"name": "first", "type": ["null", "string"]},
    {"name": "last", "type": ["null", "string"]},
    {"name": "street", "type": ["null", "string"]},
    {"name": "city", "type": ["null", "string"]},
    {"name": "state", "type": ["null", "string"]},
    {"name": "zip", "type": ["null", "string"]},
    {"name": "gender", "type": ["null", "string"]},
    {"name": "email", "type": ["null", "string"]},
    {"name": "username", "type": ["null", "string"]},
    {"name": "password", "type": ["null", "string"]},
    {"name": "phone", "type": ["null", "string"]},
    {"name": "cell", "type": ["null", "string"]},
    {"name": "ssn", "type": ["null", "string"]},
    {"name": "date_of_birth", "type": ["null", "string"]},
    {"name": "reg_date", "type": ["null", "string"]},
    {"name": "large", "type": ["null", "string"]},
    {"name": "medium", "type": ["null", "string"]},
    {"name": "thumbnail", "type": ["null", "string"]},
    {"name": "version", "type": ["null", "string"]},
    {"name": "nationality", "type": ["null", "string"]}
  ]
}
```

JsonRecordSetWriter Controller Service

Close the window for the AvroSchemaRegistry. Select the View Details button ("i" icon) next to the "JsonRecordSetWriter" controller service to see its properties:

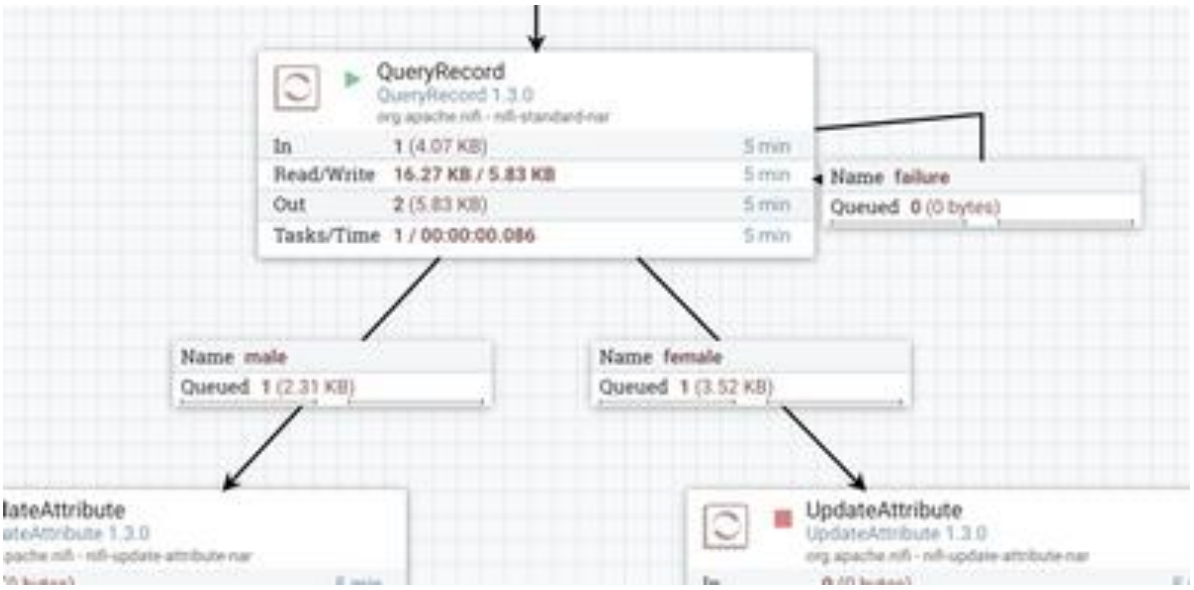


Required field

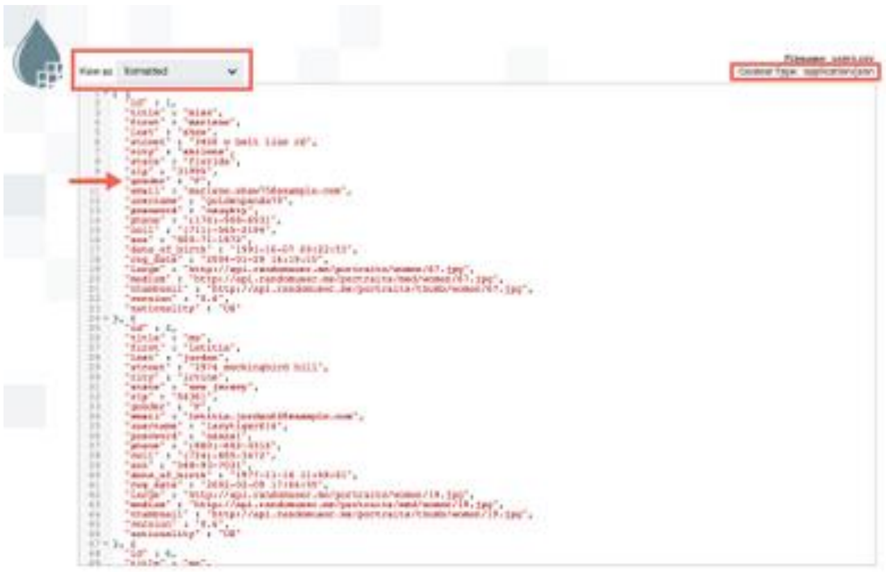
Property	Value
Schema Write Strategy	Set 'schema.name' Attribute
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	AvroSchemaRegistry
Schema Name	${schema.name}
Schema Text	${avro.schema}
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set
Pretty Print JSON	false

Schema Write Strategy is set to "Set 'schema.name' Attribute", Schema Access Strategy property is set to "Use 'Schema Name' Property" and Schema Registry is set to AvroSchemaRegistry.

Start the QueryRecord processor. The flowfile is now split into two flowfiles:

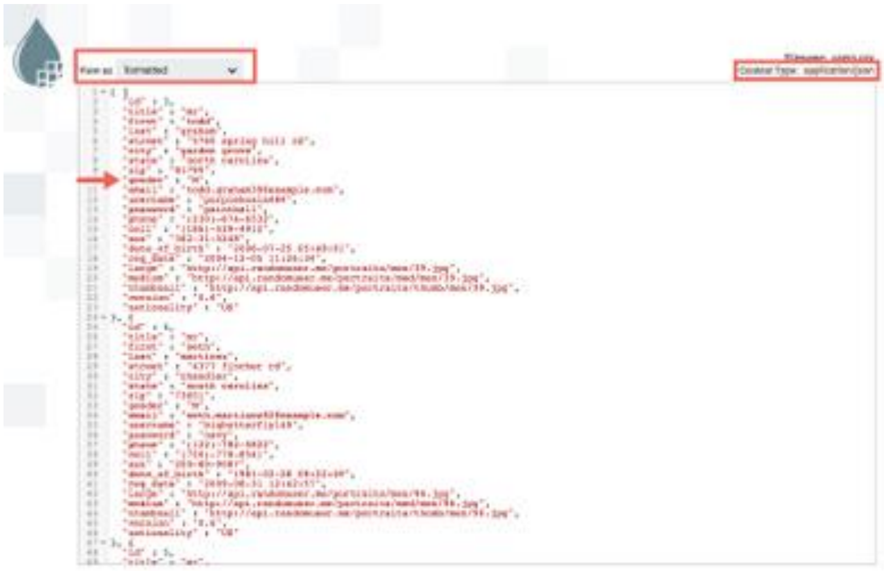


Looking at the contents of the "female" connection confirms the expected results of 6 female users in JSON format:



```
[illegible]
```

Looking at the contents of the "male" connection confirms the expected results of 4 male users in JSON format:



```
[illegible]
```

1

The SQL queries in the flow are very basic for the purposes of this tutorial. The beauty of the QueryRecord processor is that it supports the SQL for more advanced queries and operations, such as filtering specific columns/rows/fields from your data, renaming those columns/rows/fields, performing calculations and aggregations on the data.