Pipeline Blocks

Pipeline Blocks are the building blocks for the Alarm Pipeline Notification system. Each block has an input and zero or more outputs. Each block has its own unique functionality and performs a specific action for the pipeline, such as sending a notification, setting a property, or evaluating an expression.

Let's look at the different pipeline blocks that come with the Alarm Notification system. There are 8 blocks above your Pipeline Workspace that you can choose from to build your alarm notification pipeline.

Pipeline Blocks

Watch the Video

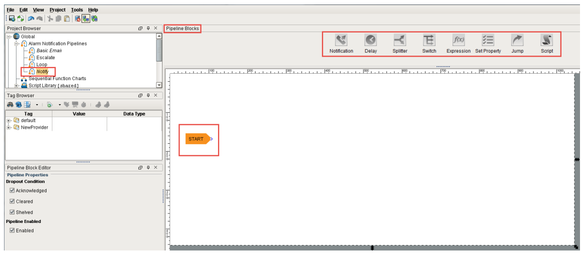
Pipeline Blocks

The **Pipeline Blocks** are described individually in this section, but here is a brief overview:

- **Notification**
  Delivers a notification through the selected Notification Profile (i.e., Email, SMS and Voice).
- **Delay**
  Blocks the alarm event for the specified amount of time.
- **Splitter**
  Forwards a single event concurrently to multiple other blocks.
- **Switch**
  Evaluates a non-boolean Expression, and forwards to an output based on the result.
- **Expression**
  Evaluates an Expression that results in either '`True`' or '`False`,' and forwards the event to that output.
- **Set Property**
  Evaluates an Expression, and sets the result as a runtime property on the alarm event.
- **Jump**
  Forwards the alarm event to a different pipeline.
- **Script**
  Executes a task outside the pipeline.

Start Block

When you create a new pipeline, you always get a **Start Block** placed in your Pipeline Workspace. You cannot delete the **Start Block** because it represents the entry point for alarm events into the pipeline.



Notification Block

As the name suggests, the **Notification Block** is responsible for sending notifications. Each notification block can target one particular notification profile, which represents one method of notification, such as Email, SMS, or Voice. Each type of notification profile will have its own relevant properties for configuration, although some features are common to all profile types. The various profiles will also behave differently, according to their capabilities, in regards to how notifications are ordered. For example, the Email profile is capable of sending a message to many recipients at once, while the SMS and Voice notification profiles must step through each contact sequentially.

Notification Block Basic Setup

There are two required settings for notification blocks: the Notification Profile and the On-Call Roster. The profile will dictate the method through which notifications occur, and the on-call roster will dictate who receives the notifications. The on-call roster provides an ordered list of contacts, which will first be pared down according to each user's schedule. The resulting contacts will be notified in order, based on how the profile operates. The settings displayed will depend on the type of profile selected.

Email Settings

- **From Address** - Email address used for the "From" field.
- **Message** - The body of the email. Like the subject, may refer to properties of the alarm.

  Message now supports HTML formatting. Simply add the <html> tag at the beginning of the Message property. An end tag (</html>) is not required.

  ```
  <html>This is my message. <br>That was a line break.
  ```

- **Consolidated Message** - The message sent when consolidation is turned on, and more than one alarm is being sent. Can refer to properties, and additionally support a special expansion syntax of "{{ ... }}", where everything inside of the double curly braces will be repeated for each alarm in the consolidation group.
- **Test Mode** - If enabled, logs to the console each time an email would be sent, instead of sending the actual email.

Voice Settings

- **Require PIN** - If true, the user will need to enter their PIN in order to hear the alarm notification messages. The user's PIN is defined in the user management section of the Gateway. If false, then anyone will be allowed to hear the messages after answering the phone.
- **Allow Acknowledgment** - If false, users will only be allowed to listen to alarms, not to acknowledge them.
- **Retries Per Contact** - The number of retry attempts per contact per alarm. The default is set to 1.
- **Delay Between Calls** - Introduces a delay between calling each contact, for each alarm. The pipeline dropout conditions are checked regularly between calls and while waiting, so this would provide time for the alarm to drop out before calling the next person. The delay is only enforced after following a "successful" call (a call that was answered). Unanswered or errored calls will move on to the next contact right away. Please note that long delays can block other alarms in the call queue. The delay is applied to all contacts for a particular alarm.

  For most phone systems, a delay of at least 1 second is advised. The system must have time between each call to hang up properly before dialing again.
- **Test Mode** - If enabled, messages will be logged to the console indicating when and to whom calls would have been made, without actually making a call.

SMS Settings

- **Message** - The message to use for single events.
- **Consolidated Message** - Like email, the message to use when multiple events are sent together, due to the block's consolidation settings.
- **Delay Between Notification** - As with voice, a delay between each message sent, to give the recipient time to acknowledge and prevent further notification.
- **Test Mode** - If enabled, logs messages that would have been sent to the Gateway console, and does not actually send them.

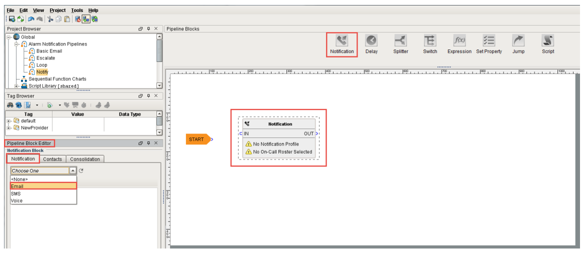
Consolidation

Notification consolidation allows you to limit the number of notifications sent by specifying a delay during which incoming events will be collected. When the delay expires, all events that have arrived during the period of time will be sent together to the notification profile. The manner in which the profile implements consolidation will vary, but in general the result will be a shorter message, or fewer messages, than would have occurred otherwise.

Consolidation is defined with two parameters:

- **Delay** - How long to wait after the first eligible alarm arrives. This setting forces a pause **before** sending a notification. The delay is used to protect against situations where an event might generate many alarms together. In other words, if an alarm comes into the Notification Block, it will be delayed for this amount of time in case there are other alarms arriving soon.
- **Frequency** - The max frequency with which the profile can send alarms. This setting forces a pause **after** sending a notification. The frequency is used to ensure that contacts aren't notified too often, for example if an alarm is rapidly going in and out of the active state.

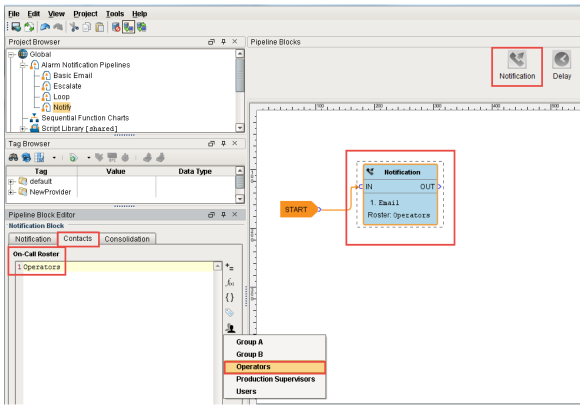
Drag the **Notification Block** to your Pipeline Workspace. Click the **Notification** tab in the Pipeline Block Editor. Choose one of the Notification profiles that you configured. For this example, choose **'Em ail.'**



Contacts

From the **Contacts** tab, you can specify the **On-Call Roster** who you want to send the alarm notification too. An On-Call Roster must first be created. In this example, several rosters were already configured. Select **'Operators'** as your on-call Roster.

You will notice that the Notification Block is now updated with your **'Email'** profile and **'Operators'** on-call roster.



The Contacts tab now accepts expressions, so it is possible to dynamically select a roster from the pipeline. The expression can make use of tag values as well as alarm properties and associated data.

The alarm properties are accessible from the Alarm Properties button { } located on the right side of the panel. Associated data can be manually entered using the "{" and "}" characters.

```
if({Group}= "A","Group A","Group B")
```

The above expression checks for associated data on the alarm named "Group". If the value is equal to "A", the Group A roster is used, otherwise the Group B roster will be notified.

When should I use quotation marks on the Roster name?

It is recommended to **always place quotation marks around roster names**. However, it is not always required for notifications to be sent out successfully.

If the expression consists of only the name of the roster, then quotes can be omitted. For example, specifying that a Notification block should always use the Roster named "**Operators**" could look like the following with quotes:

```
//This expression will evaluate successfully.
"Operators"
```

the expression would return the string **Operators**, which is a valid Roster.

However, quotation marks could be removed in this scenario. When removed, this causes a syntax error because strings in expressions must always be encased in quotation marks. However, instead of returning an error, the Notification block would assume you simply typed out the name of the roster, so the following example would also succesffully use the **Operators** roster:

```
//This will fail to evaluate as an expression, so the block will instead
look for a roster named "Operators",
//and the users will still be notified.
Operators
```

However, if the expression is any more complicated than the above example, such as using an expression function, then quotation marks must be used. The following example would check the priority of the alarm. If priority was set to "critical", then the **High Priority Roster** would be notified. For all other priorities, the **Low Priority Roster** would be used.

```
//This expression will evaluate successfully.
if({priority} > 3, "High Priority Roster", "Low Priority Roster")
```

However, if the quotes were removed from the roster names, the expression would fail with a syntax error:

```
//This expression will fail, so all the text is assumed to the name of a
Roster
if({priority} > 3, High Priority Roster, Low Priority Roster)
```

When this expression fails, the Notification block will assume that the text is actually the name of a roster, and will attempt to use a roster named "**if({priority} > 3, High Priority Roster, Low Priority Roster)**". For this reason, it is a good idea to always place quotes around Roster names, as well as check your expressions for syntax errors.

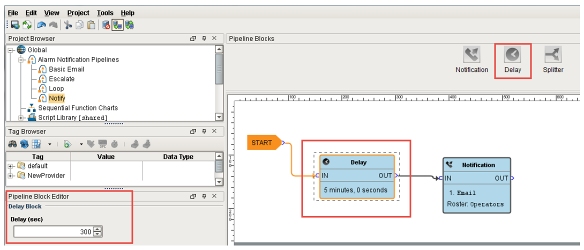
Delay Block

The **Delay Block** simply blocks alarm events for a specified period of time before moving them to the next block. They are generally used to wait for the dropout condition to become satisfied for an alarm.

For example, a 5 minute delay might be used to give operators viewing control screens a chance to acknowledge, and only send notifications if they haven't (the "active delay" deadband on the alarm could be used to delay the alarm as well, but in that case it wouldn't actually be active, and therefore not displayed, for the delay time).

Delays are often also used to control flow in a pipeline. For example, in an "escalation" scenario, a notification block might be used to send emails. Since emails are sent instantly, and acknowledgment occurs "asynchronously" (the user must see the email and click the link or log into the system), a delay block could be used to provide time for the user to acknowledge, before moving on to a voice notification block. There is no practical limit to how long a delay can be. The delay is calculated independently for each event entering the block, meaning that each event will be held for the specified time.

Click the **Delay Block** and a spinner will appear in the Pipeline Block Editor. You can specify the number of seconds to delay. In this example, if you enter **'300'** seconds, the alarm will wait 5 minutes before sending it to the Operators.



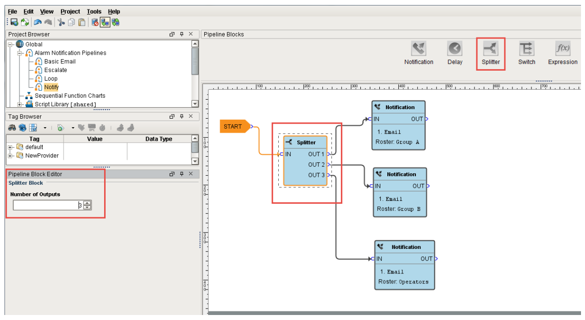
Pipeline Block Editor
Delay Block
Delay (sec) 300

Delay
5 minutes, 0 seconds

1. Email
Roster: Operators

Splitter Block

The **Splitter Block** simply forwards a single alarm event to multiple outputs at the same time. It creates a copy of the alarm event for each output and executes the alarm notifications along multiple paths in parallel. You can specify as many outputs as you want. If care is not taken, it is possible to end up with an exponential number of alarm event copies active in the pipelines at one time. Since each notification block operates on a specific on-call roster, Splitters are useful for delivering events to multiple notification blocks at once as shown in this example.



Pipeline Block Editor

Splitter Block

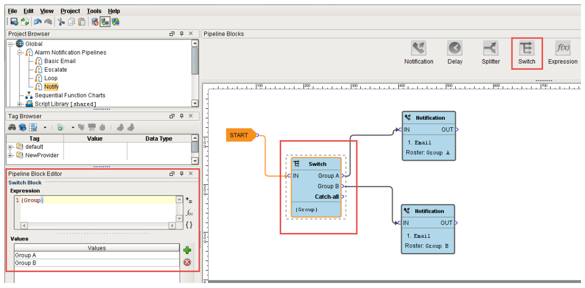
Number of Outputs: 3

Switch Block

The **Switch Block** evaluates a non-boolean expression and forwards it to an output based on a given result. It can be a number or string, and it will be looked up against a defined list of expected results. If a match is found, the alarm will follow that path, otherwise the "Catch-all" output is used.

You can create an expression and even use the alarm event's priority, associated data, value and more. The following example has associated data for the 'Group' that they belong to with 'Group A' and 'Group B' values.   The Switch Block will evaluate the expression results against these defined values, and if a match is found, the alarm will follow that path. If neither of the values are met, the Catch-all output will be used.

This example demonstrates that a pipeline can send out an alarm to 2 different lists of people based on the **'Group.'**



Pipeline Block Editor

Switch Block

Expression

```
{Group}
```

Values

Values
Group A
Group B

Expression Block

The **Expression Block** contains an Expression which is evaluated against each alarm that enters it. The result is expected to be a boolean value, either 'True' or 'False' and forwards the alarm to the specific output. The Expression executed by the block is written in the same syntax as other Expressions in Ignition. However, in this context, it is possible to refer directly to properties of the alarm event that is being evaluated by using the standard "{path}" reference syntax. In addition, the following functions are available to quickly determine the state of the alarm.

Examples

`isActive()`
This single function returns whether the current event is active or not. An Expression block like this could be used to then dispatch to an "active" pipeline, and a "clear" pipeline (both the active and clear pipeline settings on the alarm would be set to this dispatch pipeline). This kind of setup allows you to later modify how actives are handled vs. clears in one place, without having to modify many alarms.

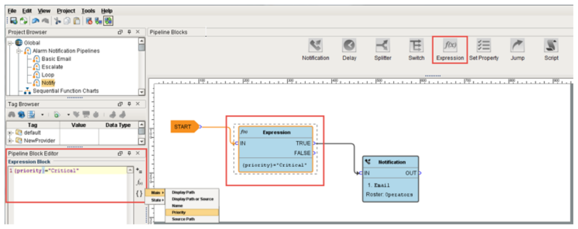
`toInt({priority})>2 && {displayPath} like "*East Area*"`
This block would forward all High and Critical alarms from the "east area" to the true path. The others would go to false, which may not actually be bound to another block, making this Expression block act like a filter.

`isPropertyDefined("isEscalated")`
This Expression checks if a property exists on the event. The "isEscalated" property is not a system defined property. Instead, in this example, it might be set using a Set Property block before forwarding back to the start of the pipeline. The first time through, this Expression would fail, but the next time, it would pass, and the "escalated" behavior could be executed.

In the **Pipeline Block Editor**, you can select the Alarm Properties from any of the icons in the Expression Block. This example uses an expression to check to see if the priority is 'Critical,' and if so, the alarm notification will follow the 'True' path. If it is not 'Critical', it will follow the 'False' path. As you can see, the 'False' output is not configured. You don't have to configure every output if you don't want too.

Priority Expression Example

```
{priority}="Critical"
```



```
{priority}="Critical"
```

Set Property Block

The **Set Property Block** allows you to define or redefine a property on the alarm event. The value is created from the Expression Block, which can refer to other properties or the same properties. Typically, the Set Property Block is used as a counter for the number of times the alarm has been looping or notifying people in your on-call roster. Settings modified in this way will only exist while the alarm is in the pipeline, they will not be stored to the journal, or show up in the status table.

Examples

If you want to attempt notification up to three times before stopping, you could create a pipeline that looked like [Notification Block] > [Set Property] > [Expression], with the Expression looping back to the notification block (perhaps with a delay in between).

The Set Property block would look like:
Property Name: counter
Expression: coalesce({counter}, 0)+

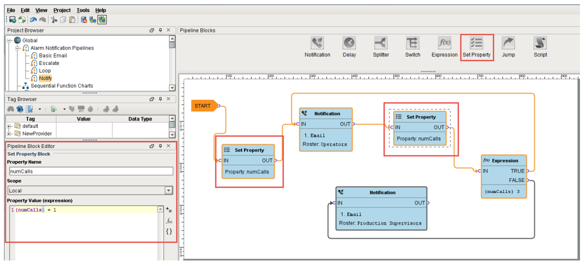
Note that the first time the block is hit, the `counter` property will not exist, and therefore the value will be null. The `coalesce` function returns the first non-null value, allowing us to start with 0. Another way to write this might be:

```
if(isPropertyDefined("counter"), getProperty("counter"), 0)+1
```

The `getProperty()` function is functionally equivalent to simply referencing the property in brackets (that is, `{counter}`).

The Expression block in this example would simply be: `{counter}`<3.

Here is another example where the operators are notified of an alarm event 3 times before being escalated to the Production Supervisors. Drag 2 **Set Property Blocks** to your pipeline workspace. Assign the first Set Property block to a Property Name **'numCalls'** and a Property Value of **'0 .'** The second Property Block you need to increment the counter. Enter **'{numCalls} + 1'** as shown in the Pipeline Property Editor. Each time the alarm goes through this block, the 'numCalls' variable increments by '1.' If no one acknowledges the alarm, it is sent to the Production Supervisor's on-call roster.



Pipeline Block Editor
Set Property Block
Property Name
numCalls
Scope
Local
Property Value (expression)

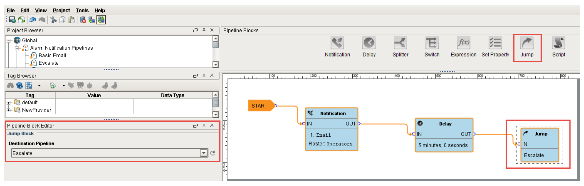
```
{numCalls} + 1
```

Jump Block

The **Jump Block** forwards an alarm event to a different pipeline. You can breakup a pipeline into several different pipelines and use the Jump Block to go back and forth between pipelines. This is perfect for an escalation pipeline because often times they can become one massive pipeline, and the Jump Block can make the escalation pipeline less complicated and more compact.

In the Pipeline Block Editor, enter the Destination pipeline called **'Escalate.'**

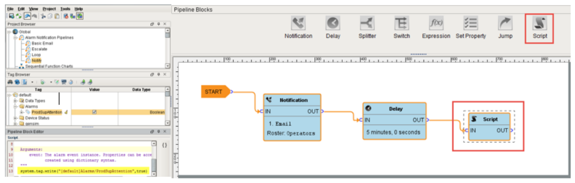
In this simple example, the **Jump Block** forwards the alarm to the Escalate pipeline and ends there.



Pipeline Block Editor
Jump Block
Destination Pipeline
Escalate

Script Block

The **Script Block** allows you to execute a task outside the pipeline like writing to another alarm tag or database. In this example, the Script Block was used to notify the production supervisors of an alarm using another alarm tag as shown in the Pipeline Block Editor.



Pipeline Block Editor
Script

```
Arguments:
    event: The alarm event instance. Properties can be acce
           created using dictionary syntax.
"""
system.tag.write("[default]Alarms/ProdSupAttention",true)
```

In the next sections, you will see how to use a combination of these different pipeline blocks to create your own unique pipelines.

Next...

- Pipeline - Filter on Alarm Priority