

Ignition 8.3 User Manual — Symbol Factory, WebDev and Cloud Connector Modules

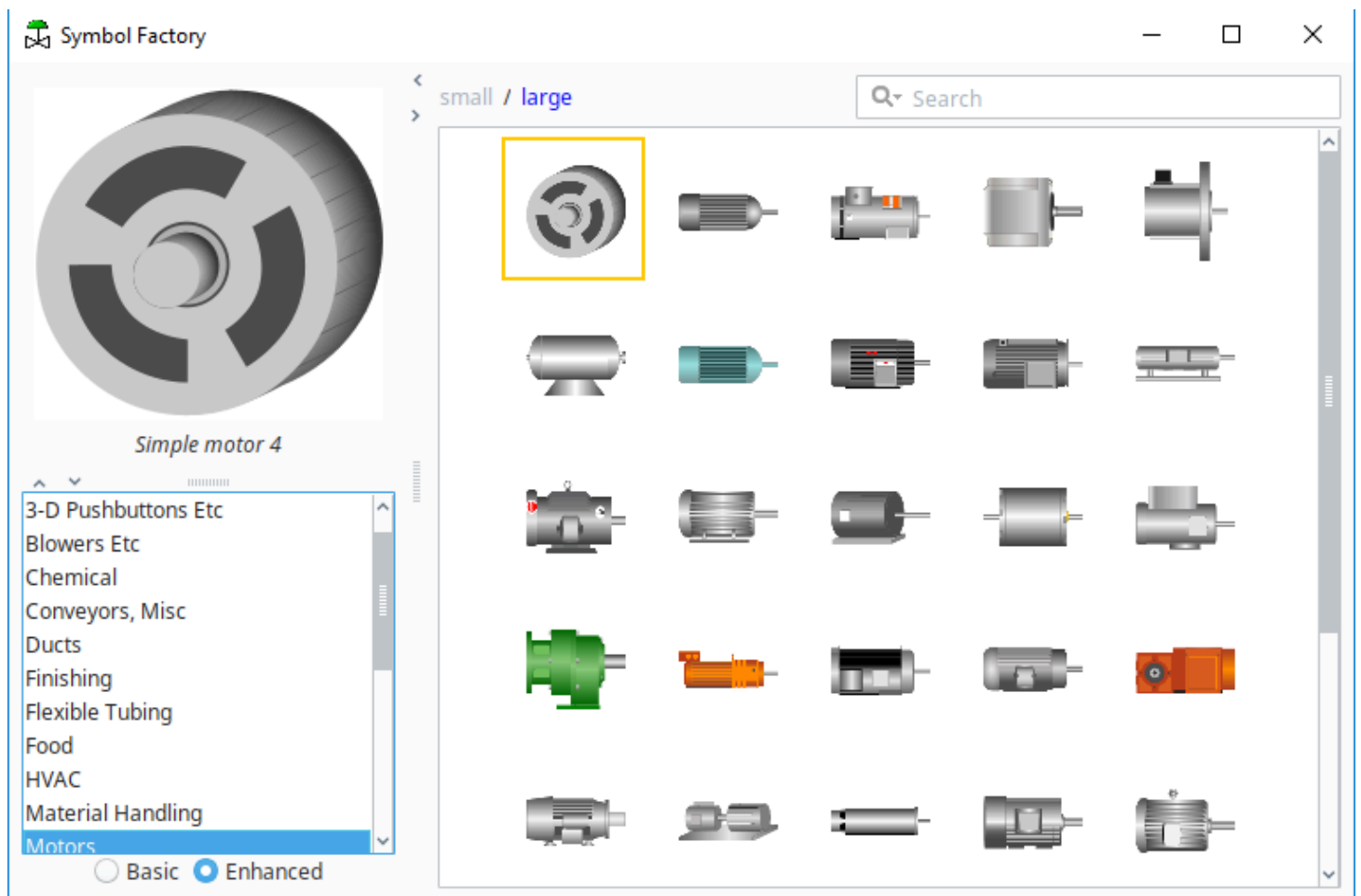
Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/8.3/>

Symbol Factory

The Symbol Factory module is included with the Vision module or the Perspective module. Symbol Factory provides nearly 4,000 industrial high quality Scalable Vector Graphics (SVG) and symbols for your projects. Vector based graphics can be resized with no pixelation or distortion. You can add these images to your project with a simple drag and drop.

In Vision, the images can also be edited or even animated. In Perspective, if you need animated symbols, try the [Perspective - Symbols Palette](#).

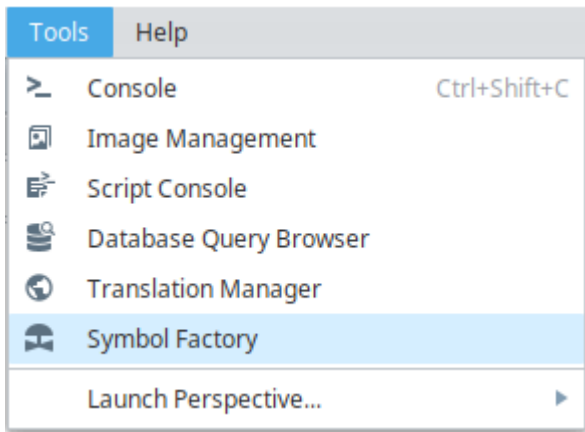
Symbol Factory images are also great for mobile responsive design, or any implementation where users view your HMI and SCADA on screens of various sizes.



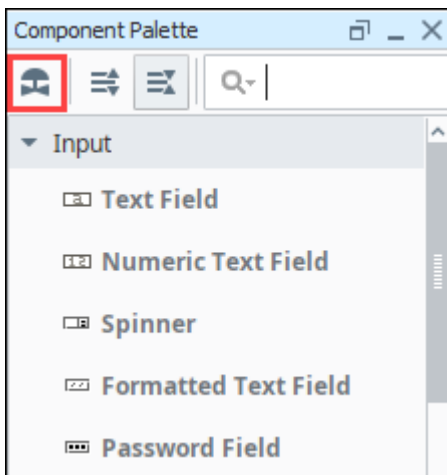
Using the Symbol Factory

1. Launch the Designer and open your project.
2. Choose **Symbol Factory** under the **Tools** menu or the project navigation tree. If you can't find these, the Symbol Factory module probably isn't installed. The Symbol Factory browser opens as

a pop-up window that stays on top of the Designer.



3. In the Vision module, there is also an icon (valve icon in the upper left) for quick access to the Symbol Factory images at the top of the Component Palette.

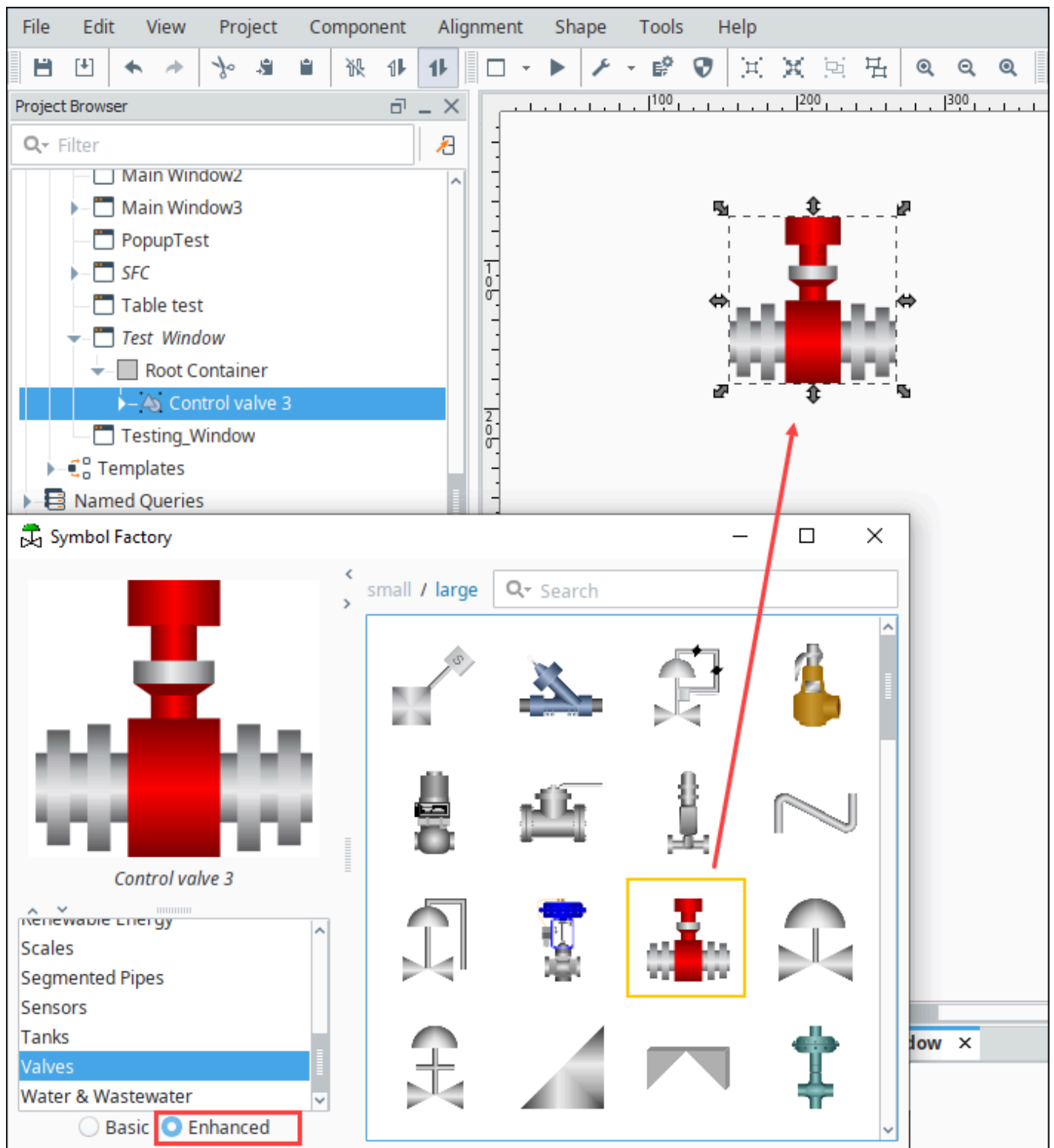


4. Browse the different categories to explore what symbols are available, or search to find a specific symbol.

NOTE

When searching Symbol Factory, it is recommended to select the Enhanced radio button in the Symbols List. The enhanced symbols are more detailed. They also have groupings that enable you to more easily animate them when used in Vision.

5. Find a symbol that you'd like to use and drag it onto an open Vision window or Perspective view.



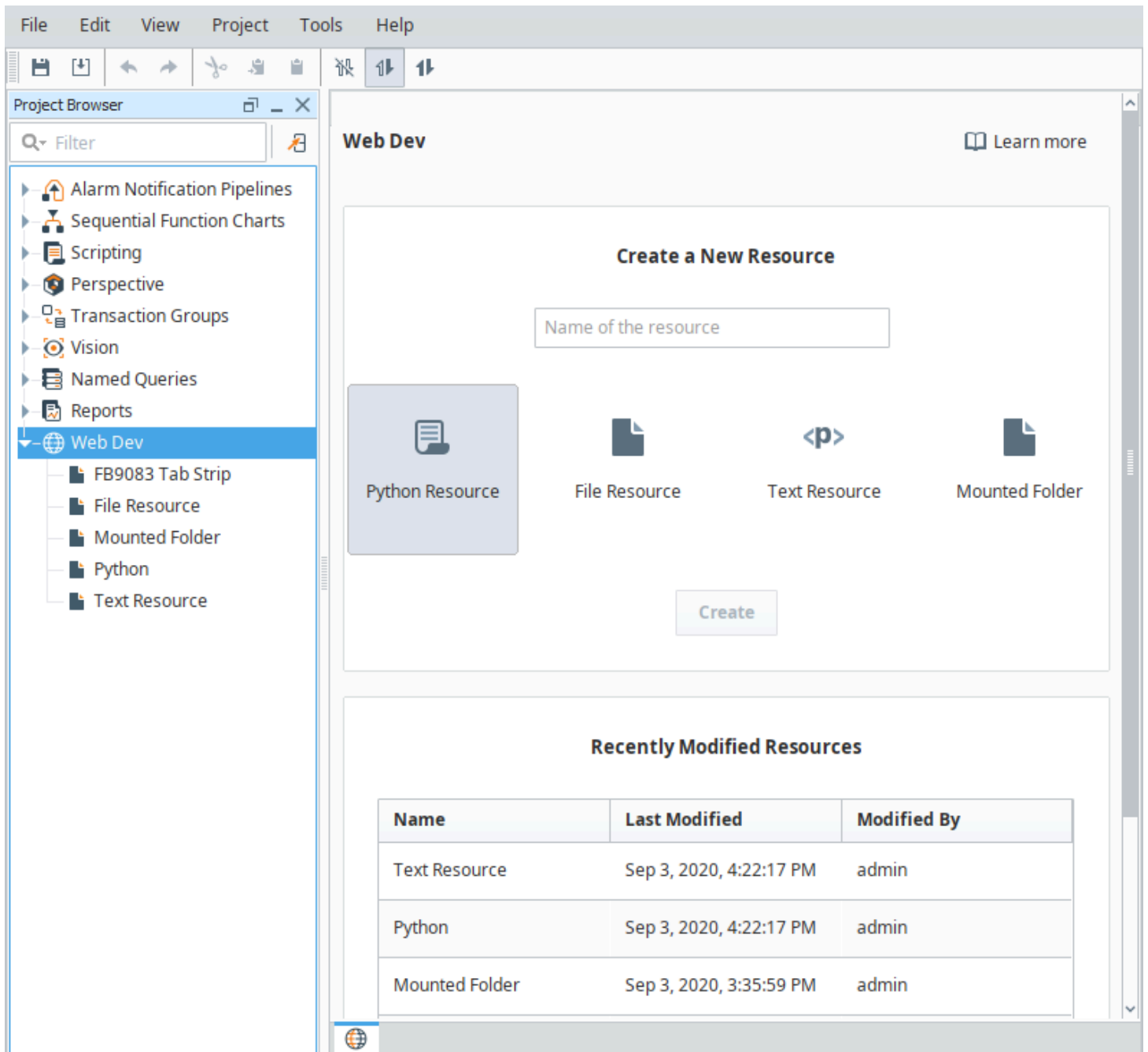
6. In Perspective, you can resize the image and change the style settings. See [Images and Icons in Perspective](#).
7. In Vision, the symbol will become a group of shapes. See [Images and SVGs in Vision](#).

Web Dev

The **Web Dev module** enables you to directly program against the web server inside the Ignition Gateway. Webpages can be built by hand using a combination of Python programming and static web resources such as images, CSS files, JavaScript files, and HTML files. Likewise, this module allows you to build RESTful web service APIs that allow external systems to interact with the Ignition server. This module follows the normal installation process.

The Web Dev Welcome tab allows you to create your program using any of the four types of resources: Python, File, Text and Mounted Folder. Each one of the resource types is a template to help you get started creating your own program. Once you select a resource type, enter a name, and press 'create', and the specific resource template will open. Each resource type will help you get started. The Web Dev Welcome tab will show you any recently modified resources along with the date it was modified and who modified it. You can double click on a recently modified query to easily open and update it.

The Web Dev Welcome tab provides a quick way to create a new resource and update existing ones.

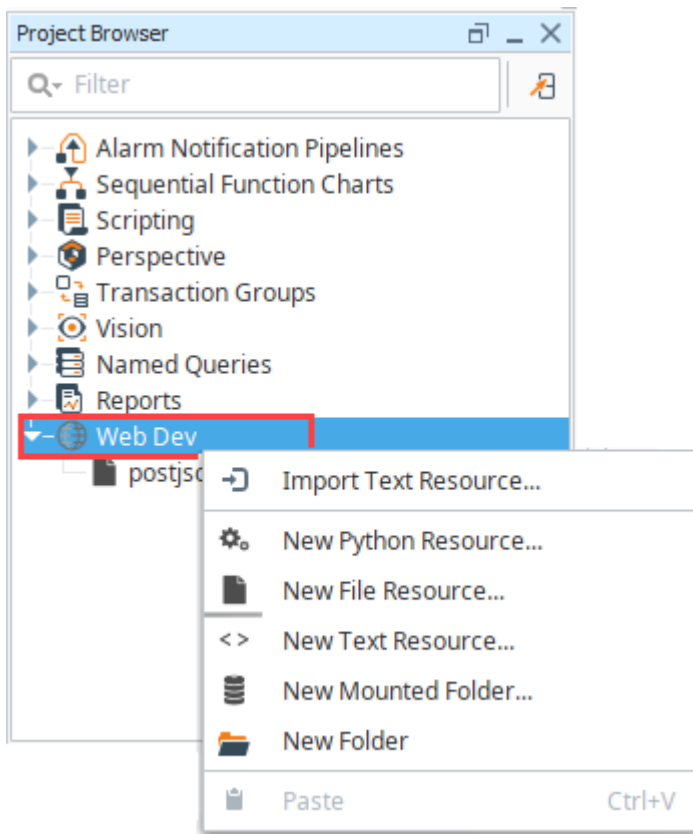


CAUTION

The Web Dev module requires specialized web-programming knowledge. The Inductive Automation support team is unable to provide detailed advice about creating a particular site. Furthermore, they are unable to provide troubleshooting beyond the basic functionality of the module.

Resources

Each type of resource may specify its content type. It is important to specify the correct content type for the contents of the resource. When the Web Dev module is installed, a new kind of project resource heading will appear in the Designer's project browser called "Web Dev". Right-clicking on this heading will allow the creation of several new types of project resources:



Web Dev resources are stored as readable `.json` and `.py` files on the Gateway's file system.

- **Python Resource** - Python resources are dynamic web resources. Each time a user browses to the URL associated with a Python resource, the script will run and generate a response in the form of a Python dictionary. See [Return Value](#) for more details on formatting this response.
- **File Resource** - A file resource is a static resource, usually a binary resource such as an image.

i NOTE

You will need to re-import a file resource if it has been changed since adding it to Web Dev.

- **Text Resource** - A text resource is a static resource much like a file resource, except that its contents may be directly edited from within the Ignition Designer. These are useful for static HTML, CSS, and JavaScript files.

Mounted Folder

Mounted Folders are a way to expose a folder from the Gateway's hard drive as a resource endpoint. For example, if the Ignition server had some directories that looked like the following:

Pseudocode - Example Directories

```
/opt/public/a.jpg  
/opt/public/info.html
```

You can create a Mounted Folder named "assets", and point it to `/opt/public/`, and then you can access those assets via:

Pseudocode - URL to Mounted Folder

```
http://host:port/system/webdev/projectname/assets/a.jpg
```

ⓘ REGARDING FILENAMES

Be mindful of the filenames in the mounted folder, as URL encoding will need to be used to access the file. For example, if a file named "My file.pdf" was placed in the mounted folder above, the file name would likely need to be encoded:

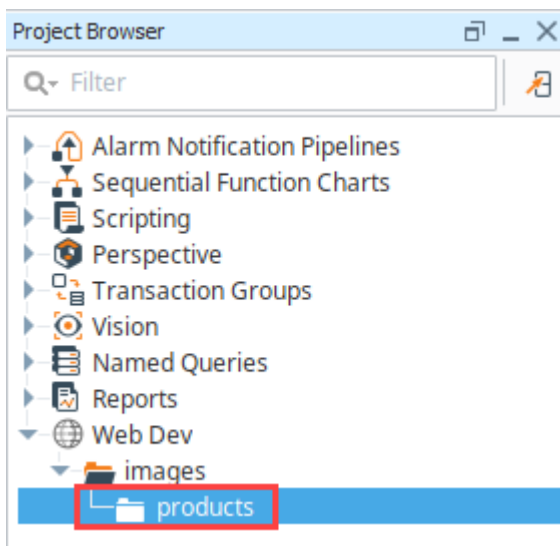
```
http://host:port/system/webdev/projectname/assets/My%20file.pdf
```

In addition, it is recommended to avoid using the following characters within filenames in mounted folders as they can result in errors:

- ' (apostrophes)
- , (commas)
- /
- ?
- #
- %
- &

Project Resources Folders

If a Mounted Folder is placed in a Project Browser folder, then the endpoint must also include the folder name. For example, if the Mounted Folder named "products" is located in the "images" folder:



The files would be accessible via:

Pseudocode - URL for Mounted Folder with Project Resource Folder

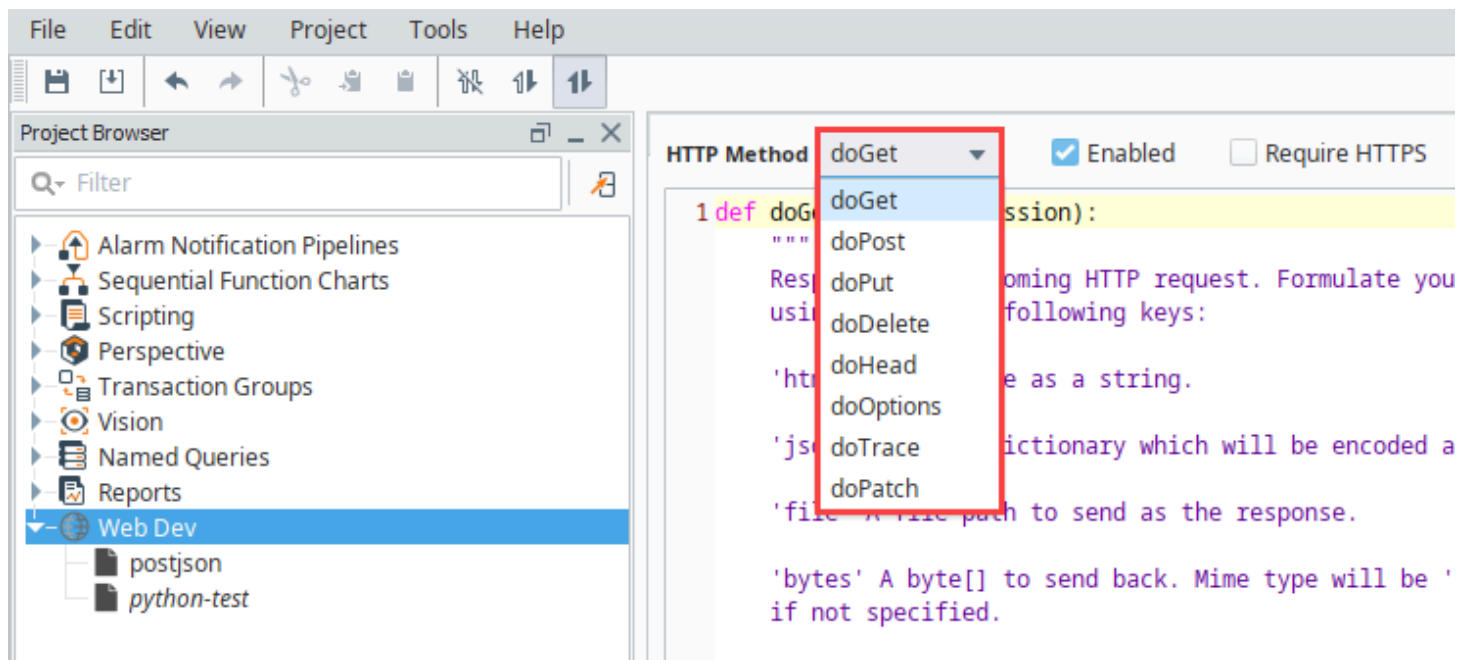
```
http://host:port/system/webdev/projectname/images/products/a.jpg
```

Python Resources

Python resources are the heart of the functionality of the Web Dev module. These resources are completely dynamic, and can handle all parts of the HTTP protocol, formulating any type of response.

Each time an incoming HTTP request arrives at a URL where a Python resource is mounted, the corresponding Python script will be run. Each Python resource can have a script for each HTTP method. In practice, most Python resources will probably only implement one or two of these, usually `doGet` or `doPost` at a minimum. The available methods are as follows:

- `doGet`
- `doPost`
- `doPut`
- `doDelete`
- `doHead`
- `doOptions`
- `doTrace`
- `doPatch`



Return Value

Return values for each do* functions can generate a response, which should always be a dictionary. In the dictionary, the keys in the table below are recognized. The keys are listed here in the order they are evaluated. For example, if you have both file and bytes, only file will take effect. The exception is the contentType key, which may be included with any of the other keys to override the default content type.

HTTP Response	Description
html	HTML source as a string. The value should be a string, which should be the source of an HTML document. Content type is assumed to be text/html.
json	The value is assumed to either be a string (which should be valid json) or to be a Python object, which will then be encoded into a json string. Content type will be application/json.
file	The value should be a string, which should be the path to a file on the server's filesystem. If no contentType is specified, then the system will attempt to probe the content type from the operating system using java.nio.Files.probeContentType. If the file key is present, but the value points to a file that doesn't exist, an HTTP 404 will be returned.
bytes	The value should be a byte array. The default content type is application/octet-stream, but you probably want to specify your own.
response	If none of the other keys are present, the system will look for the key response which will be stringified and then returned with the content type text/plain.
contentType	The mime type of the response. Needed only if ambiguous.

If your implementation of the `do*` function returns a dictionary with none of the above keys, an HTTP 500 error will be returned. However, if you return `None`, no HTTP 500 error will be returned. In this case, it is assumed that you used the `request['servletResponse']` parameter to directly formulate your HTTP response.

The default encoding for **html**, **json**, and **response** types is UTF-8.

Parameters

Each `do*` method receives the same two parameters (also called arguments): 'request' and 'session'.

Request Parameter

The request parameter is a dictionary with information about the incoming web request.

Key	Type	Description
context	object	A reference to the Gateway's context object. This provides direct to the Ignition GatewayContext. More information about this object can be found in the Ignition SDK Programmer's Guide and associated JavaDocs.
data	plain text or raw byte array	The data on the request. If the content type is application/json, it will be a Python structure (list or dictionary). If not, it will either be plain text or a raw byte array. If the incoming request body was not text, it will be available as a byte array.
postData	string	This parameter is only present for the doPost method, and its value is different based upon the value of the incoming HTTP request's Content-Type header. If the content type is 'application/json', then the <code>request['postData']</code> will be a Python dictionary structure which is the result of parsing the JSON document that was posted. If the content type starts with 'text/', then the value of <code>request['postData']</code> will be the text which was posted, as a string.
headers	dictionary	A dictionary of header : value pairs. If multiple values were returned for the same header, values will be in a tuple. The HTTP request headers will be in a dictionary under <code>request['headers']</code>. For example, you could read the User-Agent string with <code>request['headers']['User-Agent']</code>.
params	dictionary	This will be contained in a dictionary accessible via <code>request['params']</code>. Any URL parameters such as the following: Pseudocode - Request Parameter <pre>/system/webdev/project/foo?param1=value&param2=other_value</pre> For the given example, <code>request['params'] = {'param1': 'value', 'param2': 'other_value'}</code>.
remainingPath	string	<code>request['remainingPath']</code> will be <code>/bar</code>. Remaining path will be the path after the resource name if nothing is found after the resource name. This provides the remaining path.

Key	Type	Description
		text after a file resource. If you have a resource called <code>foo</code>, and a request is made to: Pseudocode - Remaining Path <code>/system/webdev/project/foo/bar</code>
remoteAddr	string	Returns the IP address of the client. Gives the remote IP address of the entity that made the web request. Note: This is from the perspective of the web server, and so may not be what you expect due to the effects of things like NAT-ing routers.
remoteHost	string	Returns the fully qualified name of the client. Gives the remote host of the entity that made the web request. Note: This is from the perspective of the web server, and so may not be what you expect due to the effects of things like NAT-ing routers.
scheme	string	The scheme is available via <code>request['scheme']</code>. The value will be <code>http</code> or <code>https</code>.
servletRequest	object	The underlying Java HttpServletRequest object. This parameter and the following servletResponse parameter give you direct access to the underlying Java servlet request and response objects. This provides direct access to the Ignition GatewayContext. More information about this object may be found in the Ignition SDK developer guide and associated JavaDocs.
servletResponse	object	The underlying Java HttpServletResponse object.

Session Parameter

The session parameter is a Python dictionary which may be used to hold information which persists across multiple different calls to your Web Dev Python Resource, or across multiple Python Resources. Session management is handled automatically, and this dictionary is created for each new client session. (The client must have cookies enabled for sessions to work). You may place any key-value pairs in the session dictionary you'd like, just make sure that the values are things that can be serialized. All normal Python types (scalar, dictionary, lists, and tuples) are serializable.

If authentication is required, will have a 'user' attribute containing information about the authenticated user, and a 'retryAttempts' attribute with the number of attempts made.

URL

Each resource will be directly accessible over HTTP and mounted beneath the /system/webdev path.

For example, if you created a Text Resource directly beneath the "Web Dev", it would be mounted at:

Pseudocode - Project Resource

```
http://host:port/system/webdev/project/resource_name
```

Notice that the project name and resource name are part of the path. If your resource is nested inside a folder, it will be part of the path too. For example:

Pseudocode - Folder Resource

```
http://host:port/system/webdev/project/folder_name/resource_name
```

Web Dev resources may have periods in their name. This means that if you upload an image file, you may include its extension directly in its name so that its path is more natural. For example, you might name an image resource "my_image.png" so that its URL is:

Pseudocode - Image Resource

```
http://host:port/system/webdev/project/my_image.png
```

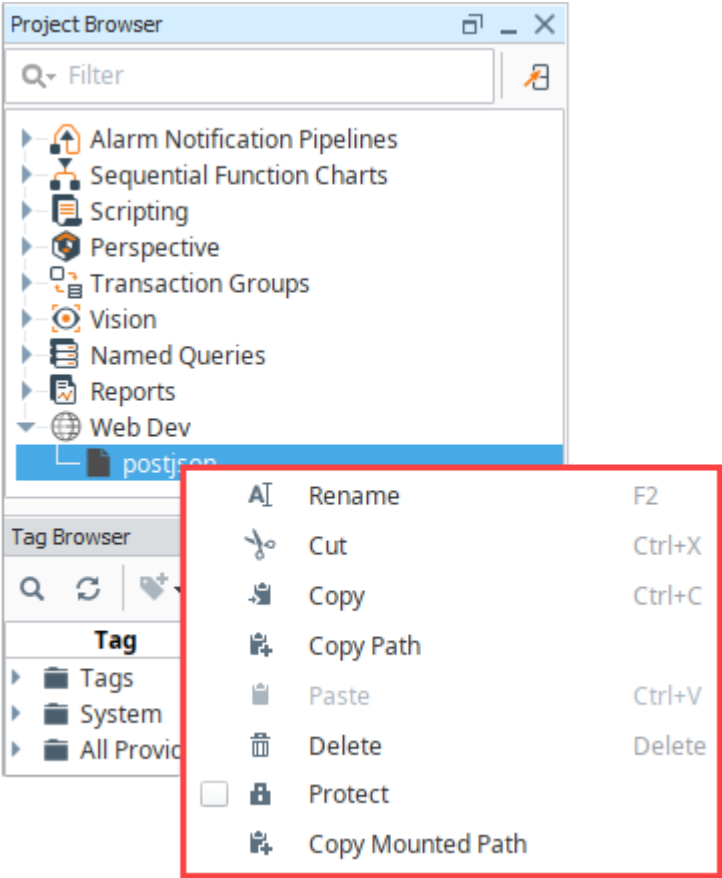
Requests to the root of your project will attempt to load a resource named "index.html". If no such resource exists, a 404 response code will be returned instead.

Pseudocode - Request to Root Project

```
http://host:port/system/project
```

Right Click Menu

The Web Dev Right Click menu is similar to other applications edit menus in that it provides basic copy/paste functionality. Options are described in the following table.



Option	Description
Rename	To rename a resource, select this option then enter a new name.
Cut	Cuts the selected resource onto the clipboard.
Copy	Copies the selected resource onto the clipboard. It can then be pasted within the Web Dev make a duplicate.
Copy Path	Copies the path of the selected resource onto the clipboard. For example, "newfile.html".
Paste	Pastes the the selected resource from the clipboard into the selected context.
Delete	Deletes the current selection. This can also be done using the delete key.
Protect	Locks the individual project resource from inside the Designer.
Copy Mounted Path	This copies the partial mounted path for the resource into your clipboard. This allows you to easily paste the path into your browser for testing. For example, "/system/webdev/MyProject/newfile.html". You need to add the path to your gateway to the beginning of this string. A full url would look like: " http://10.10.10.150:8088/system/webdev/MyProject/newfile.html "

Security Settings

There are several security settings for Python Resources. All of the settings can be set individually on each resource.

Enabled

If **Enabled** is checked, then the HTTP Method is enabled.

Require HTTPS

If this is checked, then the resource will only be accessible via an SSL connection. If a non-secure HTTP transport is used, the browser will be sent a redirect to the the Gateway's SSL port. The Gateway must have SSL enabled, of course.

Require Authentication

If this is checked, the resource will require authentication before it executes. This uses HTTP BASIC auth, and so should really be combined with the Require HTTPS option so that the credentials are encrypted. The username/password combination sent through the HTTP BASIC authentication headers will then be passed through the chosen User Source. If roles are specified, the user must have at least one of the roles. Specify multiple acceptable roles using a comma separated list. If the credentials are missing, an HTTP 401 will be returned with the WWW-Authenticate header. If the credentials are present but incorrect, an HTTP 403 will be returned.

If the credentials succeed, the Python resource will execute. In addition, the authenticated user object returned by the User Source will be accessible inside the session object as `session['user']`. Since the user is stored in session, if the client has cookies enabled, then further requests against the same session will use the stored user object and will not require additional authentication.

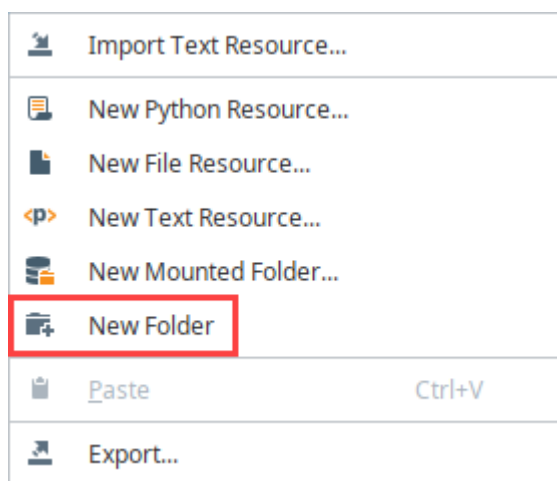
Building a Simple Web Server

This page demonstrates how to build and test a simple web server using the [Web Dev module](#). You'll create a small test environment that responds to basic HTTP `GET` and `POST` requests. The examples will use Button components in Perspective to send requests to Python Resources in Web Dev, but any external client can be used to communicate with the same endpoints.

Set Up the Web Dev Environment

The Web Dev environment used in the examples below requires a folder to store your resources.

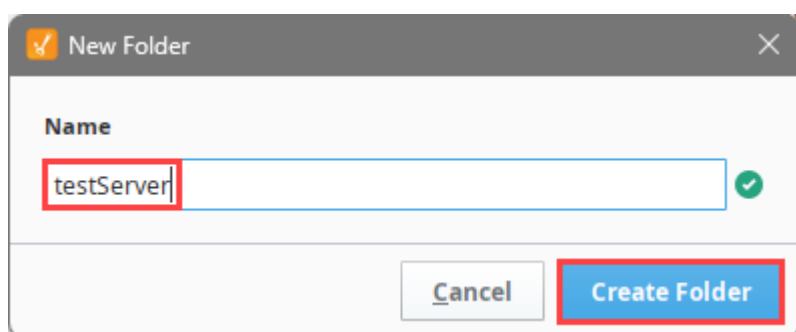
1. In the Designer, right-click **Web Dev** in the Project Browser, then select **New Folder**.



2. Name the folder **testServer**, then click **Create Folder**.

The folder name becomes part of the URL path for your resources. For example:

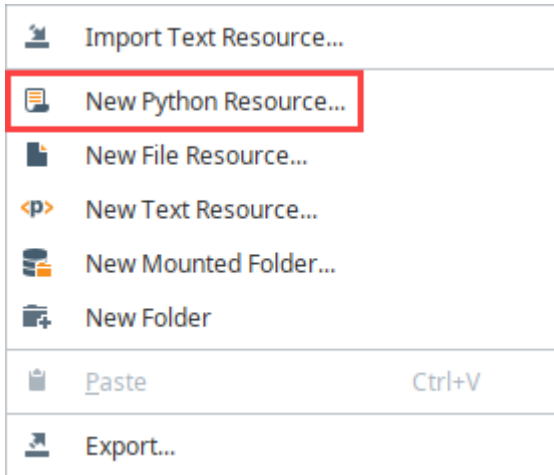
```
http://localhost:8088/system/webdev/MyProject/{testServer}/resourceName
```



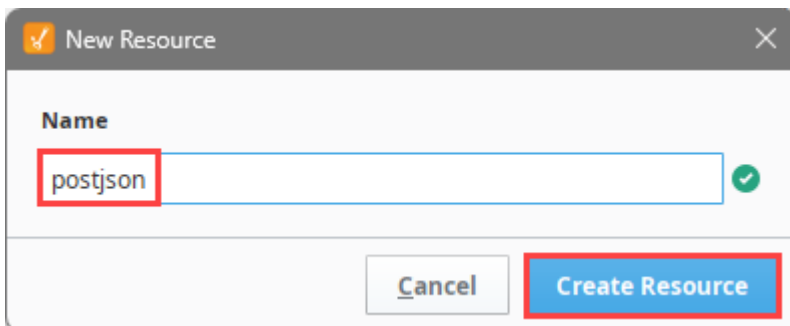
HTTP Post Example

This example demonstrates how Ignition can receive data from an external source by using a Button component in Perspective to send JSON data through an `httpPost` command to a Python Resource in Web Dev, which then receives and processes the POST request.

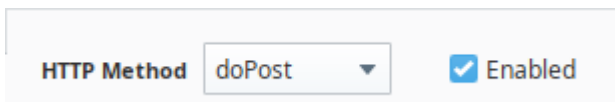
1. Right-click the **testServer** folder, then select **New Python Resource**.



2. Name the resource **postjson** and click **Create Resource**.



3. From the HTTP Method dropdown, select **doPost**, and check **Enabled**.



4. Paste the following code to accept JSON data, print it to the Gateway logs, and return a simple HTML response.

```
postjson Python Resource (Web Dev Section)
```

```

# Accept JSON data from a POST request.
# Expecting 'names' and 'values' lists of equal length.

data = request['data']
names = data['names']
values = data['values']

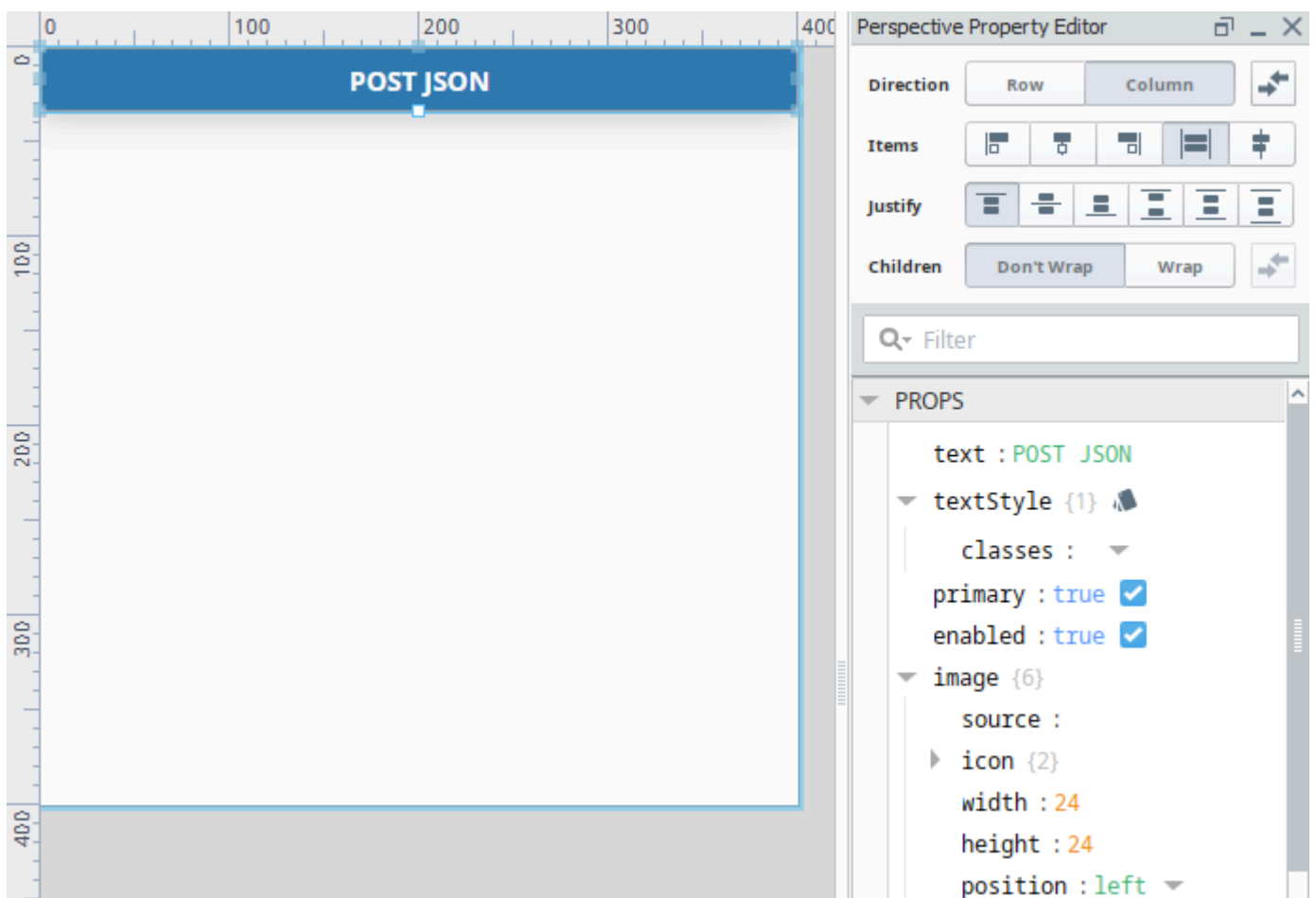
system.util.getLogger("WebDev").infoof("Names: %s | Values: %s", names, values)

html = "<html><body>"
for i in range(len(names)):
    html += "%s: %s<br>" % (names[i], values[i])
html += "</body></html>"

return {'html': html}

```

5. Open a Perspective view and add a **Button** component. In this example, we named the button **POST JSON**.



6. With the Button selected, right-click and select **Configure Events**.

	Cut	Ctrl+X
	Copy	Ctrl+C
	Duplicate	Ctrl+D
	Paste	Ctrl+V
	Paste & Replace	Ctrl+Shift+V
	Delete	Delete
	Send to Back	
	Move Backward	
	Move Forward	
	Bring to Front	
	Wrap in Container	▶
	Configure Events...	Ctrl+J
	Configure Scripts...	Ctrl+K
	Configuration Explorer	Ctrl+R

7. Under Mouse Events, select **onClick**, then add a **Script** action. Paste the following code to send a POST request to your new endpoint.

```
# Post data to the web service in JSON format.
# This allows you to use the 'data' object in the Python Resource.

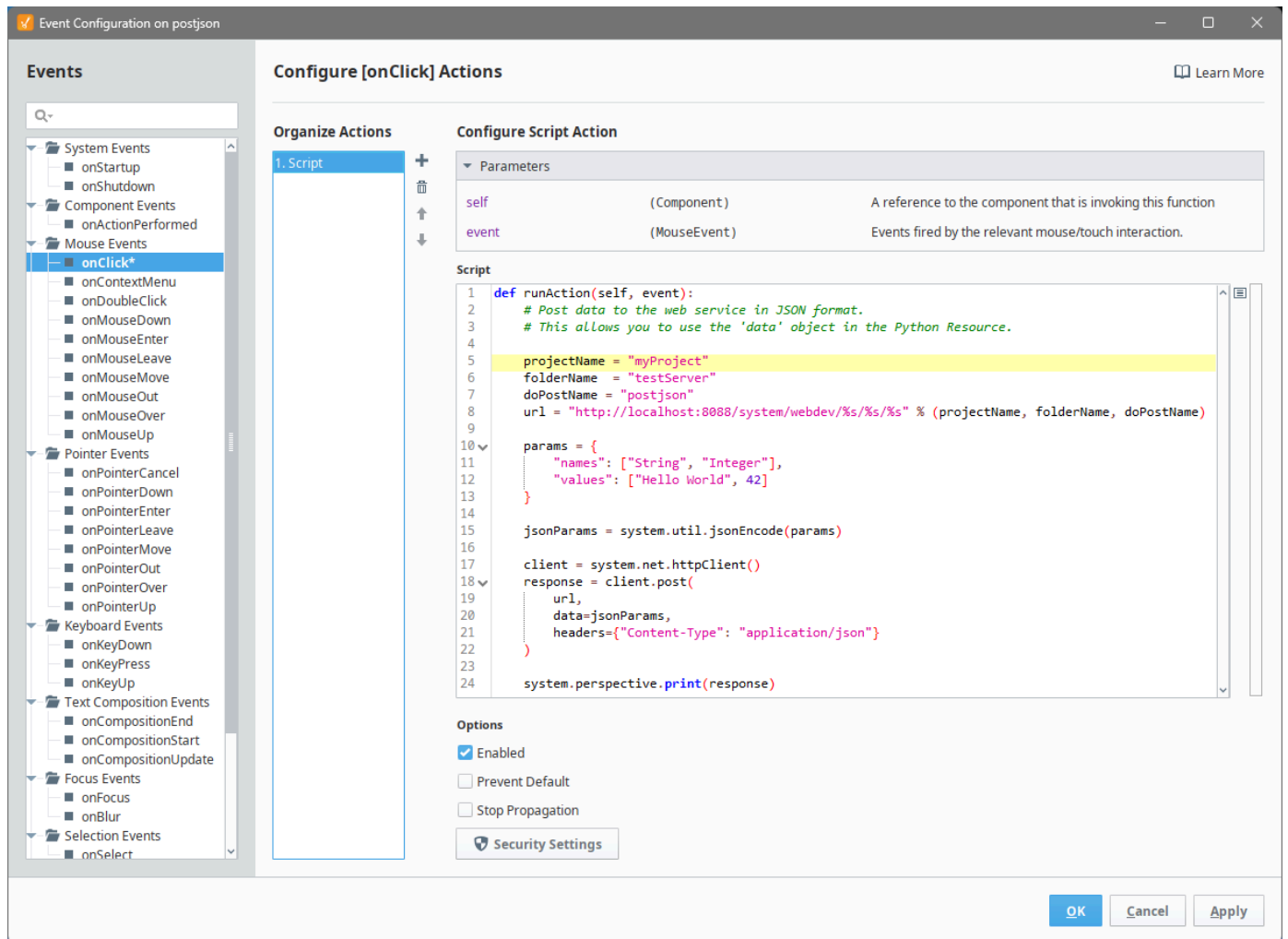
projectName = "MyProject"
folderName = "testServer"
doPostName = "postjson"
url = "http://localhost:8088/system/webdev/%s/%s/%s" % (projectName,
folderName, doPostName)

params = {
    "names": ["String", "Integer"],
    "values": ["Hello World", 42]
}

jsonParams = system.util.jsonEncode(params)

client = system.net.httpClient()
response = client.post(
    url,
    data=jsonParams,
    headers={"Content-Type": "application/json"}
)

system.perspective.print(response)
```



8. Save the project.

9. Open the Perspective Session and click the Button to send the POST request. The printed response appears in the Session Console, and the Gateway logs show the formatted data.

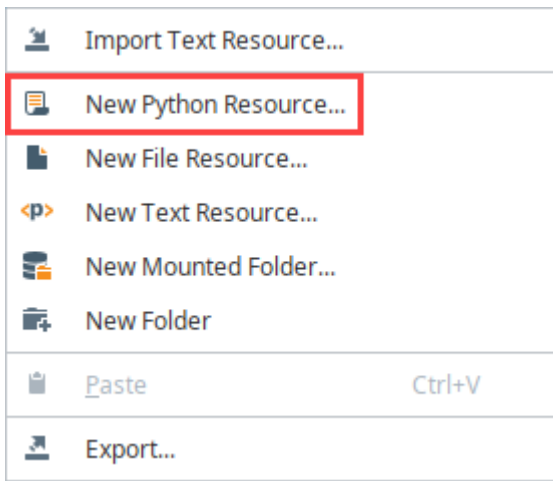
The Gateway can now receive HTTP POST requests at:

`http://localhost:8088/system/webdev/MyProject/testServer/postjson`

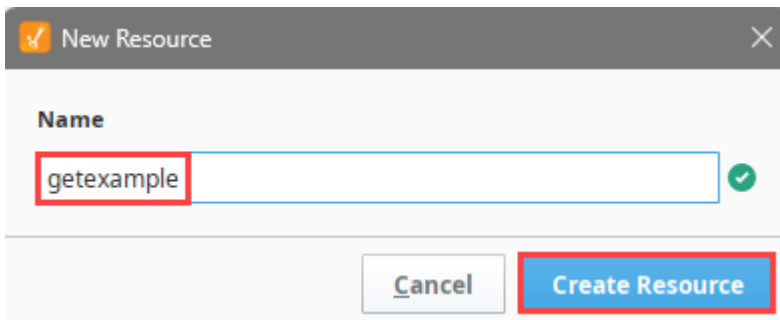
HTTP Get Example

This example shows how to respond to a simple HTTP GET request using Web Dev by creating a Python Resource that returns an HTML message, which can be viewed directly in a browser or triggered from a Perspective Event Script.

1. Right-click the **testServer** folder, then select **New Python Resource**.



2. Name the resource **getexample**, then click **Create Resource**.



3. From the HTTP Method dropdown, select **doGet**, and check **Enabled**.



4. Paste the following code to return a simple HTML message confirming that the Gateway received your GET request.

```
getexample Python Resource (Web Dev Section)

# Return a simple HTML response to confirm a GET request.

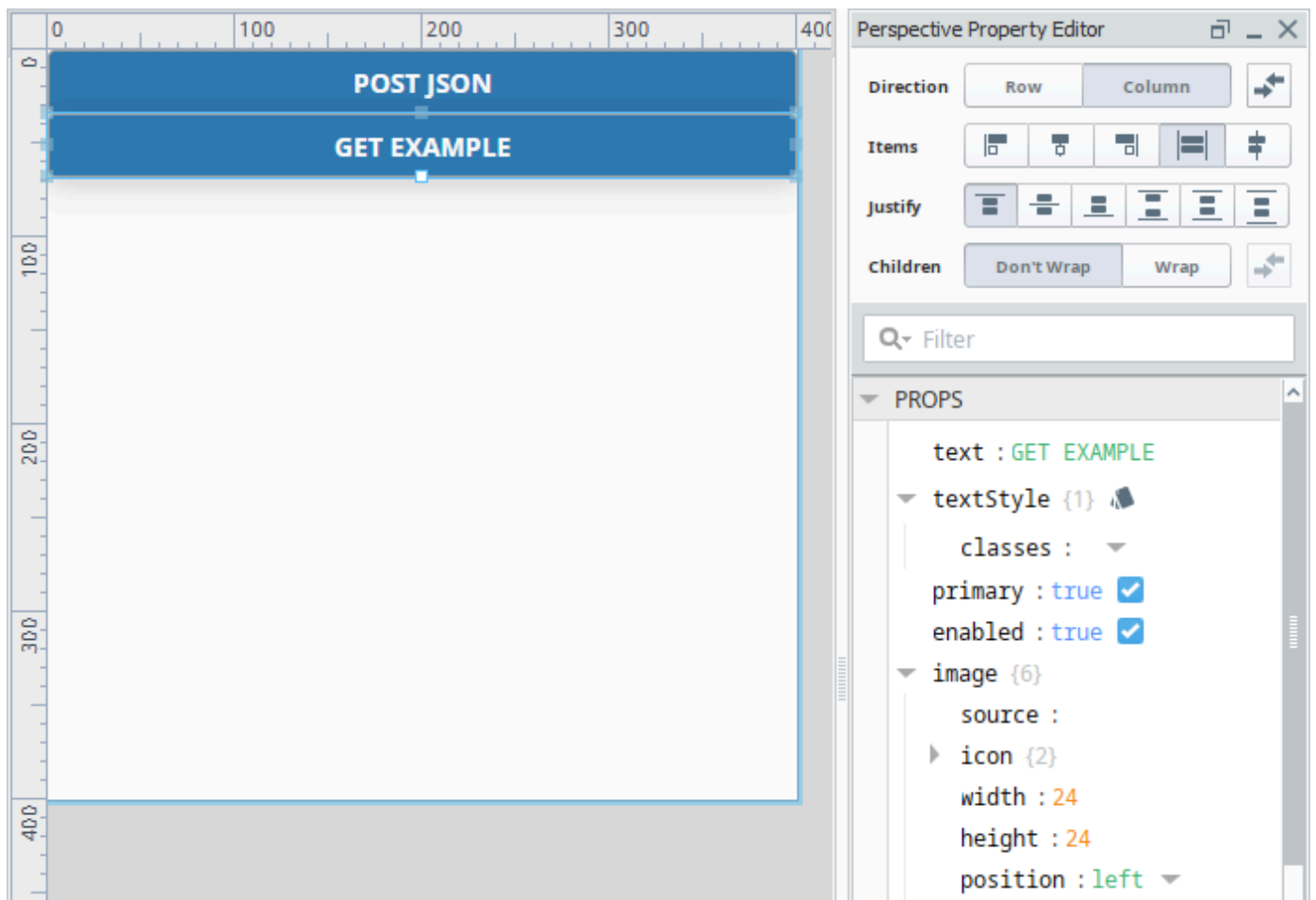
html = "<html><body><h2>Hello from Ignition Web Dev!</h2></body></html>"
return {'html': html}
```

5. Save the resource, then open a web browser and navigate to:

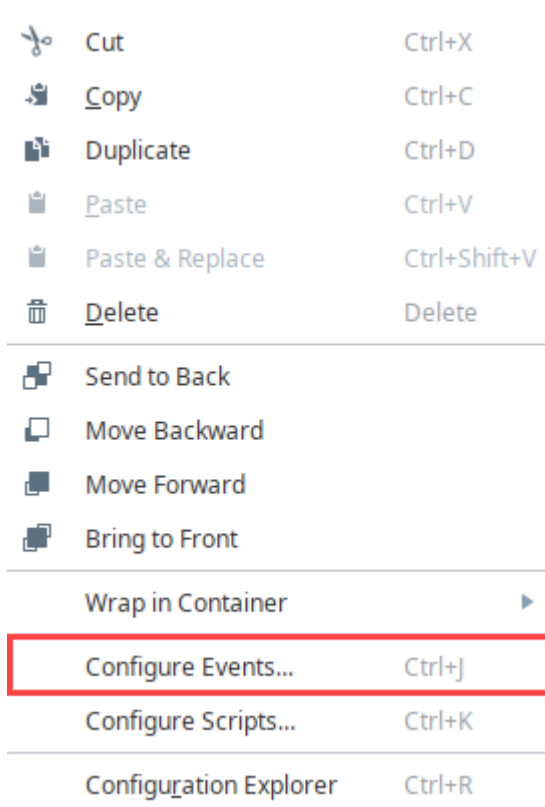
`http://localhost:8088/system/webdev/MyProject/testServer/getexample`

The HTML message appears in the browser window.

6. To test the endpoint, open a Perspective view and add a Button component. In this example, we named the Button **GET EXAMPLE**.



7. With the Button selected, right-click and select **Configure Events**.



8. Under Mouse Events, select **onClick**, then add a **Script** action. Paste the following code to send a GET request to your endpoint and print the response text in the Session Console.

```
projectName = "MyProject"
folderName = "testServer"
doGetName = "getexample"
url = "http://localhost:8088/system/webdev/%s/%s/%s" % (projectName,
folderName, doGetName)

client = system.net.httpClient()
response = client.get(url)
system.perspective.print(response)
```

9. Save the project.

10. Open the Perspective Session and click the Button. The Gateway's response appears in the Session Console, confirming that the Web Dev resource successfully handles HTTP GET requests.

The Gateway can now respond to HTTP GET requests at:

`http://localhost:8088/system/webdev/myProject/testServer/getexample`

Cloud Connector Modules

Cloud Connector modules offer extended functionality to **Ignition Cloud Edition** and standard **Ignition**. Similar to regular Ignition modules such as **Perspective** or **SQL Bridge**, Cloud Connector modules can be installed to new or configured instances of Ignition, and will integrate seamlessly with other Cloud Edition modules and existing **Ignition modules**.

Cloud Connectors

Cloud Connectors come bundled with Ignition Cloud Edition and are by design, intended for use in cloud environments. However, standard Ignition can still utilize the features of Cloud Connectors. Each Cloud Connector has a primary purpose, such as connecting to NoSQL databases using the MongoDB connector.

Cloud Connector Module	Description
MongoDB	Configure connections from Ignition to MongoDB databases. Update, insert, delete, and take aggregates of documents or list collections and connector information using included system functions . Pull information from databases to display on Perspective components using MongoDB Perspective bindings .
Kafka	Configure Kafka connections from the Ignition Gateway. Produce or consume events using event streams and included system functions .

MongoDB

MongoDB is a non-relational, NoSQL database. Unlike relational SQL databases like Microsoft SQL Server, MongoDB uses a JSON-like structure called collections instead of tables, with each JSON "row" referred to as a document. This in turn leads to lighter queries, helping optimize the performance of your Gateway and database connection.

Aside from a local installation, MongoDB can also be hosted in the cloud using the cloud service MongoDB Atlas. There are specific features which may draw users toward MongoDB Atlas, such as not needing to maintain a server or configure any clusters.

See the [official MongoDB documentation](#) for more information on MongoDB and MongoDB Atlas.

Module Information

The MongoDB Cloud Connector module comes with **Ignition Cloud Edition**, but can also be used with a standard version of Ignition. Features of the MongoDB Cloud Connector include:

- **Connectivity to MongoDB** through the Gateway interface.
- **System functions** to interact with connected MongoDB documents using a Python scripting API and expression language functions.
- A **Perspective Binding** that reads information from MongoDB documents to display on a component.

Gateway Interface

After installing the MongoDB Cloud Connector, there will be new MongoDB options available on the Gateway.

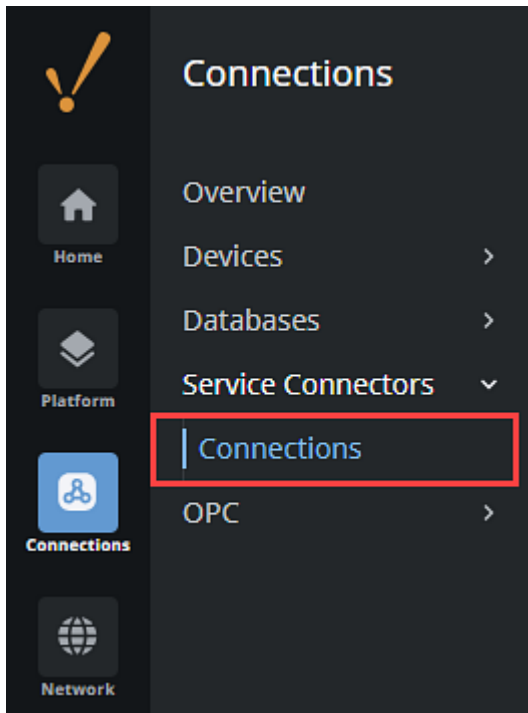
Connections - MongoDB

The Gateway Webpage's **Connections > Service Connectors > Connections** page will show information about your MongoDB connections. You can filter by name to search for a specific MongoDB connection, and change how many connections are displayed on this page.



The MongoDB option will only show up if the MongoDB Cloud Connector module is installed.

The page also allows you to make connections from Ignition to MongoDB databases.



The table below outlines the properties available when creating or editing a MongoDB connection:

General

Property	Description
Name	The name of your MongoDB connection.
Description	A description of your MongoDB connection.
Enabled	Enable or disable the MongoDB connection. Default is enabled.

Connection

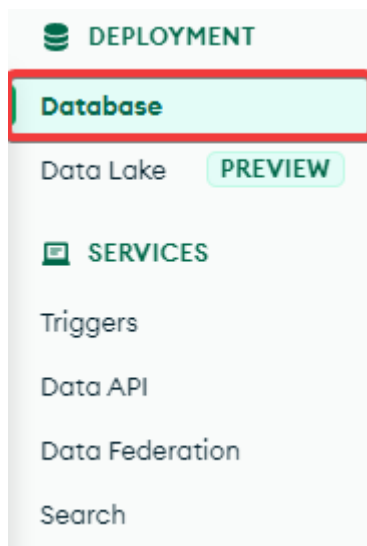
Property	Description
Connection Scheme	Prefix that identifies the connection format. Default is mongodb. Note: If you are creating a connection to MongoDB Atlas, you will need to change the Connection Scheme to <code>mongodb+srv</code> .
Connection Hosts	Host or list of hosts to connect to, comma separated. If no port is specified, the default port of 27017 is used.
Database	The name of the database to connect to.
Username	The username to use when connecting.
Password	The password to use when connecting. You can select between None, Embedded, or Referenced.
Connection Properties	Any extra connection properties you need to specify, such as enabling or disabling TLS. • If you are using <code>mongodb+srv</code> as the Connection Scheme to connect to MongoDB Atlas, TLS will be enabled by default. • If you are using <code>mongodb</code> as the Connection Scheme to connect to MongoDB, TLS will be disabled by default. Note: If you are using a trusted certificate, Ignition will automatically use the certificate without needing any configuration. If you are using a custom or self-signed certificate, you will need to specify the location of the certificate, as well as any other identifying information such as the password.

Connecting to MongoDB

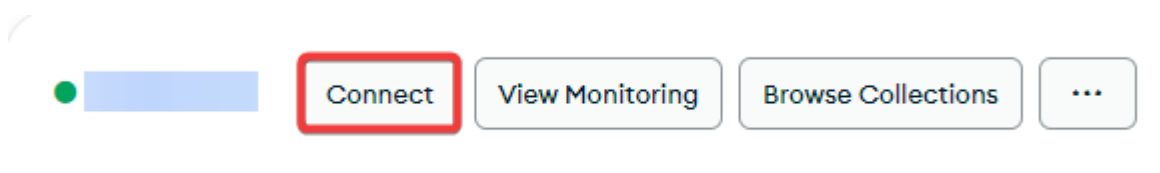
With the MongoDB Cloud Connector installed, you will be able to connect to a MongoDB database. The steps below outline how to connect to MongoDB. In this example, we will be connecting to MongoDB Atlas.

Get the Connection URL

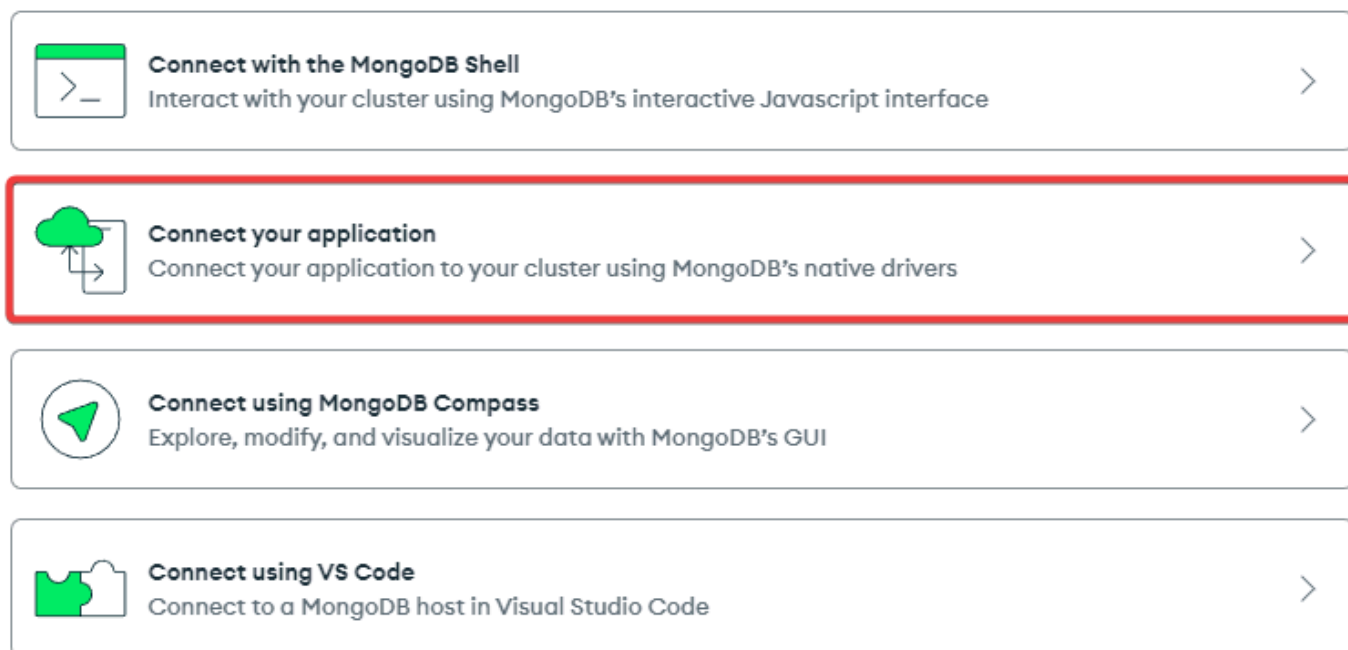
1. Go to your MongoDB Atlas Dashboard and click **Database** in the **Deployment** section.



2. On the Database Deployments window, click **Connect**.



3. You will be prompted to choose a connection method. Click **Connect your application**.



4. Choose **Java** for the driver, and select the latest version. Your connection URL will appear, giving you the information you need to connect Ignition to your MongoDB database.

1 Select your driver and version

DRIVER	VERSION
Java	4.3 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://<username>:<password>@<clustername>/?  
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **<username>** user. Ensure any option params are [URL encoded](#).

Interpreting the Connection URL

The connection URL will have all the information necessary to set up a basic connection. Note that you will still need to specify extra connection properties such as Transport Layer Security (TLS) or certificates should you need them.

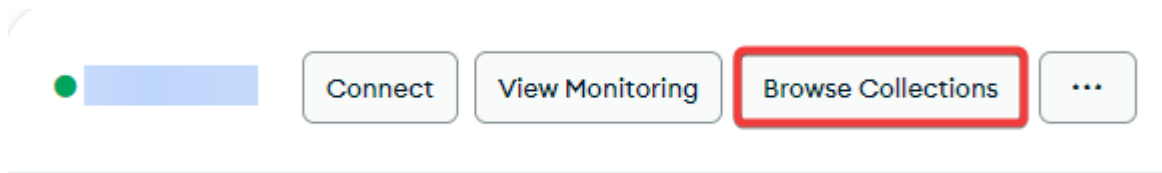
Your connection URL will look something like the example below, with specific sections of the connection URL corresponding with specific Gateway fields from the MongoDB connection setup process.

```
mongodb+srv://<username>:<password>@<clustername>.example.mongodb.net/?  
retryWrites=true&w=majority
```

Gateway Field	Connection URL Value
Connection Scheme	mongodb+srv (or mongodb if not using MongoDB Atlas)
Connection Hosts	<clustername>.example.mongodb.net
Username	<username>
Password	<password>
Connection Properties	retryWrites=true&w=majority (You can also add extra connection properties here)

Get the Database Name

1. On the MongoDB Atlas Dashboard, navigate to **Deployment > Database > Your Cluster Name > Browse Collections**.



2. Your dashboard will list any existing databases within your cluster. Choose which database to connect to from this list. In this example, we will be using the **sample_guides** database.

DATABASES: 9 COLLECTIONS: 22

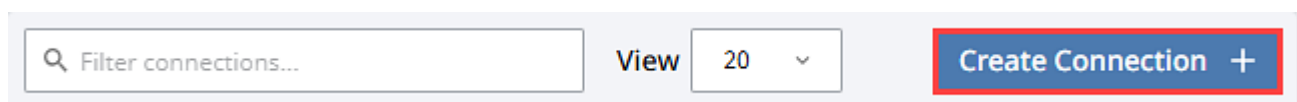
+ Create Database

Q Search Namespaces

- ▶ sample_airbnb
- ▶ sample_analytics
- ▶ sample_geospatial
- ▶ **sample_guides**
- ▶ sample_mflix
- ▶ sample_restaurants
- ▶ sample_supplies
- ▶ sample_training
- ▶ sample_weatherdata

Connect Ignition to MongoDB

1. Navigate to **Connections > Service Connectors > Connections** on the Gateway, and click **Create Connection +**.



2. Select **MongoDB** and click **Next**.

Create Services Connection

Create New Service Connection

1. Connection Type

2. Setup Service

☐ Kafka

The apache Kafka event streaming platform.

☒ MongoDB

Connect to Mongo DB services.

☐ MQTT Engine

Create an MQTT Connection

☐ MQTT Transmission Server

Create an MQTT Transmission Server

Cancel

Next

3. Fill out the Gateway fields with the appropriate information from your connection URL and the database name, then click **Create Connection**.

Create Services Connection

Create New Service Connection

1. Connection Type

2. Setup Service

GENERAL

*Required

Name *

MongoDB

The name of your {itemType} connection.

Description

Enabled

☒ Enable or disable the database connection

CONNECTION

Connection Scheme

mongodb

Prefix that identifies the connection format

Connection Hosts

clustername.example.mongodb.net, localhost, example.net

Back

Create Connection

Your MongoDB connection is now created. It may take a few seconds for your connection to update to a Valid status.

MongoDB Perspective Bindings

Users can pull data from a MongoDB database to populate [Perspective components](#). MongoDB Perspective bindings are read-only, so users will not be able to write to collections from the binding itself. See the [MongoDB Bindings in Perspective](#) page for more details.

MongoDB System Functions

While the MongoDB Perspective binding itself has no write capabilities, the included MongoDB System Functions allow users to read, update, delete, and take aggregates of collection documents. See the [MongoDB System Function](#) page for a list of available system functions.

Kafka

Kafka is a distributed event streaming system that allows you to process data in real time. The Kafka platform works by having **producers** create and publish events (also called records or messages) to a **topic**. Topics can be broken down into partitions, both of which are inside a server called a broker.

Events within topics are consumed by **consumers**, which use an offset to track which events have already been processed. Consumers can be further organized into **consumer groups**, which consume records from a set of topics.

Within a consumer group, a single partition can only be consumed by one consumer. However, multiple consumers can consume the same partition if the consumers are from different groups.

Unlike many other services, Kafka operates on a pull-based system, making it much more scalable as Kafka consumers can pull and consume topic data at varying rates. Additionally, using a pull-based system allows consumers to receive data in more optimized batches, as the consumer itself will pull all available messages that have not yet been processed. This helps eliminate any latency or buffering issues that stem from a push system.

See the [official Kafka documentation](#) for more information on Kafka.

Module Information

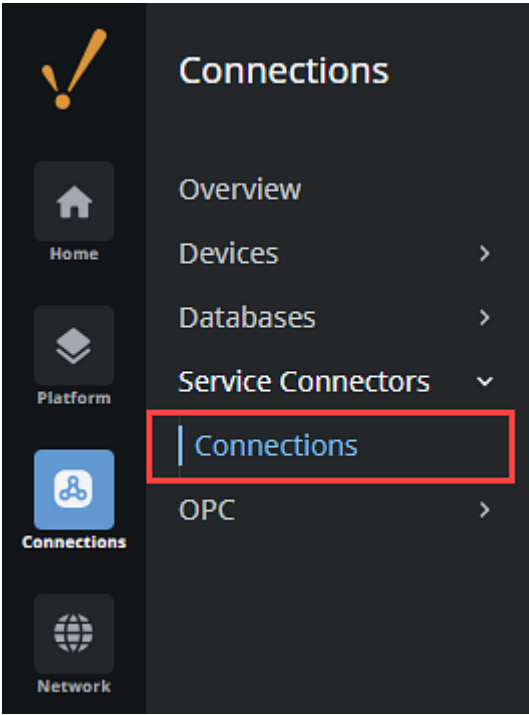
The Kafka Cloud Connector comes bundled with [Ignition Cloud Edition](#), but can also be used with a standard version of Ignition. It is not available on Edge. Features of the Kafka Cloud Connector include:

- [Connectivity to Kafka systems](#) through the Gateway Webpage interface.
- Integration with [Ignition Event Streams](#) for data processing.
- [System Functions](#) to interact with connected Kafka systems.

Gateway Interface

Installing the Kafka Cloud Connector module will result in a new configurable form on the Gateway that allows you to create Kafka connections.

These connections connect to a Kafka cluster, which can then be used for [event streams](#) or [Kafka system functions](#). To create a Kafka connection, navigate to the Gateway Webpage's **Connections** section > **Service Connectors** > **Connections**.



⚠ CONNECTING TO BROKERS

Although it's possible to connect to an individual broker, it is recommended to connect to multiple brokers for redundancy and to prevent data loss.

The tables below outline the properties available when creating or editing a Kafka connection.

Main Properties

Property	Description
Name	The name of your Kafka connection.
Description	A description of your Kafka connection.
Enabled	Enable or disable the Kafka connection. Default is enabled.
Bootstrap Servers	A list of host and ports (host:port) used for establishing a connection to the Kafka cluster, such as <code>localhost:9092</code> .
SASL Mechanism	The SASL mechanism to use for client connections. This can be any mechanism for which a security provider is available. Default is GSSAPI.
Security Protocol	The security protocol used when communicating with brokers. Default is PLAINTEXT. Options include: - PLAINTEXT: Un-authenticated, non-encrypted channel - SSL: SSL channel - SASL_PLAINTEXT: SASL authenticated, non-encrypted channel - SASL_SSL: SASL authenticated, SSL channel
Username	The username to use when connecting. This value will be injected into the <code>sasl.jaas.config</code> configuration property.
Password	The password to use when connecting. This value will be injected into the <code>sasl.jaas.config</code> configuration property. Options include None, Embedded, or Referenced.

Property	Description
Custom Client Properties	Any properties you want applied to producer and consumer clients, line separated by key value pairs. When entering multiple key value pairs, you must use the following format: <pre>key1=value1 key2=value2</pre> Note: If you are using a custom keystore, you will need to define the path to the custom keystore here using the property <code>ssl.truststore.location=<keystore_location></code>.

TLS/SSL Properties

Property	Description
Protocol	The specific SSL version to be used for the selected Security Protocol. If PLAINTEXT is selected, this setting is ignored.
Key Password	The password for the private key. Options include None, Embedded, or Referenced.
Keystore Type	The keystore type for how client and server certificates are stored.
Keystore Password	The password for the keystore. Select None if no custom keystore is selected.
Keystore Key	The password for the keystore key. Select None if no custom keystore is selected.
Truststore Password	The password for the truststore. Select None if no custom truststore exists.

Additional Properties

Property	Description
Custom Consumer Properties	A dictionary of properties applied only to consumers, as key value pairs. This property will override any duplicate keys from the Custom Client Properties. If entering multiple key value pairs, you must use the following format: <pre>key1=value1 key2=value2</pre>
Custom Producer Properties	A dictionary of properties applied only to producers, as key value pairs. This property will override any duplicate keys from the Custom Client Properties. If entering multiple key value pairs, you must use the following format: <pre>key1=value1 key2=value2</pre>

Connecting to a Kafka System

This section describes how to create a Kafka connection.

1. On the Gateway Webpage, navigate to **Connections > Service Connectors > Connections**.
2. Click **Create Connection +**.

Service Connectors

For information about setting up Service Connectors, see [this help page](#).

Status Open Metrics Dashboard

CONFIGURED CONNECTIONS

1

ERRORED CONNECTIONS

0

Connections

View 20 Create Connection

☐	NAME	DESCRIPTION	TYPE	HOST	STATUS	
☐	Connection1		Kafka Connector		CONNECTED	

Showing 1 of 1 items

3. Select **Kafka Connector**. Click **Next**.

Create Connection

Create New Service Connection

1. Connection Type

2. Setup Service

☒ **Kafka**
The apache Kafka event streaming platform.

☐ **MongoDB**
Connect to Mongo DB services.

☐ **MQTT Engine**
Create an MQTT Connection

☐ **MQTT Transmission Server**
Create an MQTT Transmission Server

Cancel

Next

4. Fill out the Gateway fields with the appropriate information from your desired Kafka broker, then click **Create Connection**.

Create Connection

Create Connection

1. Connection Type

2. Service Setup

GENERAL

*Required

Name *

Connection2

The name of your Kafka connection.

Description

Enabled

☒ Enable or disable the kafka connector connection.

Bootstrap Servers

Add Value +

HOSTS

No values.

Host or list of hosts to connect to, comma separated. If no port is specified, the default port of 27017 is used.

SASL Mechanism

GSSAPI

Back

Create Connection

Your Kafka connection is now created. It may take a few seconds for your connection to update to a Valid status.

Using Kafka with Ignition Event Streams

The main way to interact with Kafka within the Ignition platform is through event streams. Event streams can use Kafka as both a source for receiving messages and a handler for sending messages to a Kafka topic. When using the Kafka event stream source, you can configure properties such as how to receive messages, where the message should come from, any consumer properties you want to invoke, and more. Similarly, when using the Kafka event stream handler, you can configure properties such as the topic to publish the message to, headers you want to include, any producer properties you want to invoke, and more. See the [Event Streams](#) page for more details.

Kafka System Functions

Many of Kafka's features are used within an event stream as either sources or handlers. The Kafka system functions allow similar capabilities if you need to utilize specific features of an event stream, such as polling or sending records. See the [Kafka System Function](#) page for a list of available system functions.

JDBC Drivers

Bundled JDBC Driver Modules

The JDBC drivers shipped with Ignition are bundled inside modules. Currently, this includes the MariaDB, MSSQL, PostgreSQL JDBC drivers. It's important to note that these drivers will be listed on the Gateway **Modules** page, as well as the **Database Settings** page. Packaging these drivers as modules simplifies the upgrade process by eliminating the need to obtain more recent drivers yourself. Despite the differences with how driver configuration is handled, these module drivers retain the same functionality as manually installed drivers.

JAR files are considered part of the resource collection system, so they are stored in the `/data/config/resources/core/ignition/database-driver` folder within the Ignition directory.

NOTE

The SQLite driver is also shipped with Ignition, as a built-in driver, but does not have an corresponding module.

MariaDB JDBC Driver

The MariaDB JDBC Driver module provides the MariaDB driver version 3.3.3. MariaDB is a Type 4 JDBC driver, developed specifically as a lightweight JDBC connector for use with MariaDB and MySQL database servers. The MariaDB JDBC Driver module also provides the MySQL driver.

See the [JDBC Driver and Database Translator Settings](#) page for a list of the MariaDB and MySQL property settings.

MSSQL JDBC Driver

The MSSQL JDBC Driver module provides the Microsoft SQLServer JDBC driver version 9.4.0.jre11. MSSQL is ideal for enterprise-level applications and is capable of connecting to databases like Microsoft SQL Server Express and Microsoft Azure SQL Server.

See the [JDBC Driver and Database Translator Settings](#) page for a list of the MSSQL property settings.

PostgreSQL JDBC Driver

The PostgreSQL JDBC Driver module provides the PostgreSQL JDBC driver version 42.7.2 allowing Ignition to communicate and connect with a PostgreSQL database.

See the [JDBC Driver and Database Translator Settings](#) page for a list of the PostgreSQL property settings.

Configurations and Upgrades

Correct driver versioning is essential when attempting to connect to databases and maintain database connections. Since these JDBC drivers are packaged within modules, upgrading to the next version is as easy as upgrading Ignition or [upgrading your modules](#) from the **Platform > System > Modules** page. Additionally, if you remove the module, you remove the driver. If the module is disabled or faulted, the driver will be faulted.

If you are upgrading Ignition, but wish for your driver versions to remain as they are currently installed, simply select the **Keep as-is** option for each module on the Modules tab during the upgrade process.

Custom JDBC Driver Configuration

Depending on the database connection you are trying to establish, custom configurations may be required. These adjustments can be made by locating the driver on the **Connections > Databases > Settings** page of your Gateway. Click the three-dot menu icon for the driver and select **Edit** to begin making changes.

Database Settings

Drivers Translators

For information about installing driver files that are not able to be bundled with Ignition, see [this help page](#).

JDBC Drivers

NAME	DRIVER TYPE	DEFAULT TRANSLATOR	VERSION	STATUS	
☐ MariaDB	MYSQL	MYSQL	1	✓ INSTALLED	⋮
☐ Microsoft SQLServer	MSSQL	MSSQL	1	✓ INSTALLED	⋮
☐ MySQL	MYSQL	MYSQL	1	✓ INSTALLED	⋮
☐ PostgreSQL	POSTGRES	POSTGRES	1	✓ INSTALLED	⋮
☐ SQLite	SQLITE	SQLITE	1	✓ INSTALLED	⋮

Showing 5 of 5 items

Edit

Rename

Duplicate

Delete

Any changes made will override the module configurations as long as the driver name is still the default name. This happens because the modules inject the driver files at the system level, while driver changes made on the Gateway are placed as core resources. If any of your drivers have been modified, a message box will appear above the drivers list stating the number of drivers with multiple definitions. Click the **Show All** option to show the duplicates.

Database Settings

Drivers Translators

For information about installing driver files that are not able to be bundled with Ignition, see [this help page](#).

JDBC Drivers

Deployment Modes	1 resource has multiple definitions. Duplicate definitions are hidden.				Show All
NAME	DRIVER TYPE	DEFAULT TRANSLATOR	VERSION	STATUS	
☐ MariaDB	MYSQL	MYSQL	1	✓ INSTALLED	⋮
☐ Microsoft SQLServer	MSSQL	MSSQL	1	✓ INSTALLED	⋮
☐ MySQL	MYSQL	MYSQL	1	✓ INSTALLED	⋮
☐ PostgreSQL	POSTGRES	POSTGRES	1	✓ INSTALLED	⋮
☐ SQLite	SQLITE	SQLITE	1	✓ INSTALLED	⋮

Showing 5 of 5 items

The expanded list shows the results of modifying the MSSQL JDBC driver. It now has definitions on the core and system level, with the core definition as the primary option.

Database Settings

Drivers

Translators

For information about installing driver files that are not able to be bundled with Ignition, see [this help page](#).

JDBC Drivers

Deployment Modes

All resource definitions are visible.

Hide Inactive

Filter jdbc drivers...

View20

Create JDBC Driver

☐	ORIGIN	NAME	DRIVER TYPE	DEFAULT TRANSLATOR	VERSION	STATUS	
☐	system	MariaDB	MYSQL	MYSQL	1	INSTALLED	
☐	core	Microsoft SQLServer	MSSQL	MSSQL	1	INSTALLED	
☐	system	Microsoft SQLServer					
☐	core	MySQL	MYSQL	MYSQL	1	INSTALLED	
☐	system	PostgreSQL	POSTGRES	POSTGRES	1	INSTALLED	
☐	core	SQLite	SQLITE	SQLITE	1	INSTALLED	

Showing 5 of 5 items

Additional JDBC Driver Files

Depnding on the database you want to connect, you may need to add your own JDBC driver. Refer to the [JDBC Drivers and Translators](#) page for more information on adding additional JDBC drivers.

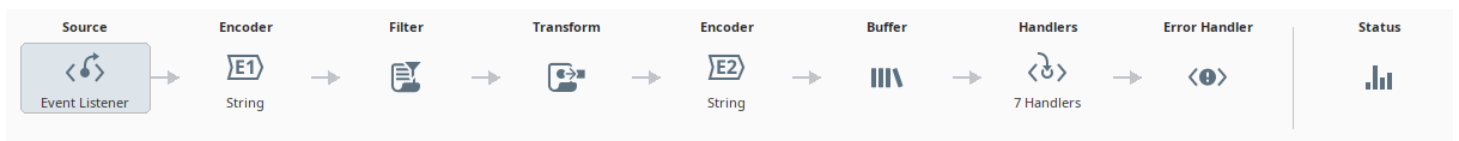
Event Streams

Event streams are project resources in Ignition that aim to allow users to have a unifying construct for handling asynchronous events, allowing you to filter, transform, and react to those events in various ways. This helps eliminate the complexity of configuring multiple scripts on separate subsystems, making data manipulation more streamlined. Event streams can be created and modified in the [Designer](#).

Unlike other mechanisms that handle data such as SQL databases, event streams are primarily event-driven or subscription-based, meaning no active solicitation of data is needed.

Event Stream Stages

Event streams are made up of several parts, called stages. While some of these stages are optional, such as the Filter and Transform stages, they will help your event stream get the result you want.



Source

The source stage, also known as the event source, is the origin of the data that the user wants to send through the event stream. Sources include:

- Kafka (Requires Kafka module)
- HTTP Endpoint (Requires Web Dev module)
- Event Listener
- Tag Event

See the [Types of Sources](#) page for more details about each source.

NOTE

Some sources may be unavailable unless the corresponding module is installed, such as [Kafka](#) for Kafka topics.

Event Objects - Data and Metadata

Source data can be broken down into event objects, which are typically JSON payloads. Event objects consist of:

- Primary data - The primary object produced by the source. Primary data can be any type of Java object, but are most commonly `byteArray` (`byte[]`) or `String` type.
- Metadata - Information describing the event. Metadata is specific to each source, and a description of available fields should be included in the source documentation. Providing metadata in your source is optional.

NOTE

Elements from an event object's primary data and metadata can be extracted in the handler stage using dot notation and the expression language. See the [Handler Expressions](#) section for more information on how to extract these elements, and the [Source Data and Metadata](#) page for more details about the elements themselves.

Encoder

The encoder is a stage within the event stream that allows users to encode data into specific formats. There are two encoding stages, the first of which follows the source stage, and a potential second which follows the transform stage (if transforms are enabled). Encoder options include:

- String
- JsonObject
- Byte[]

By default, source data will be encoded as JSON.

The encoder stage also has a table that allows you to change the encoding property from UTF-8 to a different character encoding, such as UTF-16 or BASE_64.

Encoder

String ▼

Name	Property
encoding	UTF-16

NOTE

Encoder stages are not substitutes for the transform stage, and should not be used interchangeably.

Additionally, the best encoding type for your event stream will vary depending on what kind of handler you are using.

Filter

The filter stage is an optional script that takes the event payload as an argument and returns a boolean. If the script returns false, the event is ignored and not passed down the event stream to the next stage.

The filter script takes two arguments (event and state), both of which can be further broken down.

Argument	Data Type	Subargument
event	PyDictionary	- data - The data from the event object, as a PyObject. - metadata - The metadata from the event object, as a PyDictionary.
state	PyDictionary	- lastEvent - The last event seen by the filter, which may or may not have been dropped. - lastEmittedEvent - The last event that returned True from the filter function.

Users can also add their own keys or values to PyDictionary if they want to store some state between filter calls.

Filter

☒ Enable Filter

▼ Description

Filters events from the Event Stream. For a given event, return true if, after evaluating the event, you want it to continue to the next stage in the Event Stream

▼ Parameters

event		Event provided by Source
state	(dict)	PyDictionary

Script

```
1 def filter(event, state):
2     return True
```

Filter Compilation Result

Transform

The transform stage is an optional script that takes the event payload from the source (or the filter script if returned True) as an argument and returns a different object. This new object is then used as the event payload going forward in the event stream.

Similar to the filter script, the transform script takes two arguments (event and state), both of which can be further broken down:

- event (PyDictionary)
 - data (PyObject) - The data from the **event object**.
 - metadata (PyDictionary) - The metadata from the event object.
- state (PyDictionary)
 - lastEvent - The last event seen by the filter, which may or may not have been dropped.

Users can also add their own keys or values to PyDictionary if they want to store some state between transform calls.

The transform stage can often function as data preparation. Users can manipulate the data into the form they want, making it ready for the handler stage to do something with the data.

Transform

☒ Enable Transform

Description

Transform allows you to package the data in the event to a type that handlers can process.

Parameters

event

state

Event provided by Filter or Source

(dict)
PyDictionary

Script

1def transform(event, state):

2return event.data

Transform Compilation Result

Buffer

The buffer stage acts as a staging area for events before they are passed to the handlers. Events can be batched together, depending on how the buffer stage's properties are configured. Moreover, if your handlers are already working through a set of events, the buffer stage will not pass along queued events until the events in the handler stage have been resolved. You can configure the following buffer properties.

Buffer Property	Description
Debounce	The amount of time the event stream will wait for another event before passing the current event or batch of events to the handler(s), in milliseconds. This property's value is reset back to the configured value after receiving an event. This means events will be batched together unless the Debounce time is reached, or until the Max Wait property's limit is reached, whichever comes first. Default is 100.
Max Wait	The maximum amount of time before an event or batch of events will be sent to the handler(s), in milliseconds. Default is 1000.
Max Queue Size	The maximum number of events that will be queued in the buffer stage before the event stream begins dropping events. A value of 0 means that there is no limit. Default is 0.
Overflow	Determines which event should be dropped once the Max Queue Size is reached. Options include: - DROP_OLDEST - DROP_NEWEST

NOTE

Events are dropped, not forwarded, when the Max Queue Size is reached. Events will only be forwarded to handlers if either the Debounce or Max Wait properties are reached and the handler is not currently processing events.

Buffer

Debounce(ms)

100

▲▼

Max Wait(ms)

1,000

▲▼

Max Queue Size

0

▲▼

Overflow

DROP_OLDEST

▼

Handler

The handler is the penultimate stage of the event stream and does something with the event object, depending on what kind of handler is set up. Handlers include:

- Kafka (Requires Kafka module)
- Database (Requires SQL Bridge module)
- HTTP (Requires Web Dev module)
- Gateway Event
- Gateway Message
- Logger
- Script
- Tag

See the [Types of Handlers](#) page for more details about each handler.

NOTE

Some handlers may be unavailable unless the corresponding module is installed, such as SQL Bridge for database handlers.

The handler stage itself can consist of multiple handlers, each of which are executed sequentially. You can reorder each handler step to meet your needs.

Handler Expressions

Handler expressions take advantage of Ignition's [Expression language](#). Unlike regular expressions however, handler expressions have a feature unique to event streams by allowing users to pull specific primary data or metadata from the [event object](#).

You can use dot notation encapsulated with curly brackets when attempting to extract elements from an event object within your handler. For example, if a user wanted to extract the topic metadata from a Kafka event, you can use `{event.metadata.topic}` to extract that metadata. Similarly, if you wanted to extract a specific data element such as a name from a Kafka event, you can use `{event.data.name}`. See the [Source Data and Metadata](#) page for more information on each source's extractable elements.

Depending on the handler you are using, the syntax for your handler expression may be different. For example, since script handlers process the entire batch of data at once, syntax using a singular event is not possible. You must use array syntax such as `events[0].data`. In contrast, all other handlers must use singular syntax such as `{event.data}`, since they process data per individual event.

Common Settings

The following are settings that are available for all handlers.

Enabled

Each handler has an Enabled option, allowing users to toggle the specified handler as enabled or disabled.

Failure Handling

Failure handling defines what behavior should happen if a handler is unable to process an event(s) or throws an exception. The following failure mode behaviors are available.

Failure Mode	Description
IGNORE	The error is logged, and the event stream moves on to the next handler. This also tells the source that the data was handled successfully.
ABORT	The error is logged, but the event stream does not move on to the next handler. This also tells the source that the data was not handled successfully.
RETRY	The error is logged, and the handler tries to handle the data again, after a specified delay.

The RETRY behavior can be further configured with the following properties.

RETRY Property	Description
Retry Strategy	Whether the Retry behavior is performed on a fixed or exponential rate.
Retry Count	The number of retries allowed for the event handler. Default is 1.
Retry Delay	The number of seconds before the handler is allowed to try handling the data again. If the Retry Strategy property is set to exponential, the specified value will be the initial delay, with each subsequent delay set by the Multiplier property. Default is 1.
Multiplier	The value by which to multiply the number of seconds specified in the Retry Delay property. For example, a value of 2 will double the number of seconds before each subsequent retry is allowed. Enabled only if the Retry Strategy property is set to EXPONENTIAL. Default is 1.
Max Delay	The maximum limit of time in between each subsequent retry in milliseconds, regardless of the Multiplier property. Enabled only if the Retry Strategy property is set to EXPONENTIAL. Default is 1.
Retry Failure	The behavioral logic to follow if the Retry Count is reached, but the handler still failed to handle the data. Available options include: - IGNORE - ABORT

Error Handler

The error handler is the final stage of the event stream and acts as a final catch-all for any potential issues or errors that were not caught by each individual handler's failure handling logic. This stage is invoked when a handler throws an exception and the individual handler's failure handling is set to ABORT. You can define what you want to do with a given error using the scripting area.

Error Handler

☒ Enable OnError Handling

▼ Description

Method will be invoked when a handler throws an exception.

▼ Parameters

event		Event provided by Source.
state	(dict)	PyDictionary

Script

```
1 def onError(event, state):  
2     #do something
```

Checking Event Stream Status

To the right of the event stream stages is the Status panel. You can check information such as how many events were received, how many events were filtered, and throughputs for various executions and handlers. The table below describes each status metric.

Metric	Description
Events Received	The number of events received by the event stream.
Events Filtered	The number of events that have passed through the filter.
Filter Execution	The amount of time an event takes to go through the filter stage. Throughputs measured include the average time of an event, maximum amount of time for a given event, and the time for the most recent event.
Transform Execution	The amount of time an event takes to go through the transform stage. Throughputs measured include the average time of an event, maximum amount of time for a given event, and the time for the most recent event.
Handler Execution	The amount of time an event takes to go through the specified handler. Additional rows will be added when new handlers are created. Throughputs measured include the average time of an event, maximum amount of time for a given event, and the time for the most recent event.

Status	
Events Received	0
Events Filtered	0/0 passed
Filter Execution (avg/max/most recent)	0 ms / 0 ms / 0 ms
Transform Execution (avg/max/most recent)	0 ms / 0 ms / 0 ms
Handler 1 (Logger) Execution (avg/max/most recent)	0 ms / 0 ms / 0 ms

Project Browser Status Icons

New in 8.3.8

Status icons will now appear next to the event stream name in the Project Browser. Icons will update as you edit and save your project to indicate running, modified, disabled, and failed event stream statuses. Hover over an icon to display its tooltip for more context.

Testing Your Event Stream

While creating your event stream, you can use the Test Controls panel to test your event stream. You can toggle the Test Controls panel by clicking on the Test Panels  icon on the top right of your Designer window. You can configure your test run with the following properties.

Test Controls Property	Description
Source Object	The source data you want to test your event stream with. You can choose to either test the most recent source object or manually input data to run through your event stream.
Encoder	The encoding format to use for your source data test run. You can encode the data in the following formats: • String • ByteArray (Base64) • JsonObject
Dry Run	Whether or not the event from Test Controls panel should be stored as a regular event. The test event will still run down the event stream as if it were a regular event, but enabling this property will prevent the test event from becoming an immutable event.
Run All	Run the event stream test. You can also choose to run the event stream up to the filter or transform stages by clicking on the drop down menu arrow on the right.

Test Controls

Source Object

Manual

Encoder

String

☐ Dry Run

Run All

Performing a test run will return results that reflect the event stream stage's state. Each available state is described in the table below.

State	Description
Disabled	The stage is currently disabled.
Error	The stage has processed a test event, but ran into an issue.
Faulted	The event stream has a compilation error and no test events are processed.
Good	The stage is functioning as expected and without any issues.
Ready	The stage is ready to receive and process data.

Viewing Event Stream Metrics in the Gateway

The Gateway Webpage contains a page that presents a collective of projects on the Gateway and their respective event streams, and displays metrics about them. See the [Event Streams](#) section of the Services page for more information.

Types of Sources

This page details each source available for [event streams](#).

Kafka

The Kafka source will subscribe to a Kafka topic and wait for event payloads. Data within the payload will be stored as `byte[]`, but can be changed to a different format during the encoder stage. Note that the Kafka source requires the [Kafka module](#) to be installed and enabled.

The Kafka source can be configured with the following properties.

Property	Description
Subscription Type	The way you want to receive Kafka event objects. Options include: • Consumer Group: Receives events from a specified consumer group using the <code>GroupID</code> and <code>Offset Reset</code> properties. • Partition: Receives events from a specified partition using the <code>Partition</code> and <code>Offset</code> properties.
Connector	The specific Kafka connector you want to use, configured in the Kafka Connector Gateway Webpage .
Topic	The specific topic you want to subscribe to, selectable within the dropdown menu. You can also use an expression to subscribe to multiple topics.
Group ID	The ID of the consumer group you want to subscribe to. This property will be used in conjunction with with the Consumer Group subscription type.
Partition	The partition you want to subscribe to. This property will be used in conjunction with the Partition subscription type.
Max Records	The maximum number of records you want returned on each poll. Default is 500.
Poll Timeout	The maximum amount of time the poll will wait before beginning the next poll, in milliseconds. Default is 100 milliseconds.
Offset Reset	The initial offset when first connecting to the Kafka broker. Options include: • Latest • Earliest • None This property will be used in conjunction with the Consumer Group subscription type.
Offset	The integer offset to start at when polling. This property will be used in conjunction with the Partition subscription type.

Property	Description
Additional Consumer Properties	Allows a user to specify any other consumer properties you want to add, as key value pairs. See the official Kafka documentation for available consumer configurations.

Source - Kafka ⓘ

Subscription Type

Consumer Group ▼

Configuration

Connector

▼

Topic

▼

Group ID

Max Records

500

▲▼

Poll Timeout

100

▲▼

Offset Reset

Latest ▼

Additional Consumer Properties

Key	Value
-----	-------

+
🗑️

HTTP

The HTTP source accepts HTTP POST and HTTP PUT requests to receive event objects, and will be automatically mounted at `/system/eventstream/{projectName}/{resourcePath}`. Note that the HTTP source requires the [Web Dev module](#) to be installed and enabled.

The HTTP source can be configured with the following properties.

Property	Description
Require HTTPS	Whether or not HTTPS is required to receive event objects. Default is true.
Require Authentication	Whether or not authentication is required to receive event objects. Default is false.
User Source	The user source to use if the Require Authentication property is enabled. Disabled by default.
Required Roles	The roles required, if any, that a user within the User Source property must have for authentication. Disabled by default.
Max Auth Retries	The number of authentication retries allowed. Disabled by default.

Source - HTTP ⓘ

☒ Require HTTPS

☒ Require Authentication

User Source

Required Role(s)

Max Auth Retries

0

Event Listener

The Event Listener source receives events containing a payload of serializable data from another part of Ignition, such as Gateway Messages or other event streams. Unlike other sources, the Event Listener source is not configurable.

Not Configurable

Tag Event

The Tag Event source listens for any changes to a tag's timestamp, value, or quality. Users can subscribe to a single tag path, or a list of tag paths, configurable through the Insert Tag 📌 icon on the right.

Changed in 8.3.4 You can use wildcards at both the folder level to help specify a folder path, and at the tag level to help specify a tag path. See [Wildcards in Tag Paths](#) for supported operators and patterns.

Data within the payload is stored as a Java object. The Tag Event source can be configured so that only a scalar value is published or the entire qualified value, comprised of the tag path, timestamp, value, and quality, is published.

The Tag Event source can be configured with the following properties.

Property	Description
Value	Whether or not changes to the specified tag(s) value will trigger the event stream. Default is true.
Quality	Whether or not changes to the specified tag(s) quality will trigger the event stream. Default is true.
Timestamp	Whether or not changes to the specified tag(s) timestamp will trigger the event stream. Default is true.
Skip Initial Value	Whether or not the initial values captured when saving configuration changes will trigger the event stream. Default is false. New in 8.3.2 The option to subscribe to Alarm Metrics in the Tag Event source was introduced in version 8.3.2. Since Alarm Metrics values are NULL until they are subscribed to and therefore will always need to trigger the event stream upon initialization, this property will be ignored when these values are included in the tag path.
Tag Path(s)	The area of the Tag Event source where tag path(s) are specified, with each line being a new item. You can enter tag paths manually, or use the Insert Tag icon on the right to specify a path. Tag paths will be validated as you type.

Source - Tag Event ⓘ

Change Triggers

☒ Value ☒ Quality ☒ Timestamp ☐ Skip Initial Values

Tag Path(s)

[default]newTag1
[default]newTag2
[default]newTag3



Data and Metadata

Each source may contain various elements that can be extracted using [expressions in the handler](#). See the [Source Data and Metadata](#) page to learn more about which components can be extracted.

Source Data and Metadata

This page details the extractable elements from each source. When pulling data into handlers using expressions, you will need to add curly brackets around the expression itself. See [Handler Expressions](#) on how this information is extracted.

Kafka

 NOTE

The Kafka source requires the [Kafka module](#) to be installed and enabled. If the Kafka module is not installed or is disabled, the source will not function and no extractable elements will be available.

Event Object - Data

There is no pre-existing extractable Kafka event object data, as this is provided by the user.

Event Object - Metadata

The table below explains the extractable metadata fields for Kafka event objects.

Metadata Field	Expression Syntax	Description
topic	<code>{event.metadata.topic}</code>	The topic the source event object came from. Topics are similar to folders in a file system, helping to categorize data.
partition	<code>{event.metadata.partition}</code>	The partition the source event object came from. Partitions are contained within topics, and help improve efficiency by breaking up large chunks of data into smaller partitions.
offset	<code>{event.metadata.offset}</code>	The assigned index that helps keep track of which events have already been consumed, allowing consumers to stop and resume consumption without needing to consume from the beginning.
headers	<code>{event.metadata.headers}</code>	The metadata or context specific to a message, but separate from the actual message payload. Headers are typically stored as key value pairs, and contains information such as the format of the message payload.
key	<code>{event.metadata.key}</code>	The key of the Kafka event object, which determines the partition the event object payload will go to.

HTTP

NOTE

The HTTP source requires the [Web Dev module](#) to be installed and enabled. If the Web Dev module is not installed or is disabled, the source will not function and no extractable elements will be available.

Event Object - Data

There is no pre-existing extractable HTTP event object data, as this is provided by the user.

Event Object - Metadata

The table below explains the extractable metadata fields for HTTP event objects.

Metadata Field	Expression Syntax	Description
method	<code>{event.metadata.method}</code>	The type of method the HTTP request is using, such as POST and PUT.
headers	<code>{event.metadata.headers}</code>	Additional categorical data about the HTTP request, such as the host or accepted language.
parameters	<code>{event.metadata.parameters}</code>	The parameters the HTTP request abides by to perform a specified method.
scheme	<code>{event.metadata.scheme}</code>	The protocol used for the HTTP request, such as HTTP or HTTPS.
remoteAddr	<code>{event.metadata.remoteAddr}</code>	The IP address of the domain you are performing an HTTP request on.
remoteHost	<code>{event.metadata.remoteHost}</code>	The domain name of the target you are performing an HTTP request on, such as www.inductiveautomation.com .

Alarm

Unlike the event objects above that are set up using an explicit source within the event stream, the alarm event object requires using both the **Event Stream Source block** and an event listener source.

Event Object - Data

The table below explains the extractable data fields for alarm event objects.

Data Field	Expression Syntax	Description
eventId	<code>{event.data.eventId}</code>	The UUID of the alarm event.
source	<code>{event.data.source}</code>	The qualified path of the entity that generated the alarm event.
displayPath	<code>{event.data.displayPath}</code>	The display path of the alarm.
priority	<code>{event.data.priority}</code>	The priority of the alarm, as an integer. Expected results are: • 0 (Diagnostic) • 1 (Low) • 2 (Medium) • 3 (High) • 4 (Critical)
priorityReadable	<code>{event.data.propertyReadable}</code>	The priority of the alarm, as a string. Expected results are: • Diagnostic (0) • Low (1) • Medium (2) • High (3) • Critical (4)
eventType	<code>{event.data.eventType}</code>	The condition of the alarm event, as an integer. Expected results are: • 0 (Cleared and Unacknowledged) • 1 (Cleared and Acknowledged) • 2 (Active and Unacknowledged) • 3 (Active and Acknowledged)

Data Field	Expression Syntax	Description
eventTypeReadable	<code>{event.data.eventTypeReadable}</code>	The condition of the alarm event, as a string. Expected results are: • Cleared, Acknowledged (0) • Cleared, Unacknowledged (1) • Active, Unacknowledged (2) • Active, Acknowledged (3)
eventFlags	<code>{event.data.eventFlags}</code>	A numeric bitmask flag that provides additional information about the alarm event. See the Alarm Events table for more information on eventFlags.
eventTime	<code>{event.data.eventTime}</code>	The datetime of the alarm event.

Event Object - Metadata

There is no extractable Alarm event object metadata.

Tag Event

Event Object - Data

The table below explains the extractable data fields for tag event objects.

Data Field	Expression Syntax	Description
value	<code>{event.data.value}</code>	The value of the tag event object.
quality	<code>{event.data.quality}</code>	The quality code of the tag that the tag value originates from. See the Quality Code Reference Table for more information on different qualities.
timestamp	<code>{event.data.timestamp}</code>	The timestamp of the tag value.

Event Object - Metadata

The table below explains the extractable metadata fields for tag event objects.

Data Field	Expression Syntax	Description
tagPath	<code>{event.metadata.tagPath}</code>	The tag path of the event object, as a string.
isInitial	<code>{event.metadata.isInitial}</code>	A boolean that returns true if the specified event object's value is the initial value after creating or editing the tag. Follows the same logic as a tag script's initialChange argument.
previousValue	<code>{event.metadata.previousValue}</code>	New in 8.3.5 The qualified previous value of a tag that includes the value, quality, and timestamp properties. Follows the same logic as a tag script's previousValue argument.

Types of Handlers

This page details each type of handler and what they are used for in an [event stream](#).

Kafka

The Kafka handler allows you to send source event data from your event stream to a specified Kafka topic. Note that the Kafka source requires the [Kafka module](#) to be installed and enabled.

This handler is configurable with the following properties.

Kafka Handler Property	Type	Description
Connector	String	The name of the Kafka connector to send event data to.
Topic	String	The name of the Kafka topic to send event data to.
Partition	String	The partition to send event data to.
Timestamp	String	The timestamp of the Kafka record.
Key	String	The key of the Kafka record.
Value	String	The value produced to send to the specified Kafka connector.
Headers	PyDictionary	Allows a user to specify metadata information about the produced record's message payload, as key value pairs.
Additional Producer Properties	PyDictionary	Allows a user to specify any other producer properties you want to add, as key value pairs. See the official Kafka documentation for available producer configurations.

Handlers - Configure Kafka

☒ Enabled Handler

Connector

MyKafkaConnector ▼

Topic

"MyKafkaTopic" *fx* 

Partition

1 *fx* 

Timestamp

now(10) *fx* 

Key

"employeeID" *fx* 

Value

"123456789" *fx* 

Headers

Key	Value	+ 🗑

Additional Producer Properties

Key	Value	 
buffer.memory	1000000000	

Database

The database handler allows you to send source event data from your event stream to a specified SQL database. This handler is configurable with the following properties.

Database Handler Property	Type	Description
Database	String	The database you want to send the data to. The dropdown menu will be populated with available database connections defined in the Gateway.
Mode	Enum	The action you want performed on the data. Selections in the dropdown menu include: • INSERT • UPDATE The INSERT action utilizes a standard SQL insert statement with the data provided in the Column Mapping table. The UPDATE action utilizes the Where Clause Configuration table to identify which row will be updated with new data.
Table Name	String	The table in the database you want to store data to. The dropdown menu will be populated with available tables found in your database connection(s). You can also manually enter a table name and use the Automatically Create Table property to create a new table.
Automatically Create Table	Boolean	Whether or not a new table with the required columns should be created using the entered table name. Default is enabled.
Store timestamp to	Boolean	Whether or not a timestamp should be recorded to the specified column. If enabled, the system will use the system clock to generate the timestamp. You can choose a column from the dropdown menu, or manually create a column by manually entering a column name.
Bypass Store and Forward System	Boolean	Whether or not the Store and Forward system should be used. Enabling this property will cause the database handler to ignore Store and Forward. Default is disabled.

Database Handler Property	Type	Description
Column Mapping	User Defined	Allows a user to specify which SQL column(s) data should be mapped to using the column name, SQL column type, and value. This table will be used in conjunction with the Mode property's INSERT option.
Where Clause Configuration	User Defined	Allows a user to specify where in the SQL table data should be updated by providing the column name, logical operator, and value. This table will be used in conjunction with the Mode property's UPDATE option.

Handlers - Configure Database ⓘ

☒ Enabled Handler

Database Settings

Database

SQLite

Mode

UPDATE

Table Name

TestTable

↻

☒ Automatically create table

☒ Store timestamp to

t_stamp

↻

☐ Bypass Store and Forward System

Column Mapping

Column Name	Column Type	Value Expression	
location	String	Sacramento	✓ fx
			🗑

Where Clause Configuration

Column Name	Operator	Value Expression	
location	=	Folsom	✓ fx
			🗑

Preview

UPDATE TestTable SET location = " WHERE location = Folsom

Failure Handling

Failure Mode

ABORT

HTTP

The HTTP handler sends POST and PUT requests to a specified URL. The HTTP handler is configurable with the following properties.

HTTP Handler Property	Type	Description
URL	String	The destination URL to sent the HTTP request to.
Type	Enum	The HTTP request type you are using. Options include: • POST • PUT
HTTP Version	Enum	New in 8.3.5 Whether HTTP/1.1 or HTTP/2 is used for the HTTP protocol. Default is HTTP/2.
Headers	List	The headers you want to include as part of the HTTP request. Values will be entered in the table using key value pairs as strings, with the header type as the key.
Body	Byte	The payload included as part of the HTTP request. This data will be encoded to byte format and is evaluated as an expression.
Connect Timeout	Int	The amount of time to wait before the HTTP request times out, in milliseconds. Default is 30,000.
Cookies Enabled	Boolean	Whether or not cookies should be enabled. A value of True will use cookies, while False will not. Default is True.

Handlers - Configure HTTP ⓘ

☒ Enabled Handler

URL

"http://localhost:8088/system/webdev/event-streams/exchar *fx*" 

Type

POST ▼

HTTP Version

HTTP/2 ▼

Headers

Key	Value	
Accept-Language ▼	en-US <i>fx</i>	 

Body

1 {event.data} 





Connect Timeout

30,000

Socket Timeout

60

☒ Cookies Enabled

Gateway Event

The Gateway Event handler sends events to a different event stream. Note that the receiving event stream must have an Event Listener source configured in order to receive this event. You can define the destination of your event with the following properties.

Gateway Event Handler Property	Type	Description
Project	String	The destination project containing the event stream that the event will be sent to. You can use the dropdown menu to select your project, or use an expression to define the destination project.
Path	String	The destination event stream that the event will be sent to. The dropdown menu will list available event streams defined within the specified project. You can also use an expression to define the path, allowing you to send events to different event streams.
Payload	PyDictionary	The payload that is to be sent to the destination event stream. You can also use an expression to create a dynamic payload.
Remote Gateway Name	String	The name of the remote Gateway to send events to.

Handlers - Configure Gateway Event ⓘ

☒ Enabled Handler

Project

fx ✓

Path

fx ✓

Payload

fx ✓

Remote Gateway Name

Gateway Message

The Gateway Message handler mimics the [sendMessage](#) system function by sending a [message](#) to a targeted Gateway. The Gateway Message handler is configurable with the following properties.

Gateway Message Property	Type	Description
Project	String	The project you want to send the Gateway Message's payload to. You can use the dropdown menu to select your project, or use an expression to define the destination project.
Message Handler	String	The name of the message handler in the Gateway Message you want to send.
Payload	PyDictionary	The data that will be passed onto the message handler.
Properties	PyDictionary	Allows you to specify other properties that are part of the <code>system.util.sendMessage</code> function, using key value pairs. Available parameters include: - scope - clientSessionID - user - hasRole - hostName - remoteServers See the sendMessage page for more information on each parameter.

Handlers - Configure Gateway Message ⓘ

☒ Enabled Handler

Project

test

fx

✓

Message Handler

My Message Handler

fx

✓

Payload

payload goes here!

fx

✓

Properties

Key	Value	
scope	CGS	✓ fx
hasRole	Administrator	✓ fx

+

🗑

Logger

The logger handler logs a message to the logging system when events arrive. This handler is configurable with the following properties.

Logger Handler Property	Type	Description
Name	String	The name of your logger. You can use a simple string, or use an expression to name your logger.
Log Level	Enum	The log level used by your logger handler. Available options include: • TRACE • DEBUG • INFO • WARN • ERROR
Messages	String	The message you want logged with the logging system. You can input a simple string, or use an expression to define your message.

Handlers - Configure Logger ⓘ

☒ Enabled Handler

Name

test-logger-handler *fx* ✓

Log Level

TRACE ▼

Messages

1 I am a message!

✓
+=
Σ

Script

The script handler allows users to customize how they want to handle incoming events by executing the defined script when events arrive. The payload output from this script will be a PyObject type.

Handlers - Configure Script ⓘ

☒ Enabled Handler

▼ Description

Receives a list of events when a batch is ready

▼ Parameters

events	(list)	List of events
state	(dict)	State dictionary

Script

1 def onEventsReceived(events, state):

2 system.util.getLogger("event-stream").info(str(events[0].data))

Tag

The tag handler takes data from the event stream's source and writes the data to a specified tag. The tag handler is configurable with the following properties.

Tag Handler Property	Type	Description
Tag Path	String	The tag path to use when determining which tag to write to. You can use an expression to specify a dynamic tag path based on the event payload.
Value	String	The value you want to write to the tag from the payload of the source event object. You can use an expression to break down the payload into individual data or metadata. See the Source Data and Metadata page for more information.
Quality	String, Int	The quality to write to the tag, as a string or an integer. See the Quality Codes and Overlays page for details about specific codes. You can use an expression to write a dynamic quality.
Timestamp	String	The timestamp to write to the tag. You can write a dynamic timestamp using an expression.

Handlers - Configure Tag ⓘ

☒ Enabled Handler

Tag Path

fx

Value

fx

Quality

fx

Timestamp

fx

Failure Handling

Failure Mode