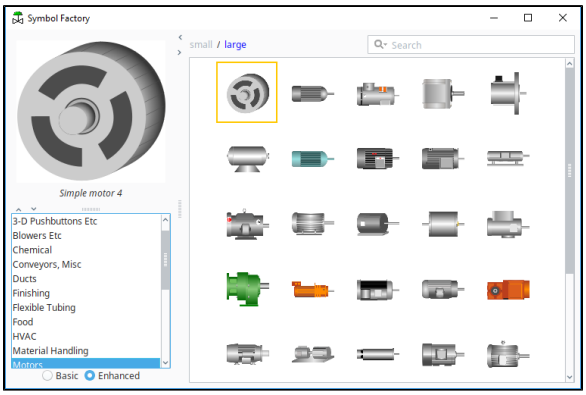
Symbol Factory

Overview

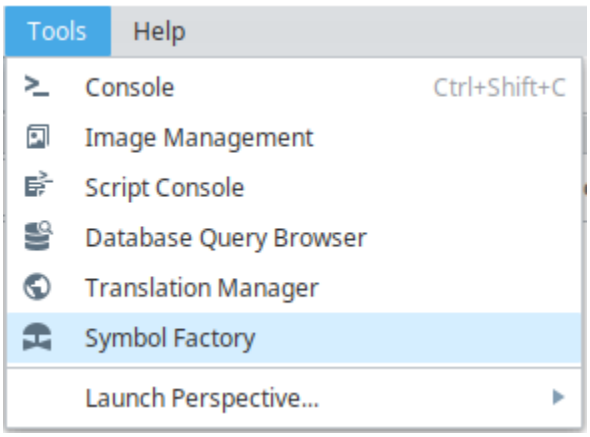
The Symbol Factory Module is included with the Vision Module or the Perspective Module. Symbol Factory provides nearly 4,000 industrial high quality Scalable Vector Graphics (SVG) and symbols for your projects. Vector based graphics can be resized with no pixelation or distortion. You can add these images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

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.

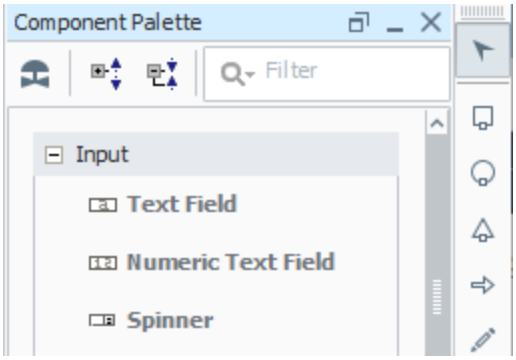


Using the Symbol Factory

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.



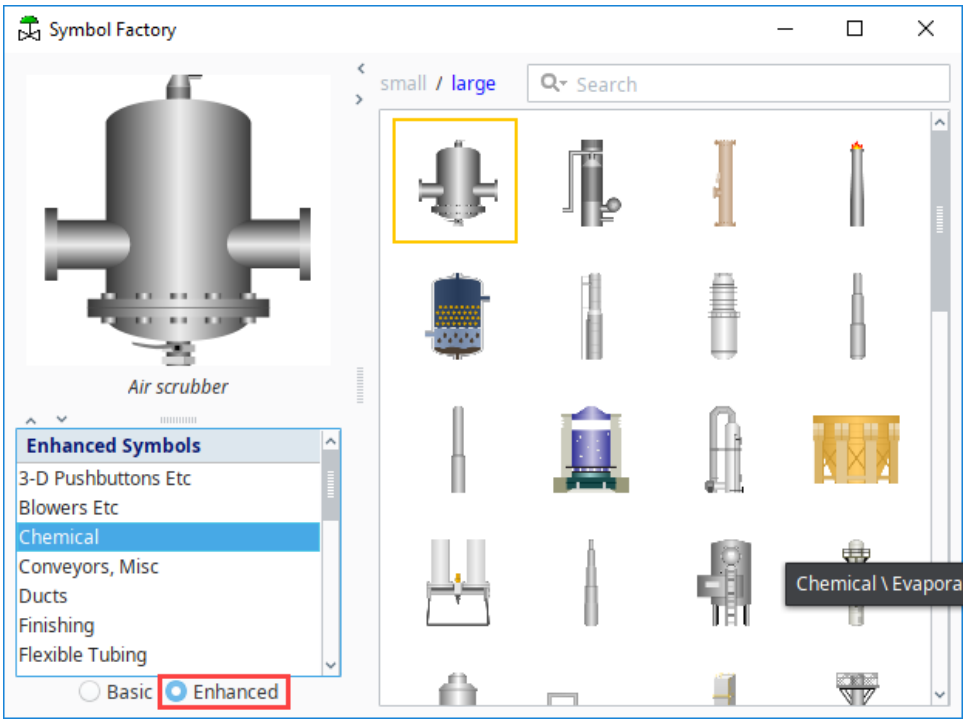
3. In the Vision module, there is also an icon (valve icon in the upper left) for quick access to the Symbol Factory images at the top of the Component Palette.



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

> ⚠ When searching Symbol Factory, it is recommended to select the Enhanced radio button in the Symbols List. The enhanced symbols are more detailed, and they have groupings that enable you to more easily animate them.

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, SVGs, and Icons in Perspective.
7. In Vision, the symbol will become a group of shapes. See Images and SVGs in Vision.

Related Topics ...

- Images, SVGs, and Icons in Perspective.
- Images and SVGs in Vision.

Web Dev

The WebDev module enables you to directly program against the web server inside the Ignition Gateway and systems running Vision Clients. 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.

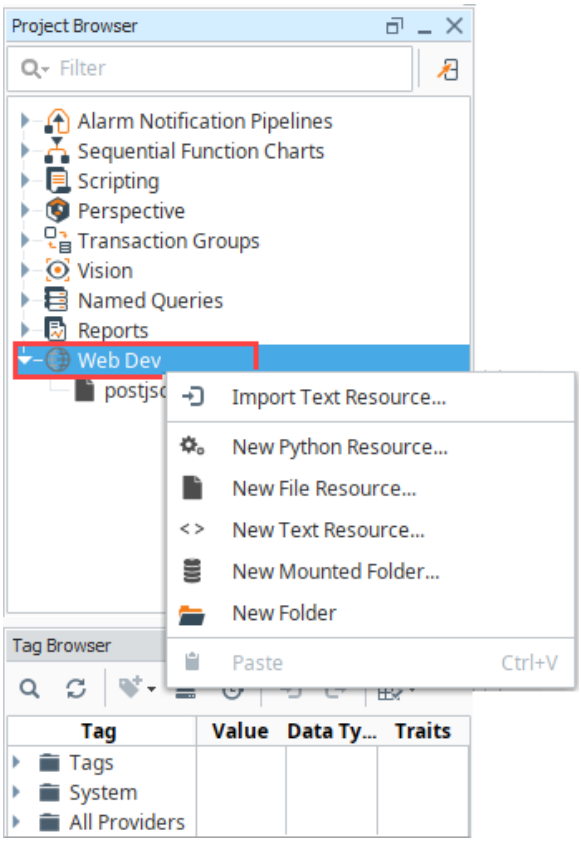
On this page

...

> **Disclaimer**
>
> The WebDev 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 WebDev 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:



> The following feature is new in Ignition version **8.0.7**
> Click here to check out the other new features

As of 8.0.7 WebDev 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. Note that you will need to re-import a file resource if it has been changed since adding it to WebDev.
- **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 look liked 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

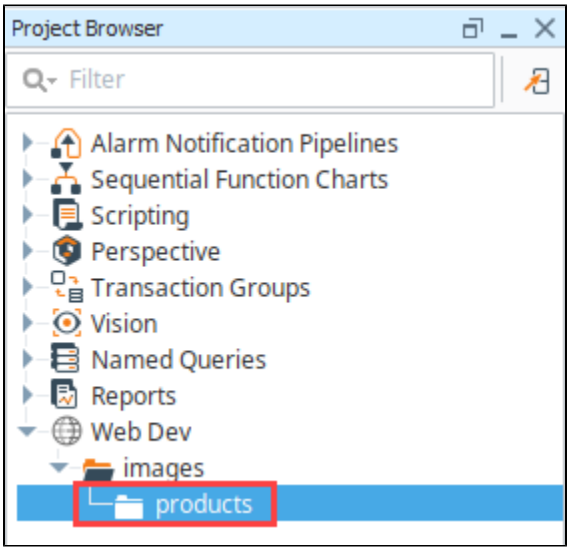
```
http://host:port/system/webdev/projectname/assets/a.jpg
```

URL Encoding
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 space in the file name would likely be encoded to `%20`

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

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

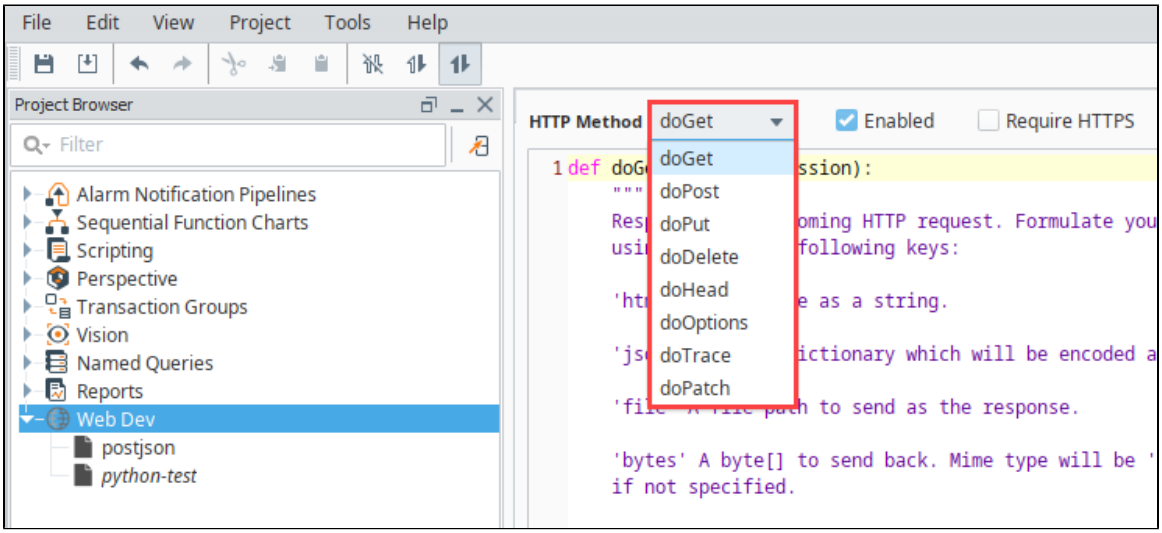
Python resources are the heart of the functionality of the WebDev module. These resources are completely dynamic, and can handle all parts of the HTTP protocol, formulating any type of response.

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

> The following feature is new in Ignition version **8.0.7**
> Click here to check out the other new features

As of 8.0.7 the HTTP method doPatch has been added.



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 access to the Ignition GatewayContext. More information about this object may 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. As of 8.0.7 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 request['postData'] 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 request['postData'] 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 request['headers']. For example, you could read the User-Agent string with request['headers']['User-Agent'].
`params`	dictionary	This will be contained in a dictionary accessible via request['params']. Any URL parameters such as the following: **Pseudocode - Request Parameter** `/system/webdev/project/foo?param1=value&param2=other_value` For the given example, request['params'] = {'param1':'value', 'param2':'other_value'}.
`remainingPath`	string	request['remainingPath'] will be "/bar". Remaining path will be "None" if nothing is found after the resource name. This provides the remaining text after a file resource. If you have a resource called 'foo', and a request is made to: **Pseudocode - Remaining Path** `/system/webdev/project/foo/bar`
`remoteAddr`	string	Returns the IP address of the client. Gives the remote IP address of the entity that made the web request. Note that 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 that 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 request['scheme']. The value will be 'http' or 'https'.
`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.

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.

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
```

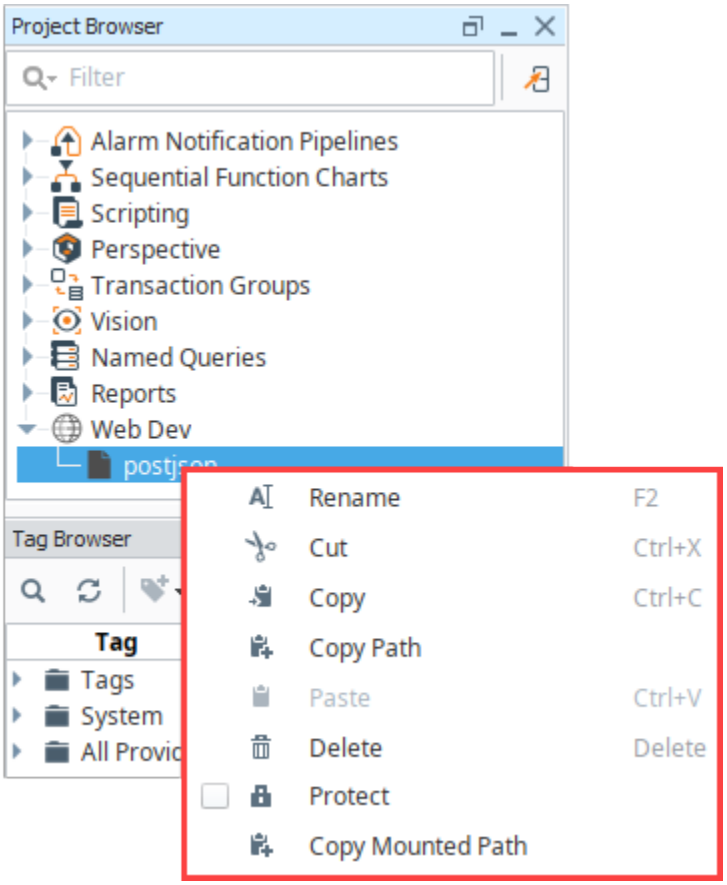
Requests to the root of your project will attempt to load a resource named "index.html". If no such resource exists, a 404 response code will be returned instead.

Pseudocode - Request to Root Project

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

Right Click Menu

The 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	The following feature is new in Ignition version **8.0.7** Click here to check out the other new features 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.

Enabled

If **Enabled** is checked, then the HTTP Method is enabled. This Enabled checkbox is new for version 8.0.7.

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.

Related Topics ...

- Web Services, SUDS, and REST
- HTTP Methods
- Installing or Upgrading a Module
- Managing Users and Roles

In This Section ...

httpPost

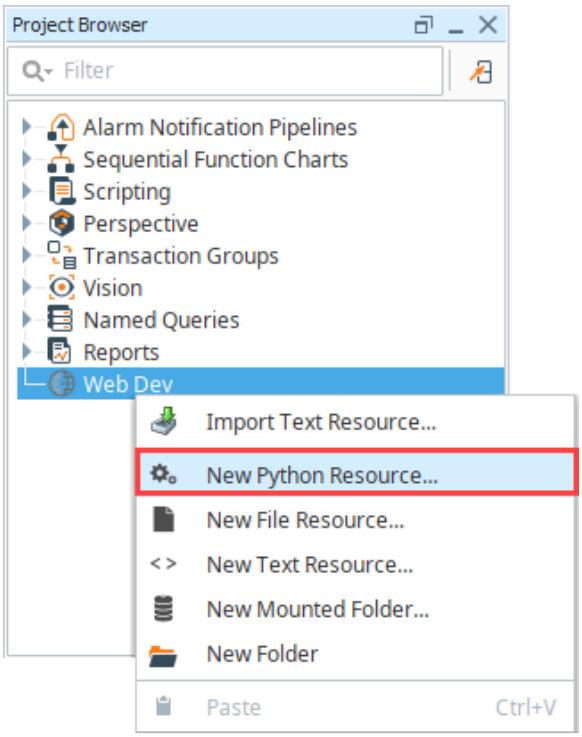
httpPost Example

On this page

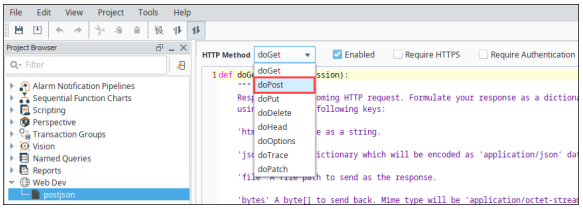
...

This example demonstrates how to allow Ignition to receive data from an external source. It uses a button to send JSON data through an httpPost command and a Python Resource in the Web Dev module to receive the post and do something with the data. This button example is for testing purposes only, the common use-case for posting data is to use another program to post data.

1. Open the Designer and right-click on the Web Dev object in the Project Browser. Select New Python Resource.



2. Name the Python resource **postjson**.
3. Select the doPost HTTP method from the dropdown in the upper left.



4. Select the **Enabled** option.
5. Copy this code into the doPost function.

"postjson" Python Resource (Web Dev Section)

```
# take in some JSON data and print it
# expecting 'names' and 'values' that are of the same length

# get the incoming parameters
data = request['postData']
names = data['names']
values = data['values']
# this will print to the wrapper.log file
print names, values

# format the string into HTML
formattedString = "<html><body>"
# loop through and add names and values
for i in range(len(names)):
        formattedString += "%s: %s, " %(names[i], values[i])
# remove the last ', ' and add closing html
formattedString = formattedString[:-2]+"</body></html>"
# this will print to the wrapper.log file
print formattedString

# return the value string
return {'html': formattedString}
```

6. Create a button on a window to test the above code. Copy the code below into your button. Make sure to change the **ProjectName** variable to the name of your project. If you used any name other than "postjson" for step 1, change the **doPostName** variable as well.

Call Web Service (Button component on a window)

```
# post data to the web service in a json format
# this allows you to use the 'postData' object in the Python
Resource

# create url to post to
projectName = "MyProject"
doPostName = "postjson"
url = "http://localhost:8088/main/system/webdev/%s/%s" %
(projectName, doPostName)
# create the dictionary of parameters to pass in
params = {}
params['names'] = ['String','Integer']
params['values'] = ['Hello World', 42]
# encode dictionary to JSON
jsonParams = system.util.jsonEncode(params)

# post to Ignition
postReturn = system.net.httpPost(url,'application/json',
jsonParams)
# print return value
print postReturn
```

7. Now test your button. Make sure to open the console to see the print out, or the wrapper.log file to see any errors caused by the doPost function.

Related Topics ...

- Diagnostics