HTTP Responses and the `Execution Location`

World of Workflow makes an excelent HTTPS server. You can quickly generate Call/Respnse workflows to build simple or complex web sites. See Building Web Pages with AI for help with this.

When you send an HTTP Response to a browser it will often contain HTML, CSS and Script.

HTTP Responses written in Liquid

The Liquid processor in World of Workflows creates the text for the HTTP Response on the server and sends the full page to the browser client. This can include data from variables, using syntax such as `{ { Variables.Company.Title } }` for the company Title field, or using a more complex loop to iterate through an array of data:

```
{% raw %}
{% assign companies = Variables.Companies %}
<table>
  <thead>
    <tr>
      <th>Title</th>
      <th>Address</th>
      <th>Website</th>
    </tr>
  </thead>
  <tbody>
    {% for company in companies %}
    <tr>
      <td> {{ company.Title }}</td>
      <td> {{ company.Address }}</td>
      <td> {{ company.Website }}</td>
    </tr>
    {% endfor %}
  </tbody>
</table>
{% endraw %}
```

The variable `Companies` will have been created earlier in the workflow. When the browser receives the HTTP Response, the table will have been composed, and the browser just displays that on the page.

Any `<script>` to be passed to the browser will be built the same way. for example

```
{% raw %}
<script>
    const companyName = '{{Variables.Company.Title}}'
    alert (companyName)
</script>
{% endraw %}
```

When this script is received by the client, it might look like this:

```
<script>
    const companyName = 'Acme Co'
    alert ('Acme Co')
</script>
```

HTTP Responses written in JavaScript

The JavaScript processor in World of Workflows creates the text for the HTTP Response on the server and sends the full page to the browser client. This can include data from variables, using syntax such as `getVariable("Company").Title` for the company Title field.

```
function generateCompanyTable(companies) {
  // Create the table element
  var table = document.createElement('table');

  // Create the table header
  var thead = document.createElement('thead');
  var headerRow = document.createElement('tr');
  var headers = [
     'Title', 'Address', 'Website'
  ];
  headers.forEach(function(header) {
    var th = document.createElement('th');
    th.textContent = header;
    headerRow.appendChild(th);
  });
  thead.appendChild(headerRow);
  table.appendChild(thead);

  // Create the table body
  var tbody = document.createElement('tbody');
  companies.forEach(function(company) {
    var row = document.createElement('tr');

    var fields = [
      company['Title'], company['Address'], company['Website']
    ];
    fields.forEach(function(field) {
      var td = document.createElement('td');
      td.textContent = field;
      row.appendChild(td);
    });

    tbody.appendChild(row);
  });
  table.appendChild(tbody);
```

```
  // Return the table as a string
  return table.outerHTML;
}

// Example usage:
var companies = getVariable("Companies");

document.body.innerHTML = generateCompanyTable(companies);
```

Written like this, the World of Workflows JavaScript processor will attempt to execute this on the server, which is not what we intend: it should be passed down to the client browser to be expected in there. However, the line

```
    var companies = getVariable("Companies");
```

cannot be executed in the client browser and must be executed on the server.

To resolve this issue, we can use the backtick (`) escape sequence. Anything written within a two backticks will be passed to the browser client to be executed there. Everything else will be executed on the server and the result will be included in the the text sent to the browser.

This is how to add the backticks to the routine :

```
`
function generateCompanyTable(companies) {
  // Create the table element
  var table = document.createElement('table');

  // Create the table header
  var thead = document.createElement('thead');
  var headerRow = document.createElement('tr');
  var headers = [
     'Title', 'Address', 'Website'
  ];
  headers.forEach(function(header) {
    var th = document.createElement('th');
    th.textContent = header;
    headerRow.appendChild(th);
  });
  thead.appendChild(headerRow);
  table.appendChild(thead);

  // Create the table body
  var tbody = document.createElement('tbody');
  companies.forEach(function(company) {
    var row = document.createElement('tr');

    var fields = [
      company['Title'], company['Address'], company['Website']
```

```
    ];
    fields.forEach(function(field) {
      var td = document.createElement('td');
      td.textContent = field;
      row.appendChild(td);
    });

    tbody.appendChild(row);
  });
  table.appendChild(tbody);

  // Return the table as a string
  return table.outerHTML;
}

// Example usage:
var companies = `+getVariable("Companies")+`;

document.body.innerHTML = generateCompanyTable(companies);
`
```

{: .key }

We added a backtick (`) at the beginning and end of the JavaScript, and then changed this line so that it executes on the server before passing the resulting data to the client browser.

```
    var companies = `+getVariable("Companies")+`;
```

In essence, follow this format to use JavaScript to display your `HTTP Response`

```
`JavaScript_that_will_execute_in_the_client_browser`+
JavaScript_that_executes_on_the_server
+`more_JavaScript_that_will_execute_in_the_client_browser`
```