

HTTP Requests and Browser Utilities

This page covers:

- [fetchHttpRequest](#) — send an HTTP request
- [Promises](#) — handle results that arrive later
- [binaryUriEncode](#), [htmlCreateBlobUrl](#), [htmlRevokeBlobUrl](#) — turn data into URLs
- [modelCallJS](#) — call a JavaScript function on the page
- [initWebUiLibrary](#)

Promises

Some functions, such as `fetchHttpRequest`, cannot return their result at once. They return a **promise**: an object that receives the result later. Attach a callback function to it.

Member	Description
<code>p.then(callback)</code>	Calls <code>callback(value)</code> when the promise completes. Returns a new promise for the callback's return value, so you can chain calls. <code>next</code> is another name for <code>then</code> .
<code>p.catch(callback)</code>	Calls <code>callback(error, stackTrace)</code> if the promise fails. Returns a new promise. <code>catchError</code> and <code>error</code> are other names for <code>catch</code> .

The callback must be a function. Otherwise `then` and `catch` return `null`.

Callbacks run after the script ends

The script does not wait for a promise. By the time a callback runs, the Script block has already set its **output** and **print** properties, so:

- the value returned by the callback does not reach the block's **output**;
- `print` in a callback is not shown.

To use the result, write it to the page model inside the callback, for example `@parent.label.text = value` or `tableSet(@parent.table, ...)`.

createPromise

```
createPromise([value])
```

Returns a promise that completes with `value` (or `null`). Use it to return a promise from a function in

every case, or to run code after the current script.

```
createPromise(21)
  .then(function(v) { return v * 2; })
  .then(function(v) { @parent.result.value = v; }); // sets 42
```

fetchHttpRequest

`fetchHttpRequest(url, [options])`

Sends an HTTP request from the browser and returns a [promise](#).

`options` is an object with these optional fields:

Field	Default	Description
method	"GET"	The HTTP method, such as "POST", "PUT", or "DELETE".
headers	none	An object of request headers, such as {"Content-Type": "application/json"}.
data	none	The request body: a string (sent as UTF-8) or a list of bytes. Ignored for GET.
binary	false	If true, the response body is a list of bytes instead of a string.
cookies	false	If true, the browser sends cookies with a cross-origin request.
enableCSRF	true	Adds the DGLux server's CSRF headers to the request. Set to false for third-party servers that do not allow extra headers.

When the response has a status from 200 to 299 (or 304), the promise completes with an object:

- `status` — the HTTP status code;
- `data` — the response body, as a string (or bytes, with `binary: true`);
- `headers` — an object of response headers.

For any other status, or a network error, the promise fails; handle it with `catch`.

The request is sent by the browser, so the normal browser rules apply: the server must allow cross-origin requests (CORS) unless it is the DGLux server itself.

```
fetchHttpRequest("https://api.example.com/sensors", {enableCSRF: false})
  .then(function(resp) {
    list = JSON.parse(resp.data);
    tableSet(@parent.table, list, ["id", "name", "value"]);
  })
  .catch(function(err) {
    @parent.errorText.value = String(err);
  });
```

Send JSON with POST:

```
body = JSON.stringify({name: "pump1", on: true});
fetchHttpUrl("/api/commands", {
  method: "POST",
  headers: {"Content-Type": "application/json"},
  data: body
}).then(function(resp) {
  @parent.status.value = resp.status;
});
```

See [JSON and XML](#) for `JSON.parse` and `JSON.stringify`, and [tableSet](#).

binaryUriEncode

`binaryUriEncode(bytes, mimeType)`

Returns a `data:` URL that contains bytes (a list of byte values, for example a binary `fetchHttpUrl` response), encoded as Base64. Returns `null` if bytes is not a list or there are not exactly two arguments.

Use it to show a downloaded image in an Image component.

```
return binaryUriEncode([72, 105], "text/plain");
```

Output: `data:text/plain;base64,SGk=`

```
fetchHttpUrl("/images/logo.png", {binary: true}).then(function(resp) {
  @parent.image.src = binaryUriEncode(resp.data, "image/png");
});
```

htmlCreateBlobUrl

`htmlCreateBlobUrl(data)`

Stores data in the browser and returns a `blob:` URL for it. `data` is a string (stored as UTF-8) or a list of bytes. Returns `null` for other values.

A blob URL is shorter than a `data:` URL, so it is better for large files. The URL is valid only on the current page. Call [htmlRevokeBlobUrl](#) when you no longer need it, to free memory.

```
csv = "name,value\npump1,3\n";
@parent.downloadLink.url = htmlCreateBlobUrl(csv);
```

htmlRevokeBlobUrl

`htmlRevokeBlobUrl(url)`

Frees a URL created by `htmlCreateBlobUrl`. Returns `null`.

```
htmlRevokeBlobUrl(@parent.downloadLink.url);
```

modelCallJS

`modelCallJS(functionName, [args])`

Calls the global JavaScript function `window[functionName]` with the values in the list `args`, and returns its result. Define the function in [custom JavaScript](#) or a library loaded by the page.

Arguments are converted to JavaScript: numbers, strings, and booleans as they are; lists and objects as JavaScript arrays and objects; tables as `{cols: [...], rows: [...]}`. Other values become `null`.

The result is returned as is. A numeric, string, or boolean result can be used directly. If the result is a JavaScript object with `cols` and `rows` arrays, it is converted to a DGLux table.

```
// In the page's JavaScript: window.addTax = function(price, rate) { return  
price * (1 + rate); };  
return modelCallJS("addTax", [100, 0.5]);
```

Output: 150

initWebUiLibrary

`initWebUiLibrary()`

Loads the DGLux web UI widget library (`lib/dcUtilities/widget.js`). Call it before you use components or JavaScript that depend on that library. Returns `null`.

```
initWebUiLibrary();
```

From:

<http://wiki.dglogik.com/> - **DGLogik**

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscrip:http_and_utilities

Last update: **2026/10/22 16:21**

