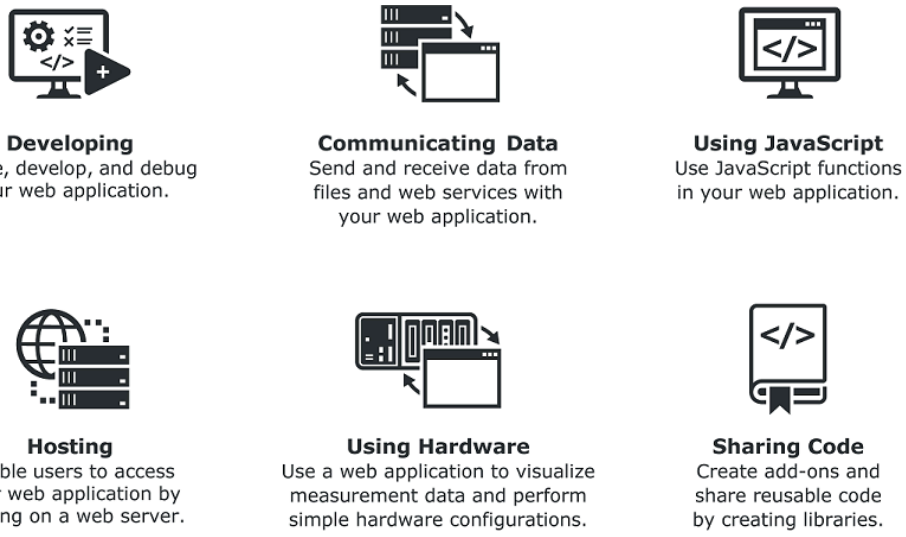
LabVIEW NXG Web Module

2025-03-20

Contents

LabVIEW NXG 5.1 Web Module Manual

Use the LabVIEW NXG Web Module to create browser-based user interfaces to visualize data from distributed systems. This manual contains step-by-step instructions for working with WebVIs and creating web applications.



- Developing—Create a web application, design responsive UIs and controls, and debug, build, and package your WebVIs.
- Communicating Data—Access WebVI resource files, web service data, and use web services like SystemLink Tag and Message in your web application.
- Using JavaScript—Add JavaScript functions to your web application using a JavaScript Library Interface (JSLI) document and create custom UI elements.
- Hosting—Learn about your hosting options during development and how to host your web application to share with users.
- Using Hardware—Determine how to integrate your web application into your hardware system to share measurement data and interact with your hardware.
- Sharing Code—Build a library and add custom palettes to share code and create add-ons.

Tip To access examples, on the **Learning** tab, click **Examples** » **Programming WebVIs**.

Manuals for Related Products

- NI Web Server
- SystemLink
- SystemLink Cloud

LabVIEW NXG Web Module 5.1 New Features and Changes

LabVIEW NXG Web Module 5.1 adds new functionality to the previous release.

New Features

- Use the configuration pane to configure high color, low color, and fit type for intensity graph scales.
- Programmatically set cursor style, cursor shape, and crosshair style of graphs and charts.
- Programmatically read and write disabled indexes in a Listbox when running a VI.
- Programmatically read and write disabled items in a Tree when running a VI.
- Programmatically read and write graph axis names.
- Programmatically read and write column headers of a data grid.

Developing a Web Application

Learn about the different tasks you complete to develop a web application using LabVIEW NXG Web Module.

- Create a web application—Write a program to run in a web browser.
- Create user interfaces—Design responsive interfaces and customize controls for your web application.
- Debug WebVIs—Learn about strategies to debug your WebVIs.
- Build a web application—Build and view your application in a web browser
- Link WebVIs in a web application—Link multiple WebVIs to create multiple web

pages.
- Package a web application—Learn how to configure a package or package installer for your web application.

Creating a Web Application

Use the web application project template to create an application that can run in a web browser.

1. In the Projects tab, click **Web Application Project**, name your project, and click **Create**.

   A web application project includes the following parts:

   2. **Web application document (.gcomp)**—The container for all the files in your application.
   3. **WebVI (.gviweb)**—A specialized VI that generates HTML, JavaScript, and CSS files after you build a web application. Each WebVI marked as top-level in your application generates one HTML file, which corresponds to one web page after you build your web application.
   4. **Other files**—(Optional) Any other files you want to include in your web application, such as CSS, JavaScript, and image files.
   5. **Web Server target**—The item in SystemDesigner that represents the web server that hosts the web application.
6. Open or create a WebVI within the application document and customize the WebVI for your unique programming goals.
7. (Optional) If you want to customize the appearance of your WebVI or add interactive functionality, create CSS and JavaScript files and include them in your application.
8. Test and debug your code.
   Select **Run** » **Run in browser** to test your WebVI in a browser.

   Note Unlike desktop VIs, WebVIs do not currently support debugging tools, such as execution highlighting, probes, and breakpoints.
9. Repeat steps 2–4 to create additional WebVIs.

Once you finish creating and testing your application, build it into a web application that can run in a web browser.

Related tasks:

- Building a Web Application
- Debugging a WebVI

Designing a Responsive User Interface

You can design the user interface of a WebVI to respond to different screen sizes.

Suppose you want your WebVI to run on both a computer monitor and a tablet device; or you want the WebVI to run on a mobile phone that users expect to scroll only vertically. Instead of creating a user interface for each use case, you can design a single user interface that adapts the panel and its contents to the screen size.

1. Create a WebVI within your application document.
2. On the Panel tab, set **Panel layout** to **Flexible**.
3. Create containers by placing objects on the panel.
   - On an empty panel, place a control or indicator anywhere on the panel to create a container.
   - Place a control or indicator outside an existing container to create a new container. You can create multiple containers in a column or in a row.

   A container appears as a border when you hover over objects on the panel.
4. Select a container and configure the layout options in the **Layout** section of the Item tab to determine how objects in the container arrange and align when the screen size changes.
5. Select an object and configure the layout options in the **Layout** section of the Item tab to set the object size and whether the object resizes when the screen size changes.
6. Click **File** » **Save all** to save all files.
7. Select **Run** » **Run in browser** to view your application in a web browser. Resize the web browser and ensure the controls and indicators resize appropriately.

To make your web application available to other users, host your build output on a web server that is accessible to other users.

Related tasks:

- Aligning and Arranging Objects in a Responsive User Interface
- Building a Web Application
- Hosting a Web Application on a Server

Aligning and Arranging Objects in a Responsive User Interface

A ***container*** groups together controls and indicators and determines their arrangement on the panel. If the screen size on which you view your application changes, the objects in a container rearrange according to the layout settings of the container. Containers exist only on panels whose layout type is flexible.

Before you begin, create a WebVI whose layout type is **Flexible**.
Select a container and configure the layout options in the **Layout** section of the Item tab to determine how objects in the container arrange and align when the screen size changes.

1. Set the **Direction** to one of the following options:

Option	Description
Row	Arranges objects in a single row as the screen width allows. As the screen width decreases, objects rearrange into multiple rows.
Column	Arranges objects in a single column.

2. Set the **Horizontal alignment** to one of the following options:

Option	Description
Left	Aligns objects to the left of the container.
Center	Aligns objects to the center of the container.
Right	Aligns objects to the right of the container.
Space between	Arranges objects with space between the left and right of objects.
Space around	Arranges objects with space before, between, and after objects.

3. Set the **Vertical alignment** to one of the following options:

Option	Description
Top	Aligns objects to the top of the container.
Center	Aligns objects to the center of the container.
Bottom	Aligns objects to the bottom of the container.
Space between	Arranges objects with space between the top and bottom of objects.
Space around	Arranges objects with space above, between, and, below objects.

Select an object and configure the layout options in the **Layout** section of the Item tab to set the object size and whether the object resizes when the screen size changes.

4. Set the **Width** and **Height** of the object.
5. Set **Flexible resize** to one of the following options:

Option	Description
Do not resize	Does not resize the object.
Resize height and width	Resizes the height and width of the object while maintaining aspect ratio.
Resize width	Resizes only the width of the object.

Note The flexible resize options are not available for all objects, as some objects have fixed width and/or height. For example, text objects do not support flexible resize options. You must enable the **Auto size** property for text objects to resize to the container or disable the **Auto size** property to prevent text objects from resizing.

6. **Optional:** Set **Minimum height** and **Minimum width** of the objects. This prevents the objects from becoming too small to read or use when the screen size decreases.
7. Click **File** » **Save all** to save all files.
8. Select **Run** » **Run in browser** to view your application in a web browser. Resize the web browser and ensure the controls and indicators resize

appropriately.

Related tasks:

- Designing a Responsive User Interface

Customizing the Appearance of Controls in a WebVI

Use ***Cascading Style Sheets*** (CSS) to customize the appearance of controls in a WebVI. You can modify properties such as the font, color, shape, or layout of a control.

Before you begin, complete the following:

- Familiarize yourself with CSS on the Mozilla Developer Network
- Open or create a web application project

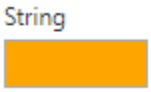
> **Note** If you modify the CSS to customize the appearance of controls, these modifications may not persist from release to release. If you modify the CSS from NI defaults, plan to test your code each time you upgrade to a new version of the G Web Development Software and make manual updates to your CSS styles if necessary.

1. Open the panel of a WebVI and click **Edit HTML Source** (`</>`).
2. In the `<head></head>` tags, add the following lines of code after the `<style ni-autogenerated-style-id=""></style>` tags.

```
<style>
    selector {
        property 1: value 1;
        property 2: value 2;
        ...
        property n: value n;
    }
</style>
```

CSS Item	Description
`selector`	The element you want to customize. You can select an element by its tag name, ID, class, or attribute. Example: `ni-string-control`
`property: value`	The property of the element that you want to customize and the value you want to change it to. Example: `--ni-control-background-color: orange;`

For example, you can use the following code to change the background color of a string control from the default white to orange.




```
<style>
    ni-string-control {
        --ni-control-background-color: orange;
    }
</style>
```

CSS Item	Description
`ni-string-control`	This selector targets every string control in a WebVI.
`--ni-control-background-color: orange;`	This sets the CSS Custom Property named `--ni-control-background-color` to the color orange. The `--ni-control-background-color` CSS Custom Property targets the background color of string controls.

3. Click **OK** and verify your changes.

Search within the programming environment to access the following installed example:

Customize with CSS

Related reference:

- Best Practices for Customizing the Appearance of Controls in a WebVI
- Best Practices for Customizing the Appearance of Controls in a WebVI

Best Practices for Customizing the Appearance of Controls in a WebVI

Use a combination of configuration options in NXG, NI-defined custom properties, and browser-defined Cascading Style Sheet (CSS) properties to customize the controls in your WebVI. By following these best practices, you can create, edit, and maintain the appearance of your controls more easily.

Note If you modify the CSS to customize the appearance of controls, these modifications may not persist from release to release. If you modify the CSS from NI defaults, plan to test your code each time you upgrade to a new version of the G Web Development Software and make manual updates to your CSS styles if necessary.

When to Use CSS Properties

Always use configuration options within LabVIEW NXG instead of NI-defined custom properties or browser-defined CSS properties when configuration options are available. Refer to the following examples of control configurations with LabVIEW NXG Web Module:

- If the control supports the Configuration Pane UI to accomplish the same effect.
- If the control supports Property Node configuration to accomplish the same effect.

When Web Module defines custom CSS properties, such as `--ni-fill-background`, you should use these over other browser-defined CSS properties whenever possible. You can use browser-defined CSS properties, such as `font` or `margin`, to format your controls if NXG configuration options are not available.

Styling Guidelines

Use the following guidelines to style your WebVI controls.

Goal	How to implement
Style the entire page.	Select `body`
Style the panel canvas.	Select `ni-front-panel`
Style a specific control in my WebVI.	Set an HTML class attribute in the IDE and use that class selector in the CSS: `.your_class_selector { --ni-true-background: yellow; --ni-true-foreground-color: #067bc2; }`  **Note** To use an HTML class attribute as a CSS class selector, you must prefix the class attribute with a period in the CSS. In the example above, the HTML class attribute referenced is `your_class_selector`. See W3schools CSS Selectors for more information.
Style a group of controls in my WebVI.	Set multiple HTML class attributes in the IDE and apply the styles to those classes in the CSS: `.activation-progress { font-weight: bold; } .activation-progress.warning { --ni-fill-background: yellow; } .activation-progress.danger { --ni-fill-background: red; }`  **Note** When specifying multiple HTML class attributes in the IDE, the list of classes must be space-delimited. In the example above, the HTML class attribute for the control is `activation-progress`. To make the control bold and yellow, you

Goal	How to implement
	would use the HTML class attribute of `activation-progress warning`. Notice the class is not prefixed with a period and the class names are separated with a space.

For recommendations on how to style specific controls, such as setting border styles on a button or configuring font styles in text boxes, see the ***How do I customize a specific control on my WebVI Panel?*** section of CSS Frequently Asked Questions.

Related tasks:

- Customizing the Appearance of Controls in a WebVI

Debugging a WebVI

Unlike desktop VIs, WebVIs do not currently support debugging tools, such as execution highlighting, probes, and breakpoints. However, you can still debug a WebVI using the following strategies.

Choose between the following debugging options.

Use Case	Instructions
You want to see the error output of diagram objects.	WebVIs support an alternative to automatic error management that logs unhandled errors to the Output tab or browser development console. Because this does not provide the full capabilities of automatic error management, make sure you follow the guidelines for programmatic error management. **Note** To view your error output, create an error indicator and place it on the panel.

Use Case	Instructions
Use this debugging strategy in either of the following cases. • **You need to debug a subVI in your web application.** • **You want to log error information to the console of your web browser.**	To write log error information to the console of your web browser or LabVIEW NXG output window, go to the diagram palette and select **Interoperability** » **Operating System** » **Write to System Log**. To view the information output by **Write to System Log** within the editor, on the **Navigation pane**, open the **Output** tab by clicking **Tool Launcher** » **All** » **Output**. To view the information output by **Write to System Log** in a web browser, use the console of the web browser development tools.
You need to debug your top-level WebVI.	Wire extra indicators at any point in your WebVI code to imitate probes and display the data that flows through a wire. **Note** WebVIs do not currently support G types as controls or indicators on the panel. If you create an indicator from a G type on the diagram, no corresponding indicator is available on the panel.

Related tasks:

- Building a Web Application

Related information:

- Error Management

Building a Web Application

To view your completed web application in a web browser, build your application to generate HTML, CSS, JavaScript, and other files that are included in your application.

Before you begin, open or create a web application project.

1. On the Project Files tab, double-click the application document to open it.
2. In the application document, enable the **Top-level VI** checkbox for each WebVI you want to output as a separate web page.
   Each top-level WebVI in an application document generates one HTML file, which corresponds to one web page.
3. On the Document tab, click **Build**.
   Monitor the status of your build in the Build Queue tab.
4. In SystemDesigner, right-click your application document and select **Run** to view your application in a web browser.

To make your web application available to other users, host your build output on a web server that is accessible to other users.

Related tasks:

- Creating a Web Application
- Hosting a Web Application on a Server

Linking WebVIs in a Web Application

Before you begin, open or create a web application project with multiple top-level WebVIs. Each WebVI that you mark as top-level outputs as a separate web page.

1. On the Project Files tab, double-click the web application document to open it.
2. In the web application document, select the top-level WebVI you want to link to.
   On the Item tab, copy the **Relative URL**.
3. Open the top-level WebVI you want to link from.
4. Switch to the panel and add a Hyperlink Control.
5. Select the Hyperlink Control. On the Item tab, in the **URL** field, paste the relative URL of the WebVI that you want to link to.

   You may need to modify the relative URL depending on how your files are

organized. Consider the following file structure.

```
Application
|
|———One.gviweb
|
|———Two.gviweb
|
|———Namespace_A
|    |
|    |———Three.gviweb
|
|———Namespace_B
     |
     |———Four.gviweb
     |
     |———Namespace_C
          |
          |———Five.gviweb
```

Link from	Link to	Relative URL
One.gviweb	Two.gviweb	Two.html
One.gviweb	Three.gviweb	Namespace_A/Three.html
Three.gviweb	One.gviweb	../One.html
Three.gviweb	Four.gviweb	../Namespace_B/Four.html
Five.gviweb	One.gviweb	../../One.html
Five.gviweb	Three.gviweb	../../Namespace_A/Three.html

6. Build your application and open the HTML output in a web browser to test the link you created.

Search within the programming environment to access the following installed example:

Multiple Top-Level WebVIs

Considerations for Packaging a Web Application

Web applications have unique deployment factors and dependencies you need to know before configuring a package or package installer.

You can package web applications using the same steps as non-web applications. However, you must deploy web applications to a web server. After you add your web application to the package, LabVIEW NXG Web Module automatically adds NI Web Server to the dependencies list. The NI Web Server location is the default root directory for your web application.

Note You can view this file path by opening **SystemDesigner** in your web application project and selecting **Web Server** on the diagram. The file path appears in the configuration pane on the right under **NI Web Server.**

Web Module automatically adds `/<web application name>` to the end of the file path when you add a component to the package document in one of the following ways.

- Right-click the component in the project tree and choose **Create Package/Installer with item**.
- Open a distribution document and choose **Add Files...**.

For web applications that use SystemLink APIs, such as Tags and Messages, you must add those dependencies to the package manually.

Related information:

- Packaging an Application
- Packaging a Library

Communicating Data with a Web Application

Send and receive data from files and web services with your web application.

- Access resource files from a WebVI—Use WebVIs to include CSS, JavaScript, and other files in your web application.
- Access data from a web service—Learn more about sending requests to and receiving responses from a web service, including same-origin and cross-origin

requests, and configuring the LabVIEW Web Service for access to your web application resources.
- Communicate with a web service using a WebVI—Use WebVIs to communicate with web services, such as SystemLink Tags and Messages, on the same network as your web application or cloud web services.

Accessing Resource Files from a WebVI

You can include CSS, JavaScript, images, and other types of files in your web application and access them from a WebVI.

Note For information about including JavaScript files for a JavaScript Library Interface document, refer to the Integrating a JavaScript File Into a Web Application help topic.

Before you begin, create or open a web application project.

1. In the Project Files tab, right-click the application document file (`.gcomp`) and select **Import files**.

   Note You must save the application document before you can import files.

2. Navigate to the file you want to add and click **Open**.
3. Open the WebVI you want to reference the resource file from, and choose one of the following options based on file type.

Type of file	How to reference the file from your WebVI
CSS	In the `<head></head>` tags, add the following lines of code: `<link rel="stylesheet" href="my-style-sheet.css" />` Refer to the Mozilla Developer Network for

Type of file	How to reference the file from your WebVI
	more information on using CSS to customize the appearance of your UI.
JavaScript	In the `<head></head>` tags, add the following lines of code: `<script src="my-script.js"></script>` Refer to the Mozilla Developer Network for more information on using JavaScript to add custom functionality to your web application.
Image	◦ For images that are hosted externally, complete the following steps: a. Switch to the panel and add a URL Image control to the panel. b. On the Item tab, enter the path to your image file in the **Source URL** field. ◦ For images on disk, complete the following steps: a. Import the image into your application document file (`.gcomp`). b. Switch to the panel and drag the image from the Project Files tab onto the Panel.

Related tasks:

- Integrating a JavaScript File Into a Web Application

Retrieving Data From a Web Service

Before you begin, open or create a web application project.

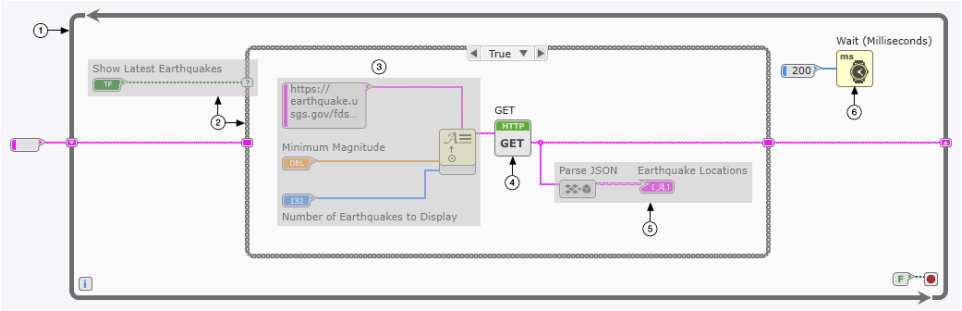
What to Use

- GET
- While Loop
- Wait (Milliseconds)
- Case Structure

What to Do

Create the following diagram to retrieve data from a web service and display that data on the panel.

Customize the gray sections for your unique programming goals.



①	If you want your web application to run continuously in a web browser, place your code in a While Loop with a False constant wired to the `condition` terminal. Otherwise, your web application runs only once in the web browser. **Tip** You can run your web application again by refreshing the page.
②	Place the code that calls the web service within a Case Structure. The Case Structure prevents the GET node from making a request to the web service in each iteration of the While Loop. Making a GET request each time the While Loop iterates is unnecessary because the data you're accessing probably doesn't change as quickly as the loop iterates. Most web services also limit how many requests you can make per second and may even ban your IP address if you make too many requests.
③	Enter the URL of the web service that you want to call. You can replace this code with a string constant containing the URL of a web service. You can also create

	code that programmatically creates a URL based on user input. In this example, Minimum Magnitude and Number of Earthquakes to Display determine what values make up the URL.
④	The GET node sends a request to the web service and returns data from that web service. In this example, the GET node sends a request to the U.S. Geological Survey web service and returns the latest earthquake data.
⑤	Create a subWebVI that parses the data that the web service returns. The web service in this example returns data in the JSON format, which is what most web services return. Other common data formats are XML, CSV, and YML. If a web service returns data in a format other than JSON, you can use other String nodes to parse that data.
⑥	To reduce load and improve performance when you run your built web application on a web browser, add a Wait node to any WebVI that uses an infinite While Loop.

Troubleshooting

If the GET node doesn't return any data or returns unexpected data, verify the following conditions:

- The URL is correct. Test this by navigating to the URL in a web browser.
- The website or web server you want to access is running. Test this by navigating to the URL in a web browser.
- You have a working internet connection.

If each of these conditions is true, try one of the following:

- To check the status of an HTTP GET request, wire an indicator to the `status code` output of the GET node.
- To check detailed information about the HTTP GET request, including its status, wire an indicator to the `headers` output of the GET node.

Refer to the W3C website for more information about HTTP status codes and headers.

Examples

Search within the programming environment to access the following installed examples:

- ***Call a 3rd Party Web Service***
- ***Call LabVIEW Web Service***

Considerations When Accessing Data from Web Services

If you want to create a web application that sends requests to and receives responses from a web service, you need to know the origin of the web service that hosts the web application as well as the origin of the target web service.

For example, a web application hosted at http://localhost:8080/Demo/MyExample.html has the origin http://localhost:8080. A server origin contains three parts:

Part	Definition	Example
Scheme	Protocol the web service uses.	http://
Host	Domain name or IP address of the service.	localhost
Port	TCP port of the web service. If you don't specify a port, the web service uses the default port.	8080

Once you know the origin of the host web service and the target web service, determine whether requests from the web application to the target web service will be same-origin requests or cross-origin requests. If both origins are identical, requests from the web application to the web service are same-origin requests. If there is any difference between the two origins, requests from the web application to the web service are cross-origin requests.

Browsers running web applications do not impose restrictions on same-origin requests. Web applications that perform cross-origin requests are subject to the ***Cross-Origin Resource Sharing*** (CORS) mechanism. By default, a web browser blocks all cross-origin requests made to a target web service that is not configured to support CORS.

Related concepts:

- Types of Cross-Origin HTTP Requests

- Configuring CORS for a LabVIEW Web Service
- Hosting a Web Application During Development

Related tasks:

- Enabling CORS for a Web Service
- Sending a Credentialed Cross-Origin HTTP Request

Types of Cross-Origin HTTP Requests

Cross-origin HTTP requests, either simple or non-simple, determine whether the browser asks the target web service before sending a request.

Simple requests are cross-origin HTTP requests that do not require prior approval from the target web service before the request can be sent by the browser. An HTTP request must meet the following criteria to execute as a simple request:

- Use a GET, HEAD, or POST node to make the request.
- Include only CORS-safelisted request-headers in the request.
- Set the Content-Type request header to one of the following values:
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`

Non-simple requests are cross-origin HTTP requests that must get approval from the target web service to send the actual HTTP request. The web browser sends a CORS ***preflight request*** to the target web service to ask for approval. The response to the CORS preflight request determines if the web browser can proceed to send the actual HTTP request. A cross-origin HTTP request executes as a non-simple request if it violates any of the criteria for a simple request.

CORS also enables ***credentialed requests***. Credentialed requests may use HTTP cookies and HTTP Authentication headers, or allow TLS client certificates. By default, browsers do not include credentials with a cross-origin HTTP request. However, you can use the Configure CORS node in a WebVI to include credentials with a cross-origin request. The Configure CORS node has no effect on same-origin HTTP requests.

Related concepts:

- Considerations When Accessing Data from Web Services

Related tasks:

- Hosting a Web Application on the NI Application Web Server

Enabling CORS for a Web Service

To incorporate resources from a different origin into your web application, configure CORS for the server hosting those resources.

1. Refer to the web service documentation or contact the web service administrator to verify if the web service allows cross-origin requests using CORS.

   > **Note** The target web service must enable cross-origin requests to allow HTTP requests from a WebVI executing in the development environment. Refer to Hosting a Web Application During Development for more information.

2. Determine whether the web service requires credentialed CORS requests.
   If the web service requires credentialed requests, refer to ***Sending a Credentialed Cross-Origin HTTP Request*** to configure your application to send credentialed requests.
3. Ensure the web service administrator includes an `Access-Control-Allow-Origin` header in HTTP responses to enable cross-origin HTTP requests to the web service.

   - For non-credentialed simple CORS requests, set the `Access-Control-Allow-Origin` header to one of the following values:

     - The wildcard value (*) that allows any origin to access a resource. NI recommends using this value only during the development of your application.
     - The origin of the web application performing the HTTP request.

   - For credentialed simple requests, ensure the web service includes the

following header values in responses to the web application:

- `Access-Control-Allow-Origin` header set to the origin of the web application performing the HTTP request.
- `Access-Control-Allow-Credentials` header set to True.

Refer to the Mozilla Developer Network documentation on HTTP access control for additional information on advanced CORS configurations, including:

- Responding to a CORS preflight request during a non-simple CORS request
- Handling headers for browser caches
- Accepting additional request headers from a browser
- Allowing the browser to have access to additional response headers

Related concepts:

- Considerations When Accessing Data from Web Services
- Configuring CORS for a LabVIEW Web Service

Related tasks:

- Sending a Credentialed Cross-Origin HTTP Request

Configuring CORS for a LabVIEW Web Service

You may need different CORS configurations for each LabVIEW Web Service in your application during development and deployment.

Note The following content only applies to LabVIEW Web Services deployed to the LabVIEW Application Web Server. NI recommends deploying LabVIEW Web Services to the NI Web Server instead.

To determine the origin of a web application executing in the development environment, refer to Hosting a Web Application During Development. During development using a LabVIEW Web Service, configure CORS to test your application in the LabVIEW editor.

LabVIEW Web Services only allows simple cross-origin requests and cannot respond to CORS preflight requests from a web browser. Use the following table to find the best CORS configuration for your web application.

Note You do not need to enable CORS if your WebVI and LabVIEW Web Service have the same origin. To achieve this, host the built WebVI in the Public Content Folder of the LabVIEW Web Service the WebVI is making HTTP requests to when running your web application.

Goal	CORS Configuration	Configuration Instructions
Share your web application resources publicly and receive requests from any origin. **Note** This configuration is the least secure. Any origin could access your web service.	• No credentials • Unfiltered Origins	Configure a LabVIEW Web Service CORS for Open Access
Allow specific origins you define or well-known origins to access your web application resources that do not require credentials.	• No credentials • Filtered Origins	Configure a LabVIEW Web Service CORS for Filtered Access
Allow specific origins you define or well-known origins to access your web application resources and you want to enable sharing credentials, such as cookies.	• Credentials • Filtered Origins	Configure a LabVIEW Web Service CORS for Credentialed Access

Related concepts:

- Considerations When Accessing Data from Web Services

Related tasks:

- Configure a LabVIEW Web Service CORS for Open Access
- Configure a LabVIEW Web Service CORS for Filtered Access

- Configure a LabVIEW Web Service CORS for Credentialed Access

Configure a LabVIEW Web Service CORS for Open Access

Configure CORS to allow any origin access to your WebVI resources.

This CORS configuration is the least secure configuration. Any origin can access your web service and application resources. For example, any web page the browser visits could run scripts that make requests to the web service.

Note The following content only applies to LabVIEW Web Services deployed to the LabVIEW Application Web Server. NI recommends deploying LabVIEW Web Services to the NI Web Server instead.

What to Use

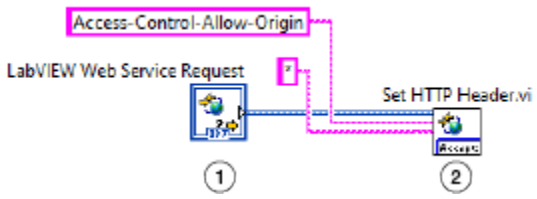
You can find the Web Services API on the Connectivity palette in LabVIEW.

Note LabVIEW NXG does not support creating web services.

- LabVIEW Web Service Request
- Set HTTP Header

What to Do

Create the following diagram in a Web Resources VI to configure a LabVIEW Web Service to allow CORS from any origin.



①	A new Web Resources VI automatically adds the LabVIEW Web Service Request class to the block

	diagram and terminal pane.
②	Set HTTP Header sets the HTTP header value in response to a request. Use the `headerAccess-Control-Allow-Origin` to indicate if the origin making the request can access the response of your Web Service VI. Use wildcard (`*`) as the `header value` to allow any origin to access your WebVI resource.

Troubleshooting

If you encounter errors, try the following troubleshooting strategy:

- Verify the `header` and `header value` are correct.

Configure a LabVIEW Web Service CORS for Filtered Access

Configure CORS to allow specific origins you define or well-known origins access to your WebVI resources.

Note The following content only applies to LabVIEW Web Services deployed to the LabVIEW Application Web Server. NI recommends deploying LabVIEW Web Services to the NI Web Server instead.

What to Use

You can find the Web Services API on the Connectivity palette in LabVIEW.

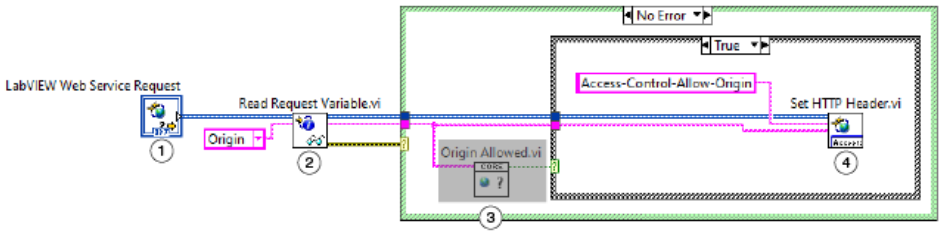
Note LabVIEW NXG does not support creating web services.

- LabVIEW Web Service Request
- Read Request Variable
- Set HTTP Header

What to Do

Create the following diagram in a Web Resources VI to configure a LabVIEW Web Service to allow CORS with specific or well-known origins.

Customize the gray sections for your unique programming goals.



①	A new Web Resources VI automatically adds the LabVIEW Web Service Request class to the block diagram and terminal pane.
②	Read Request Variable checks the request for the variable you define. To check the origin of the request, enter `Origin` for the value of `variable`. Use a Case Structure to create two cases: a case to continue if an origin is found in the request and a case to do nothing if an origin is not found in the request. **Note** Web browsers in a same-origin configuration and HTTP clients outside of web browsers may not have an `Origin` header in the request. These should be handled by the web resource.
③	Define how to filter the origins you want to access your web resources. Some common

	implementations include the following: • Have a strict list of origins to check against • Check for a prefix on the origin, such as http://localhost • Use the LabVIEW Web Service Request and Read Request Variable to check for additional information, such as Remote Address Use a Case Structure to create two cases: a case to continue if the origin passes the filter you implement and a case to do nothing if the origin does not pass the filter.
④	Set HTTP Header sets the HTTP header value in response to a request. Use the `headerAccess-Control-Allow-Origin` to indicate if the origin making the request can access the response of your Web Service VI. The `header value` populates with the origin the filter approves.

Troubleshooting

If you encounter errors, try the following troubleshooting strategies:

- Check the implementation of your filter for errors.
- Verify the `header` is correct.

Configure a LabVIEW Web Service CORS for Credentialed Access

Configure CORS to allow specific origins you define, well-known origins, and origins with credentials access to your WebVI resources.

Note The following content only applies to LabVIEW Web Services deployed to the LabVIEW Application Web Server. NI recommends deploying LabVIEW Web Services to the NI Web Server instead.

What to Use

You can find the Web Services API on the Connectivity palette in LabVIEW.

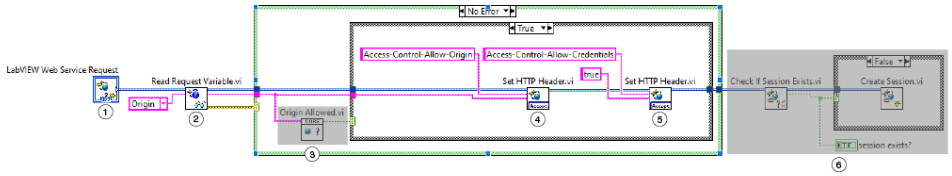
Note LabVIEW NXG does not support creating web services.

- LabVIEW Web Service Request
- Read Request Variable
- Set HTTP Header

What to Do

Create the following diagram in a Web Resources VI to configure a LabVIEW Web Service to allow CORS with specific or well-known origins and origins with credentials.

Customize the gray sections for your unique programming goals.



1	A new Web Resources VI automatically adds the LabVIEW Web Service Request class to the block diagram and terminal pane.
2	Read Request Variable checks the request for the variable you define. To check the origin of the request, enter `Origin` for the value of `variable`.

	Use a Case Structure to create two cases: a case to continue if an origin is found in the request and a case to do nothing if an origin is not found in the request. **Note** Web browsers in a same-origin configuration and HTTP clients outside of web browsers may not have an `Origin` header in the request. These should be handled by the web resource.
③	Define how to filter the origins you want to access your web resources. Some common implementations include the following: • Have a strict list of origins to check against • Check for a prefix on the origin, such as http://localhost • Use the LabVIEW Web Service Request and Read Request Variable to check for additional information, such as Remote Address Use a Case Structure to create two cases: a case continue if the origin passes the filter you implement and a case to do nothing if the origin does not pass the filter.
④	Set HTTP Header sets the HTTP header value in response to a request. Use the `headerAccess-Control-Allow-Origin` to indicate if the origin making the request can access the response of your Web Service VI. The `header value` populates with the origin the filter approves.

	Note The wildcard (`*`) value is not valid for credentialed cross-origin requests. You must choose a filtered origin.
⑤	Set HTTP Header sets the HTTP header value in response to a request. Use the `headerAccess-Control-Allow-Credentials` to indicate if the origin making the request can access the response of your Web Service VI and accept credentialed information, such as cookies. Use the `header valuetrue` to allow the filtered origin to access your Web Service VI. You must make a corresponding change in the Configure CORS node in your WebVI. See Sending a Credentialed Cross-Origin HTTP Request for more information.
⑥	Add code that only functions if the origin has supported credentials defined in the request. The code above uses session VIs that require browser cookies to function.

Troubleshooting

If you encounter errors, try the following troubleshooting strategies:

- Check the implementation of your filter for errors.
- Verify the `header` is correct for each Set HTTP Header.
- Check the implementation of the Configure CORS node in your WebVI.

Related tasks:

- Sending a Credentialed Cross-Origin HTTP Request

Sending a Credentialed Cross-Origin HTTP Request

Enable credentials in your WebVI, such as including and storing HTTP cookies, adding HTTP Authorization headers, or permitting TLS client certificates to interact with a web service.

In addition to configuring a web service to support cross-origin credentialed requests, configure the client handle to include credentials during a CORS requests.

Note Same-origin requests do not have to perform CORS configuration on the client handle of a WebVI to include credentials in a request. During a same-origin request the result of using the Configure CORS node is ignored.

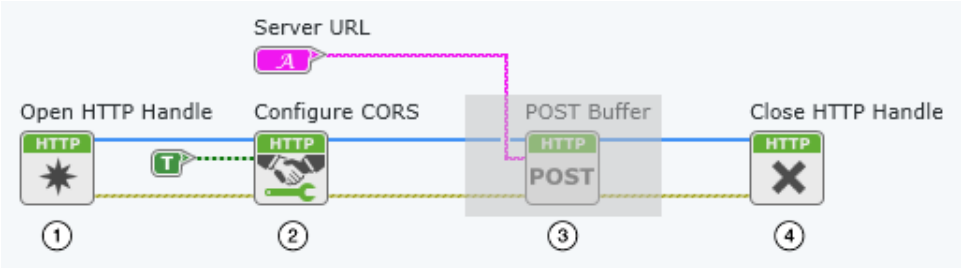
What to Use

- Configure CORS
- Open HTTP Handle
- Close HTTP Handle

What to Do

Create the following diagram to send a credentialed cross-origin HTTP request.

Customize the gray sections for your unique programming goals.



①	Open HTTP Handle creates a client handle that

	preserves the headers you want to add to HTTP requests the application sends.
②	Configure CORS adds configuration information, such as whether credentials should be included in CORS requests, to the client handle.
③	Configure the HTTP node to access resources for your application.
④	Close HTTP Handle closes the client handle and deletes any authentication credentials and HTTP headers associated with the handle. **Note** The browser cache independently stores and manages cookies a credentialed CORS request or same-origin request creates. Closing the client handle does not affect the cookies in the browser cache.

Troubleshooting

Some servers may require that your application not include credentials with a cross-origin HTTP request. If the server you send requests to responds with the `Access-Control-Allow-Origin` header set to `*`, the wildcard value, the HTTP request fails with a CORS configuration error. To resolve the error, set the `include credentials during CORS` input of Configure CORS to False. To enable credentialed access for a LabVIEW Web Service, see Configure a LabVIEW Web Service CORS for Credentialed Access.

Related concepts:

- Considerations When Accessing Data from Web Services
- Configuring CORS for a LabVIEW Web Service

Communicating Data with Web Services Using WebVIs

You can communicate with web services on the same network and cloud web services using LabVIEW NXG WebVIs.

Use the following tables to choose the best communication option for your needs.

Table 1. Communicating with Web Services on the Same Network

Goal	Web service to use	How to implement
Track a piece of data, such as a hardware measurement.	Tags Service	Transferring Data Using Tags
Send commands, status updates, and data between web applications and servers.	Messages Service	Sending Messages Between Systems
Share waveform data stored in a TDMS file.	TDM Reader API Service	Reading Measurement Data from TDMS Files
Host a WebVI to make it accessible from web browsers.	NI Web Server	Hosting a Web Application on the NI Web Server
Create a custom HTTP REST API to interface with an existing system, such as a database.	LabVIEW Web Service	Creating and Accessing a LabVIEW Web Service **Note** Download the latest LabVIEW Help for the most up-to-date content.
		Configuring CORS for a LabVIEW Web Service
		Hosting a Web Application on the LabVIEW Application Web Server
Stream data with low latency.	3rd party WebSocket service **Note** LabVIEW and LabVIEW NXG do not support listening for WebSocket connections at this	WebSocket G APIs

Goal	Web service to use	How to implement
	time.	

Table 2. Communicating with Cloud Web Services

Goal	Web service to use	How to implement
Share tags and messages securely over the internet.	SystemLink Cloud Tags and Messages	Connecting to SystemLink Cloud from LabVIEW NXG Web Module
		Sharing Data Across Systems
Host a WebVI securely on the internet.	SystemLink Cloud Visualizations	Hosting a Web Application on SystemLink Cloud
Make local tags and messages available securely on the internet.	SystemLink Cloud Connector	Connecting to SystemLink Cloud

Related concepts:

- Considerations When Accessing Data from Web Services

Related tasks:

- Retrieving Data From a Web Service
- Hosting a Web Application on a Server

Using JavaScript with a Web Application

Call JavaScript functions in your web application and use JavaScript to create UI elements.

- Calling JavaScript functions in your web application—Use a JavaScript Library Interface (JSLI) to create entry points to JavaScript functions in your web application.
- Creating UI Elements with JavaScript—Create UI elements to place on the panel of

your web application using the Placeholder HTML Container and JavaScript Library Interface document (JSLI).
- JavaScript Resources—Research JavaScript concepts using NI-recommended resources.

Calling JavaScript Functions in a Web Application

Use a JavaScript Library Interface (JSLI) to call globally accessible JavaScript functions in your web application.

A JSLI is a document in which you create ***entry points***, or defined calls, to JavaScript functions.

When you create a new entry point, a visual representation of the entry point appears on the software palette of the diagram. You can place and wire entry points like nodes.

As the web application executes, the entry point calls the JavaScript function. Input data passes from the diagram to the JavaScript function, and output data returns from the JavaScript function to the diagram.

Complete the following tasks to call JavaScript functions in your web application:

1. Prepare your code(Optional)—Create wrapper code for the JavaScript file you want to use.
2. Create a JSLI—Define the JavaScript functions you want to call in your web application.
3. Integrate the JavaScript file into your web application—Connect the JavaScript file to your web application and create calls to the JavaScript functions you defined in the JSLI.
4. Call an asynchronous JavaScript function(Optional)—Add code to your JavaScript file to use asynchronous JavaScript calls in your WebVI.

Examples

Search within the programming environment to access the following installed example:

- ***Call JavaScript From a WebVI***

After you create a JSLI and add entry points to your web application, if necessary, troubleshoot common issues with a JSLI.

Preparing Your Code For Use With a JavaScript Library Interface

Create wrapper code, if necessary, to make your JavaScript functions compatible with the JavaScript Library Interface (JSLI).

Before you begin, create a web application and acquire a JavaScript library or find a global, built-in browser function you want to use. For more information on JavaScript concepts, refer to JavaScript Resources.

1. Determine if you need to create wrapper code.
   When calling JavaScript functions in your web application, you may need to create wrapper code to make your JavaScript code compatible with the JSLI. The following table contains common examples of when to create wrapper code and descriptions of the wrapper code to create.

Situation	Solution
You need to call functions that use or return unsupported data types. JSLIs currently support the following data types: ◦ booleans ◦ strings ◦ numerics ▪ 8-bit signed integer ▪ 16-bit signed integer ▪ 32-bit signed integer ▪ 8-bit unsigned integer ▪ 16-bit unsigned integer ▪ 32-bit unsigned integer ▪ Single-precision, floating-point numeric ▪ Double-precision, floating-point numeric ◦ JavaScript references ◦ 1D array of numerics (represented as a TypedArray)	Create wrapper functions that convert between unsupported JavaScript types and types supported by the JSLI.

Situation	Solution
◦ 1D array of JavaScript references (represented as Array of JavaScript values)	
You need to use the `new` operator to create an instance of an object or you need to invoke a method on an instance of an object.	Complete the following steps: a. Create a wrapper function that invokes the `new` operator and stores a reference to the new object instance. b. Create a wrapper function that looks up the reference to that object and invokes a method.

2. Create wrapper code for your JavaScript library. NI recommends you create one JavaScript file per JSLI document that contains all the wrapper functions you create.

   Model the following code as a best practice for creating wrapper code. Customize the following code for your unique programming goals.

```
              /*1*/
(function () {
   'use strict'; /*2*/

    var counter = 1; /*3*/

    /*4*/
    var logWithCount = function (message){
      console.log(counter + ' > ' + message); /*5*/
      return counter++; /*6*/
    };

    window.logWithCount = logWithCount; /*7*/
}());
```

①	Wrap the contents of the entire wrapper JavaScript file in an immediately invoked function expression (IIFE). The IIFE creates a new lexical scope which prevents you from unintentionally adding objects to the global scope.
②	Configure the IIFE to use JavaScript strict mode. Strict mode improves error handling and allows browsers to make certain optimizations.

③	Create a private variable by declaring it within the IIFE. In this example, the private variable is named `counter`. Because this variable is private, it cannot be unintentionally modified by other code. This variable is not accessible to the JSLI because you did not add it to the global scope.
④	Create a new function and add the desired parameters to the function. In this example, we create a function named `logWithCount` that passes a parameter to a built-in browser function named `console.log`.
⑤	Call the function you are wrapping, in this example, the `console.log` function.
⑥	Return a value from your wrapper function. If you configure the return parameter in the JSLI document to be of a type other than void, you must pass a return value of the type you configured in the JSLI document to the return statement.
⑦	Add the function you want to call in your web application to the global scope to allow the JSLI document to access the function. In this example, we use the `logWithCount` function. In a web browser, the name of the global object is `window`, so the code `window.logWithCount = logWithCount;` places `logWithCount` on the global scope.

Related tasks:

- Defining Calls to JavaScript Functions using a JavaScript Library Interface

Defining Calls to JavaScript Functions using a JavaScript Library Interface

To call JavaScript functions in your web application, define calls to the JavaScript functions using a JavaScript Library Interface (JSLI) document.

Before you begin, create or open a web application project and prepare your JavaScript code for use with the JSLI. For more information on JavaScript concepts, refer to JavaScript Resources.

Note For the JSLI to access a JavaScript function, the function must be accessible from the global scope.

To create and configure a JSLI document, complete the following steps:

1. On the **Project Files** tab, add a JSLI to a web application component or library component in your project.
   A JSLI document appears in your web application.
2. Define the prototype for the JavaScript function you want to call. A defined call to a JavaScript function in a JSLI is called an ***entry point***.
   a. In the JavaScript global form field, enter the name of the JavaScript function you want to call. The name you enter must exactly match the function name in the JavaScript code. You must specify all functions relative to the global scope. Use dot notation to refer to nested objects. For example, if you want to call the absolute function abs in the object Math, you must specify `Math.abs` in the JavaScript global form field.

      
      Note The JavaScript global form field is case sensitive.

   b. Click **Add function** to create an entry point. The default entry point name is created from the JavaScript global.
   c. Click **Add parameter**.
   d. On the **Item** tab, specify the parameter name, data type, and JavaScript representation.
      Consider the following information when you add parameters to your prototype:
      - You can choose any name for the parameters you add to a JSLI document, but you must add them in the same order as they appear in the JavaScript function.
      - The parameter and return data types you specify in the JSLI document must match the parameter and return data types in the JavaScript function.
      - The parameter in the top position in the JSLI document is always the return type.
3. Repeat step 2 to create as many entry points as you need for your project.

Related tasks:

- Waiting for Asynchronous JavaScript Operations to Complete in a Web Application
- Debugging JavaScript Library Interfaces

Integrating a JavaScript File Into a Web Application

Connect a JavaScript file to your web application and call functions you defined in a JavaScript Library Interface (JSLI) in your web application.

Before you begin, create and configure a JLI.

1. Import the JavaScript file(s) you want to use into your web application.
   a. Save your project.
   b. On the **Project Files** tab, right-click a web application document or library component document (`.gcomp`) and select **Import files**.
   c. Navigate to the JavaScript file(s) you want to add and click **Open**. Click **Copy** to close the Copy existing file(s) dialog box.
2. Integrate the JavaScript file into your web application.
   a. Open the JSLI document associated with the JavaScript file and enter the relative path to the JavaScript file in the HTML script and link dependencies form field.
   Example:

   ```
   <script src="library.js"></script>
   ```

   > **Note** The JavaScript file must be in the same component as the JSLI document and must be marked as **Always include** in the component document.

   b. Click **Apply Changes**.
3. Add the entry points you defined in the JSLI to any web application in your project.
   a. Open the WebVI in which you want to call the JavaScript function.
   b. On the diagram palette, click **Project Items** » **Software** » **<YourWebAppName>**. Click the folder with the name of your JSLI document to show each entry point you defined in that JSLI.
   c. Drop the entry points you want to use on the diagram.
   d. Wire the entry points and complete the diagram.
   e. Run the WebVI. Input data passes from the diagram to the JavaScript code, and output data returns from the JavaScript code to the diagram.

Waiting for Asynchronous JavaScript Operations to Complete in a Web Application

Use asynchronous JavaScript code to wait on tasks or requests while the rest of the

code in your WebVI continues to execute.

Synchronous code means that your code executes in order. Asynchronous code means that a portion of your code waits for a task to complete while the rest of your code continues to execute. For more information on JavaScript concepts, refer to JavaScript Resources.

What to Use

- External JavaScript file
- JavaScript Library Interface (JSLI) document

What to Do

1. Create the following code in a JavaScript (.js) file to use an asynchronous JavaScript function in your web application to run two loops in parallel. Customize the following code for your unique programming goals.

```
(function () {
    'use strict';

    /*1*/
    const synchronousDivide = function (numerator, denominator) {
        if (denominator === 0) {
            throw new Error('Cannot divide by zero');
        } else {
            return numerator / denominator;
        }
    };

    const sleep = function (time) {
        return new Promise(function (resolve) {
            setTimeout(resolve, time);
        });
    };

    /*2*/
    const asynchronousDivide = async function (numerator, denominator) {
        await sleep(1000); /*3*/

        /*4*/
```

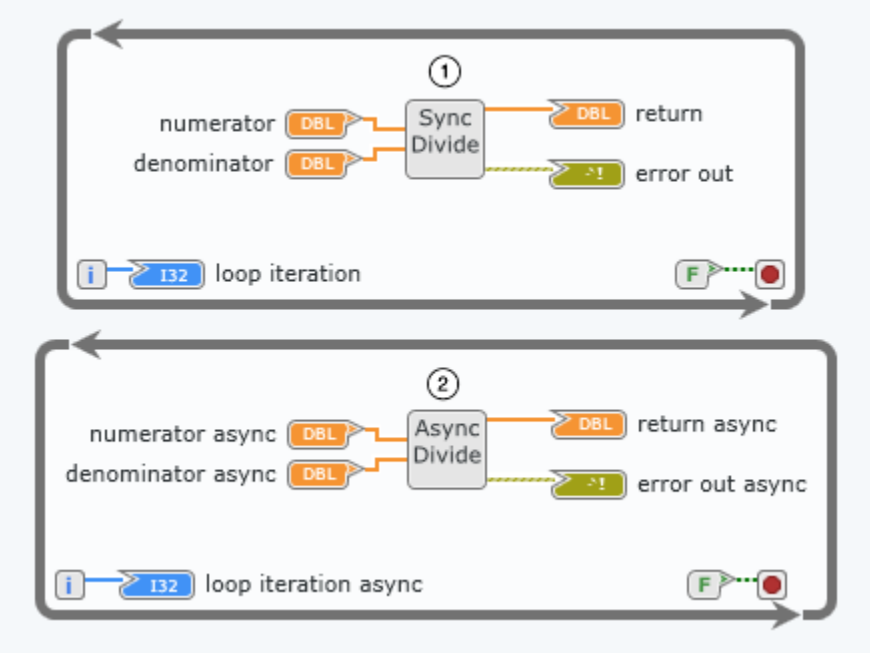
```
        if (denominator === 0) {
            throw new Error('Cannot divide by zero');
        } else {
            return numerator / denominator;
        }
    };

    /*5*/
    window.synchronousDivide = synchronousDivide;
    window.asynchronousDivide = asynchronousDivide;
}());
```

①	Create a JavaScript function that completes synchronously named `synchronousDivide`. **Note** To notify G Web Development Software that a JavaScript error has occurred in a synchronous function, you must throw an error in your JavaScript code. In your diagram code, the `error out` output on the JSLI node returns an error.
②	Create an async JavaScript function named `asynchronousDivide`. An async JavaScript function results in a JavaScript Promise that notifies G Web Development Software that the function will run asynchronously.
③	Add code that runs asynchronously. This example uses the sleep function which asynchronously pauses execution for 1000 milliseconds.

④	Return a value or throw an error from the async function. When using an async JavaScript function for asynchronous programming, returning a value or throwing an error uses syntax similar to synchronous programming.
⑤	Add the functions you want to call in your web application to the global scope to make the functions accessible to the JSLI document. In this example, we place both the `synchronousDivide` and `asynchronousDivide` functions on the global scope.

2. Create and configure a JSLI document.
3. Integrate the JavaScript file into your web application.

4. Create the following diagram to use the asynchronous JavaScript function to execute two loops in parallel in your WebVI.



①	Run the WebVI. The Sync Divide JSLI node executes continuously without waiting.
②	The Async Divide JSLI node waits 1000 milliseconds before continuing execution. Because the `asynchronousDivide` function executes asynchronously in your JavaScript code, the second loop doesn't block execution of the first loop. The Async Divide JSLI node behaves synchronously in your diagram code, so you do not need to wait on the asynchronously executing JavaScript code manually in your diagram code.

Related tasks:

- Defining Calls to JavaScript Functions using a JavaScript Library Interface
- Integrating a JavaScript File Into a Web Application
- Debugging JavaScript Library Interfaces

Debugging JavaScript Library Interfaces

Solve common problems that may occur when configuring and using JavaScript Library Interfaces (JSLIs).

Note Most errors occur due to mismatches between the JSLI configuration and the functions in the JavaScript library. Mismatch errors may not occur at the exact time the JavaScript function call executes on the diagram.

Use the following table to resolve common errors.

Issue	Solution
You receive a **`Function not found`** error.	Verify the JSLI uses the correct spelling and case

Issue	Solution
	sensitivity for the entry points you defined.
The JSLI node does not appear in the software palette on the diagram.	Check both of the following items. • On the **Project Files** tab, the JSLI document is in a web application document. • On the **Project Files** tab, double-click the web application document. Ensure the **Select target** drop-down box is set to Web Server.
The changes you make to the HTML source panel do not take affect.	Verify your HTML is valid.
You do not get the results you expected from your entry point.	Verify that the parameters are in the correct order in the JSLI document.
Your function does not appear to be called.	Add temporary calls to `console.log('my custom message')` to your JavaScript code to print a custom message to the console when it is called. **Note** To view messages printed to the console, see Debugging a WebVI.

Creating UI Elements with JavaScript

Create UI elements to place on the panel of your web application using the Placeholder HTML Container and JavaScript Library Interface document (JSLI).

You can add custom UI elements, such as controls, to your web application by creating a placeholder on the panel. Using the placeholder, you can set the dimensions and other properties of the element that work for both flexible and absolute layouts. You then create a control reference and wire the reference into a JSLI, directly exposing the underlying HTML for the control to your JavaScript code.

Before you begin, create or open a web application.

What to Use

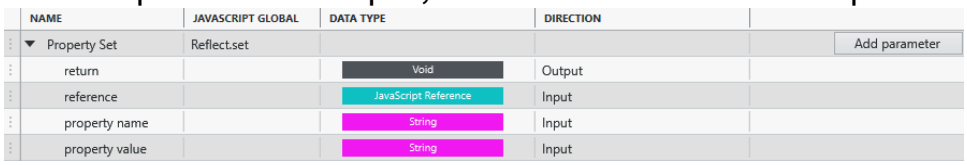
- Placeholder HTML Container
- Obtain JavaScript Reference

What to Do

1. Open or create a JSLI and add the JavaScript function you want to call from your web application. Add the parameters with one input having a data type of **JS Reference**.

   > **Note** In order to directly access your HTML element, your function must have an input with a matching data type of JS Reference. You can also include an array of JS references.

   In the following example, the JavaScript function is **Reflect.set**, a function available in most browsers, with the name **Property Set**. In addition to the JavaScript Reference input, there are two additional inputs for the function.

   > **Note** You can rename the function and properties, however the JavaScript Global function name must be the name of the function you are referencing.

2. Open or create a WebVI (`.gviweb`) and on the panel, select **Decorations** » **Placeholder HTML Container** and place it on the panel.
3. Use the Configuration pane on the right to configure the container.

   > **Note** Use the Configuration pane or property nodes to configure the HTML container instead of injecting HTML to configure the placement and sizing.

   The container is empty by default and you can populate it with a control at

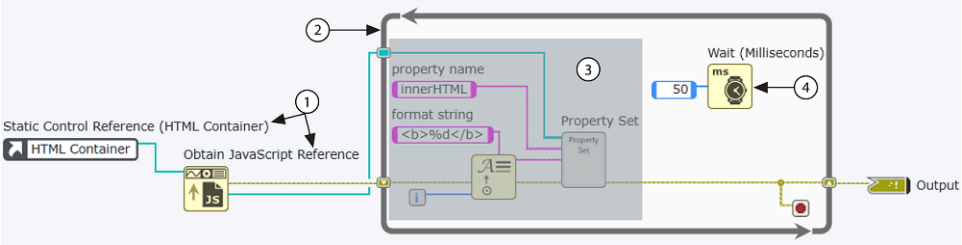
runtime. If you use the Configuration pane to add a placeholder image, the placeholder image is visible in the container at edit time. Select **Image visible at runtime** to make the placeholder image also visible at runtime until the contents of the container are available.

4. With the **HTML Container** selected, click **Create reference** in the Configuration pane.

   The **HTML Container** does not have a terminal, so you must create a reference to interact with it on the diagram.

5. Create the following diagram to populate the HTML container on the panel with the JavaScript function you defined in the JSLI.

   Customize the gray sections for your unique programming goals.



①	The Static Control Reference represents the **HTML Container** you placed on the panel. **Tip** You can wire the HTML container to a property node to configure the size and position of your control. Obtain JavaScript Reference creates a JavaScript reference with a value of the HTML element within the container. **Note** The references do not have controls and are only visible on the diagram. You can wire references to terminals on a connector pane.
②	If you want your web application to run continuously in a web

	browser until it encounters an error, place your code in a While Loop with the conditional terminal wired to the `error out` parameter of your JavaScript function. In this example, the Property Set JavaScript function is wired to the conditional terminal. If Property Set returns an error, the conditional terminal will stop the loop from executing.
③	The JavaScript function uses the JavaScript reference from Obtain JavaScript reference. You can configure the function using the properties you defined in the JSLI document along with any other supporting code. In this example, the Property Set function outputs the number of loop iterations by converting the iteration counter output into a string using the HTML `<b>` tag.
④	To reduce loading time and improve performance when you run your built web application on a web browser, add a Wait node to any WebVI that uses an infinite While Loop.

Troubleshooting

Ensure you properly define the JavaScript calls you make in the JSLI. Refer to the following resources for more information.

- Defining Calls to JavaScript Functions using a JavaScript Library Interface
- JavaScript Resources

Related concepts:

- Considerations When Creating UI Elements with JavaScript
- JavaScript Reference Functionality

Considerations When Creating UI Elements with JavaScript

JavaScript UI elements may require special sizing considerations to ensure they appear correctly after you deploy your web application or share the UI element in a

library.

The size of the UI element you create can change if your web application has a flexible layout or if your user configures the element using property nodes. Consider using the following strategies to ensure the UI element retains the appropriate size to fit in the Placeholder HTML Container.

- Configure with CSS—Use a style sheet to set the size of the UI element.
- Configure with the ResizeObserver API—Use the browser-based ResizeObserver API to make adjustments programmatically.

> **Note** The ResizeObserver API is supported in most browsers. Ensure your browser supports this API before implementing it in your code. Refer to the Mozilla Developer Network documentation for more information about the ResizeObserver API.

Related concepts:

- Creating UI Elements with JavaScript

JavaScript Reference Functionality

JavaScript references behave similarly to references for other data types, but have some unique functionality.

When using JavaScript references in your web application, you need to consider the following:

- Primitive values **null** and **undefined**—JavaScript references treat primitive values **null** and **undefined** the same as other primitives, such as strings, booleans, numbers, etc. If a value of null or undefined is returned, it generates a new JavaScript reference that you need to clean up to prevent memory leaks.
- Not a Number/Path/Refnum? behavior—The node tells you if the value is valid, but it does not tell you if the JavaScript reference value is null or undefined. There is not an existing node to check for values of null or undefined.
- JavaScript values—JavaScript references can hold any JavaScript value, including primitives and objects, such as JavaScript functions, HTML elements, and class

instances.
- Type definitions containing a JavaScript reference—JavaScript references do not have front panel representations, so you will need to use the following process to create a type definition containing a JavaScript reference:
  1. Create a VI (.gvi) with a web server target in your web application project.
  2. Add your JavaScript reference to the diagram.
  3. In the Configuration pane, click **Change to Type Definition**.

Related concepts:

- Creating UI Elements with JavaScript

JavaScript Resources

Research JavaScript concepts using NI-recommended resources.

The items are some concepts you may want to research when using JavaScript Library Interfaces (JSLIs) and calling JavaScript functions in your web application. Each item links to the appropriate Mozilla Developer Network (MDN) documentation resource. If the JavaScript concept you want to learn about is not listed, NI recommends you search the MDN JavaScript documentation.

- Asynchronous JavaScript
- Built-in Objects
- Callback Function
- Global Scope
- Immediately Invoked Function Expression (IIFE)
- Objects
- Strict Mode
- TypedArray Object

Hosting a Web Application on a Server

To make your web application available to other users, host your build output on a web server that is accessible to other users.

You can host your web application on any server you choose. This includes local servers and servers in the cloud. Use the following table to choose and set up a server for hosting your web application.

Options for Hosting a Web Application

Required Products	Hosting Option	Use Case	Hosting Instructions
G Web Development Software or NI SystemLink	NI Web Server	You want to manage your own server from within your network, either through G Web Development Software or NI SystemLink.	NI SystemLink and the G Web Development Software both use the NI Web Server. Follow the instructions for Hosting a Web Application on the NI Web Server.
SystemLink Cloud	SystemLink Cloud Server	Choose this server in either of the following cases: • You want to access your web application from the SystemLink Cloud website and on mobile devices without sacrificing security. • You want to avoid placing the server burden on your measurement hardware,	Follow the instructions for Hosting a Web Application on SystemLink Cloud.

Required Products	Hosting Option	Use Case	Hosting Instructions
		such as when using real-time hardware.	
LabVIEW 2013 or later	LabVIEW Application Web Server	Choose this server in either of the following cases: • You want to host a web application directly on a LabVIEW real-time target, such as a CompactRIO. • You want to use the security features provided by LabVIEW Web services.	Follow the instructions for Hosting a Web Application on the G Web Development Software Web Server.
3rd party software	3rd party server	You need functionality that is not included in the NI server options.	Before you begin, you must build a web application. Copy your entire web application build output to the server directory. To navigate to your web application output on your machine, click **Locate directory in Windows Explorer** on

Required Products	Hosting Option	Use Case	Hosting Instructions
			the Document tab of your web application component document. **Note** You may need to configure a 3rd party server to support the application/wasm MIME type for files with the `.wasm` file extension.

Related tasks:

- Building a Web Application

Hosting a Web Application on the NI Web Server

To make your application available to other users, host your build output on a web server that is accessible to other users.

Before you upload your application or WebVI on NI Web Server, you must complete the following tasks:

- Remove the URL, username, and password from the panel and the diagram to maximize the security of your web application. Refer to the ***Hosting Authentication Credentials Securely*** section of Security in NI Web Technology for more information.

NI SystemLink and G Web Development Software both use NI Web Server. Complete the following steps to host your web application on NI Web Server.

1. Open your web application project (.gwebproject), navigate to **SystemDesigner**, and open **Design** view.
2. Select **Web Server** on the diagram.
   In the configuration pane under **Item » NI Web Server**, you will find the NI Web Server root directory path. The root directory path populates SystemDesigner based on your configuration settings in the NI Web Server Configuration Utility. Here is an example of a root directory path: `C:\Program Files\National Instruments\Shared\Web Server\htdocs\`.
3. Copy your entire web application output directory into the web server directory. To navigate to your web application output on your machine, click **Locate item in Windows Explorer** on the Document tab of your web application component document.
4. Open a web browser and navigate to `http://localhost/WebApp_Web%20Server/Main.html`, localhost is the IP address of the server and `Main.html` is the file name of the top-level WebVI in your web application.

Related information:

- Choosing Remote Settings

Hosting a Web Application on the NI Application Web Server

To make your web application available to other users, host your build output on a web server that is accessible to other users.

Before you begin, you must complete the following tasks.

- Install a version of LabVIEW from 2013 or later.
- Build your web application.

To host your web application on the Application Web Server, choose from the following options.

Option	Description
Host your web application on the Application Web Server through LabVIEW Web Services.	a. In your LabVIEW Web Service, create a public content folder. For more information, refer to integrating static content into a web service. b. Copy your entire web application output directory into the public content folder you created. c. Publish your LabVIEW Web Service. **Note** Download the latest LabVIEW Help for the most up-to-date content.
Manually host your web application files on the Application Web Server.	a. Navigate to the Application web server document root. Example if the 32-bit Application Web Server is enabled: `C:\Program Files (x86)\National Instruments\Shared\NI WebServer\www` Example if hosting on a CompactRIO: `/var/local/natinst/www` b. Copy your entire web application output directory into the Application web server document root. To navigate to your web application output on your machine, click **Locate directory in Windows Explorer** on the Document tab of your web application component document. c. Open a web browser and navigate to `http://localhost:8080/WebApp_Web%20Server/Main.html`, where `Main.html` is the file name of the top-level WebVI in your web application.

Option	Description
	Note You can update the port in the URL, if necessary, to match the Application Web Server HTTP port.

Hosting a Web Application During Development

Host WebVIs running in the LabVIEW NXG editor with NI Web Server or the LabVIEW NXG embedded web server.

NI recommends hosting your WebVI with NI Web Server. NI Web Server helps you avoid additional configurations when you use other APIs NI Web Server hosts, such as SystemLink Tags and Messages. Refer to the NI Web Server Manual for more information.

Use the LabVIEW NXG embedded web server if you are unable to configure NI Web Server on your development PC. If you choose the embedded web server, you cannot connect to APIs hosted on the NI Web Server, such as the SystemLink APIs, unless you configure NI Web Server to support the cross-origin resource sharing (CORS) mechanism. Refer to Choosing Remote Settings to learn more about enabling CORS for NI Web Server.

Note The LabVIEW NXG embedded web server has an origin of http://localhost:<port>. Your operating system assigns the port based on availability and varies between instances of the editor. Any HTTP request made by a WebVI hosted in the editor that does not target a user resource is considered a cross-origin request.

To set your hosting preference in LabVIEW NXG, select **File** » **Preferences** » **Web Server**.

Related concepts:

- Considerations When Accessing Data from Web Services

Hosting a WebVI in SystemLink

Host a WebVI in SystemLink to securely share it with users on the server.

Before hosting a WebVI in SystemLink, create a package (.nipkg) in LabVIEW NXG that contains the WebVI.

1. In SystemLink Web Application, under Data Visualization, click **WebVIs**.
2. Click **Import**.
3. Select the package you want to upload from your local machine.
   The WebVI you import must be 20 MB or smaller.
4. Enter a name and optional description for the WebVI.
5. Select the workspace you want to host the WebVI in.
6. Click **OK**.
7. Click the WebVI to run it.
8. To update an existing WebVI, select the WebVI and click ↑ to upload the updated file.

Users in the workspace you selected who have WebVI permissions can interact with the WebVI you uploaded.

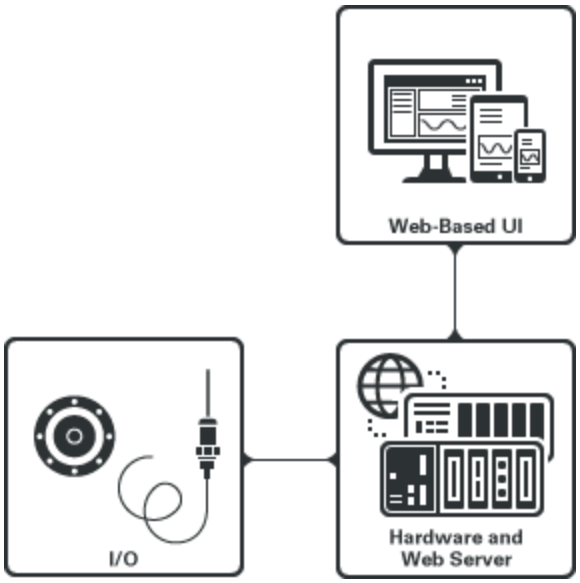
Using Hardware with a Web Application

Use a web application to share measurement data and interact with hardware.

The following diagrams show three architectures for how to use your hardware with a web application. Use the diagrams to choose the best architecture for your application.

Note You can use both real-time and non-real-time hardware devices with your web application.

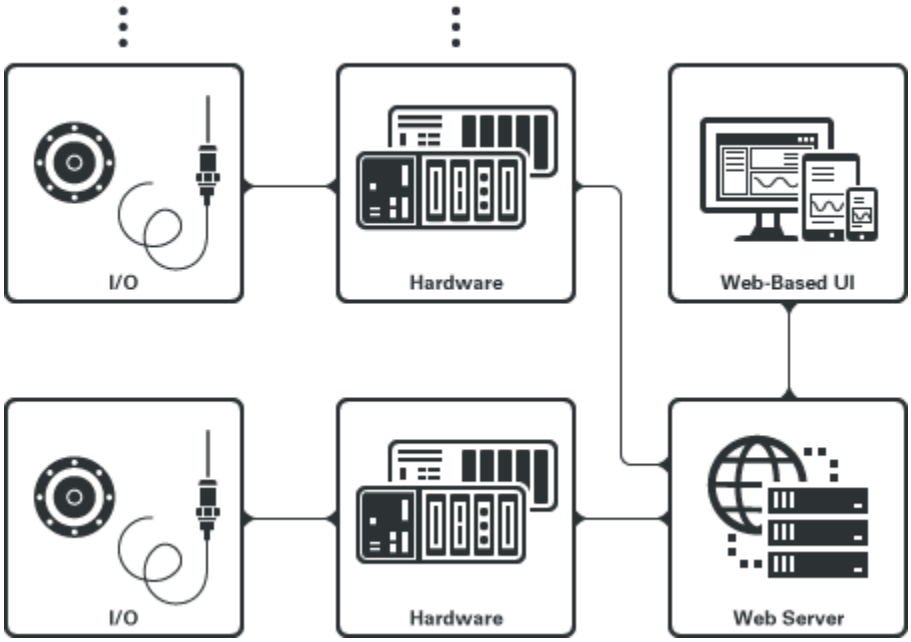
Access a Web Application on Your Hardware



In this architecture, the web service runs on the hardware itself and the hardware also stores the web application. This architecture works best for light-weight web applications, such as browser-based simple configurations and human-machine interfaces for your hardware.

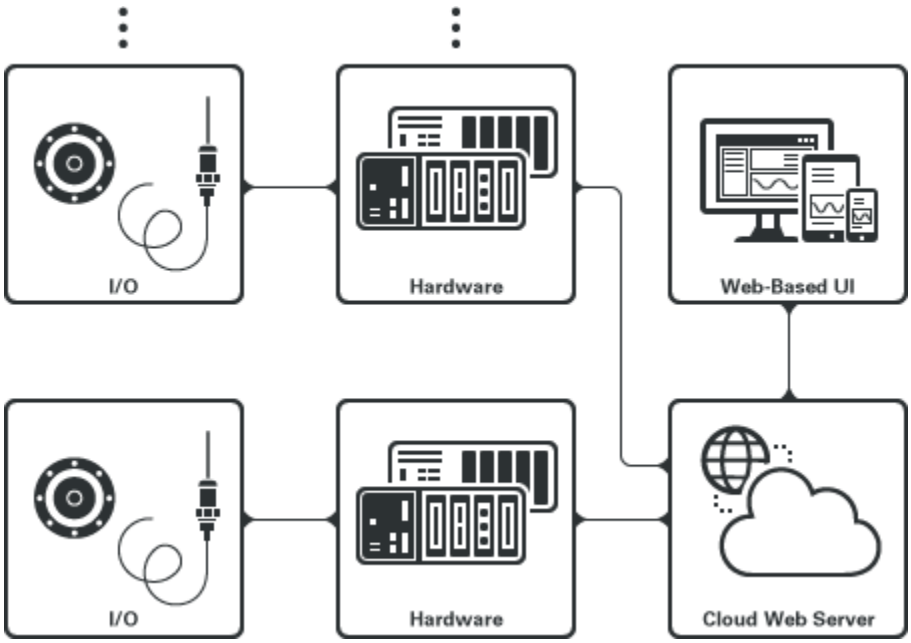
Note You cannot use SystemLink APIs using this architecture.

Access a Web Application on a PC



In this architecture, the web service runs and web application is stored on a PC on the same network as your hardware. This architecture keeps the server burden off of your hardware and works best for sharing measurement data dashboards and applications using SystemLink Tag and Message APIs.

Access a Web Application on the Cloud



In this architecture, your hardware shares data with the web application hosted in the cloud. This architecture works best for accessing the web application from a desktop or mobile device over the internet to securely share your measurement data.

Using a Web Application with a CompactRIO Device

Leverage the following architectures to use your web application with a CompactRIO device.

- Access a web application on your CompactRIO.
- Access a web application on a PC.
- Access a web application on the cloud.

Use the following table to determine the best architecture for your system and web

application.

Architecture	Use case	How to access	How to implement
Access a web application on your CompactRIO.  **Note** You cannot use SystemLink APIs using this architecture.	Light-weight web applications: • Browser-based simple configurations • Human-machine interfaces	Enter the IP address of your CompactRIO in a browser on the same local network as the device.	• Create and Access a LabVIEW Web Service on the CompactRIO. • Host using the LabVIEW Application Web Server. • Configure cross-origin resource sharing (CORS) for your LabVIEW Web Service running on a CompactRIO while developing the web application in the LabVIEW NXG editor.  **Note** NI recommends disabling CORS access to the LabVIEW Web Service when you complete development and deploy the web application to the CompactRIO. For more information about how to protect your web application and data, refer to

Architecture	Use case	How to access	How to implement
			> Security in NI Web Technology.

Access a web application on a PC.	• Measurement data dashboards
	• Use SystemLink Tag and Message APIs
	• Avoid putting the server burden on your CompactRIO
		Enter the IP address of the PC in a browser on the same local network as the PC and CompactRIO.	• Access the data from your CompactRIO device using one of the following options:
			  ◦ Use NI Web Server Data Services, such as SystemLink Tag and Message APIs.
			  ◦ Create and Access a LabVIEW Web Service on the PC.
			• Host using the NI Web Server or a third-party web server.
			• Configure cross-origin resource sharing (CORS) if your development PC and hosting PC are different machines.

Access a web application on the cloud.	• Use a desktop or mobile device to access the web application over the internet • Host and securely share your measurement data with SystemLink	Enter the URL for the web application in a browser connected to the internet.	• Use SystemLink Cloud to share data across systems by directly connecting your hardware or make local tags and messages available using SystemLink Cloud Connector API. • Host using SystemLink Cloud Server to take advantage of the

Architecture	Use case	How to access	How to implement
	Cloud		accessibility and security of SystemLink. • Connect to SystemLink Cloud using an API key. **Note** Refer to SystemLink Cloud FAQ to learn more about SystemLink Cloud requirements and features.

Building Shareable Libraries

Use your web application library to share reusable code.

You can create an add-on for your web application and distribute it in a package using the following steps.

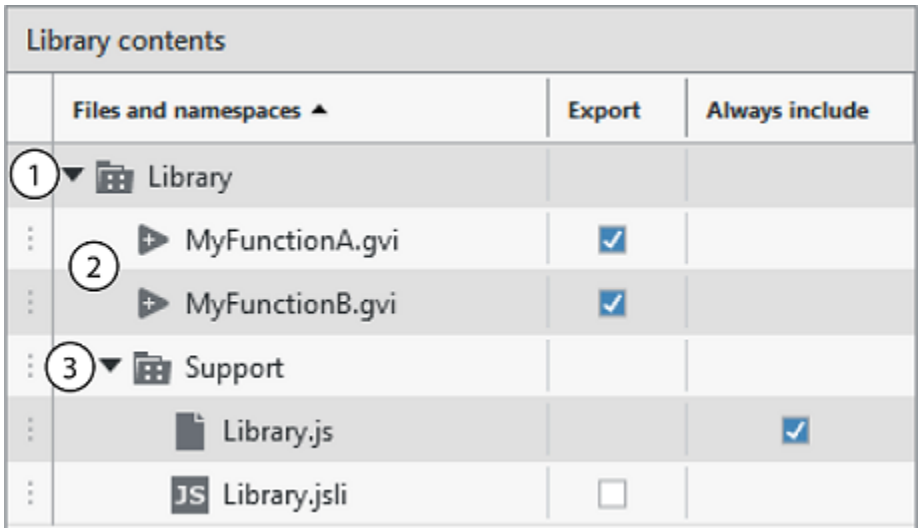
1. Create a library—Add all of the source files for your web application to the library.

   Note Ensure you create the library on the web server target.

2. Add a palette to the library—Add a HTML panel palette and all supporting functions to the web application library.
3. Add custom UI elements to the library—Add a palette with your custom UI elements to the web application library.
4. Package the library—Build the web application library into a package or package installer.

Recommendations for Building a Web Application Library

Consider the following recommendations when building a reusable library for your web application.



1. Use VIs (.gvi) instead of WebVIs (.gviweb)—Use a WebVI as the top-level VI in your web application and VIs with the .gvi file extension for all other VIs and libraries. Using VIs with the .gvi file extension enables you to use your libraries across targets.
2. Wrap JavaScript Library Interface (JSLI) nodes in VI (.gvi) files and uncheck **Export** for JSLI documents—Rather than exporting nodes defined in the JSLI document, wrap the nodes in VI (.gvi) files and ensure they are not accessible outside of the library. The wrapper VI isolates the JSLI nodes from the library user, giving you the flexibility to add functionality, such as supporting additional data types or having multiple outputs.
3. Put JavaScript resources and JSLI in a support namespace—Use a separate namespace for JavaScript resources, such as CSS, and JSLI documents to organize web application specific resources.

If you want to add custom visual elements to your web application library, refer to Adding a Custom UI Palette to Your Web Application Library.

Adding a Custom UI Palette to Your Web Application Library

Add custom UI elements you created with the Placeholder HTML Container to a palette

in your web application library.

Before you create a palette for your custom UI elements, complete the following tasks:

- Enable the preview feature—Navigate to **File » Preferences » Preview Features** and in the Web Module section, select **Enable Placeholder HTML Containers in WebVI library palettes** to enable this feature. Refer to Preview Features to learn more about LabVIEW NXG preview features.
- Create UI elements—Use the Placeholder HTML Container and Obtain JavaScript Reference node to create custom UI elements for your web application.

Use a control definition file to make the custom UI elements you create available in the palette when someone installs your web application add-on.

1. Create or open a library in your web application project.

   Tip Make sure this library has an extension of `.gcomp` and has a Web Server target.

2. Create a control definition file and add it to the library.
3. Add your Placeholder HTML Container to the control definition file and configure how you want it to appear when placed on the front panel.

   Note If you use the Image URL option, you cannot use an image file in the library at deployment. You must use an image URL from the public internet or a self-contained data URL. If you try to reference an image file in the library component, it may not resolve in the correct location once deployed.

4. In the library with the control definition file selected, click **Create palette file** from the configuration pane.
5. Select **G HTML panel palette** for the palette type.
6. (Optional) Use the configuration pane to configure your palette.