Zebra
Aurora™ Vision

Aurora Vision Studio 5.7

WebHMI

Created: 11/10/2026

Product version: 5.7.5.80527

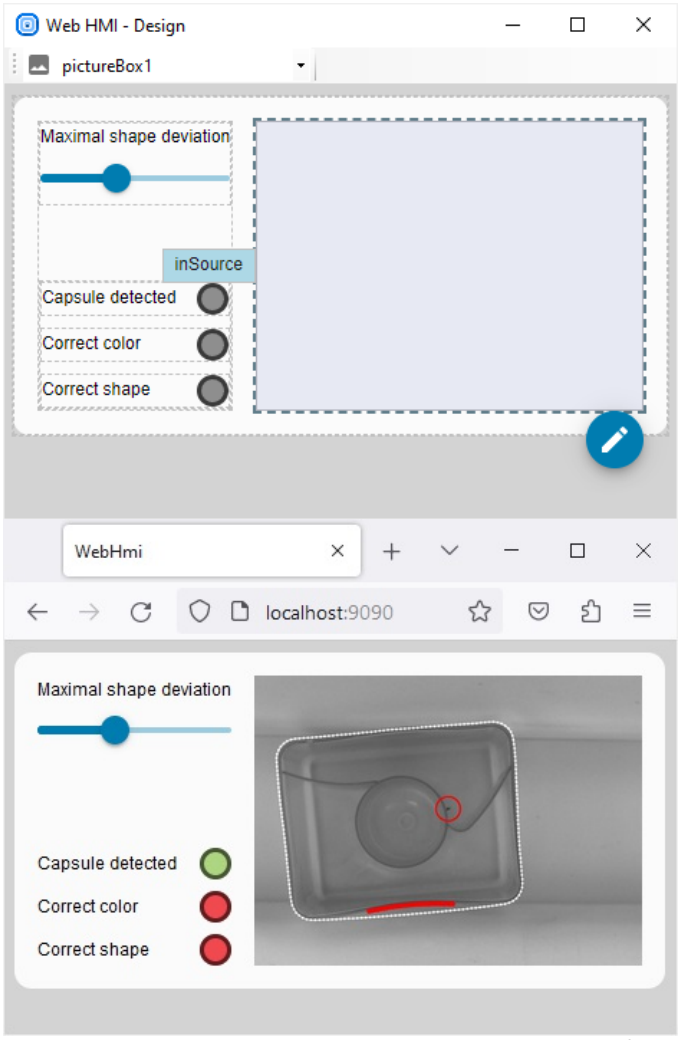
Table of content:

- Designing WebHMI
- Handling WebHMI Events
- Authorization in WebHMI
- Securing WebHMI with SSL

Designing WebHMI

Introduction

Aurora Vision Studio, apart from the classic desktop HMI, allows users to create an interactive webpage version of the HMI which can be displayed in any web browser. The webpage is created within Aurora Vision Studio IDE in the WYSIWYG-like editor by manipulating the WebHMI controls on the page.



A simple WebHMI example: at the design time (left) and at run time displayed in the browser (right).

The WebHMI page is composed from the WebHMI controls. Some of the controls may contain other controls to build the hierarchy, e.g., the StackPanel and the TablePanel provide specific layout types, whereas the TabControl allows to arrange controls in different tab pages. At the top of the control hierarchy is the Window control, which may contain any single control.

The appearance of controls can be modified using the properties listed in the *Properties Control* when the control is selected and the WebHMI designer tab is active.

Communication with Aurora Vision application is achieved through connections with control ports and control events:

- From a filter's output to a control's input, e.g., for displaying an image or text result.
- From a control's output to a filter's input, as data sources for setting various parameters in a program with e.g., with sliders or checkboxes.
- With a special kind of macrofilters executed in response to some action on the control, e.g., Button.Click.

Note: Communication between Aurora Vision Studio and the WebHMI frontend application is unsecured as it is transmitted over HTTP and WebSockets, both of which lack encryption. Therefore, we advise against making WebHMI accessible on public networks. It is possible to secure the WebHMI either by enabling built-in SSL support (see Securing WebHMI with SSL) or through the use of a proxy server, such as nginx.

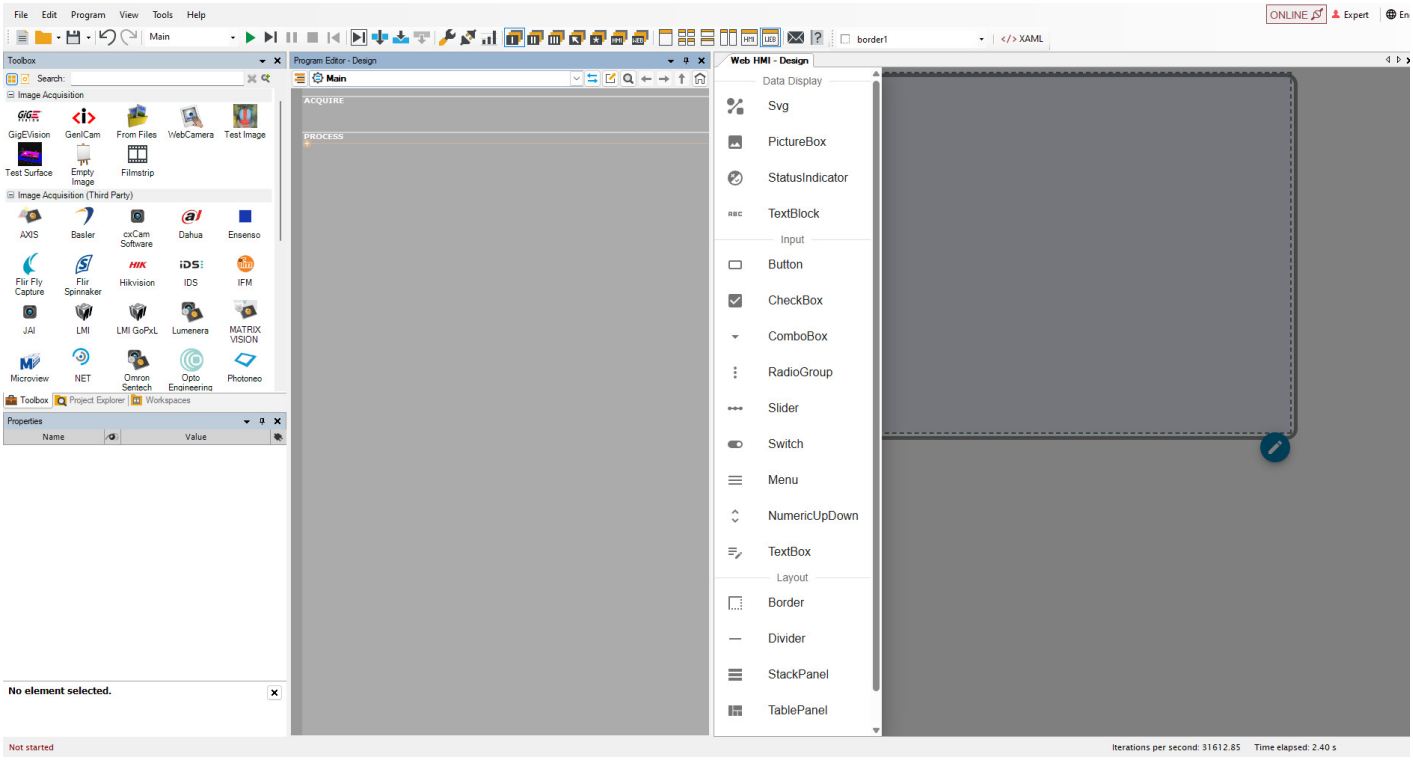
Adding the WebHMI to the Project

To add a WebHMI subsystem, choose *Edit » Add Web HMI...* or click the *WebHMI* button on the Toolbar. Then, select a WebHMI template that fits the target layout or choose the target layout. The template is just a starting point that can be freely modified. Modifications include:

- Adjusting selected control properties in the *Properties Control*
- Content modifications through the
  
  action button, e.g., adding content, removing selected control, etc.

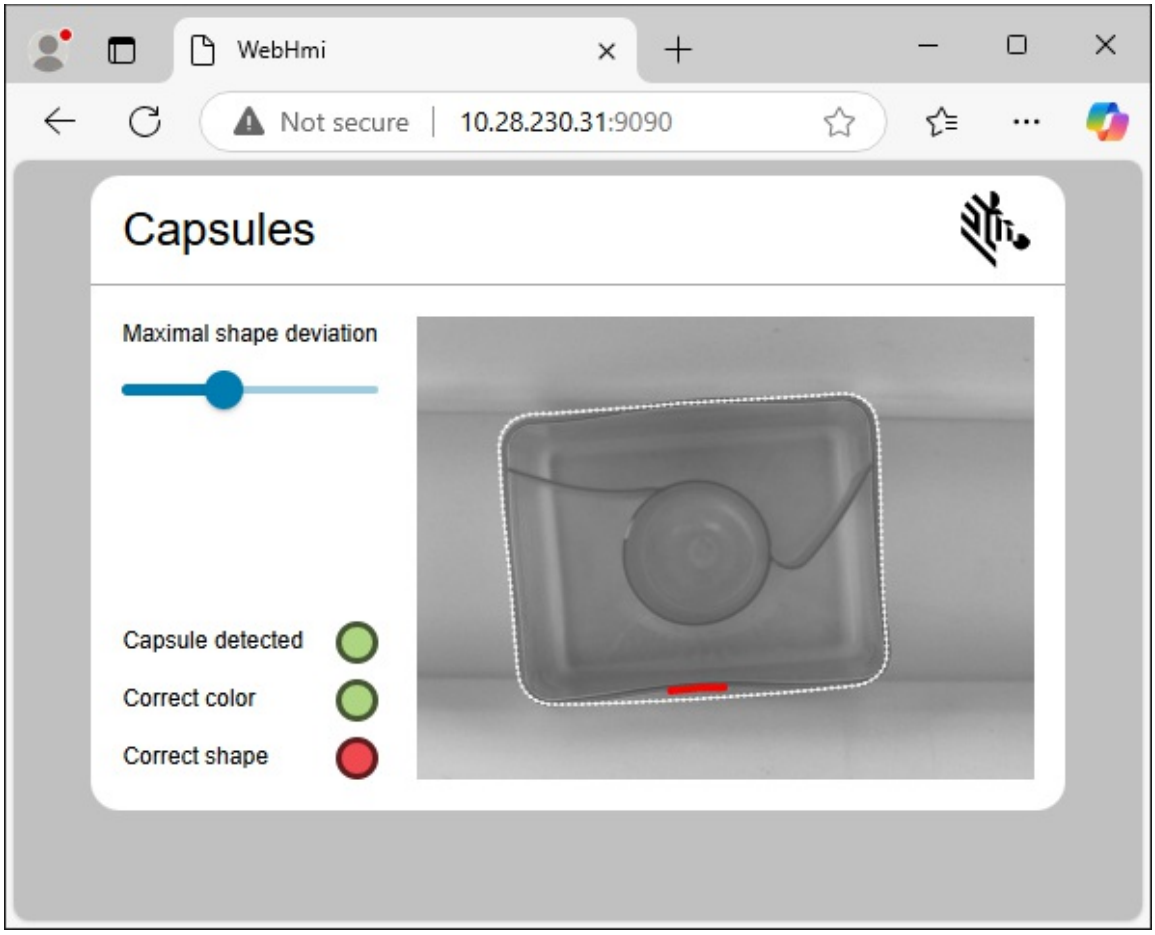
In particular, to set a PictureBox content for the Border control:

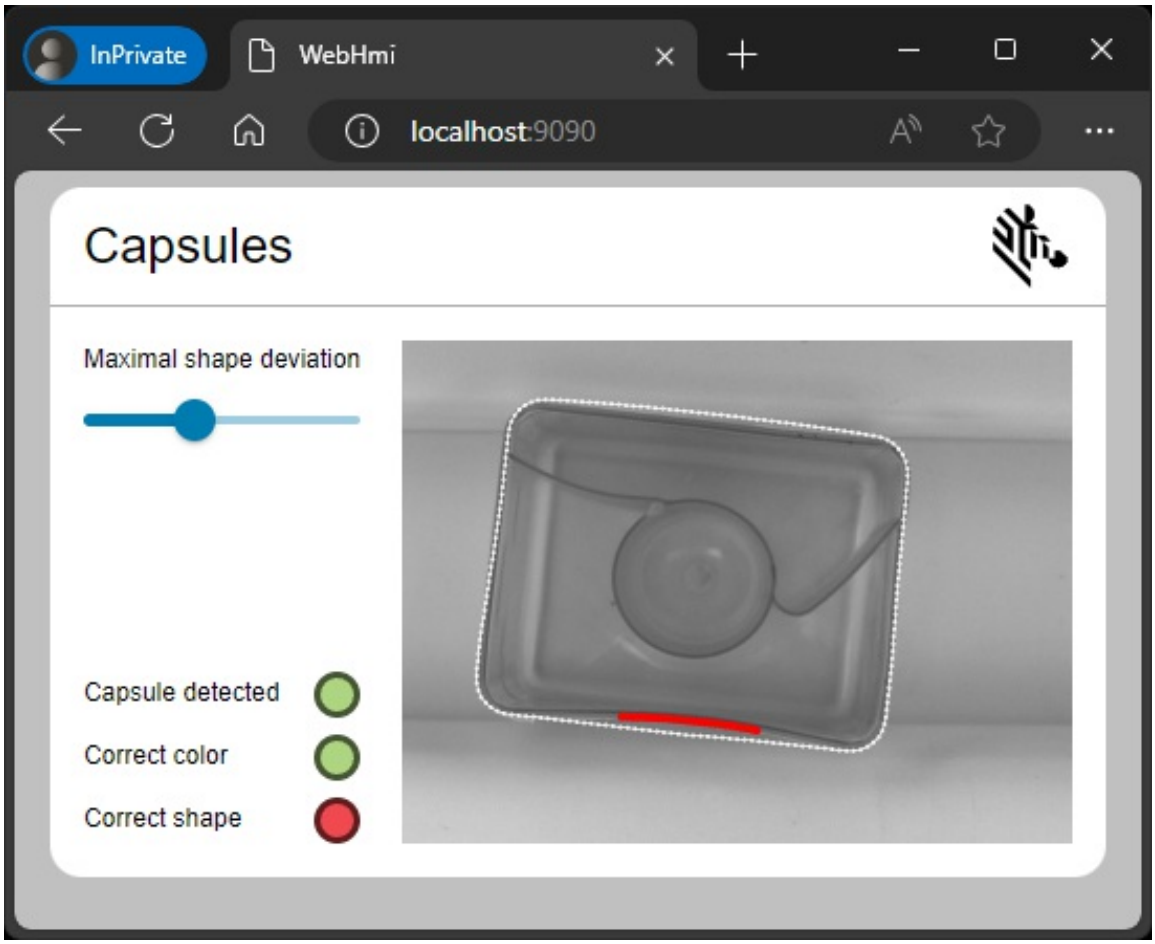
1. Select the Border control,
2. Click the
   
   action button
3. Click the *Set Content...* command
4. Select the PictureBox control from the control list



WebHMI Display

To display the WebHMI, open your web browser and, in the address bar, type the IP address of the PC running the Aurora Vision application, followed by the WebHMI Server Port number.
If you want to display the WebHMI in a web browser on the PC that runs the application, you may type ***localhost*** instead of the IP address.



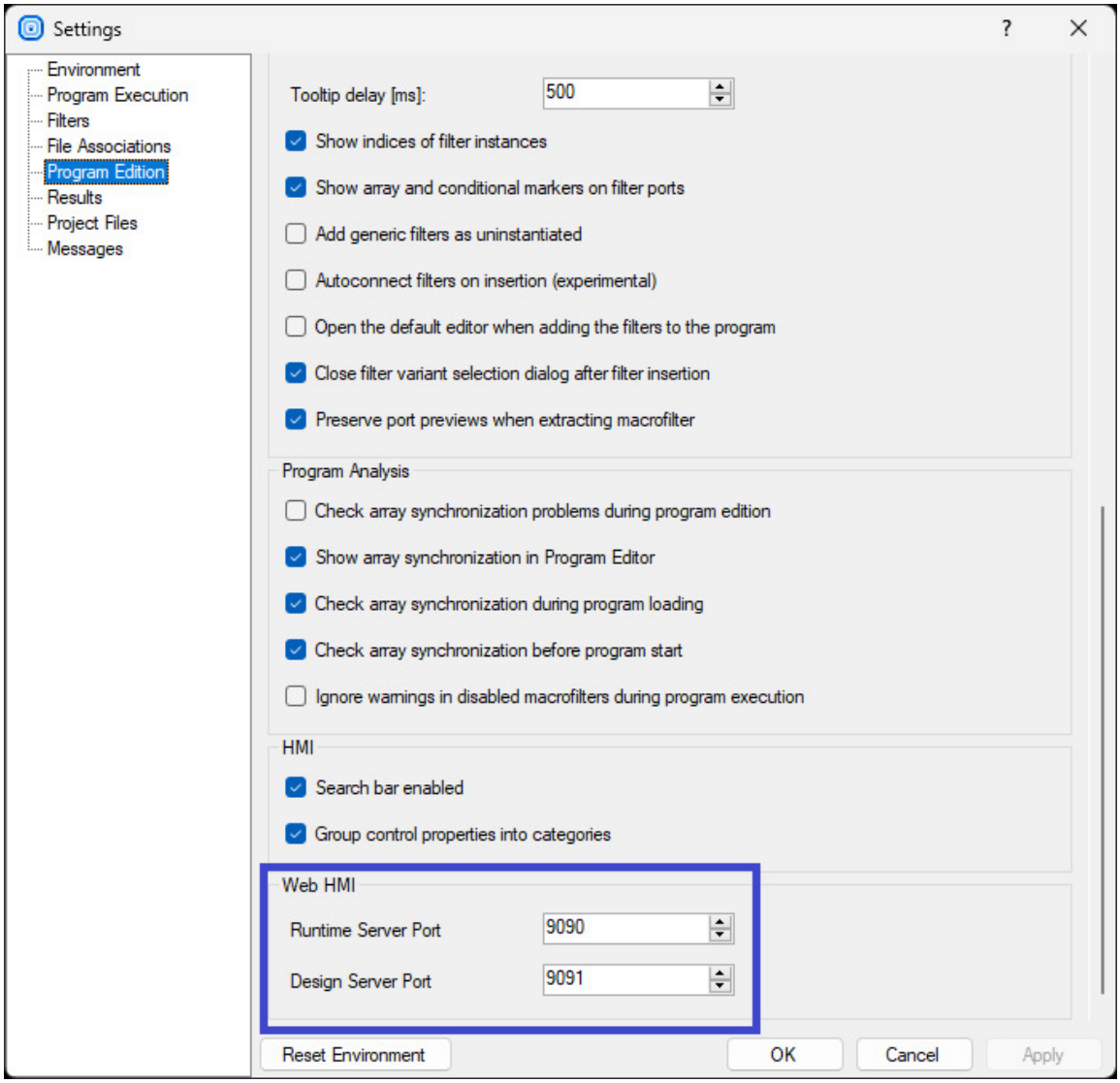


The number of the WebHMI Server Port is available in *Tools » Settings » Program Edition » Web HMI* section.

The default values are **9090** for the **Runtime Server** and **9091** for the **Design Server**.

There are two variants of the WebHMI server: **Designing** and **Runtime**.
The **Designing** server allows the user to access WebHMI design mode via a web browser. The controls can be modified, and it displays the available ports that can be connected to the program.

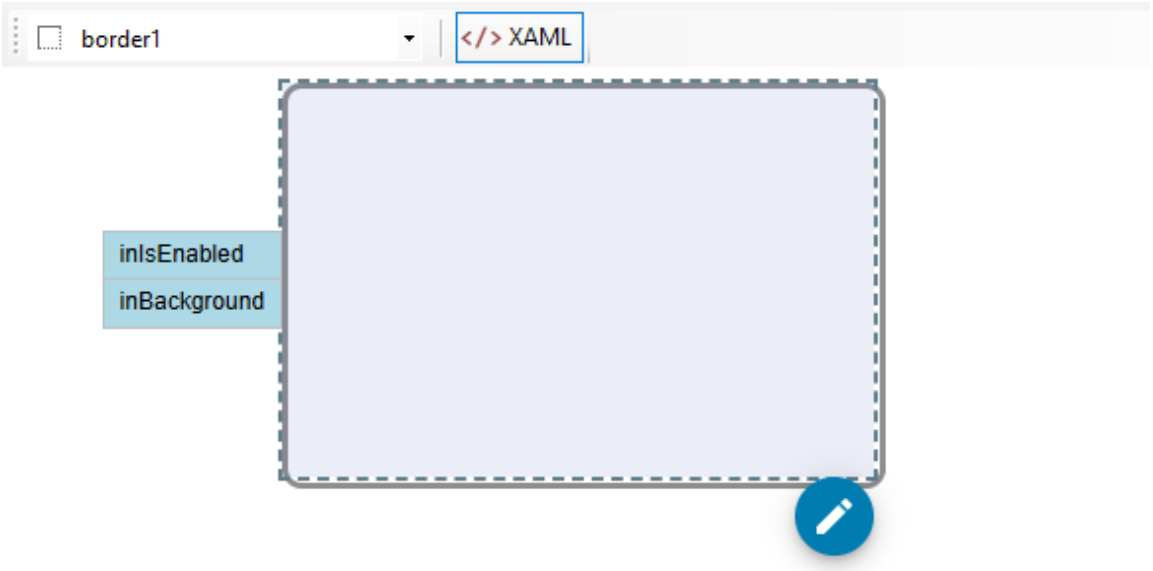
The **Runtime** server presents the WebHMI during program execution. It doesn't allow modification of the WebHMI layout, permitting only the intended interaction with the controls during runtime.



WebHMI Code view

You can modify the WebHMI view and its controls directly in its code, not only in the **Properties** window.

To show the code, click on the **XAML** button on the Toolbar.




```
<Window
    Name="window">
    <Border
        Width="300px"
        Height="200px"
        Margin="auto"
        BorderBrush="#8E8E8E"
        BorderStyle="Solid"
        BorderThickness="3px"
        CornerRadius="10px"
        Name="border1" />
</Window>
```

Removing the WebHMI

The WebHMI subsystem can be removed from the project with the *Edit » Remove Web HMI* command.

Handling WebHMI Events

Introduction

The WebHMI mechanism in Aurora Vision Studio operates on a loop basis, that is, at the beginning of each iteration, all the inputs from the WebHMI are read by the filters and macrofilters that are connected to those inputs. However, the results coming from those filters and macrofilters will not in turn be output to the WebHMI until after the given iteration has been fully executed. The WebHMI's workflow is thus analogous to that of the basic HMI (consult this entry).

It is very convenient to use in typical machine vision applications focused on fast image acquisition loops and where the WebHMI is mainly used for setting parameters and observing inspection results. In many applications, however, it is not only required to use parameters set by the end user, but also to handle events such as button clicks.

To perform certain actions immediately after an event is raised, Aurora Vision Studio also comes with a WebHMI subsystem that handles events independently of the program execution process. This allows to create user interfaces that respond almost in real-time to user input, e.g., clicking a button, moving the mouse cursor over an object or changing an associated value. None of these actions would be possible were events handled solely within the workflow dictated by the program execution process.

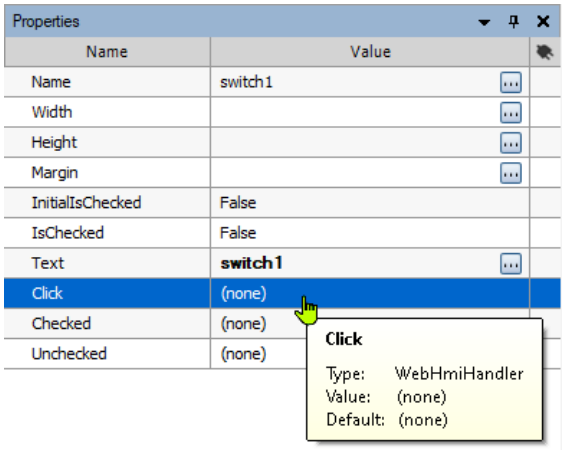
Event Handlers

An event handler is a subprogram in Aurora Vision Studio that determines behaviors to be implemented when an associated event occurs. It is in fact simply a macrofilter that is executed once for each received event. If there are several WebHMI controls which should trigger the same procedure, one event-handling macrofilter can be used for all of them.

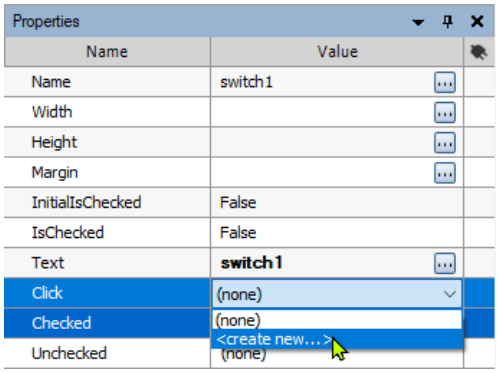
Creating Event Handlers

To create an event handler:

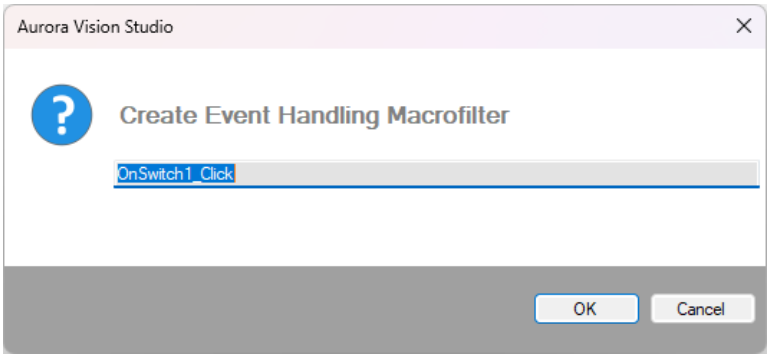
1. Left-click the WebHMI control that will be the source of an event and look up its properties. To find which controls have defined events, refer to the Standard WebHMI Controls entry.
2. In the properties, you will find WebHmiHandlers:



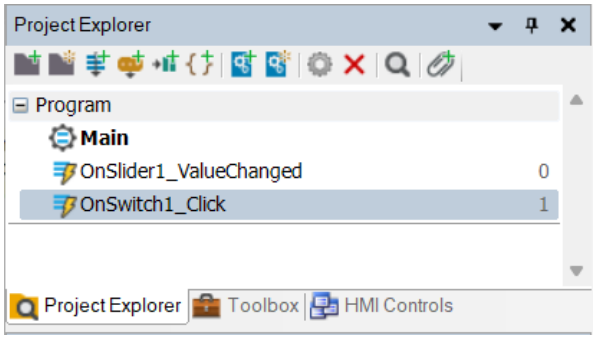
3. Click the drop-down list next to the event name and select <create new...>, as shown below:



4. Define the name of the event handler in the pop-up window:



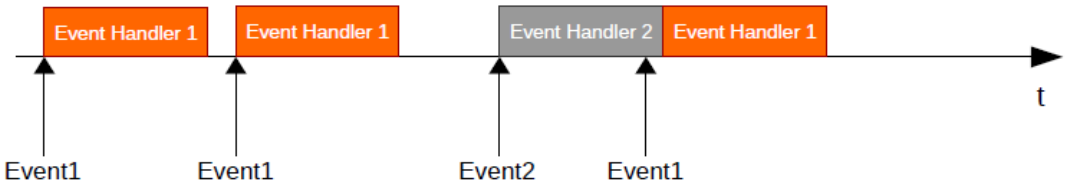
5. Once the event handler has been created, it becomes available in the Project Explorer:



Queuing Event Handlers

To create applications that handle events efficiently, it is important to understand the mechanism of event-handling macrofilter execution. Especially the way how the aforementioned WebHMI subsystem queues them when several events come at short intervals.

When a WebHMI control sends an event to signal the occurrence of an action, the WebHMI subsystem retrieves the event and stores it in a FIFO queue. This queue is then processed in a separate thread dedicated to handling WebHMI events.



Sample events sequence and timeline showing the order in which they are handled

This has important implications:

- Event handlers should not contain any time-consuming algorithms that could delay the execution of subsequent events.
- Events are always handled one by one. Therefore it is essential to make event-handling procedures as short as possible.
- All event handlers are executed in parallel to the main program loop, but no two event handlers are executed at the same time.
- If you use thread limit-consuming data analysis filters in event handlers, they will be considered additional thread limit-consuming data analysis threads and may thus report exceeding the license restrictions.

Authorization in WebHMI

1. WebHMI Display
2. Introduction
3. Key Concepts
    1. Authorize Property on Windows
    2. Login Endpoint
    3. Authorization Providers
4. Setting Up Authorization - Step by Step
    1. Step 1: Use the Login Template
    2. Step 2: Create Protected Windows
    3. Step 3: Configure OAuth Providers
5. Authorization Flow
6. User Information in Windows
7. Security Guidelines
8. Summary
9. Introduction
10. Prerequisites
11. Enabling SSL - Step by Step
12. Accessing WebHMI over HTTPS
    1. Automatic Redirect to HTTPS
13. Troubleshooting
14. Security Guidelines

Introduction

Aurora Vision WebHMI provides authorization to protect sensitive windows. The system supports two authentication methods: **Local Authentication** (username/password managed by the application) and **OAuth 2.0** (delegated to Google, Microsoft, etc.).

Key Concepts

Authorize Property on Windows

The **Authorize** property on a Window control determines whether authentication is required. When set to true, unauthenticated users are redirected to the login endpoint. Windows with Authorize="false" remain public.

Login Endpoint

The Login Endpoint is a public window where users authenticate. It displays the login form or OAuth buttons and handles redirection to protected content after successful authentication.

Authorization Providers

Two provider types are available: **Local Provider** (validates credentials internally) and **OAuth Provider** (delegates to Google, Microsoft, etc.).

Setting Up Authorization - Step by Step

Step 1: Use the Login Template

When adding a new WebHMI to the project, select the **Login** template from **Edit » Add Web HMI...**

The template automatically creates:

- A login window with pre-configured controls for Local Authentication (username/password input)
- Error message display for failed authentication attempts
- OAuth buttons for multiple providers (Google, Microsoft, GitHub, etc.)
- Pre-configured LocalAuthProvider and OAuthProvider components in the application

The login window is set to Endpoint="login" and does not have Authorize="true", making it publicly accessible for all users.

Step 2: Create Protected Windows

Create additional windows for your application's protected content and set their Authorize property to true. When users try to access these windows without authentication, they are automatically redirected to the login endpoint.

Step 3: Configure OAuth Providers

If the Login template was used in Step 1, the authorization structure is already pre-configured in the application:

- **LocalAuthProvider** is ready for local username/password authentication
- **OAuthProvider** elements are pre-configured for common providers (Google, Microsoft, GitHub)
- **LoginEndpoint** is set to login

To enable OAuth authentication, configure each provider with the required credentials:

- **ClientId** – Obtain from your OAuth provider's developer console
- **ClientSecret** – Obtain from your OAuth provider's developer console
- **RedirectUri** – Set to your application's callback URL (e.g., http://localhost:9090/auth/callback)

Authorization Flow

1. User visits a protected window
2. System checks for valid authentication token
3. If unauthenticated, user is redirected to the login endpoint
4. User authenticates via Local or OAuth
5. Token is issued and user is redirected to original destination
6. Protected content is displayed

When the token expires or the user logs out, re-authentication is required to access protected windows.

User Information in Windows

After authentication, the Authorization system tracks:

- **CurrentUserName** – Display name of the authenticated user
- **LogoutUrl** – URL for signing out

Bind these to controls in the designer. Example:

```
<TextBlock Text="{Binding #authorization.CurrentUserName, StringFormat='Welcome, {0}!'}" />
```

Security Guidelines

- **HTTPS only in production** – All communication must be encrypted; see Securing WebHMI with SSL for setup instructions
- **Provide logout buttons** – Allow users to sign out and clear session tokens
- **Token expiration** – Tokens expire after a configured time period; re-authentication is then required
- **Selective protection** – Use Authorize="true" only on windows that require authentication
- **Secure WebSockets** – Use WSS (WebSocket Secure) instead of unencrypted WS

Summary

Authorization in WebHMI requires four steps:

1. Create a public login window at the configured endpoint
2. Set Authorize="true" on protected windows
3. Configure Authorization settings in the App control (Local or OAuth provider)
4. Optionally bind CurrentUserName or LogoutUrl to controls

Securing WebHMI with SSL

1. WebHMI Display
2. Introduction
3. Key Concepts
    1. Authorize Property on Windows
    2. Login Endpoint
    3. Authorization Providers
4. Setting Up Authorization – Step by Step
    1. Step 1: Use the Login Template
    2. Step 2: Create Protected Windows
    3. Step 3: Configure OAuth Providers
5. Authorization Flow
6. User Information in Windows
7. Security Guidelines
8. Summary
9. Introduction
10. Prerequisites
11. Enabling SSL – Step by Step
12. Accessing WebHMI over HTTPS
    1. Automatic Redirect to HTTPS
13. Troubleshooting
14. Security Guidelines

Introduction

By default, Aurora Vision WebHMI communicates over plain HTTP and unencrypted WebSockets, which means the data exchanged between the browser and Aurora Vision Studio can be read or tampered with by anyone able to observe the network traffic. To protect this communication, Aurora Vision Studio can enable SSL/TLS for the WebHMI Runtime server, so that the WebHMI is served over HTTPS instead.

Prerequisites

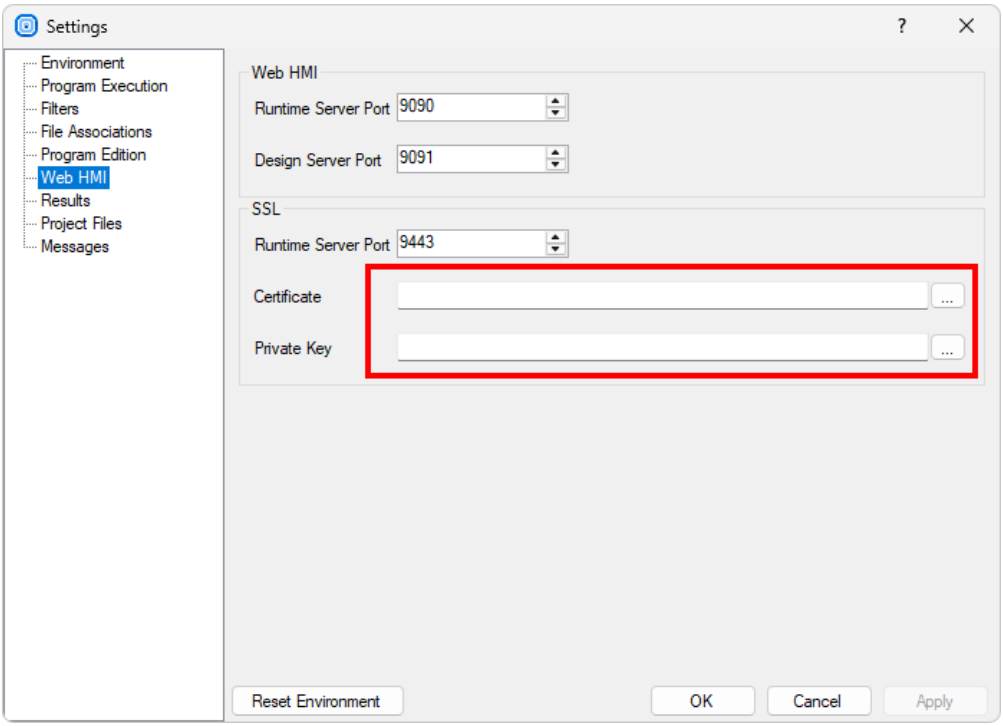
To enable SSL, you need two files:

- **Certificate** – a PEM-encoded certificate file. If your certificate was issued by an intermediate Certificate Authority (CA), this file should contain the full certificate chain.
- **Private Key** – the PEM-encoded private key matching the certificate.

A self-signed certificate can be used for testing, but web browsers will display a trust warning for it. For production use, obtain a certificate issued by a trusted Certificate Authority.

Enabling SSL – Step by Step

1. Open *Tools » Settings » Web HMI*.
2. In the **SSL** section, set the **Runtime Server Port** that the HTTPS server should listen on (the default is **9443**).
3. Click the ... button next to **Certificate** and select your certificate file.
4. Click the ... button next to **Private Key** and select your private key file.
5. Click **Apply** or **OK**.



Both the certificate and the private key must be provided for SSL to be enabled. If either of them is missing, the WebHMI Runtime server starts without SSL support.

Accessing WebHMI over HTTPS

Once SSL is enabled, the WebHMI Runtime server can be reached over HTTPS by typing https://<address>:<ssl-port> in the browser's address bar, using the SSL Runtime Server Port configured above. SSL applies only to the **Runtime** server; the **Design** server used while editing the WebHMI in Aurora Vision Studio is not affected. See WebHMI Display for more information about the Runtime and Design servers.

Automatic Redirect to HTTPS

When SSL is enabled, the WebHMI Runtime server automatically redirects requests made to its regular HTTP port to the corresponding HTTPS address on the SSL port (an HTTP 307 redirect). This means that once SSL is configured, opening http://<address>:<port> in a browser takes you to the secure https://<address>:<ssl-port> page automatically, without needing to update the address manually.

Troubleshooting

If the configured certificate or private key file cannot be found, Aurora Vision Studio reports an error naming the missing file when the WebHMI server starts. Verify that the paths configured in the Settings dialog point to existing files and that the certificate and private key are PEM-encoded.

Security Guidelines

- **Use a CA-issued certificate in production** – self-signed certificates should only be used for testing.
- **Protect the private key file** – restrict access to the private key file on disk to trusted users.

Zebra
Aurora™ Vision

This article is valid for version 5.7.5

Legal Terms of Use Privacy Policy

ZEBRA and the stylized Zebra head are trademarks of Zebra Technologies Corp., registered in many jurisdictions worldwide. All other trademarks are the property of their respective owners. ©2026 Zebra Technologies Corp. and/or its affiliates.